Hazard-Free Implementation of Speed-Independent Circuits

Alex Kondratyev, *Senior Member, IEEE*, Michael Kishinevsky, *Senior Member, IEEE*, and Alex Yakovlev, *Member, IEEE*

Abstract— **This paper develops a theoretical framework for the hazard-free gate-level implementation of speed-independent circuits specified by event-based models, such as signal transition graphs (for processes with AND causality and input choice) or their extension, called change diagrams (which allow OR-causality). It presents sufficient conditions, called the generalized monotonous cover requirements, for a hazard-free circuit to be built within a standard implementation structure. This structure consists of two-level simple-gate combinational logic and a row of latches, either a C-element or an *RS*-latch. A set of semantic-preserving transformations is defined that can be applied to an original behavioral description of the circuit so as to produce its specification in the form that satisfies the monotonous cover requirement. The transformations are applied at the event-based representation level (to avoid state explosion) and proved to be effective. The main result of the paper is therefore twofold: 1) the proof that any speed-independent behavior can be implemented at the gate level without hazards and 2) an efficient method for constructing such an implementation. Experimental results show that the proposed method compares very favorably, in area and performance, to the previously known techniques.**

Index Terms— **Asynchronous circuits, gate-level implementation, hazard freedom, logic synthesis, monotonous cover, speed independence.**

I. Introduction

AUTOMATED synthesis of asynchronous, or self-timed, circuits has recently become an important issue in the list of problems confronting the very large scale integration (VLSI) computer-aided design (CAD) community. The scope of application for asynchronous circuits is increasing due to a number of *potential* advantages:

- greater modularity since one can design, reuse, and maintain components one at a time;
- no problems with clock signal skew and thus no area or time penalty for the fast and reliable driving of clock lines since the system does not use the clock at all;
- operational scalability, that is, if one part works slower, the system slows down but does not fail;
- robustness to parameter variations, for example, temperature and supply voltage [24];
- reduction of power dissipation due to the absence of clock activity and performing signal transitions only when they are needed [35].

One approach for the design of asynchronous circuits is to rely upon known *bounds on gate/wire delays* and/or to use a restricted input-output signaling protocol [14], [20], [31], [34], [43]. A typical example of the latter is the *fundamental mode* protocol, where a new input pattern may be applied to the circuit only when the circuit is completely settled after the previous pattern. This assumption implies the imposition of certain relational constraints on the response delay of the environment.

In contrast, *speed-independent circuits* [15], [23], [28] rely on the *unbounded delay model*, which uses pessimistic assumptions for gate delays (no bounds are known) and realistic assumptions for wire delays (the skew of wire delays at multiple fan-outs is less than one gate delay). These circuits do not impose global constraints on the environment. Instead, each input change is *acknowledged* by the circuit to indicate that the environment is allowed to apply the next input pattern. Therefore, speed-independent circuits are more *robust* to irregular parameter variations (process changes, voltage and temperature variations, noise). Speed-independent circuits enjoy the property of being *self-checking to stuck-at faults* [26], [38] and are *easier to verify* than bounded delay circuits [12], [15], [20].

A. Problem Description

An event-based model, called the signal-transition graph (STG), has become popular as the specification language for synthesis of asynchronous circuits [4], [15], [20], [27], [37], [41]. A simple example of the STG is shown in Fig. 1(a). It defines causal relations between rising (denoted with $+$) and falling (denoted with $-$) signal transitions in the circuit by means of arrows. For example, the rising transition of signal x has two fan-in arrows $-a \rightarrow +x$ and $+d \rightarrow +x$, which mean that transition $+x$ can only occur when *both* transition $-a$ *and* transition $+d$ have occurred. The dots (●) placed on the arcs indicate the particular initialization of the circuit. Such a representation of the circuit behavior can be viewed either as an interpreted Petri net [13] or as a formalization of timing diagrams [20].

If an STG specification satisfies certain correctness criteria (discussed in Section II-B), one can derive a state graph from

Manuscript received December 8, 1994; revised April 22, 1998. The work of A. Kondratyev was supported in part by the Engineering and Physical Sciences Research Council under Grant GR/L24038. This work of A. Yakovlev was supported by the Engineering and Physical Sciences Research Council under Grants GR/J52327/K70175 and GR/K70175/L28098. This paper was recommended by Associate Editor A. Saldanha.

A. Kondratyev is with the University of Aizu, Aizu-Wakamatsu 965 Japan.
M. Kishinevsky is with the Strategic CAD Lab, Intel Corp., Hillsboro, OR 97124 USA.
A. Yakovlev is with the University of Newcastle upon Tyne, Newcastle NE1 74U England.

Publisher Item Identifier S 0278-0070(98)06765-7.

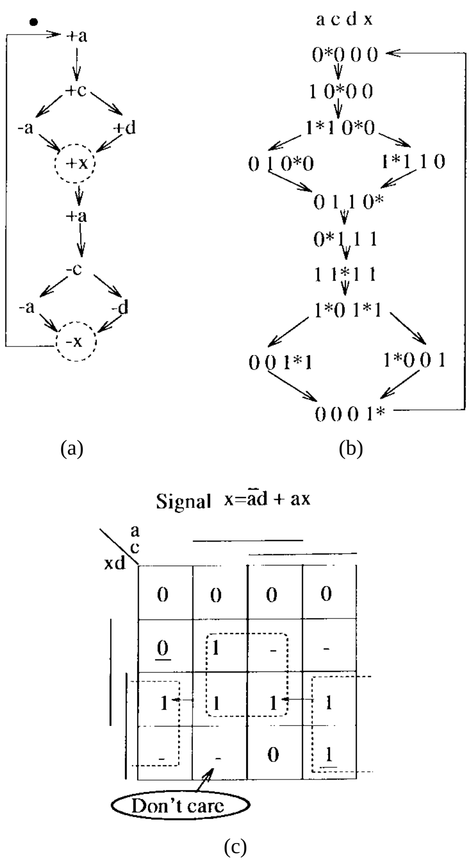


Fig. 1. (a) A signal-transition graph, (b) the corresponding transition diagram, and (c) the Karnaugh map for signal x.

such an STG. The state graph, called a transition diagram (TD) following the tradition of [26], and [28], has vertices, called states, which are labeled with binary codes, and arcs between the states that correspond to the allowable signal transitions. A TD corresponding to the STG from Fig. 1(a) is shown in Fig. 1(b). To indicate that a signal can perform a transition the signal's value is marked with an asterisk (0* and 1*).

If any two states of a TD have different Boolean codes or if states with the same code output signals have the same next value, i.e., the complete state coding requirement is met (see Section III-A), then the TD defines an *incompletely specified logic function* for each output signal. The following three sets of Boolean vectors can be derived from the TD: the *on-set,* the set of states with the signal value equal to 1 or to 0*, where the function evaluates to 1; the *off-set,* the set of states with the signal value equal to 0 or 1*, where it evaluates to 0; and the *dc-set* (don't care set), the set of states that are not reachable from the initial state and wherein the function is not specified. From the Karnaugh map given in Fig. 1(c), one can derive the logic function $x = ax + \overline{a}d$.

The next stage of synthesis is technology mapping into standard gate cells. The function for signal x may not be implementable with one standard logic gate due to three major reasons.

1) Signal a occurs in the function both in the inverted and noninverted form. This requires an extra inverter for delivering these two values. The use of extra inverters can introduce logic hazards at the output of the cell that implements signal x when signal a performs rising or falling transitions (shown by the arrows inside the Karnaugh map in Fig. 1(c).
2) The function for x is self-dependent, i.e., it depends on x (itself) and consequently requires a feedback. An actual implementation of feedback structures must involve at least two negative gates inside the feedback loop and is typically based on latches.
3) In general, the logic function for x may be too complex for one library cell. Due to the problem of logic hazards, decomposing "large" logic functions into "smaller pieces" is nontrivial.

It was proved in [38] that any logic function derived for a speed-independent specification can be decomposed into an RS-latch, built of two *complex* gates, without hazards. Fig. 2(a) shows a Karnaugh map for the S_x function of an RS-latch that implements signal x in a dual-rail form, and Fig. 2(b) shows the RS-implementation of signal x. Although the S_x function implemented as a cube $\overline{a}d$ can change its value nonmonotonously $1 \rightarrow 0 \rightarrow 1$ in the state sequence 0110* → 0*111 → 11*11 → 1*01*1 → 001*1, as shown in the Karnaugh map, this does not imply hazards at the outputs of the RS-latch. Indeed, the stable "0" is held at the output of the complex gate $\tilde{x}$ by the "1" from the x output of the latch when cube $\overline{a}d$ is changing its value. [A complementary metal–oxide–semiconductor implementation of the complex gate for $\tilde{x}$ is shown in Fig. 2(c).]

However, a simple gate implementation based on a trivial decomposition of a complex gate latch into a standard simple gate latch and a sum-of-product two-level combinational circuit for functions S_x and R_x is hazardous, as exemplified in Fig. 2(d). A hazard-free implementation using simple gates can be obtained by including redundant literals into S_x and R_x: $S_x = \overline{a}dc$ and $R_x = \overline{a}\overline{d}\overline{c}$ [Fig. 2(e)]. In general, it is not always possible to find a nonprime or redundant cover for the hazard-free implementation of the S and R functions for the RS-latch. If this is impossible, a transformation of the initial specification is needed.

The main results of the paper can be summarized as follows.

For the speed-independent implementation of a correct STG specification, we use a *standard* implementation structure. Two possible such structures are considered: an *RS-implementation* and a *C-implementation.* They are based on the use of two-level combinational logic and a latch (either an RS latch or a C-element) for each output signal. We prove that if each cube implemented by an AND-gate in the first level of combinational logic obeys the *monotonous cover (MC) requirement,* then both standard implementations are speed independent. This requirement is formulated in terms of regions in the state space of the transition diagram that corresponds to the initial STG.[1]

[1]The term "MC requirements" refers to restrictions that will be imposed on cover cubes in the sum of products (SOP's) of the S and R functions of an implementation. Bearing in mind the link between a TD (and hence STG) and the ON and OFF sets of the S and R functions it produces, we can therefore use this term for the TD's (STG's) that generate the excitation functions satisfying the MC requirements.

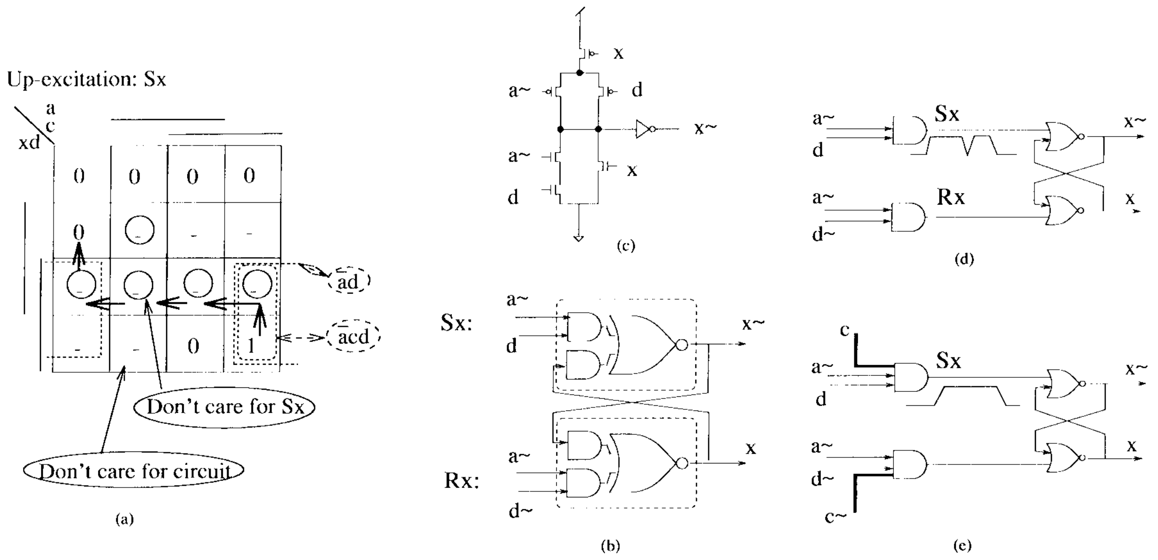


Fig. 2. Hazards in RS-implementation of signal x.

We then switch our focus to model transformations, where we present a constructive procedure that allows us to transform an initial STG specification to the form that meets the monotonous cover requirement. From the latter, a speed-independent standard implementation can be directly derived. The method is based on inserting additional internal signal transitions, and it always converges. Only very limited transformations that preserve the language generated by the specification are used here. Neither concurrency reduction [36] nor signal reshuffling [23] is allowed. In other words, by equivalence in this paper, we mean the trace equivalence with respect to externally observable signals.

We extend these results in several ways. First, we relax the monotonous cover requirement to allow gate sharing in the combinational logic and discuss several optimization techniques. Then we generalize the monotonous cover requirements to allow hazard-free implementation for a wider class of specifications, such as those that are not restricted by the unique entry condition (defined in Section III-C) and those with OR-causality.

B. Related Work

A great deal of research activity has been recently focused on the synthesis of asynchronous circuits from the STG specifications. We will only discuss the work that directly relates to the hazard-free gate-level implementation of speed-independent circuits.

In [2], Beerel and Meng suggest a synthesis method that is based on the use of the standard C-implementation structure. They define an implementation condition that is equivalent to our monotonous cover requirement for the case of decomposition into simple gates. This work has been an important step toward the implementation of speed-independent circuits into standard gate cells. Our work generalizes the results of [2] by 1) considering a *wider class of specifications* that can be handled in the synthesis process and 2) extending the theory of monotonous cover to support more aggressive optimization.

The requirement for specifications to satisfy the unique entry conditions, central in [2], is overly restrictive even for the implementation of circuits with AND causality between signal transitions. We show in this work how to avoid this restriction, and apply our methods to specifications that have both AND and OR causal relations between signal transitions.

The conditions of [2] and the "basic" monotonous cover requirement are both targeted at the architecture in which every transition of an output signal in the specification is implemented by a separate cone of logic realized in simple gates (AND, NAND, OR, and NOR). To improve the efficiency of the implementation, we produce the generalized monotonous cover requirements (where a single cone of logic can be shared to implement several signal transitions) and the polyterm monotonous cover for the implementation with complex gates (AND-OR, AND-NOR, etc.).

This work also suggests a different (from [2]) way of organizing logic that implements signal transitions. The monotonous cover requirement is defined in accordance with *one* cube (that leads to a very simple and, we believe, efficient architecture: one excitation region—one AND gate in combinational logic), while the conditions in [2] are defined for a *set of cubes* covering the excitation region. The latter makes logic more complicated (see the experimental results) and calls for a computationally hard (circuit-level) transformation of the circuit based on using additional acknowledgment wires from the gates of the combinational logic. This method does not always converge and sometimes needs extra signals added into the specification. The problem of logic decomposition with sharing was recently solved in [10].

Chu [4], [7] suggests a way of implementing TD specifications that are produced by free-choice safe STG's. His logic synthesis is based on complex gates, which may not be implementable as single units, satisfying both the delay model requirements and the constraints of a given technology.

Moon *et al.* [27], describe an implementation architecture consisting of two-level sum-of-product logic and set/reset-dominant RS latches. They consider STG specifications with

AND causality only. Their hazard elimination procedure is not strictly complete, as it does not always guarantee hazard-free implementation.

Varshavsky and Kishinevsky [38] present a formally justified method of implementing autonomous (having no external inputs) speed-independent circuits by complex AND-NOR gates and by two-input NAND and NOR gates, as well as distributive circuits without OR-causality by two-input NAND (or alternatively NOR) gates only, thus proving theoretical results on the implementability of these classes. However, the offered constructions are area inefficient and thus impractical.

Yu and Subrahmanyam [42] proposed to use the property of a separable cube to define necessary and sufficient conditions for hazard-free implementation of a limited class of STG's. Their analysis, however, assumes another implementation architecture (sum-of-product combinational logic without separate asynchronous latches), and it is applied under rather severe constraints on the specification, namely, only marked graphs (STG's without choice) and nonrepetitive transitions of noninput signals are allowed.

A preliminary discussion of the monotonous cover condition accompanied with a synthesis method was presented in [17] and [18]. This method was formulated as a set of Boolean constraints under the rules of introducing additional signals. The solution can be found using Boolean satisfiability solvers. This approach allows handling only relatively small specifications because Boolean constraints are described in terms of the state graph rather than at the signal-transition level. Furthermore, the control of the quality of logic obtained is rather limited. The latter is due to a limited opportunity to influence the proof run in the Boolean satisfiability solvers.

Pastor *et al.* [33] used the theory of monotonous covers presented in this paper for developing structural methods of synthesis of speed-independent circuits using cube approximations derived from STG's. Last, [10] presents the state-of-the-art method for decomposition and technology mapping of speed-independent circuits, which is built on the monotonous cover theory.

This paper is further organized as follows. Section II introduces asynchronous circuits and STG's. TD's and their properties are presented in Section III. Section IV defines two standard implementation architectures based on C-elements and RS-latches. Section V presents the monotonous cover requirements. Section VI discusses reductions of STG specifications to the form satisfying monotonous cover requirements. To this end, a set of equivalent transformations at the STG level is presented. Section VII shows the extensions of monotonous cover theory to support several optimizations and to capture processes with OR-causality. Last, Section VIII presents experimental results and comparison (on a benchmark list) with the previously known methods.

II. Modeling Circuit Behavior

A. Asynchronous Circuit Model

A circuit is described as an interconnection of logic gates. Every gate is represented by a combination of a function evaluator, which evaluates the corresponding logical function instantly, and an *unbounded* delay, which is attached to the gate's output. The type of delay can vary depending on the assumptions about particular physical properties of the electronic devices and wires implementing the circuit. We will adopt the most pessimistic view on the properties of the gate delay: any glitch in its input can propagate to the output (more details on delay models can be found in [1] and [26]). All the delays involved in switching and transmission of signals within a gate and through its output wire prior to a fork are assumed to be reduced to the output delay. The skew of signals in the wire delays after the fork is assumed to to be less than the minimum gate delay. A delay induced by a wire may be modeled explicitly, if need be, by inserting an auxiliary component, a buffer, into the wire break.

A *circuit* $\mathcal{C}$ is a set of gates $Z = \{z_1, \cdots, z_m\}$ and a set of input nodes $X = \{x_1, \cdots, x_k\}$ with each gate input connected to strictly one gate output or one input node and with no two outputs tied together. A circuit is called *autonomous* if $X = \emptyset$. The total set $Y = X \cup Z$ will simply be called a set of signals, where $x_1, \cdots, x_k$ are the *input signals* and $z_1, \cdots, z_m$ are the *output signals*. Some of the output signals serve as the *external* output signals, i.e., they are observable by the circuit environment, and the others are *internal* signals, i.e., not observable by the environment. The input signals all come from the environment.

A *state* of a circuit is a binary vector of the signal values on the outputs of gates and on the input nodes. The behavior of the ith gate can be described by the Boolean equation $z_i' = f_i(z_1, \cdots, z_i, \cdots, z_m, x_1, \cdots, x_k)$, where $z_1, \cdots, z_{i-1}, z_{i+1}, \cdots, z_m, x_1, \cdots, x_k$ are the signal values in the nodes corresponding to the inputs of the ith gate, z_i is the current value of the gate output, z_i' is its next (after some delay) value, and $f_i(z_1, \cdots, z_i, \cdots, z_m, x_1, \cdots, x_k)$ is the logic function of the ith gate. Obviously, the irredundant representation of the gate's function should only involve a subset of signals, those that are physically connected to the gate.

The *model of circuit* $\mathcal{C}$ (with m gates) is a set of m simultaneous equations of the above type [28].

A circuit is *initialized* if its initial state is defined as a binary vector $s_0 \in \{0,1\}^n, n = m + k$, each component s_0^i of which is the state (or value) of the corresponding Boolean variable $y_i \in Y$. Hereupon, we will only consider initialized circuits, assuming that the initialization process already has been performed.

An initialized circuit may change the values of its output signals. A signal can either be *stable* in state s—if its value in this state, zero or one, is equal to the value computed by the logic function corresponding to z_i—or *excited,* otherwise. For example, an AND gate with output at one and at least one input at zero is *excited* (its output is denoted by 1*). In any state, the excited signals tend to become stable and possibly change their state, which would then match their function's value. This generates the dynamic behavior of the circuit, i.e. transitions from the initial state to other states, which can be determined according to the equation system. This state-transition behavior can be formally captured in a TD (defined in Section III).

The transient behavior of the circuit may also proceed in parallel with the signal transitions on the input nodes. In other words, the environment may change some of the input values in response to the external output changes according to some protocol. This style of circuit-environment interaction is often called the *input-output mode*. It is more general and robust than the traditional *fundamental mode* [14], [20], [31], [34], [43], which restricts the environment's action by allowing input transitions only when all output signals are stable.

A spontaneous deviation from the intended circuit behavior, called *hazard,* can occur in the implementation of an asynchronous circuit. This appears as a short spiky pulse, which does not correspond to any signal transition in the specification.

In the framework of speed-independent design, a hazard-free behavior is captured by the notion of semimodularity [26], [28] that is defined as follows. An excited signal can either perform the enabled transition (e.g., transition $1^* \rightarrow 0$ at the output of an excited AND gate with output at one and at least one input in zero) or become *disabled* because the gate inputs change (e.g., if both inputs of an AND gate go to one before the output changes to zero). The latter behavior is considered hazardous because in such conditions, a spurious pulse might appear on the output of the gate. A circuit is *semimodular* with respect to a state s_0 and a signal z_i if, when the circuit is initialized in s_0, no transition of the signal z_i can be disabled during the circuit operation. A stricter definition will be given in Section III. In the framework of this paper we will use the terms "speed-independent" and "semimodular" circuit as synonyms because semimodular circuits are the widest subclass of speed-independent circuits that are free of hazards under both inertial and pure delay models [41]. (For the original analysis of the relationship between semimodularity and speed independence, one may look into [26] and [28].)

B. STG's

An STG is defined as a Petri net (PN) whose transitions are labeled with the transitions of signals in a modeled circuit. We assume here the standard definition of a PN [29]: $\mathcal{N} = \langle P, T, F, M_0 \rangle$, with P being a set of places, T a set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ a flow relation, and M_0 the initial marking, which is a function $P \rightarrow N$ where N is a set of nonnegative integers. A PN marking is depicted by tokens in the places, whose number is determined by M_0. In this paper, we will mainly consider a sufficiently powerful subclass of PN's called free-choice PN's.

A *free-choice net* is a PN in which, if two or more transitions share one predecessor place, then this is the only predecessor place for all of them. Such a place is called a free-choice place. A *marked graph* (MG) net is a PN in which each place has exactly one predecessor and one successor transition. MG's allow concurrency but cannot model choice.

An STG is a free-choice PN whose transitions are interpreted as signal transitions on the circuit inputs (input transitions) or gate outputs (output transitions). A signal transition can be represented by $+a_j$ or $-a_j$, where a is the name of the signal; "$+$" or "$-$" is the sign of the signal transition, with "$+$" for the transition of a from zero to one, and "$-$" for the opposite transition (only binary systems are considered here); and j is a subscript denoting the instance number of the $+a$ or $-a$ transition (in one cycle of the circuit operation the signal on an input or gate output may change several times). This subscript can be omitted if the signal transition can occur only once in the circuit operation cycle. $*a_j$ is used to denote either a "$+a_j$" transition or a "$-a_j$" transition.

Definition 2.1: An STG G is a triple $\langle \mathcal{N}, A, \lambda \rangle$, where $\mathcal{N}$ is a free-choice PN, A is the set of signal transitions, and λ: $T \rightarrow A$ is a function that labels each transition with a signal transition.

The functioning of an STG is similar to that of Petri nets [29]. A transition is enabled if all its predecessor places are marked. When an enabled transition fires, the marking of each predecessor place is decremented, and the marking of each successor place is incremented. The new marking of the STG obtained through the firing of the transition can again make some transitions enabled. This leads to the dynamic behavior of the STG, analogous to that of a circuit. We can therefore talk about sequences of transitions that fire under the markings reachable from the initial marking M_0. Such sequences will be called *feasible sequences* of the STG.

An STG is graphically represented as a directed graph with transitions denoted by their names and places by circles, where places that have only one predecessor and successor transition are usually omitted. Transitions of input signals are underlined. The subscripts of signal transitions, if necessary, are placed in the same line with the signal change (using commas).

The following two properties of PN's (and hence STG's) reflect their ability to define finite and cyclic processes in circuits.

A PN is called *k-bounded* (*safe*) if for every reachable marking M and each place $p \in P$, $M(p) \leq k$ $(M(p) \leq 1)$, where $M(p)$ denotes the number of tokens in place p under the marking M. It is called *bounded* if there exist a finite k for which it is k-bounded.

A PN is called *live* if for every reachable marking M and each transition $t \in T$, there can be reached another marking M' such that t is enabled at M'.

Two binary relations, called *precedence* and *concurrency,* are defined for signal transitions. Let $*a_j$ and $*b_k$ be two arbitrary signal transitions. If in every feasible sequence generated in the STG with respect to the initial marking M_0 every ith occurrence of $*a_j$ is met before the ith occurrence of $*b_k$, then $*a_j$ *precedes* $*b_k$, denoted by $*a_j \Rightarrow *b_k$.

If there is a marking M reachable from the initial marking M_0 such that two sequences of signal transitions $*a_j, *b_k$ and $*b_k, *a_j$ can fire from M, then $*a_j$ and $*b_k$ are said to be *concurrent,* denoted $*a_j || * b_k$.

An example of STG is shown in Fig. 3(a). This STG has only one free-choice place $p1$ that is initially marked. This place is a predecessor one for transitions $+a_1$ and $+b_2$, and both of them are enabled initially. The firing of any transition, say, $+a_1$, disables $+b_2$ and enables the events $+c_1$ and $+d_1$ that are successors for $+a_1$. One of the firing sequences that is feasible in this STG is $+a_1 + c_1 - a_1 + d_1 + b_1 - c_1 - b - d \cdots$. It is clear that in this example transition, $+a_1$ precedes $+b_1$, and transition $+d_1$ is concurrent to $-a_1$.

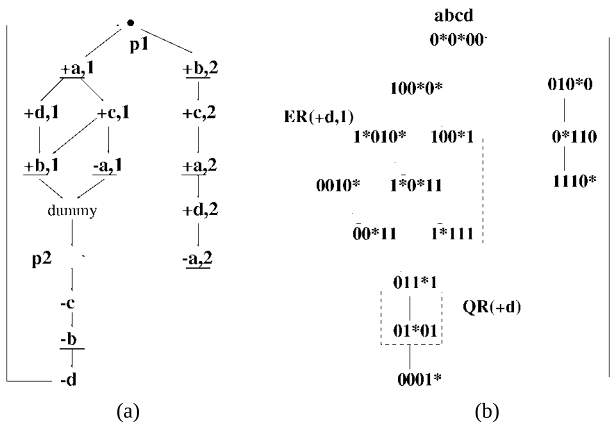


Fig. 3. (a) Example of STG and (b) corresponding TD.

Now we will introduce the notion of STG correctness that points out the necessary and sufficient conditions for the correspondence between the STG model and its correct implementation as a semimodular circuit [15], [39]. Not every STG is implementable by a circuit. In the previous section, we considered initialized circuits, which have a single initial state. Hence, when targeting at the implementation by an initialized circuit, we should require the similar property from the original STG, i.e., in its initial marking, the value of each signal must be determinate (zero or one). We will call an STG *well initialized* if for every signal a in any feasible sequence starting from m_0 the first transition of a has the same direction (either falling or rising). Clearly, in a well-initialized STG, the value of each signal in the initial marking is determinate.

Any sequence of signal transitions in a binary circuit possesses the following property, called *switchover correctness:* the rising and falling transitions of the same signal must alternate. Last, for an STG to specify the finite behavior (of a circuit), the set of markings during the STG operation cannot grow infinitely, i.e., the underlying Petri net has to be *bounded*.

Our goal is a semimodular implementation of the STG behavior without internal nondeterminism. In this paper, we do not consider implementations involving arbiters [11]. Hence, the free-choice places in the STG must only be associated with external nondeterminism, which corresponds to the designer's partial knowledge of the environment's reactions. Thus, for a correct STG, all the successor transitions of a free-choice place may only be *input* signal transitions.

Definition 2.2: An STG is *correct* iff it satisfies the following four conditions.

1) The STG is well initialized.
2) Each feasible sequence of signal transitions is switchover correct.
3) The STG is bounded.
4) Any free-choice place precedes only input transitions.

It was proved in [39] that STG correctness (in terms of Definition 2.2) is necessary and sufficient for implementability of an STG with a semimodular circuit.

STG descriptions are more compact than TD's and are therefore more appropriate for specification and verification. However, in synthesis of Boolean functions for circuit signals, it is often more convenient and efficient to derive Boolean vectors corresponding to the markings of STG's.

III. Transition Diagrams

A TD is a directed graph where each vertex is in one-to-one correspondence[2] with the markings reachable from the initial marking M_0 of a given STG. A TD vertex (state) is labeled with a Boolean vector $s = \langle s^1, \cdots, s^n \rangle$ representing the value of STG signals (n is the number of signals in the STG). Two states s_1 and s_2 corresponding to markings M_1 and M_2 are connected with an edge in the TD $(s_1 \rightarrow s_2)$ if M_2 is reachable from M_1 by the firing of some event $*a$ in the STG. From this, it follows that each edge of the TD can be labeled with the name of the corresponding transition: $s_1 \xrightarrow{*a} s_2$. This transition is called *enabled* in state s_1.

Definition 3.1: [5] A TD has a *consistent state assignment* iff for each pair of states s_1 and s_2 connected with an edge $(s_1 \rightarrow s_2)$, the following conditions are met.

1) If the edge is labeled by $+a$ transition, then signal a is equal to zero in s_1 and to one in s_2.
2) If the edge is labeled by $-a$ transition, then signal a is equal to one in s_1 and to zero in s_2.
3) In all other cases, the value of signal a in s_1 and s_2 is the same.

It can be proved that a TD obtained from a correct STG always has a consistent state assignment [15]. Fig. 3(b) shows the TD that corresponds to the STG from Fig. 3(a). It is easy to check the consistency of its state assignment.

A. Complete State Coding

Each state s of a TD defines a vertex in the *on-set* or the *off-set* of a signal function f_i. If $f_i(s) = 1$, i.e., signal z_i has either value 1 or 0^* in s, then s belongs to the *on-set.* Otherwise, s belongs to the *off-set.* All states that do not belong to the TD of the circuit, i.e., that are not reachable from the initial state(s), belong to the *dc-set* of f_i. Using the minimization of the incompletely specified logic functions defined by their *on-sets, off-sets,* and *dc-sets,* we derive the gate functions of the implementation.

Unfortunately, such a procedure is not always immediately possible. Two different markings of an STG may correspond to the TD states that have *identical Boolean codes,* even in the case when the TD has a consistent state assignment. If two states of a TD with identical Boolean codes have different excitation of output signals, then the TD violates the *complete state coding* (CSC) requirement [5], and the two states are said to be in a CSC conflict.

A violation of the CSC requirement means that the circuit being in the same binary state has to produce different transi-

[2]In principle, in certain nonlive STG's, one marking can be associated with several different signal encodings, and hence with different states. We, however, prefer not restrict ourselves with the liveness condition in synthesis (e.g., to enable synthesis of circuits whose behavior is nonrepetitive) and therefore discard such STG's here. It is not difficult to include such cases by means of considering an STG state as a pair "marking and signal vector" (see [16]).

tions on the gate outputs. This is possible only if the circuit can distinguish such conflict states by means of an additional memory, i.e., internal signals that do not exist in the original specification.

If an STG is correct, then there is a semimodular circuit such that its output behavior is equivalent to the STG specification. This circuit can, however, have more signals than the original STG, since some additional state signals are required to ensure the CSC property. Thus, the equivalence between the STG and the circuit behavior is considered only with respect to the signals from the initial STG specification.

We will not discuss here how to ensure the CSC property. There have been several formal techniques proposed recently for the elimination of CSC conflicts by inserting extra internal signals [8], [15], [22], [32], [37]. Thus, we will further consider only TD's that satisfy the CSC property. For such TD's, it is always possible to derive logic functions for all output signals. Usually, some of the logic functions are too complex to be implemented by a single gate. Therefore, the function has to be decomposed into several gates, and this decomposition must not introduce any hazards, i.e., be semimodular.

B. Semimodular Transition Diagrams

We defined a TD as an object generated through the token flow simulation of an STG. However, in order to formally check the properties of a circuit, it is convenient to relate an initialized circuit with its TD. The TD can be constructed through circuit simulation using state traversal, following the way suggested in Section II-A. Efficient techniques of TD generation and asynchronous circuit analysis can be found elsewhere [28], [38]. Drawing upon the correspondence between the circuit and its TD, we can formally define the notion of semimodularity in the TD terms and speak about a semimodular circuit as a circuit whose TD satisfies certain formal properties.

Definition 3.2 (Conflict) [28], [38]: State w of a circuit is said to be a *conflict state* with respect to signal a iff a is excited in w and there is another signal b excited in w such that $w \xrightarrow{b} u$ and signal a becomes stable in state u. If a is an output signal, then w is said to be an *output conflict state;* if a is an input, while b is an output signal, then w is said to be an *input-output conflict state;* when both a and b are inputs, w is said to be an *input conflict state.*

Hazards at the gate outputs can occur if a circuit reaches an output conflict state. Input-output conflict states specify the conditions under which output signals disable inputs. This might impose nonimplementable constraints for the behavior of environment and should be avoided in speed-independent implementation. Input conflicts identify states in which the environment determines the direction of control flow. For example, a random-access memory (RAM)-controller has a read and a write mode controlled by the read and the write input signals. This type of input control is represented in the TD of the RAM-controller by means of input conflict states, in which both read and write input signals are excited, but always only one of them (the environment's decision) performs a transition.

Definition 3.3 (Semimodularity): A TD is said to be *semimodular* (*output semimodular*) with respect to state u iff no conflict state (output and input-output conflict state) is reachable from u.

Let us return to the TD example in Fig. 3(b). In the initial state 0^*0^*00, both a and b signals are excited but the firing of either of them disables the excitation of the other (see states 100^*0^* and 010^*0). Thus, 0^*0^*00 is a conflict state and this TD is not semimodular. As a and b are both input signals, state 0^*0^*00 is an input conflict state. There are no other conflict states in this TD, so it is output semimodular and can be implemented by a semimodular circuit.

The definition of semimodularity is given for TD's. However, we will equally apply it directly to the circuits associated with the TD's in question. Bearing in mind that we consider speed independence and semimodularity of circuits as synonyms, we have the following correspondence among circuits, STG's, and TDs: speed-independent circuits $\Rightarrow$ output semimodular TD's $\Leftarrow$ correct STG's.

C. Properties of Transition Diagrams

This section produces a finer analysis of the structural properties of TD's. The foundation laid here will be further refined into a set of properties that the TD derived from the correct STG must satisfy in order to allow its hazard-free implementation in a number of implementation architectures.

The following definitions relate signal transitions with states of TD's.

Definition 3.4 (Excitation Region): An *excitation region* of signal a in transition diagram G is a maximally connected set of states in which a has the same value and is excited.

The excitation region corresponding to transition $*a_i$ will be denoted as $ER(*a_i)$. Note that there can be several excitation regions for a, corresponding to multiple transitions of a. We will call an excitation region that corresponds to a "$+a$" ("$-a$") transition an *up-excitation region* (*down-excitation region*).

Definition 3.5 (Quiescent Region): [2] The quiescent region corresponding to transition $*a_i, QR(*a_i)$ is the maximally connected set of states s reachable from $ER(*a_i)$ such that 1) a is stable in s and 2) s is not reachable from any other $ER(*a_k)$ without going through $ER(*a_i)$.[3]

Excitation region $ER(+d_1)$ and the following quiescent region $QR(+d_1)$ are shown by dashed lines in Fig. 3(b).

Definition 3.6 (Minimal State): A state u is said to be a *minimal* state for excitation region $ER(*a_i)$ if it has no predecessors within the region. It is denoted as $u_{\min}(*a_i)$.

Definition 3.7 (Unique Entry Condition): An excitation region is said to satisfy the *unique entry condition* (UEC) if it has exactly one minimal state.

The notion of "unique entry condition" is important because it is a sufficient condition for the existence of a *single cube* to cover an excitation region.

[3] Note that contrary to [2] and [18], in this paper, we consider only the *restricted* quiescent regions, which do not include states reachable directly from two different ER's of the same signal.

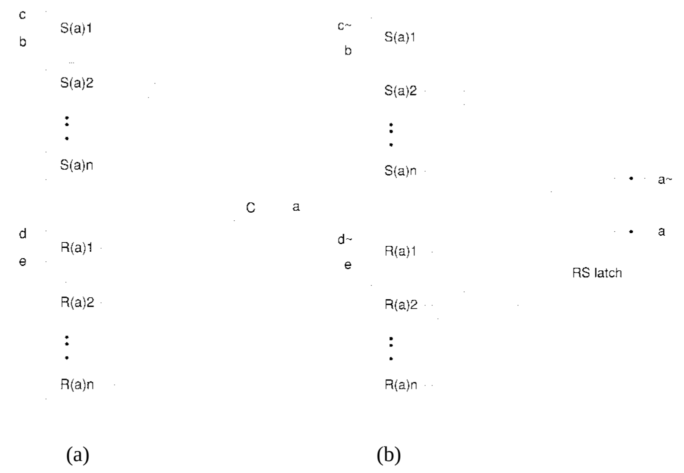


Fig. 4. Signal network for standard (a) C- and (b) RS-implementation structures.

Definition 3.8 (Trigger Signals): A transition $*b_j$ (as well as its underlying signal b) is called a *trigger* for a transition $*a_i$ if by firing $*b_j$ we can enter the excitation region $ER(*a_i)$. Formally, $*b_j$ is a trigger for $*a_i$ if and only if there exists a state v in the transition diagram such that $v \notin ER(*a_i)$ and $v \xrightarrow{*b_j} u, u \in ER(*a_i)$.

Definition 3.9 (Ordered and Concurrent Signals): A signal b is said to be *ordered* with respect to a transition $*a_i$ if no transition of signal b is excited within $ER(*a_i)$. Otherwise, b is said to be *concurrent* with $*a_i$.

Definition 3.10 (Persistency): [5] A trigger transition $*b_j$ is *nonpersistent* to $*a_i$ if b is concurrent to $*a_i$; otherwise, it is said to be *persistent*. A transition diagram G is said to be *persistent* if for each excitation function $ER(*a_i)$ the corresponding transition $*a_i$ is persistent to its trigger signals.

If a trigger transition $*b_j$ is nonpersistent to $*a_i$, there has to be a state v in $ER(*a_i)$ in which $*b_k$ is excited. Let us illustrate nonpersistency using the TD from Fig. 3(b). The minimal state of $ER(+d_1)$ (100^*0^*) can be reached only by firing transition $+a_1$. This transition is the only trigger to $ER(+d_1)$. However, inside $ER(+d_1)$, transition $-a_1$ is excited, which leads to the nonpersistency of transition $+a_1$ with respect to $+d_1$.

IV. Basic Implementation Structures

In our synthesis method, we consider two implementation structures based on two different types of asynchronous latches: one using Muller C-elements and the other using RS-latches. Both structures are essentially the same except that the latter is *dual-rail* encoded.

A. Standard C-Implementation

A two-input Muller C-element is an asynchronous memory element with inputs a and b and one output c. The next state equation is $c = ab + c(a + b)$. The implementation structure using C-elements is shown in Fig. 4(a). This structure is called a *signal network*. A signal network for output signal a is constructed in the following way.

1) For each *up-excitation region* $ER(+a_i)$, a *region function* $S_a(i)$ is derived as a single cube implemented by an AND gate.
2) The region functions $S_a(i)$ are combined by an OR-gate to create an *up-excitation function* S_a for a (the *down-excitation function* R_a is obtained in a similar way).
3) Up- and down-excitation functions are connected to the inputs of a C-element (directly and via an inverter, respectively).

Such an implementation is called a *standard C-implementation*. It requires, in addition to C-elements, AND gates, OR gates, and inverters.

If we consider all input inverters as independent gates, the standard C-implementation will no longer be speed independent. To justify the use of input inverters, we consider some realistic bounds on gate delays.

Theorem 4.1 [19]: Assume that a standard C-implementation $\mathcal{C}_1$ of some STG is output-semimodular. Let $\mathcal{C}_2$ be the same standard C-implementation except that all input inversions of AND-gates are implemented by separate inverters. Let $d_{\rm inv}^{\max}$ be the maximal possible delay of one inverter and $D_{sn}^{\min}$ be the minimal possible delay of one signal network (it consists of the AND-gate delay + the OR-gate delay + the C-element delay). $\mathcal{C}_2$ is hazard free under any distribution of gate delays satisfying the following condition: $d_{\rm inv}^{\max} < D_{sn}^{\min}$.

The relational constraint on the value of inverter delays given by the latter statement is realistic. This allows us to use the standard C-implementation without considering precise bounds on gate delays.

B. Standard RS-Implementation

We can also use an RS latch to implement the signal network. An RS latch is an asynchronous memory element with inputs S and R and dual-rail outputs F and F'. The dual-rail outputs are inverses of each other. The next state equations are $F = \overline{F' + R}$ and $F' = \overline{F + S}$.

When RS latches are used, all the internal signals of a circuit are already implemented in the dual-rail form that allows us to replace all the inverse occurrences of each internal signal in the region functions by the signal from the inverse output of a corresponding RS latch. If all input signals are also presented in the dual-rail form, all the region functions can be implemented simply by an AND-gate without input inversions.

In practice, it is more efficient to use NAND and NOR gates. With RS latches, this requirement comes quite naturally: a two-level AND-OR function can be replaced by a two-level NAND-NAND function with the same inputs and output. The corresponding structure is shown in Fig. 4(b). Such an implementation is called a *standard RS-implementation*. The possibility of using NAND and NOR gates is an advantage of the RS-implementation over the C-implementation. However, it requires either working under the dual-rail interface with environment or using special input-output converters to and from the dual-rail form.

V. Monotonous Cover Condition

We now investigate the necessary and sufficient conditions for deriving a speed-independent circuit from an STG using the basic implementation structures described in the previous section. However, bearing in mind the relationship between STG's and TD's, we will formulate these conditions in TD terms.

To develop the sufficient conditions for a hazard-free implementation under the unbounded gate delay model, we first introduce the notion of a *cover cube* and its correspondence to the excitation regions.

Definition 5.1 (Cover Cube): A cube $c = c_1, \cdots, c_k$ is said to be a *cover cube* for an excitation region $ER(*a_i)$, denoted as $c(*a_i)$, if each literal c_j corresponds to some signal b ordered with a_i and $c_j = b$ if b has the value "1" in $ER(*a_i)$; otherwise, $c_j = \overline{b}$.

In a TD that satisfies the UEC condition, the smallest (in its dimension) cover cube $c_{\min}(*a_i)$ can be derived by deleting from a min-term corresponding to the state $u_{\min}(*a_i)$ all signals that are concurrent to $*a_i$ together with signal a_i itself.

Definition 5.2 (Correct Covering): A cover cube $c(*a_i)$ covers $ER(*a_i)$ *correctly* if:

1) for $*a_i = +a_i$ it does not cover states where the value of function for signal a equals zero (i.e., $c(*a_i) \cap (ER(-a) \cup (QR(-a)) = \emptyset)$);
2) for $*a_i = -a_i$ it does not cover states where the value of function for signal a equals one (i.e., $c(*a_i) \cap (ER(+a) \cup QR(+a)) = \emptyset)$).

Theorem 5.1: In a transition diagram G, every cover cube $c(*a_i)$ covers the corresponding $ER(*a_i)$ correctly only if G is persistent.

Proof: (*By Contradiction*) Let the original transition diagram G be nonpersistent. Thus, it contains an excitation region, say, $ER(*a_i)$, with signal b being a trigger signal to $*a_i$ but b concurrent to $*a_i$. In this case, $c(*a_i)$ does not contain signal b. As b is a trigger signal for $*a_i$ by Definition 3.10, a state v exists in G such that v does not belong to $ER(*a_i)$ and $v \xrightarrow{b} u, u \in ER(*a_i)$.

Evidently, $c(*a_i)$ covers all states of $ER(*a_i)$, and so it also covers state u. But since $c(*a_i)$ does not contain signal b, and states u and v differ only in the value of $b, c(*a_i)$ will also cover v. The latter contradicts the condition that $c(*a_i)$ covers the $ER(*a_i)$ correctly. □

For TD's in which the UEC conditions are satisfied, the following corollaries from Theorem 5.1 can be shown.

Corollary 5.1: The largest (in its dimension) cube $c_{\max}(*a_i)$ that can cover $ER(*a_i)$ correctly can be derived by deleting from a min-term corresponding to the state $u_{\min}(*a_i)$ all signals except for the trigger ones to $*a_i$.

Corollary 5.2: For any cube $c(*a_i)$ that covers $ER(*a_i)$ correctly $c_{\min}(*a_i) \subset c(*a_i) \subset c_{\max}(*a_i)$.

We now define the key definitions that essentially capture the sufficient condition for a speed-independent implementation. Monotonous cover requirements are very similar to the conditions from [2] with the exception that here they are first formulated with respect to a single cube, while [2] considers the arbitrary logical functions to implement region functions. We believe that the construction "region function → AND-gate" leads to a simpler and more efficient implementation in the basis of simple gates (see experimental results). Further, we generalize the monotonous cover requirements to allow the sharing of logic between different region functions and consider several exceptions that can optimize the monotonous cover implementations by softening the requirements.

Definition 5.3 (MC): A cover cube $c(*a_i)$ is said to be an MC for $ER(*a_i)$ if:

1) *cover condition:* $c(*a_i)$ covers all states $ER(*a_i)$;
2) *monotonicity condition:* in every sequence of states inside $ER(*a_i) \cup QR(*a_i)$, $c(*a_i)$ changes at most once;
3) *one-hot condition:* $c(a_i)$ does not cover any reachable state outside $ER(*a_i) \cup QR(*a_i)$.

This definition implies that cube $c(*a_i)$ with the MC property covers $ER(*a_i)$ correctly.

Definition 5.4 (MC Requirement): A transition diagram is said to satisfy the MC requirement if and only if for every excitation region $ER(*a_i)$ of an output signal there exists a monotonous cover cube $c(*a_i)$.

This definition implies that if the TD satisfies the MC requirement, for every output signal a, all the cubes corresponding to the transitions of a are disjoint (do not cover common states).

Theorem 5.2: If the excitation functions R_a and S_a for each output signal a in the TD derived from a correct STG are represented as the sums of cubes $c(-a_1), \cdots, c(-a_k)$ and $c(+a_1), \cdots, c(+a_k)$, respectively, where $c(*a_i)$ corresponds to the monotonous cover of $ER(*a_i)$, then both standard RS- and C-implementations are semimodular.

Theorem 5.2 gives a sufficient condition that guarantees implementation correctness. To optimize logic for the region functions, we can allow the cover cubes for different excitation functions to be implemented by a single AND-gate. To deal with such a gate sharing, let us generalize the notion of MC.

The definitions of cover cube and correct cover can be generalized to cater for a set of transitions. If F is a set of transitions $(F = \{a_{i1}, *b_{i2}, \cdots\})$ and $F' = \{ER(*a_{i1}), ER(*b_{i2}), \cdots\}$ is the corresponding set of excitation regions, then $c(F')$ is a cover cube for F' if $c(F')$ is a cover cube for any excitation region from F'; and $c(F')$ covers F' correctly if it correctly covers each excitation region from F'.

Definition 5.5 (Generalization of MC): A cover cube $c(F')$ is said to be an MC for a set of excitation regions F' if:

1) $c(F')$ covers all states in $ER(*a_{i1}) \cup ER(*b_{i2}) \cup \cdots \cup ER(*d_{ik})$;
2) $c(F')$ changes at most once in every sequence of states inside each of regions $ER(*a_{i1}) \cup QR(*a_{i1}), ER(*b_{i2}) \cup QR(*b_{i2}), \cdots, ER(*d_{ik} \cup QR(*d_{ik})$;
3) $c(F')$ does not cover any reachable state outside $ER(*a_{i1}) \cup QR(*a_{i1}), ER(*b_{i2}) \cup QR(*b_{i2}, \cdots, ER(*d_{ik}) \cup QR(*d_{ik})$.

All three conditions in this definition generalize the corresponding conditions of Definition 5.4. The last condition ensures that only one cube and, respectively, one AND-gate can be turned on in the excitation region. Due to this condition, if the signal network for signal a contains an AND-gate for cube $c(F')$ (i.e., some $ER(*a_i) \in F'$), then all the excitation regions of a that intersect $c(F')$ have to be covered by $c(F')$ completely; otherwise, some other cube is required for correct cover, and thus more than one cube can be turned on inside one excitation region.

The generalized MC conditions give a theoretical basis for the optimization based on sharing gates between different region functions. The idea of gate sharing under the standard implementation is not novel. For example, [2] suggests that if two region functions correspond to adjacent cubes with the same set of variables (e.g., abc and $ab\overline{c}$), then they can be merged into one gate implementing cube ab. However, this condition covers only a particular case of sharing, and it is applied *after* the region functions have been derived, i.e., for this particular cover implementation. Our generalized MC requirement considers the problem *before* the derivation of region functions and hence can be used for deriving a minimal cover. It gives more general conditions for a proper (hazard-free) sharing of logic. However, we must admit that the search space for choosing the corresponding set of transitions is quite large, and the selection should be guided heuristically.

The importance of the generalized MC requirement is shown by two facts. First, we prove that the standard implementation based on MC cubes will have the behavior equivalent to the original specification (i.e., conformance of implementation to specification). Second, we show that the implementation is free from any hazards (in internal gates and outputs).

To establish the relationship between the behavior of the implementation and that of the specification (STG), we need to define a formal notion of equivalence.

The behavior of an STG and a circuit can be compared by the languages they realize. The languages are characterized by the set of *traces*, i.e., feasible sequences, of signal transitions. The equivalence relation between signal transitions in the STG and the circuit is given by the relationship between any transition $+a_i$ $(-a_i)$ in the STG and any rising (falling) transition of signal a in the circuit.

Definition 5.6 (Projection): For a trace q over a set of signals S_A, the projection of q on a set of signals $S_B, S_B \subset S_A$ is a sequence $q \downarrow S_B$, which is obtained from q by deleting all transitions whose signals are not in S_B.

The projection of a set of traces of STG D $(L(D))$ on a set of signals S_B is the set of projections of all traces from $L(D)$ on S_B (denoted by $L(D) \downarrow S_B$).

Definition 5.7 (Trace Equivalence): Two STG's $D1$ and $D2$ with signal sets S_{A1} and S_{A2} are trace equivalent for a set of signals $S_B, S_B \subseteq S_{A1} \cap S_{A2}$ if $L(D1) \downarrow S_B \equiv L(D2) \downarrow S_B$.

Both an STG and a circuit behavior can be characterized by their trace sets. Thus, one can compare in this way two different STG's, or two circuits, or an STG and a circuit. When an STG and a circuit are compared, we will always assume that the circuit is initialized in the state in which all internal signals are stable. Note that for standard implementations, this is a reasonable assumption because all the latches are connected to the external outputs, and hence when the latches are initialized to a proper state, all the internal signals will eventually become stable.

Theorem 5.3: If the excitation functions R_a and S_a of each output signal a in the TD derived from a correct STG are represented as the sums of cubes $c(-a_1), \cdots, c(-a_k)$, and $c(+a_1), \cdots, c(+a_k)$, respectively, where $c(*a_i)$ corresponds to the monotonous cover of some excitation regions $ER(*a_{i1}), ER(*b_{i2}), \cdots, ER(*d_{ik})$ and each region $ER(*a_i)$ is covered by exactly one cube, then both standard RS- and C-implementations are trace equivalent to the original STG with respect to the set of external signals.

Proof: We will refer to the outputs of AND- and OR-gates in the SOP's of the excitation functions as internal signals so as to distinguish them from the external signals that are present in the original STG.

Let us prove the statement by the induction on the length L of feasible sequences in the STG and in the circuit.

1: Let $L = 1$.

1.1: Assume $*a_i$ is feasible in the STG. We must show that an equivalent sequence exists in the implementation.

Clearly, transition $*a_i$ is enabled in the initial state of the STG. The initial states of the STG and the implementation coincide in their external signals. Therefore, if a is an input signal, the environment should enable a at the input of implementation, and $*a$ is feasible in the implementation. If a is an output signal, then some cover cube in the excitation function $ER(*a_i)$ must be "ON" in the initial state of STG. Let, e.g., $*a_i = +a_i$. The cover cube in $ER(+a_i)$ corresponds to an AND-gate in the SOP of S_a. Every internal signal of the implementation is stable in the initial state, which means that the output of the chosen AND-gate should be in state 1. From the similar consideration, it follows that output S_a, being an internal signal, is also in state 1. By the MC requirements, no cover cube of any $ER(-a_j)$ is turned "ON" in the initial state of the STG. Therefore, in the initial state of the implementation, all AND-gates in the SOP of the R_a, and the R_a gate itself, have their outputs at zero. Hence, the external output a is enabled in the implementation, and sequence $+a$ is feasible.

1.2: Suppose sequence $+a$ is feasible in the implementation. Let us show that the corresponding sequence is feasible in the STG. Every internal signal is stable in the initial state of the implementation. From this, it follows that a is an external signal. If a is an input signal, then, following the earlier consideration, we can conclude that some input transition $+a_i$ must be enabled in the initial state of the STG. If a is an output signal, the firing of $+a$ in the implementation is possible only if $S_a = 1$ while $R_a = 0$. Therefore, some AND-gate in the SOP of S_a must have its output at 1. This gate corresponds to a cover cube in the excitation function $ER(+a_i)$, and because in the initial state of STG the value of a is 0, a is enabled in the STG and sequence $+a_i$ is feasible.

Thus, we have showed that for any feasible sequence of length "1" in the STG, there is an equivalent sequence in the implementation and vice versa.

2: Let us now show that, if the statement is true for all feasible sequences of length $L = n$, then it remains true for all feasible sequences of length $L = n + 1$.

2.1: Feasible sequence in the implementation $\Rightarrow$ feasible sequence in the STG.

Assume there is a sequence q produced by the implementation such that the projection of q on the set of external signals equals $q' = r', *b, *a$, where sequence r' has length $n-1$, and let, e.g., $*a = +a$. Then q can be represented as $q = r, *b, p, +a$, where p is a subsequence containing only internal signals of the implementation.

Let us consider the signal network for function S_a.

The up transition of signal a has to be caused by transition $+S_a$ on the output of the SOP, and $+S_a$, in turn, by the switching of some AND gate T. Consider the state s of sequence q in which gate T is first time enabled to ensure the considered change $+a$. The projection of s on the set of external signals gives the state s', in which $a = 0$ and some cover cube of S_a is turned "ON." Therefore, in the TD derived from the STG, $s' \in ER(+a_i)$.

Case 1: State s is reached in q before $*b$.

Sequence $r', *b$ is feasible in the STG due to the inductive assumption. Starting from some point inside this sequence, signal a becomes enabled, and it should keep its enabledness in the final state $s1'$ of sequence $r', *b$ because the STG is correct. Therefore, sequence $r', *b, +a$ is feasible in the STG. This sequence is trace equivalent to q with respect to the external signals.

Case 2: State s is reached in the sequence q after $*b$.

According to the rules of the standard implementation, the inputs of the AND gate T are the external signals only. Therefore, in state $s1'$, which is the final state of sequence $r', *b$, signal a should be enabled, and we can again construct a feasible sequence $r', *b, +a$ in the STG.

2.2: Feasible sequence in the STG $\Rightarrow$ feasible sequence in the implementation.

Consider an arbitrary feasible sequence $q' = r', +a_i$ in the STG, where the subsequence r has length n. By the induction assumption, there exists a sequence r feasible in the implementation such that r and r' are equivalent by the set of external signals. Let state s be the final state of r. r' is equivalent to r, and therefore the projection s' of state s on the set of external signals belongs to the TD derived from the STG. In the TD state s', signal a is enabled because of the feasibility of $r', +a_i$. Therefore, $s' \in ER(+a_i)$ and has to be covered by some monotonous cube c. This cube is implemented by an AND gate T and, after the firing of r', this gate is either enabled or is already in state 1. If it is enabled, then by choosing a proper value for its delay, we can make feasible the sequence $r', +T$ in the standard implementation. The same is valid for the OR gate S_a. Thus, without loss of generality, we can consider that after firing the sequence r, signal S_a is equal to one. Signal a will not be enabled under such a condition only if $R_a = 1$. However, in state s', all cover cubes in R_a must be turned off. This means that any AND gate in R_a is either enabled and is going to switch to zero or is already in state 0. Then, again by choosing proper delays for the gates, we can make a feasible sequence r, p in the implementation, after which all AND gates of R_a and R_a itself will be set to zero (p contains only internal signals). Sequence r, p differs from r only in having extra (if any) transitions of the internal signals, and thus r, p is also equivalent to r' (because the equivalence relation is defined by external signals). After firing r, p, we have $S_a = 1$, while $R_a = 0$, and therefore signal a is going to change to one, i.e., in the implementation, the sequence $r, p{+}a$ is feasible. This sequence is equivalent to $r', +a_i$ that finishes the proof. $\square$

Theorem 5.4: If the excitation functions R_a and S_a for each output signal a in the TD are represented as the sums of cubes $c(-a_1), \cdots, c(-a_k)$, and $c(+a_1), \cdots, c(+a_k)$, respectively, where $C(*a_i)$ corresponds to the monotonous cover of some excitation regions $ER(*a_{i1}), ER(*b_{i2}), \cdots, ER(*d_{ik})$ and each region $ER(*a_i)$ is covered by exactly one cube, then both standard RS- and C-implementations are semimodular.

Proof: We have three groups of gates in such an implementation: gates for the output signals, OR-gates for the excitation functions, and AND-gates for the cubes of the excitation functions. (Since all the cubes $c(*a_i)$ for a are mutually disjoint, the standard implementation will always associate each such cube to a unique AND-gate.)

Let us consider each gate separately.

Case 1 (AND-Gates): Semimodularity can be violated only in transitions $0^* \rightarrow 0$ and $1^* \rightarrow 1$.

Assume that the output of a gate T, which corresponds to cube $c(+a_i)$, is in 0^*, and that cube $c(+a_i)$ covers the excitation regions $F' = \{ER(*a_{i1}), ER(*b_{i2}), \cdots\}$ of transitions from $F = \{*a_{i1}, *b_{i2}, \cdots\}$. It means that $C(+a_i) = 1$ in the corresponding state s of the external signals and $s \in ER(+a_i)$. Transition $0^* \rightarrow 0$ can occur on the output of T only if the state of the external signals wherein $c(+a_i) = 0$ is reached. By the definition of monotonous cover and from the equivalence between the standard implementation and the original STG (see Theorem 5.3), it can only happen within the quiescent region $QR(+a_i)$ (by condition 1) of Definition 5.5 $c(+a_i)$ can switch off only outside $ER(+a_i)$, while by condition 3), it should be "OFF" before $ER(+a_i) \cup QR(+a_i)$). In $QR(+a_i)$, signal a has already fired to one. Since in the excitation region $ER(+a_i)$ no cubes except for $c(+a_i)$ can have value "1," no AND-gates except for T can cause the firing of transition $+a$. So, to produce a $0^* \rightarrow 0$ transition on the output of T, cube $c(+a_i)$ has to be switched in the following way: $1 \rightarrow 0 \rightarrow 1 \rightarrow 0$ within the $ER(+a_i) \cup QR(+a_i)$, which contradicts condition 2) of Definition 5.5. Thus, transitions $0^* \rightarrow 0$ cannot occur on the T output.

Let us now show that transitions of the form $1^* \rightarrow 1$ also cannot occur on the output of T. If $T = 1^*$ in a state s of the external signals, then the value of $c(+a_i)$ is equal to zero and $s \notin ER(+a_i)$. T can become a stable "1" in a state $s1$ where $c(+a_i) = 1$. This can only happen via entering another excitation region, $ER(*b_j)$, where $*b_j \in F$. Let us show that for any sequence q from s to $s1$, there exists, e.g., a transition $+d_k$ ($+d_k \in F$) such that q should cross the excitation region $ER(-d_m)$. Indeed, if q passes only through the states within the $QR(+d_k)$, then in $ER(+d_k) \cup QR(+d_k)$, cube $c(+a_i)$ changes from one (in the states of $ER(+d_k)$) to zero in state s, and then to one again in state $s1$, which contradicts condition 2) of Definition 5.5. Therefore, $-d_m$ should fire in sequence q between s and $s1$. This firing occurs at state $s2$, in which $R_d = 1$ and $S_d = 0$. Then, in the implementation, before reaching state $s2$, the following transitions must take place: $T \to 0, S_d \to 0$. The firing of $-d_m$ at state $s2$ acknowledges these transitions on T and S_d. Therefore, upon reaching state $s1$, gate T is already reset and cannot be in state 1^*. Therefore, transitions $1^* \to 1$ cannot happen.

Case 2 (OR-Gate): For an OR-gate, at any moment only one input can be in state 1, and the sequence of input patterns is always $I \to O \to I$, where I denotes a pattern with one component in "1" and the remaining in "0," and O denotes the pattern where all components are in "0." So the output of an OR-gate repeats the changes on its "hot" input T. Hence, similar considerations that proved the semimodularity of T are also valid for OR-gates.

Case 3 (Output Gates): The semimodularity of output signals immediately follows from the equivalence of standard implementation and original STG with respect to output signals. □

Theorem 5.4 provides sufficient conditions for guaranteeing implementation correctness. There are several cases when these conditions can be softened.

- *Degenerative SOP:* An SOP implementation becomes degenerative, for example, if a cube c consists of one literal (b or $\overline{b}$) and/or the corresponding excitation function (e.g., S_a) consists of cube c only. Then we can remove the AND and OR-gate from its implementation and connect the output b directly to the corresponding input of C-element or RS latch. In this case, it is sufficient to demand from c that it be a correct cover, not necessarily a monotonous one. Stated differently, when c is turned "ON" and "OFF" in $QR(+a)$, it does not influence the output signal a, as the latter is already in state 1. At the moment of entering $ER(-a)$, the cube c will be reset (see Definition 5.2 for correct cover conditions).
- *Extending "don't cares" for the standard C-implementation*[4]: To operate correctly, an RS-latch based on NOR-gates requires that the R and S functions be disjoint, i.e., the following condition should be met: $R \cdot S \equiv 0$. The dual condition, $\overline{R} \cdot \overline{S} \equiv 1$, must be met for an RS-latch based on NAND-gates. For the standard C-implementation, neither of these conditions is necessary. That allows one to expand the *dc-set* for the R and S functions. Indeed, if the output a of a C-element is in state 1 and its input S_a also has the value "1," the value on the other input $\overline{R_a}$ is of no importance for the output behavior. Therefore, function R_a can be set to one *before* $ER(-a)$, i.e., in the states of $QR(+a)$ where $S_a = 1$, which formally violates the condition for correct covering (see Definition 5.2). Consequently, this condition can be relaxed for the case of standard C-implementation. Note that due to the monotonicity requirement for the function R_a, it has to go high in $QR(+a) \cup ER(-a)$ only once, except for those cases when function R_a consists of one literal. In the latter case, we do not need to preserve the monotonicity of R_a (see the previous item), and can consider all states of $QR(+a)$ where $S_a = 1$ as "don't cares" for R_a.

[4]The goal of this approach is analogous to the one described in [3]. The latter exploits the "observability don't cares" optimization technique, well known in synchronous circuit synthesis.

VI. Reduction to the MC Form

If the initial specification does not satisfy the MC requirement, it has to be altered to allow a hazard-free implementation.

One of the obvious ways is to change the specification semantically, e.g., by the reduction of concurrency [36] or by the signal reshuffling [23]. Such a process will certainly converge because, in the worst case, we may end up with a fully sequential specification without any parallelism. For such specifications to satisfy the MC-requirement, it is sufficient to ensure CSC by one of the existing methods. This approach, however, is subject to two important factors: performance (typically, it slows down the circuit operation) and I/O protocol (concurrency reduction must not change the behavior of the environment).

In this paper, we will rely on a more restricted sort of transformations, those preserving the language generated by the specification. The task thus stated is to reduce the specification to the MC-form by adding extra signals in such a way that these signals will be internal for the implementation, while for the outer observer, the implementation will show the same behavior as the initial specification, except, probably, for timing/performance characteristics.

The equivalence of our transformations is based on the notion of trace equivalence, introduced in Definition 5.7. Trace equivalence does not require the equivalence of input-output interfaces, i.e., the sets of input and output signals in trace-equivalent models can be different. If one has to preserve the input-output interface, then a stricter equivalence notion is needed, which requires the equivalence of input and output sets in both objects.

Let two STG's $G1$ and $G2$ have the same sets of input signals $I \subset A_1 \cap A_2$, and external output signals $O \subset A_1 \cap A_2$. If STG's G_1 and G_2 are trace equivalent with respect to the set $I \cup O$, then these STG's will simply be called *equivalent*.

The task of reduction to the MC form is now formalized as follows: add new signals to the original STG in such a way that the obtained STG will satisfy the MC conditions and will be equivalent to the original one.

Two questions however arise here: 1) Is such a reduction always possible? and 2) How to do it? We will see further that for a correct STG, the answer to the first question is positive. This fact will be proved in a constructive way, i.e., by presenting an efficient technique that allows the reducing of any STG to its MC form, so the solution for the second problem will be shown altogether.

The first attempt to find a general method of ensuring the MC requirements was made in [17] and [18]. The MC requirement was formulated there as a set of Boolean constraints under the rules of introducing additional signals. If the additional signals satisfy these constraints, the solution must have the MC properties. This solution can be found using the state-of-the-art Boolean satisfiability solvers. However, this approach allows handling of only small specifications, and the quality of the logic obtained is low.

We present here another approach based on direct transformations on the STG level, which is computationally more efficient and allows more flexibility in the reduction of logic.

In this section, we will restrict ourselves to correct STG's, with the additional requirements for STG's to be safe and for the corresponding TD's to satisfy UEC's and to have no states with the same binary codes.[5] Such STG's are called *strongly correct* STG's.

In fact, this does not limit the descriptive power of correct STG's. Similar to the methods of reduction to CSC form [8], [15], [22], [37], a TD, and the corresponding STG, can always be reduced to the form with all states encoded by different binary codes. We will assume that this transformation is performed beforehand. The unique entry condition is also not too restrictive. In Section VII-B, we will generalize this condition and show how to ensure this generalized property in an arbitrary STG. We have chosen to work with safe STG's because the procedures of reduction to the MC form are much simpler for them. Moreover, it was proved in [15] that any unsafe STG can be unfolded into an equivalent safe one. Besides, unsafe specifications are relatively rare in practice.

Let us solve the problem of STG reduction to the MC form in two steps:

1) STG reduction to the persistent form;
2) reduction of the persistent STG to the MC form.

A. Eliminating Nonpersistency

The methods for eliminating nonpersistency presented in this section generalize those of [38]. These methods apply structural transformations at the STG level, such as insertion of new events (see [29]), which preserve trace equivalence. For comparison, model transformations presented in [25] have been based on concurrency reduction.

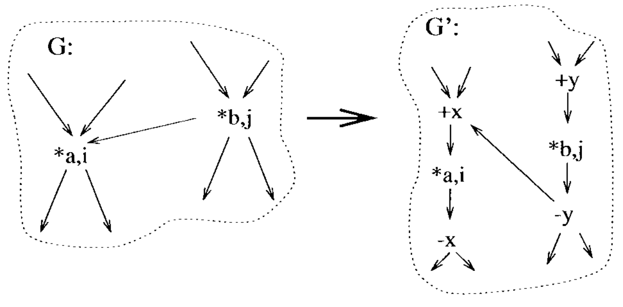
The following theorem gives an upper bound on the number of signals that have to be inserted to remove nonpersistency for a given pair of transitions in STG's with choice. Moreover, it shows where these signals have to be inserted.

Theorem 6.1: Assume that there is a nonpersistency between a pair of transitions $*b_j$ and $*a_i$ in a *strongly* correct STG. This nonpersistency can be eliminated by introducing two additional signals, and no new nonpersistencies arise.

Proof: Let $*b_j$ be a trigger nonpersistent transition to $*a_i$, i.e., $*b_{j+1} || *a_i$. We shall prove this theorem by developing an effective method for eliminating nonpersistency between $*a_i$ and $*b_j$ by inserting two extra signals x and y into STG. Three properties have to be checked:

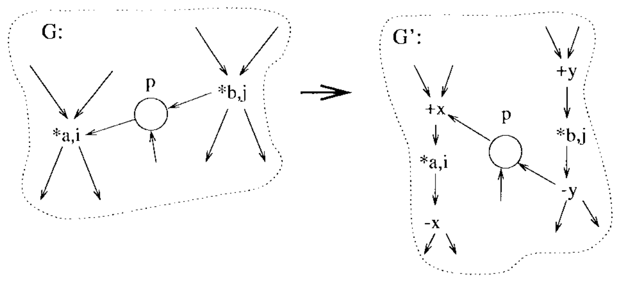
- the nonpersistency between $*a_i$ and $*b_j$ is eliminated and no new nonpersistencies have been created;
- the modified STG, G', is equivalent to the initial one, G;
- if the initial STG, G, has CSC and consequently allows direct circuit implementation, then the modified STG, G', also meets the CSC requirement.

[5]The latter requirement, called unique state coding, is stronger than the CSC condition and is taken just to simplify the presentation of this paper.



(a) (b)

Fig. 5. Elimination of nonpersistency without an intermediate place.



(a) (b)

Fig. 6. Elimination of nonpersistency with an intermediate place.

As $*b_j$ is a trigger transition to $*a_i$, it must be a predecessor of $*a_i$. There are only two cases to consider.

1) $*b_j$ is a direct predecessor of $*a_i$ [see Fig. 5(a)].
2) The relation of direct precedence between $*b_j$ and $*a_i$ is mediated by place p [see Fig. 6(a)]. This place is explicit in the STG and thus can have more than one predecessor transition.

Case 1: Assume that there is an arc between $*b_j$ and $*a_i$. Additional signals are inserted in the way shown in Fig. 5(b), i.e., all transitions that are the direct predecessors of $*a_i$ $(*b_j)$ in G become the direct predecessors of $+x$ $(+y)$, while all transitions that are the direct successors of $*a_i$ $(*b_j)$ in G become the direct successors of $-x$ $(-y)$.

Consider whether G' is persistent to all the newly introduced arcs. Arcs $(+x, *a_i)$ and $(+y, *b_j)$ are persistent because $+x \rightarrow *a_i \rightarrow -x$ and $+y \rightarrow *b_j \rightarrow -y$. Arc $(*a_i, -x)$ is persistent because in a correct STG, all the transitions of one signal have to be ordered, and thus $-x$ is ordered with all the transitions of signal a that are next to $*a_i$ (see the rules of transformation). The same holds for the arc $(*b_j, -y)$. Arc $(-y, +x)$ can be nonpersistent only if $+x || +y$. From the safeness of the original STG, it follows that $*b_j$ cannot happen for the second time without the firing of $*a_i$. It is easy to see that, under the considered rules of transformation, it would imply that $+y$ cannot happen twice without the firing of $-x$. Hence, arc $(-y, +x)$ is also persistent. All other input and output arcs of transitions $+x, -x, +y, -y$ can be nonpersistent only if the arcs corresponding to them in the original STG, G, were nonpersistent. This concludes the proof that the nonpersistency between $*a_i$ and $*b_j$ is eliminated and no new nonpersistencies have been created.

STG G' is obviously equivalent to the initial STG, as can be shown by projecting G' on the set of the original signals in G.

Let us prove that in the suggested transformation, all states of the TD are encoded by different binary codes, i.e., the strong correctness of the STG is preserved. Consider the excitation regions that correspond to the added transitions $+y, -y, +x, -x$. Let us exclude from the states of $ER(+y)$, e.g., the additional bits corresponding to signals x and y. Clearly, the obtained projection of $ER(+y)$ will coincide with the set of states $ER(*b_j)$ in the original TD. The original TD has no states with the same binary code. Hence, in $ER(+y)$, no state will be encoded with the code of some other state of the TD. Similar consideration is valid for all other excitation regions introduced diring this transformation.

We can therefore state that if G satisfies the conditions of the theorem, then G' also obeys the requirements of strong correctness.

Case 2: Assume that $*b_j \rightarrow p \rightarrow *a_i$, as shown in Fig. 6(a). In this case, a simple modification of the transformation for case 1 will exclude the nonpersistency between $*b_j$ and $*a_i$ [see Fig. 6(b)]. The equivalence between G' and G, as well as the preservation of CSC, can be shown in the same way as for case 1. □

Theorem 6.1 shows how any strongly correct STG can be reduced to the persistent form. For practical purposes, in many cases more efficient ways of eliminating nonpersistencies exist. A useful set of heuristics is presented in [19]. They include methods for eliminating a single nonpersistency by one signal, sharing of signals to eliminate several nonpersistencies at once, etc.

B. Reduction of Persistent STG to MC-Form

According to Theorem 5.1, the persistency property is necessary for the MC requirements, and it can be achieved using the results in Section VI-A. Thus, we can now consider the reduction to the MC form for persistent, strongly correct STG's.

First, we illustrate the method of reducing an STG to the MC form using a simple example of the specification shown in Fig. 7(a), with input signal c and output signals a, b, d. In this STG, all output signals are persistent. The Karnaugh map corresponding to the logic function of signal a is shown in Fig. 7(c). Excitation region $ER(+a)$ consists of three states. The only signal that is ordered with $+a$ is b, and thus cube b is chosen to cover $ER(+a)$. However, this cover is neither monotonous nor even correct because cube b also covers two states from the *off-set* of the function for a. To make this cover correct, we need to split cube b into two cubes $b\overline{d}$ and c, but this violates the MC requirement because *two* cubes will now be turned on in state 0110 (underlined in Fig. 7) inside *one* excitation region. Thus, function S_a does not meet the MC requirement, and is therefore hazardous when implemented by simple gates. For all other excitation functions, the MC requirement is satisfied, and the logic for their implementation is hazard free. The obtained set of excitation functions is

$$R_b = acd + \overline{a}d\overline{c}, \quad S_b = \overline{d} + a\overline{c}, \quad R_d = a\overline{b}, \quad S_d = c, \quad R_a = b\overline{c}d, \quad S_a = b\overline{d} + c. \tag{1}$$

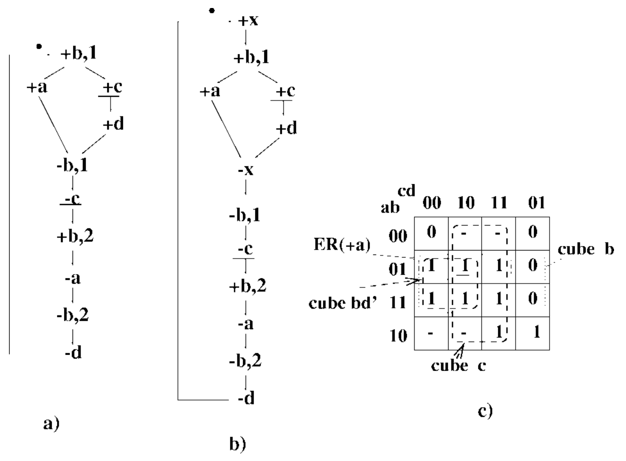


Fig. 7. Insertion of additional signal x to ensure MC-conditions.

To reduce the cover for $ER(+a)$ to the MC form, we need to distinguish between the case when signal b is equal to one after firing $+b_1$ (a has to be equal to one) and the case when signal b is still in one (after $+b_2$) but a has to become zero after $-a$. This can be done, for example, by adding signal x, which will be high in $ER(+a)$ and will become low before $-a$ [see Fig. 7(b)]. Such a transformation will result in the following logic:

$$b = a\overline{c} + x, \quad R_d = \overline{a}\overline{b}, \quad S_d = c, \quad R_a = \overline{c}b\overline{x}, \quad S_a = bx, \quad R_x = ad, \quad S_x = \overline{d}. \tag{2}$$

The area estimate for logic corresponding to the standard C-implementation for 1) and 2) (using the SIS library from [21]) shows that the hazard-free implementation is even smaller than the original hazardous one: 464 area units for 1) and 374 for 2). The area of hazard-free implementation can be further reduced to 344 units by extending "don't cares" for signal x. This allows one to implement signal x simply by a C-element with an inverted output. The result strongly depends on the place where the transitions of signal x are inserted. In our case, the logic is reduced because after adding signal x, signal b can be implemented by a hazard-free combinational circuit without a latch.

This is an issue of the logic optimization strategy.

The following statement, similar to Theorem 6.1, holds for the reduction to the MC form.

Theorem 6.2: In a persistent, strongly correct STG G, let the MC-requirement be violated for some excitation region $ER(*a_i)$ that corresponds to transition $*a_i$. Then, by inserting new signals, a persistent, strongly correct STG G', equivalent to G, can be derived, where this violation is eliminated and no new violations of the MC-requirement arise.

Proof: We first consider the case when $*a_i$ has no places in its predecessor set, i.e., $*a_i$ is directly preceded by transitions $\{*b1_{j1}, \cdots, *bn_{jn}\}$ (without loss of generality, we can assume $*bk_{jk} = +bk_{jk}, k = 1, n$ and $*a_i = +a_i$). We will introduce additional signals x and $y1, \cdots, yn$ into G in the way shown in Fig. 8.

Consider the MC properties of $+a_i$ and the newly introduced transitions $+x, -x, +y1, -y1, \cdots, +yn, -yn$.

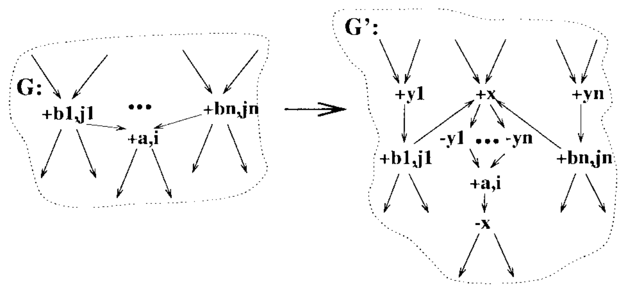


Fig. 8. Reduction to MC-form.

1) If $ER(+bk_{jk})$ was covered by MC cube $c_{bk_{jk}}$ in G, then we will ensure the MC cube $c_{bk_{jk}}\overline{bk}$ (if $*bk_{jk} = -bk_{jk}$, then this cover cube will be $c_{bk_{jk}}bk$). Indeed, $c_{bk_{jk}}\overline{bk}$ evidently covers all states of $ER(+yk)$. The correctness of cover $c_{bk_{jk}}\overline{bk}$ follows from the correctness of cover $c_{bk_{jk}}$, since $c_{bk_{jk}}\overline{bk}$ cannot be turned on outside $QR(+yk)$, because it would imply that $c_{bk_{jk}}$ had to be turned on somewhere outside $QR(+bk_{jk})$. Last, cube $c_{bk_{jk}}\overline{bk}$ changes only once in any trace inside $QR(+yk)$ because it is reset by the $+bk_{jk}$ transition and cannot be set again inside $QR(+yk)$ due to the persistency of G. In fact, the next transition of $bk, *bk_{kj+1}$ (a down transition, $-bk_{kj+1}$, for our choice of transition polarity) can occur only after $+a_i$ in G, and thus $*bk_{kj+1}$ occurs after $-x$ in G'.[6]
2) From the similar consideration, it is easy to show that cube x is an MC for $ER(-yk)$, and cube $x\overline{y1}\cdots\overline{yn}$ is an MC for $ER(+a_i)$.
3) Cube a is a correct cover for $ER(-x)$ because in all states where signal x has to be at one, signal a is at zero and cube a is turned off ($-a_i$ cannot be concurrent to $+x$; otherwise, in the original G, $-a_i$ is concurrent $+a_i$, which contradicts the correctness of G). As this cube consists of only one literal, the correctness of G guarantees hazard-free behavior (see the notes about the degenerative cases discussed at the end of Section V), so we need not check the monotonicity of the cover.
4) Let us show that the MC for $+x$ can be represented by cube $c_x = y1\cdots yn\; b1\cdots bn$. Indeed, the monotonicity of this cube follows from the persistency of G (any changes of signals bk should happen only after $-x$), while the correctness of c_x is ensured by the reset of yk immediately after $+x$.
5) For all direct successors of $+bk_{jk}$ or $+a_i$, we will replace, in the corresponding cover cubes, literals bk and a with $bk\; yk$ and $a\overline{x}$, respectively. This clearly will not violate their MC-properties.

STG G' is obviously equivalent to the initial STG, as can be shown by finding the projection G' on the set of the main signals from G.

[6]If $+bk_{jk}$ has no monotonous cover in G, then it might be that $+yk$ has no monotonous cover in G'. However, as cube yk is an MC for $ER(+bk_{jk})$ in G', the insertion of $+yk$ does not introduce new MC violations, which is necessary for progress.

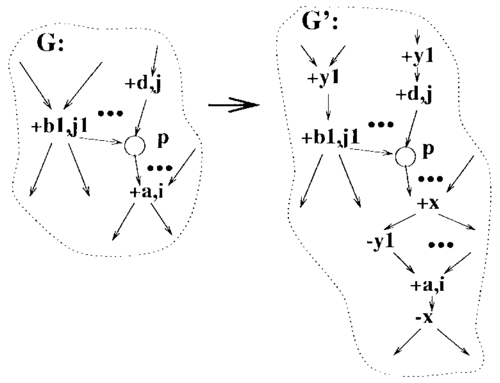


Fig. 9. Reduction to MC-form with an intermediate place.

Last, this transformation preserves CSC in exactly the same way as in Theorem 6.1.

6) The case when $+a_i$ has an input place p among its direct predecessors can be treated in a similar way. However, p can have several transitions directly preceding it, and therefore to preserve the consistency of the STG, the up transitions of the corresponding additional signal should be inserted before each of them (see Fig. 9). □

Theorems 6.1 and 6.2 prove that any strongly correct STG G can be transformed into an equivalent STG G' with MC properties. This implies that for the adopted architectures (SOP combinational logic plus a latch), we have found a method for the hazard-free implementation of an STG in the basis of simple gates.

In the following section, we extend the reduction methods in two ways. First, we will show how simple modifications to the standard implementation structures can help ensure the MC requirement. Then we will generalize the MC-theory to allow the hazard-free implementation of a wider class of specifications. STG's without the UEC and models with OR-causality will be considered.

VII. Extensions of the MC-Theory

A. Extending Implentation Structures

1) Inverse Feedback: The MC conditions can be violated for a cover cube if this cube has nonmonotonous behavior in the corresponding quiescent region (changes more than once in it). One way to ensure the monotonicity of the cube is to reset it *immediately* when entering the quiescent region. Note that in all states of excitation region $ER(*a_i)$, the value of signal a is inverse to its value in all states of the following quiescent region. For example, in the states of $ER(+a_i)$, signal a is equal to zero, and in all states of the quiescent region $QR(+a_i)$, a is equal to one. Signal a has a constant value in $ER(*a)$ and so can be added to the cover cube. This will restrict the cover only by the states in the excitation region. In such a case, the requirement for the cover cube to change only once inside the quiescent region is fulfilled automatically.

The use of *self-dependent covers* of excitation regions allows softening of the MC-requirements and reformulation of Definition 5.3 in the following way. A cover cube $a'c(*a_i)$, where $a' = a$ if $*a_i = -a_i$ and $a' = \overline{a}$ if $*a_i = +a_i$, is a monotonous cover for $ER(*a_i)$ if 1) $c(*a_i)$ covers all states of $ER(*a_i)$ and 2) $c(*a_i)$ does not cover any reachable state outside $ER(*a_i) \cup QR(*a_i)$.

At the implementation level, a self-dependent cover involves an additional inverse feedback wire from the latch output to its corresponding signal network.

This optimization is possible because the inverse feedback is never used to switch the output of a latch. It is only used to reset the gates in the corresponding signal network after the switching of the latch has already occurred. However, some precautions should be taken if inverse feedbacks are used together with other optimizations (e.g., it limits the possibilities of logic sharing).

2) Polyterm Covers: Let us extend the basis of standard gates for the implementation of SOP structure to complex gates (AND-OR). Clearly, since the basis is more powerful, the requirements for hazard-free implementation can be relaxed. Despite such a relaxation, the main objectives of the monotonous cover of excitation functions remain valid. These are:

1) to preserve the *one-hot discipline,* which assumes that in the SOP structure of the signal network, only one gate can be turned on at a time;
2) to switch the output of the gate that triggers the latch monotonously, i.e., only *one* transition can occur at the output of this gate before the latch changes its output value.

Definition 7.1 (Polyterm Monotonous Cover): The union of cubes $C = \{c_1(*a_i) + \cdots + c_r(*a_i)\}$ is a *polyterm monotonous cover* for $ER(*a_i)$ if:

1) C covers all states in $ER(*a_i)$;
2) the logic function corresponding to C changes at most once in any trace of states inside $ER(*a_i) \cup QR(*a_i)$;
3) C does not cover any reachable state outside $ER(*a_i) \cup QR(*a_i)$.

Definition 7.1 extends the MC conditions for a polyterm implementation. Evidently, similar to a single-term monotonous cover, the polyterm cover can be generalized for the case of several excitation regions covered by the same set of cubes C (see Definition 5.5).

In practice, however, the complex gate implementation requires additional correctness criteria since achieving atomic behavior for a complex gate with input inverters is not always possible. To justify the use of input inverters (see Section IV), we showed that the standard implementations are robust because a malfunciton (i.e., nonhazard-free operation) can only happen if the delay of the inverter is greater that the delay of the entire region network. When a complex gate depends both on the direct (a) and inverted $(\overline{a})$ values of some signal a, then the conditions for correct operation are determined by the result of the race between the inverter $\overline{a}$ and the wire a at the inputs of the complex gate. Clearly, the conditions for hazard-free operation in this case should be stricter than in the case when input inverters are in a race with the whole signal network.

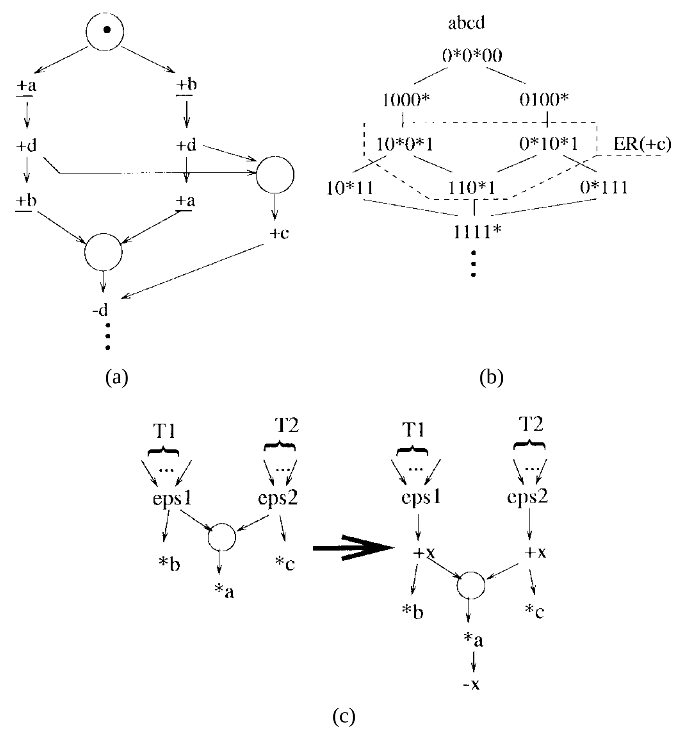


Fig. 10. Generalized unique entry condition.

For hazard-free operation of a complex gate with input inverters, to guarantee the atomic behavior of a gate, the following sufficient requirement can be added to the three conditions of Definition 7.1:

4) Any signal transition $u \overset{*b}{\rightarrow} v$ inside $ER(*a_i)$ is an internal transition for at least one of the cubes in C, i.e., both u and v are covered by some $c_j \in C$.

This condition implies that any transition within one excitation region is *internal* for at least one of the cover cubes. It can be informally viewed as a generalization of the static hazard elimination condition, which is used to implement Boolean functions in the fundamental mode [30], [34], [38]: every input transition should be internal for at least one cube of the cover that implements the function.

Condition 4) would be, for example, violated for a polyterm cover $\{c = a\overline{b} + \overline{a}b\}$ corresponding to an XOR gate if both its inputs, a and b, change concurrently within the same excitation region of signal c.

B. Extending Class of Specifications

1) Revising the Unique Entry Condition: Up to now, we required the excitation regions of STG to satisfy the UEC requirement, i.e., to have only one minimal state. To illustrate that the violations of UEC are not necessarily always dangerous, let us consider the example. $ER(+c)$ in Fig. 10(b) has two minimal states: 10*0*1 and 0*10*1. However, both of them are triggered by the rising transitions of d, and cube d is a monotonous cover for $ER(+c)$.

Hence, there is a simple way for generalizing the unique entry condition.

Definition 7.2 (Generalized Unique Entry Condition): An excitation region is said to satisfy the *generalized unique entry* *condition* if each of its minimal states has the same set of trigger transitions.

If an excitation region satisfies the generalized UEC, it can be covered by a single cube using the technique from Section VI.

Let us consider a TD that violates the generalized UEC condition, i.e., it has an excitation region $ER(*a)$ in which there are minimal states that are entered by different sets $T1$ and $T2$ of trigger signals. It can be shown that in correct STG's, such states are reached only by alternative branches. Therefore, the reduction of an STG to the generalized UEC form can be done by inserting an additional signal x [see Fig. 10(c)], where $eps1$ and $eps2$ are silent transitions of STG. It can be observed that after this transformation, there will be two seprate excitation regions for $+x$, each with a single minimal state. At the same time, the generalized UEC will be satisfied for $*a$. This technique, described here only informally, appears to be effective in most practical cases.

2) Modeling Processes with OR*-Causality:* The class of STG's considered so far allows only one type of causal relation between signal transitions, which simply inherits the enabling and firing rules of the underlying Petri net transitions. It is called AND-causality, as every transition can occur only if *all* of its direct predecessors have occurred. To specify arbitrary processes in semimodular circuits, OR-causal relations between signal transitions may also be needed [15], [38]. In the latter case, a transition can occur when *at least* one of its direct predecessors has occurred. It has been proven [40] that OR-causality cannot be adequately captured in the class of free-choice PN's (and hence in the class of correct STG's) at the level of net transitions without involving complex labeling mechanisms.[7] OR-causality can, however, be expressed if we introduce in the model an additional type of arc specific to OR-causal relations. Such a model, called a *change diagram,* (CD) was originally proposed in [39] (without choice) and further developed (to allow choice) in [40]. This extension is of importance because any circuit with an OR-gate (or, dually, an AND-gate) has an OR-causal relation if *concurrent* rising (dually, falling) transitions occur at the inputs of the gate.

The formal definition of change diagrams is based on two types of precedence relations between transitions in asynchronous circuits.

1) The *strong precedence* relation between transitions $*a$ and $*b$, usually depicted by a solid arc in the graphical representation of change diagrams, means that that $*b$ cannot occur without the occurrence of $*a$.
2) The *weak precedence* relation between transitions $*a$ and $*b$, usually depicted by a dashed arc in the graphical representation, means that $*b$ may occur after an occurrence of $*a$. But $*b$ may also occur after some other transition $*c$, which is also weakly preceding $*b$, without the need for $*a$ to occur.

[7]Modeling the same event (corresponding to a single excitation region) with several transitions at the net level gives rather powerful means but may lead to an explosion in the size of the net.

Definition 7.3: A CD G is a triple $\langle \mathcal{D}, A, \lambda \rangle$, where $\mathcal{D} = \langle T, \mathcal{S}, \mathcal{W}, M_0 \rangle$ is a marked directed graph[8] with a set of vertices T; two types of arcs, $\mathcal{S} \subseteq (A \times A)$ for *strong* precedence and $\mathcal{W} \subseteq (A \times A)$ for *weak* precedence; the initial marking $M: \mathcal{S} \cup \mathcal{W} \rightarrow N$ on the arcs; A is the set of signal transitions; and $\lambda: T \rightarrow A$ is the unique labeling function.

The strong and weak precedence relations must satisfy the following: 1) they are mutually exclusive, i.e., $(*a, *b) \in \mathcal{S}$ implies $(*a, *b) \notin \mathcal{W}$ and vice-versa, and 2) all the predecessors of a transition $*a$ must be either of the *strong* type or of the *weak* type. Hence, the set of transitions A is partitioned into AND-*type* transitions (with strong predecessors) and OR-*type* transitions (with weak predecessors).

The firing rule of CD's is similar to that of PN's, with arcs playing the role of places and flow relation elements at the same time. Each arc is assigned an integer *marking*, which, unlike a PN marking, can be *negative*. Initially, each arc in M has a marking of one, and each arc not in M has a marking of zero.

- An AND-*type* transition is enabled if *all* its predecessor arcs have a marking greater than zero.
- An OR-*type* transition is enabled if *at least one* predecessor arc has a marking greater than zero.

When an enabled transition fires, the marking of each predecessor arc is decremented and the marking of each successor arc is incremented.[9] Similar to STG's, CD's generate state-transition semantics represented by TD's. Due to the absence of choice in CD's, the TD's generated by them have no conflict states. It was proven in [15] that the modeling power of correct CD's exactly coincides with that of semimodular TD's. The definitions of liveness, boundedness, safeness, and (strong) correctness, applied to STG's in Section II-B, can be trivially extended to CD's.

Fig. 11(a) shows an example of a process with OR-causality specified by a CD. Unlike all other (AND-type) transitions, transition $+c$ (OR-type) fires if at least one of the transitions $+a$ or $+b$ have fired. The TD generated by this CD is shown in Fig. 11(b).

3) Implementation of Processes with OR*-Causality:* At the TD modeling level [Fig. 11(b)], OR-causality always leads to the violation of UEC conditions. Contrary to the violations of UEC in STG's, however, in CD's, different minimal states of an excitation region are entered by different *concurrent* (not conflict) transitions. Hence, the technique of reduction to the generalized UEC presented in Fig. 10(c) will not work here.

Nevertheless, a simple gate implementation of OR-causal processes can be found. For example, it was proven [38] that any semimodular TD can be implemented in the functional basis of two-input NAND and two-input NOR gates. Together with the above-mentioned result of [15] about a tight relationship between CD's and semimodular TD's, this

[8]Note the difference between this type of marked graph and the one that is a PN subclass, defined in Section II-B.

[9]The marking of a predecessor arc $(*a, *b) \in \mathcal{W}$ of an OR-*type* transition $*b$ can become negative as a consequence of a firing of $*b$ due to positive marking on some other arc $(*c, *b) \in \mathcal{W}$. It can then return to zero when $*a$ fires. After the next $*a$ firing, the marking of arc $(*a, *b) \in \mathcal{W}$ can become zero or positive again.

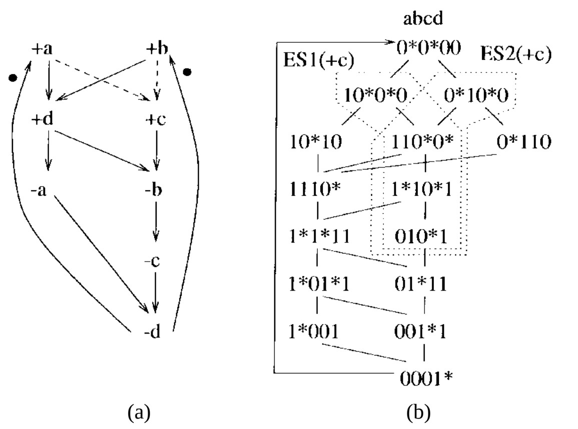


Fig. 11. (a) Example of a CD with OR-causality and (b) corresponding semimodular TD.

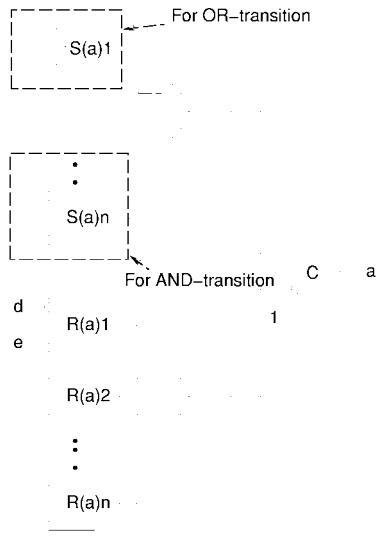


Fig. 12. Generalized standard C-implementation.

implies that any correct CD is also implementable in the same logic basis. The generic implementation constructions proposed in [38], however, are area ineffcient and hence impractical.

We are therefore interested in an extension of the basic MC-form that would allow OR-causal signal transitions to be produced by their signal networks. The major problem with the use of MC-form in an OR-causal operation is as follows. The number of cubes that cover the excitation region corresponding to an OR-transition is at least equal to the number of minimal states. Moreover, these cubes can be simulateneously turned on in this excitation region, thereby breaking up the correctness of the MC-based simple gate implementation. For example, in the model of Fig. 11, the *subregions* $ES_1(+c)$ and $ES_2(+c)$, induced by their respective minimal states 10*0*0 and 0*10*0, give rise to the two-cube, complex cover $a\overline{d} + b$. Both cubes cannot, however, be separated into two simple AND gates without hazards—these cubes are both turned on in state 110*0*, which belongs to $ER(+c)$.

To avoid the above problem in a simple gate implementation, we must make sure that the cubes that cover excitation regions with OR causality consist of single literals and hence do not need explicit AND gates in the first layer of the SOP structure. Thus, any MC implementation of a CD with OR would have simple OR gates in the first layer of its signal network, as shown in Fig. 12.

Informally, an *OR-monotonous cover* is a cover in which an excitation region of an OR-transition is covered monotonously by a set of single literal cubes. A more formal treatment requires the following definitions.

Let $ER(*a)$ be an excitation region of an OR-transition $*a$. Let $ER(*a)$ have a set of minimal states $\{u_1, \cdots, u_k\}$. An *excitation subregion* $ES_i(*a)$ is a subset of states in $ER(*a)$ that are reachable inside the region from minimal state u_i. Evidently, if for each subregion $ES_i(*a)$ we can find its own monotonous cover cube c_i, then the set of cubes $\{c_1, \cdots, c_k\}$ will satisfy the monotonous polyterm cover conditions for the whole region $ER(*a)$. In the CD of Fig. 11, the *subregions* $ES_1(+c)$ and $ES_2(+c)$ satisfy the polyterm cover $a\overline{d} + b$.

A way toward a simple gate implementation of CD's with OR-transitions is given by the following lemma.

Lemma 7.1: Assume that $*b_1$ is a direct predecessor of an OR-transition $*a_i$ and that signal b has nonmultiple transitions—i.e., only one pair, $*b_1$ and $*b_2$, of opposite transitions (one rising and one falling) occurs in the CD. Let the minimal state u_j corresponding to subregion $ES_j(*a_i)$ of $ER(*a_i)$ be entered by transition $*b_1$. If transition $*b_2$ is ordered with $*a_i$ and the next transition of $a, *a_{i+1}$, such that $*a_i \Rightarrow *b_2 \Rightarrow *a_{i+1}$, then cube b is a monotonous cover of the subregion $ES_j(*a_i)$.

Proof: Let us assume, without loss of generality, that $*a_i = +a_i$ and $*b_1 = +b$ and, consequently, $*a_{i+1} = -a_i, *b_2 = -b$. Cube b satisfies the following MC-conditions.

1) Cube b covers all states of $ES_j(+a_i)$ because it will be turned off only after $ER(+a_i)$ will be passed $(+a_i \Rightarrow -b)$.
2) Cube b cannot cover any state outside $QR(+a_i)$ because it will be turned off before passing $QR(+a_i)$ $(-b \Rightarrow -a_i)$, and $+b$ is the only transition that can turn it on.
3) Inside $QR(+a_i)$, cube b changes monotonously because transitions $+b$ and $-b$ are nonmultiple. □

If all direct predecessors of an OR-transition satisfy the conditions of Lemma 7.1, we will say that the corresponding excitation region has an *OR-monotonous cover*.

Corollary 7.1: If an OR-transition $*a_i$ with direct predecessors $+b, -c, \cdots$ has an OR-monotonous cover, then the excitation function for $ER(*a_i)$ can be implemented in a hazard-free way by an OR-gate $b + \overline{c} + \cdots$.

According to this corollary, to implement a CD in the basis of simple gates, it is sufficient to ensure the OR-monotonous cover conditions for all OR-transitions. This would satisfy the C-implementation shown in Fig. 12. The following theorem is a direct consequence of the theorems in Section V and the above corollary.

Theorem 7.1: If each excitation region (OR-excitation region) for an output signal of a correct CD satisfies the monotonous (OR-monotonous) cover conditions, then the generalized RS- and C-standard implementations are semimodular.

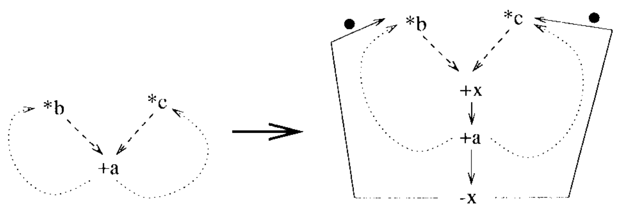


Fig. 13. Introducing the general synchronizer in CD.

The next question is whether any correct CD can be implemented in an MC-form. The proof of the following theorem presents a basic technique of how a CD that violates the OR-monotonous cover condition can be transformed to an MC-implementable form.

Theorem 7.2: In any correct, safe, live CD, the conditions of OR-monotonous cover can be ensured for all OR-transitions by introducing additional signals.

Proof: Let us show that a safe, live CD can always be transformed to a behaviorally equivalent CD, such that for any OR-transition $*a$ there will be found a transition that is ordered with $*a$ and with all its direct predecessors. This transition is called the *general synchronizer*. Without loss of generality, we can assume that $*a = +a$, and it has only two direct predecessors, $*b$ and $*c$. Then, the safeness of the CD implies that $*b$ and $*c$ cannot fire twice without the occurrence of $+a$. It means that for any i, $+a^i \Rightarrow *b^{i+1}; +a^i \Rightarrow *c^{i+1}$, where $+a^i$ denotes the ith instantiation of $+a$. Thus, without violating the initial ordering, we can the transform the CD by adding signal x, as shown in Fig. 13 (dotted arcs denote the transitive ordering relations).

(We now assume that a general synchronizer $*g$ is found in the original CD or that the CD is transformed to the form with the explicit general synchronizer like in Fig. 13.)

There are two possible cases where a general synchronizer $*g$ may be positioned relative to $+a$.

Case 1: If $*g$ precedes the transition $-a$, where $-a$ is the next (to $+a$) transition of signal a, then the transformation shown in Fig. 14(a) will ensure the OR-monotonous conditions for $+a$. (The dotted lines in Fig. 14(a) denote the transitive ordering relations).

Case 2: If $-a$ precedes $*g$, then the cube corresponding to one of the OR-causes will be turned on even after the signal a is reset, which violates the conditions of correct cover. In this case, the OR-transition itself must also be replaced [see Fig. 14(b)]. □

The transformations suggested in this proof are not aimed to ensure optimality in achieving the OR-monotonous cover conditions. In practice, the number of additional signals can be much less. To illustrate this, let us return to the CD from Fig. 11(a). The OR-monotonous cover conditions are violated for OR-transition $+c$ because $-a||+c$ and $-a||-c$, which imply that cube a for the OR-cause $+a$ will still be on after c goes low. One additional signal x ensures the OR-monotonous cover for $+c$ (see Fig. 15). The up-excitation function for c will be $S_c = x + b$. Note that the introduction of x does not cause any new noncorrectness.

In such cases, we first need to insert additional signals reducing the specification to an OR-monotonous form (see transformation in Fig. 16) and then apply the generalized standard implementation with OR-gates.

The above implementability results were proven for CD's, which do not allow input choice in specifications. To have a combination of OR-causality and choice within the same model, one could use a unified formalism, e.g. causal logic nets (CLN's), defined in [40]. This model, pictorially similar to STG's, associates with every transition a Boolean function, defined on the set of predecessor places, thus determining the type of causality. An interested reader may refer to [40] for details, including a classification of OR-causality types and different types of firing rules. In principle, the transformation techniques aimed at OR-monotonicity, described here for CD's, can be applied to a subclass of correct CLN's, in which places involved in OR-causal enabling functions are not used for choice.

VIII. Experimental Results

The proposed approach has been tested on the examples presented in this paper and on the known set of benchmarks from [21]. The CAD tool "FORCAGE" [15] was used to derive the excitation functions and check the implementations with respect to their freedom from hazards.

To evaluate the efficiency of our approach, the obtained circuits were realized in the SIS library of simple gates, and their area and delay were estimated. The closest method of implementation in simple gates was suggested by Beerel *et al.* [2]. The circuits obtained by this method are semimodular and thus are hazard free under any distribution of gate delays. We will give the comparison of our solutions to those obtained by [2]. Unlike both the technique of [2] and our method, the approach developed in [21] ensures hazard-free properties by selecting the right ratios between gate delays (implemented in the SIS tool). We will compare our method to that of [21] using the area and delay estimates from [2] taken for bounded delay synthesis from SIS tool.

The delays of implementations in [2] and [21] were evaluated through the worst case delays of the networks for the signal implementation. This is not completely relevant to the circuit performance, as neither critical cycles nor cycle times are taken into account. However, it relates to the speed properties of a circuit because the faster the components, the higher the circuit speed. The delay of a signal network depends upon its depth and upon the delays of its gates, which in turn are determined by the gates' fan-in. To be in correspondence with the experimental results from [2] and [21], we used the same strategy for the evaluation of the circuit delay.

Almost all specifications from the set of benchmarks described in [2] and [21] satisfy the MC properties. However, by choosing the appropriate type of architecture (RS- or C-standard), one can achieve area and/or delay reduction. A particular optimization strategy can also strongly influence the circuit's area and performance.

The classical asynchronous logic transformations [34] are not speed-independence preserving. Our method allows optimization at different levels. At the level of MC transformations, one can optimize the circuit area and delay by inserting

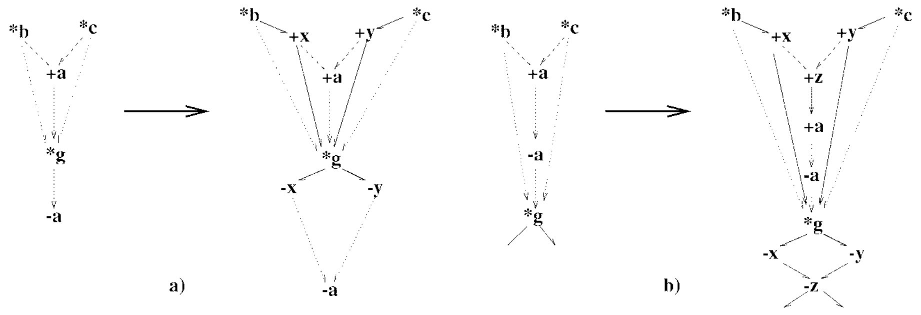


Fig. 14. General ways of CD reduction to OR-monotonous form.

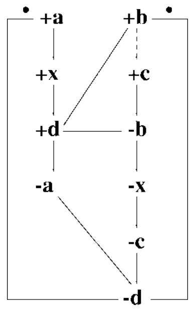


Fig. 15. Reduction of CD to OR-monotonous form.

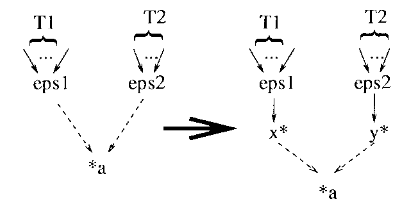


Fig. 16. Process with OR causality.

TABLE I
EXPERIMENTAL RESULTS

Circuit	*trans./ states*	SIS Area/Del	Stanford Area/Del	C-impl Area/Del	RS impl Area/Del	Area opt. Area/Del/Type
chu133	14/24	352/5.2	240/4.8	216/4.8	312/3.6	216/4.8/C
chu150	14/26	232/7.0	240/4.8	232/4.8	312/3.6	232/4.8/C
chu172	13/12	104/1.6	152/3.6	112/2.6	112/2.4	112/2.4/RS
converta	14/18	432/6.8	520/6.0	320/4.8	432/3.6	320/4.8/C
ebergen	14/18	280/5.6	344/4.8	280/4.8	328/3.6	280/4.8/C
full	8/16	224/5.2	112/2.4	112/2.4	208/3.6	112/2.4/C
hazard	10/12	296/6.6	256/4.8	240/3.6	224/3.6	224/3.6/RS
hybridf	16/80	274/6.6	152/2.4	152/2.4	312/3.6	152/2.4/C
nowick	16/20	264/6.6	504/6.0	456/6.0	376/3.6	376/3.6/RS
pe-send-ifc	53/117	1232/12.2	2072/7.2	1632/7.2	1480/4.8	1480/4.8/RS
qr42	14/18	280/5.6	344/4.8	280/4.8	328/3.6	280/4.8/C
vbe10b	22/256	1008/10.0	800/6.0	784/6.0	640/4.8	640/4.8/RS
vbe5b	12/24	272/4.2	264/4.8	192/3.6	232/3.6	192/3.6/C
vbe5c	12/24	224/5.2	264/4.8	200/3.6	224/3.6	200/3.6/C
wrdatab	24/216	824/7.0	840/4.8	744/4.8	816/4.8	744/4.8/C
var1	8/12	184/3.0	256/5.0	240/3.8	240/5.0	240/3.8/C
xyz	6/8	120/3.2	200/4.8	192/4.8	160/3.6	160/3.6/RS
totals		6602/102	7560/82	6384/74.8	6736/65.0	5960/67.40

TABLE II
COMPARISON WITH SIS TOOL AND STANFORD TOOL

Parameter	SIS	Stanford	C-impl	RS-impl	Area opt.
Area	1	1.15	0.97	1.02	0.90
Delay	1	0.80	0.73	0.64	0.66

extra signals. If the specification already satisfies the MC properties, then we choose for each signal a monotonous cover with the minimum literal count among all possible monotonous covers. For speed-independent standard C-implementation, we use gate-level local optimization, similar to that described in [2] but more aggressive. This includes gate sharing and converting AND and OR gates into faster and more area-efficient NAND and NOR gates. Whenever there is a monotonous cover for a signal a such that $S_a = \overline{R_a}$, the signal network for signal a can be implemented as a simple two-level combinational circuit for S_a, without a latch. We also actively use the expansion of the *dc-sets* for the R and S functions in the C-implementation, as described in Section V.

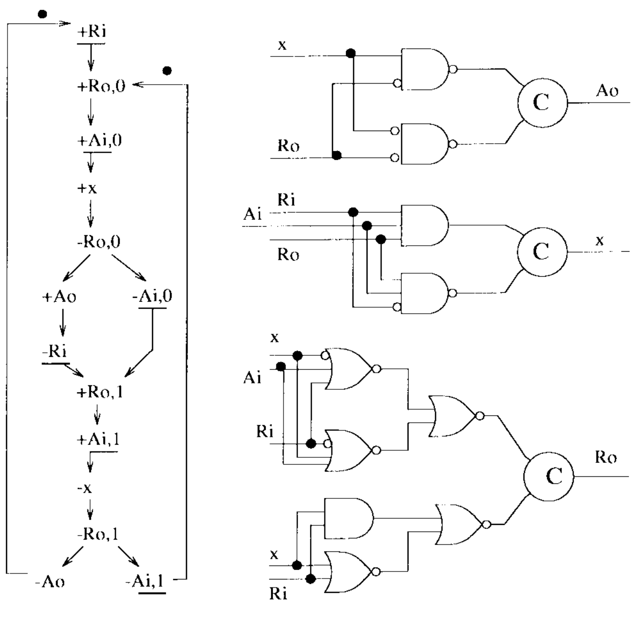
For the standard RS-implementation, we additionally allow merging the OR gate from the two-level combinational net with the NOR-gates that implement the RS-latch. This reduces the circuit delay (the depth of the signal network decreases by one) and area while still keeping the circuit in the simple gate basis.

The results are shown in Table I. The column labeled "Trans./states" shows the number of signal transitions in the initial STG specification and the number of states in the corresponding transition diagram. The columns labeled "Area" give the total area (excluding routing) of each circuit, using a "generic" standard cell library from SIS. The columns labeled "Del" give the maximum delay inside *one* signal network based on the SIS conventions for delay estimates. Although this method does not allow one to observe the actual cycle time of the circuit, we had to choose it in order to be compatible with the delay estimate for the Stanford and SIS methods. The columns labeled "SIS" and "Stanford" are directly borrowed from [2]. The numbers for SIS were obtained in [2] under the assumption of the fixed delay model (when the lower and upper bounds for gate and wire delays coincide) with an optimization script to minimize area. The column labeled "C-impl" presents the area and delay estimate for the locally optimized (for area) standard C-implementation. The column labeled "RS-impl" gives the area and delay estimate for the area-optimized standard RS-implementation. The last column shows the best between standard C- and RS-implementations with respect to area.

We summarize the experimental results in Table II. It puts the area and delay for the Stanford method and the methods presented in this paper against the implementation obtained by the SIS tool. The total area and the delay for all circuits from the benchmark implemented by SIS was taken equal to one.

These results show that, in comparison with the SIS implementation, our area-optimized standard implementation on average reduces the area by about 10% and increases the performance by about 34%. The C-implementation and the RS-implementation, separately, produce approximately the same area (-3 and $+2\%$) and increase the speed by about 27 and 36%, respectively. SIS needs to pad extra delay lines to ensure hazard freedom. These delays represent a significant fraction of the overall delay through the circuit. By contrast, speed-independent standard implementations provide hazard freedom by construction. This is the reason for our speed advantage in comparison with SIS.

In comparison with the Stanford method, our optimized implementation reduces on average the area by about 21% and increases the speed by about 18%. The C-implementation and the RS-implementation separately give an area reduction of 16 and 11% and speed increase of 9 and 20%, respectively. Fig. 17 shows why our C-implementation gives some benefits in comparison with [2] for the "converta" example. The initial STG is shown in Fig. 17(a). Fig. 17(b) shows the circuit from the benchmark of [2], with the area 520 units and delay 6.0 units. Fig. 17(c) shows our standard C-implementation with the area 360 units and delay 3.6 units. The circuit area was reduced due to using the extended *dc-set* for the up-function of signal Ro, choosing cover cubes with all literals noninverted for signal x and better local optimization for signal Ao. The area can be further reduced, to 320 units, by substituting an XOR-gate, available in the asynchronous library from SIS, for the three gates as shown in Fig. 17(c).



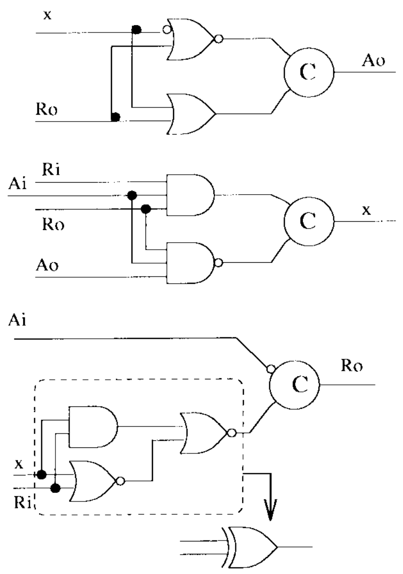


Fig. 17. (a) STG specification for "converta." (b) Stanford's implementation with area 520 units. (c) Two versions of standard C-implementation with area 360 and 320 units.

IX. Conclusion

In this paper, we have presented a theoretical framework for the hazard-free gate-level implementation of speed-independent circuits specified by STG's or their extension, called change diagrams. We first considered the most well known class of behaviors, defined by free-choice safe STG's, which only allow specifying circuits with AND causality between signal transitions. Toward the end, the most general class of speed-independent specifications, described with CD's, with both AND and OR causality, was tackled. We have provided sufficient conditions for hazard-free implementation by the standard structures combining two-level simple-gate combinational logic with latches (either a C-element or RS-latch). We described these conditions using transition diagrams, a state model that can be generated from the initial event-based specification defined in the form of an STG or CD.

We have developed a set of transformation techniques that can be applied at the event-based specification level (which would make them more efficient) to obtain the corresponding specification in the form satisfying the monotonous cover requirement. Here, in a step-by-step manner, we have shown the methods of eliminating nonpersistence and nonmonotonous covering, generalizing the MC conditions to allow optimization based on sharing logic gates and handling excitation regions with multiple entry states. An important feature of our approach is that all of the allowed transformations are semantic preserving, i.e., adding new signals does not change the original order between signals as defined by the initial STG or CD. In other words, we allow only very limited alterations in the model, e.g., forbiding concurrency reduction or signal reshuffling.

We have also presented a number of techniques to improve efficiency and hence practicality of the monotonous cover approach under certain relaxation of the original implementation structures, such as using the inverse feedbacks and complex (AND-OR) gates with appropriate modification to the MC condition.

We pursue an approach that is in some sense opposite to the one in [2], which achieves hazard freedom by applying some computationally hard, logic-level transformation procedures, yet without a firm guarantee of finding a correct solution.

Our method does not fail because it is applied at the level of the behavioral model and can be guided by the chosen implementation structure. That is, it can always guarantee a hazard-free solution for any speed-independent specification if the designer has a chance to compromise on a number of factors: the gate/wire delay model (inertial or distributed inertial), the model of a complemented literal (a separate inverter or inhibitor input to the gate), possibility to use AND-OR-NOT gates, RS-latches, C-elements, and, last, area/speed overheads concerned with the addition of auxiliary state signals. Unlike [17] and [18], the approach described in this paper reduces specifications to the MC form by direct and constructive transformations instead of reducing the problem to the Boolean satisfiability task, where the possibilities to control the quality of the logic area and performance are relatively poor.

The experimental results have shown that our present approach to hazard-free implementation of speed-independent circuits appears to improve over the previous work in both qualitative and quantitative domains.

The state-of-the-art implementation of the proposed method is in the tool `petrify` [9]. This tool also implements new methods for logic decomposition and technology mapping of speed-independent circuits in a library of gates with restricted fan-in [10]; these methods are built upon the MC theory.[10]

As the main direction for future work, we consider a two-level minimization guided by the monotonous cover constraints and an optimized transformation of STG and CD specifications to the monotonous cover form. It would also be interesting to relax the equivalence criterion for possible signal reshuffling and concurrency reduction.

Acknowledgment

The authors wish to thank J. Cortadella for the efficient implementation of the MC theory in the tool `petrify`. They are grateful to P. Vanbekbergen and B. Lin for numerous discussions that have led to the creation of MC conditions and for their common work on the implementation of these conditions in the satisfiability framework [17], [18]. They thank L. Lavagno for many useful comments and assistance with the SIS tool, and P. Beerel for providing them with detailed information on his benchmark. They are also indebted to the anonymous reviewers who provided many useful comments.

References

[1] P. Beerel and T. Meng, "Semi-modularity and testability of speed-independent circuits," *Integration, VLSI J.*, vol. 13, no. 3, 1992.

[2] ——, "Automatic gate-level synthesis of speed-independent circuits," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1992, pp. 581–587.

[3] ——, "Logic transformations and observability don't cares in speed-independent circuits," in *Proc. ACM Int. Workshop Timing Issues in the Specification and Synthesis of Digital Systems (TAU'93)*, Malente, Germany, Sept. 1993.

[4] T.-A. Chu, "Synthesis of self-timed VLSI circuits from graph-theoretic specifications," in *Proceedings of ICCD-87*. New York: IEEE Computer Society Press, 1987, pp. 220–223.

[5] ——, *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*, Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, June 1987.

[10]Tool `petrify` can be downloaded from http://www.ac.upc.es /~vlsi/petrify/petrify_files.html.

[6] ——, "Automatic synthesis and verification of hazard-free control circuits from asynchronous finite state machine specifications," in *Proc. ICCD'92*, Cambridge, MA, Oct. 1992, pp. 407–413.

[7] ——, "Synthesis of hazard-free control circuits from asynchronous finite state machine specifications," in *Proc. ACM Int. Workshop Timing Issues in the Specification and Synthesis of Digital Systems*, Princeton, NJ, Mar. 1992.

[8] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Complete state encoding based on the theory of regions," in *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, Mar. 1996, pp. 36–47.

[9] ——, "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.

[10] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, E. Pastor, and A. Yakovlev, "Decomposition and technology mapping of speed-independent circuits using Boolean relations," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1997, pp. 220–227.

[11] J. Cortadella, L. Lavagno, P. Vanbekbergen, and A. Yakovlev, "Designing asynchronous circuits from behavioral specifications with internal conflicts," in *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, Salt Lake City, UT, Nov. 1994, pp. 106–115.

[12] D. L. Dill, *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. Cambridge, MA: MIT Press, 1988.

[13] M. Hack, "Analysis of production schemata by petri nets," Massachusetts Institute of Technology, Cambridge, Technical Report, Tech. Rep., Project MAC, Feb. 1972.

[14] D. A. Huffman, "The synthesis of sequential switching circuits," *J. Frank. Inst.*, vol. 257, pp. 161–190, 275–303, Mar. 1954.

[15] M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky. *Concurrent Hardware: The Theory and Practice of Self-Timed Design*. London: Wiley, 1993.

[16] A. Kondratyev, J. Cortadella, M. Kishinevsky, E. Pastor, O. Roig, and A. Yakovlev, "Checking signal transition graph implementability by symbolic BDD traversal," in *Proceedings of the European Design and Test Conference*. New York: IEEE Computer Society Press, 1995, pp. 325–332.

[17] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev, "On the conditions for the gate-level speed-independence of asynchronous circuits," in *Proc. ACM Int. Workshop Timing Issues in the Specification and Synthesis of Digital Systems (TAU'93)*, Malente, Germany, Sept. 1993.

[18] ——, "Basic gate implementation of speed-independent circuits," in *Proc. Design Automation Conf.*, June 1994, pp. 56–62.

[19] A. Kondratyev, M. Kishinevsky, and A. Yakovlev, "Monotonous cover transformations for speed-independent implementation of asynchronous circuits," Aizu University, Japan, Tech. Rep., Mar. 1994.

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

[21] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli, "Algorithms for synthesis of hazard-free asynchronous circuits," in *Proc. Design Automation Conf.*, 1991, pp. 302–308.

[22] L. Lavagno, C. Moon, and R. Brayton, and A. Sangiovanni-Vincentelli, "Solving the state assignment problem for signal transition graphs," in *Proc. Design Automation Conf.*, 1992, pp. 568–572.

[23] A. Martin, "Synthesis of asynchronous VLSI circuits," in *Formal Methods for VLSI Design*, J. Staunstrup, Ed. Amsterdam, The Netherlands: North-Holland, 1990.

[24] A. J. Martin, S. M. Burns, T. K. Lee, D. Borkovic, and P. J. Hazewindus, "The first asynchronous microprocessor: The test results," *Comput. Architecture News*, vol. 17, no. 4, pp. 95–110, June 1989.

[25] T. H.-Y. Meng, *Synchronization Design for Digital Systems*. Norwell, MA: Kluwer, 1991.

[26] R. E. Miller, *Sequential Circuits and Machines*, vol. 2 of *Switching Theory*. New York: Wiley, 1965.

[27] C. W. Moon, P. R. Stephan, and R. K. Brayton. "Synthesis of hazard-free asynchronous circuits from graphical specifications," in *Proceedings of ICCAD-91*. New York: IEEE Computer Society Press, 1991, pp. 322–325.

[28] D. Muller and W. Bartky, "A theory of asynchronous circuits," in *Annals of Computation Laboratory*. Cambridge, MA: Harvard Univ. Press, 1959, pp. 204–243.

[29] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, pp. 541–580, Apr. 1989.

[30] S. M. Nowick and D. L. Dill, "Exact two-level minimization of hazard-free logic with multiple-input changes," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 986–997, Aug. 1995.

[31] S. M. Nowick and B. Coates, "Automated design of high-performance unclocked state machines," in *Proc. ACM Int. Workshop Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, Sept. 1993.

[32] E. Pastor and J. Cortadella, "Polynomial algorithms for complete state coding and synthesis of hazard-free circuits from signal transition graphs," Universitat Politecnica de Catalunya, Spain, Tech. Rep. RR 93/17 UPC/DAC, Sept. 1993.

[33] E. Pastor, J. Cortadella, O. Roig, and A. Kondratyev, "Structural methods for the synthesis of speed-independent circuits," in *Proceedings of the European Design and Test Conference*. New York: IEEE Computer Society Press, 1996, pp. 340–347.

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

[35] K. van Berkel, R. Burgess, J. Kessels, A. Peeters, M. Roncken, and F. Schalij, "A fully-asynchronous low-power error corrector for the DCC player," in *ISSCC 1994 Dig. Tech. Papers*, San Francisco, CA, 1994, vol. 37, pp. 88–89.

[36] P. Vanbekbergen, F. Cathoor, G. Goossens, and H. De Man, "Time and area performant synthesis of asynchronous control circuits," in *Proc. ACM Int. Workshop Timing Issues in the Specification and Synthesis of Digital Systems*, Vancouver, Canada, Aug. 1990.

[37] P. Vanbekbergen, B. Lin, G. Goossens, and H. De Man, "A generalized state assignment theory for transformations on signal transition graphs," in *Proc. ICCD'92*, Cambridge, MA, Oct. 1992.

[38] V. Varshavsky, Ed., *Self-Timed Control of Concurrent Processes*. Dordrecht, The Netherlands: Kluwer, 1990.

[39] V. Varshavsky, M. Kishinevsky, A. Kondratyev, L. Rosenblyum, and A. Taubin, "Models for specification and analysis of processes in asynchronous circuits," *Sov. J. Comput. Syst. Sci.*, vol. 26, no. 2, pp. 61–76, 1989.

[40] A. Yakovlev, M. Kishinevsky, A. Kondratyev, and L. Lavagno, "OR causality: Modeling and hardware implementation.," in *Proc. 15th Int. Conf. Application and Theory of Petri Nets*, Zaragosa, Spain, June 1994, pp. 568–587.

[41] A. Yakovlev, L. Lavagno, and A. Sangiovanni-Vincentelli, "A unified signal transition graph model for asynchronous control circuit synthesis," in *Proc. ICCAD'92*, Santa Clara, CA, Nov. 1992, pp. 104–111.

[42] M. Yu and P. Subrahmanyam, "Hazard-free asynchronous circuit synthesis," in *Proc. Working Conf. Asynchronous Design Methodologies*, Manchester, England, Mar. 1993.

[43] 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.

Alex Kondratyev (M'94–SM'97) received the M.S 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. Petersburg. He is a coauthor of *Concurrent Hardware: The Theory and Practice of Self-Timed Design* (Wiley, 1994). He was a Cochair of the Async'96 Symposium, Cochair of the CSD'98 Conference, and has been a member of the program committee for several conferences. His research interests include several aspects of computer-aided design, with particular emphasis on asynchronous design and theory of concurrency.

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

He was a Researcher with the St. Petersburg Mathematical Economics Institute Computer Department, Russian Academy of Science, during 1979–1982 and 1987–1989. From 1982 to 1987, he was with a software company. From 1988 to 1992, he was a Senior Researcher with the R&D Co-op TRASSA. In 1992, he joined the Department of Computer Science, Technical University of Denmark, 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 Corp., Hillsboro, OR. His current research interests include design of asynchronous and reactive systems and theory of concurrency. He has coauthored two books on asynchronous design and has published more than 50 journal and conference papers.

Alex Yakovlev (S'94–M'97) received the M.Sc. and Ph.D. degrees in computing science from the Electrotechnical University of St. Petersburg, Russia.

From 1980 to 1990, he was with the Electrotechnical University of St. Petersburg, where he worked in the area of asynchronous and concurrent systems. During 1982–1990, he was an Assistant and then Associate Professor in the Computing Science Department. He first visited Newcastle, England, in 1984–1985 for research in very large scale integration (VLSI) and design automation. He returned to Britain in 1990 and was with Politechnic of Wales (now University of Glamorgan) for one year. Since 1991 he has been a Lecturer, and quite recently a Reader, in computing systems design at the Newcastle University Department of Computing Science, where he is heading the VLSI Design research group. His current interests and publications are in the field of modeling and design of asynchronous, concurrent, real-time, and dependable systems.