Automatic Synthesis of Extended Burst-Mode Circuits: Part I (Specification and Hazard-Free Implementations)

Kenneth Y. Yun, *Member, IEEE,* and David L. Dill, *Member, IEEE*

***Abstract*— We introduce a new design style called *extended burst-mode*. The extended burst-mode design style covers a wide spectrum of sequential circuits ranging from delay-insensitive to synchronous. We can synthesize multiple-input change asynchronous finite state machines and many circuits that fall in the gray area (hard to classify as synchronous or asynchronous) which are difficult or impossible to synthesize automatically using existing methods. Our implementation of extended burst-mode machines uses standard CMOS logic, generates low-latency outputs, and guarantees freedom from hazards at the gate level. In Part I, we formally define the extended burst-mode specification, provide an overview of the synthesis methods, and describe the hazard-free synthesis requirements for two different next-state logic synthesis methods: two-level sums-of-products implementation and generalized C-elements implementation. We also present an extension to existing theories for hazard-free combinational synthesis to handle *nonmonotonic* input changes.**

***Index Terms*—Asynchronous controller, extended burst-mode, generalized C-element, hazard-free synthesis.**

I. Introduction

TODAY'S system components typically employ the synchronous paradigm primarily because of the availability of the rich set of design tools and algorithms and, perhaps, because of the designers' perception of "ease of design" and the lack of alternatives. Even so, the interfaces among the system components do not strictly adhere to the synchronous paradigm because of the cost benefit of mixing modules operating at different clock rates and modules with asynchronous interfaces. There is little doubt that today's *heterogeneous* system concept at the board level will carry over to tomorrow's systems-on-a-chip, because miniaturization does not make global synchronization any simpler. This paper addresses the problem of how to synthesize asynchronous controllers operating in a heterogeneous environment, i.e., in a system composed of components using different synchronization mechanisms.

In this paper, we are mainly concerned with designing correct and efficient asynchronous controllers. There are three factors that affect the quality of the final design and the scope of its applications: *specification method*, *target implementation*, and *synthesis method*. The *design style* is a general term which refers to a combination of all of these. Our goal is to develop a design style that attains a good combination of expressiveness and implementability, which are conflicting goals. We will briefly survey existing design styles and zero in on a design style called *extended burst-mode*, which has a reasonable combination of expressiveness and implementability.

Translation Methods

Several design styles use high-level languages of concurrency as their user-level specification formalisms. Martin's method [1] is based on Hoare's *CSP*; Brunvand [2] uses *occam*; Ebergen's method [3] is derived from *trace theory*. The synthesis procedures typically involve syntax-driven translation [4] or algebraic transformation [5] of the specifications into *delay-insensitive* (DI) or *speed-independent* (SI) circuits.

An advantage of the compilation methods over other methods is that complex concurrent systems can be described elegantly and concisely in high-level constructs without low-level timing concerns, which makes it easier to modify and verify the system behavior. However, because it is difficult to utilize global optimization techniques during the translation process, the automated synthesis often produces inefficient results. In general, the circuits generated using the compilation methods tend to incur considerably more area than those synthesized by other methods. Recently, some efforts have been made to address the optimization problems. Gopalakrishnan *et al.* [6] use *peephole optimization*: i.e., translate a group of DI modules into *burst-mode* specifications and resynthesize using the *3D* tool described in this paper.

There has been a concerted effort to synthesize commercial-scale circuits to demonstrate the practicality of these methods, such as the DCC error corrector chip developed at Philips Research Laboratories by van Berkel *et al.* [7] using a synthesis tool called *Tangram* compiler.

Graph-Based Methods

Almost all of the graph-based methods use the Petri net or a restricted form of the Petri net as the specification formalism. A *Petri net* is a graph model used for describing concurrent systems. Chu [8] introduced a restricted form of the Petri net called *Signal Transitions Graph* (STG) to specify asynchronous circuits. Chu's initial definition of STG

Manuscript received October 1, 1997; revised October 9, 1998. This work was supported in part by a gift from Intel Corporation and by a National Science Foundation CAREER Award MIP-9625034. This paper was recommended by Associate Editor M. C. Papaefthymiou.

K. Y. Yun is with the Department of Electrical and Computer Engineering, University of California, San Diego, La Jolla, CA 92093-0407 USA (e-mail: kyy@ucsd.edu).

D. L. Dill is with the Computer Science Department, Stanford University, Stanford, CA 94305 USA (e-mail: dill@cs.stanford.edu).

Publisher Item Identifier S 0278-0070(99)01008-8.

(interpreted free-choice Petri net), however, allowed only a limited mechanism to select alternative responses of the circuit. Meng [9] extended Chu's work and developed an automatic synthesis tool. More recent works in STG-based synthesis include methods of Lavagno [10], Vanbekbergen [11], and Ykman-Couvreur *et al.* [12].

In general, the strong suit of the STG is its ability to express concurrency. Its main weakness is the awkwardness in specifying input choices. That is, the mechanisms to guide the responses of the machine are limited. In a free-choice STG specification, the machine selects the course of its future behavior solely based on input transitions. The machine cannot handle choices based on input *levels*. Vanbekbergen [11] introduced the *generalized* STG which allows input choices based on signal levels. However, his synthesis method cannot guarantee the generatation of hazard-free circuits. Cortadella *et al.* [13] extended the STG to handle internal conflicts, i.e., arbitration. Some graph-based methods, such as Varshavsky's [14], Beerel's [15], and Kondratyev *et al.*'s [16], use *state graphs* to avoid syntactic problems associated with STG's.

Although most graph-based synthesis methods generate SI circuits, some methods use *bounded wire delay model*, a circuit model in which the delay of each gate and wire has a lower and an upper bound. Lavagno's method inserts fixed delay elements to avoid internal hazards, though it makes no assumptions about the circuit's environment. Myers's method [17] uses an STG-like specification formalism called *Event-Rule Systems* [4]. His tool, ATACS, synthesizes very compact area-efficient *generalized C-element* circuits by exploiting all known delays, both internal and external.

Asynchronous Finite State Machines

Asynchronous finite state machines (AFSM's) have been around for the past 30 years. The work on AFSM's was pioneered by Huffman and others. Early AFSM's [18], [19] assumed that the environment operates in fundamental mode, that is, the environment generates a single input change and waits for the machine to stabilize before it generates the next input change. Recent work in AFSM's allows the multiple-input change fundamental mode operations. We focus on a recently introduced multiple-input change machine called *burst-mode* machine. Burst-mode asynchronous finite state machines were first introduced by Davis *et al.* [23] and formalized by Nowick and Dill [21], [22]. Burst-mode machines have been implemented using a method developed at HP Laboratories called *MEAT* [23], the *locally clocked* method [22], the *3D* method [22], and the *UCLOCK* method [25].

A *burst-mode* specification is a variation of a Mealy machine that allows multiple-input changes in a burst fashion—in a given state, when all of a specified set of input edges appear, the machine generates a set of output changes and moves to a new state. The specified input edges can appear in arbitrary order, thus allowing input concurrency, and the outputs are generated concurrently. The advantages of a burst-mode specification over STG specifications are that it is similar to the synchronous Mealy machine with which designers are familiar, that the input choice is more flexible than that of the STG, and that the state encoding is more flexible in the implementations. Its main practical disadvantage is that it does not allow input changes to be concurrent with output changes. The input choice mechanism is more flexible than the STG but still primitive. For example, it cannot handle choices between two sets of concurrent events if one set is a subset of the other.

The extended burst-mode design style described in this paper is a superset of burst-mode with two new features[1]: *directed don't cares* and *conditionals*. Directed don't cares allow an input signal to change concurrently with output signals, and conditionals allow control flow to depend on the input signal levels, in the same way synchronous state machines regulate control flow. Thus this design style not only supports burst-mode multiple-input change asynchronous designs with added input/output concurrency, it also allows the automatic synthesis of *any synchronous Moore machine*, in which the synchronous inputs are represented as conditional signals, and the clock is the only nonconditional signal. Moreover, this design style covers a wide range of circuits between burst-mode and fully synchronous.

We summarize the main contributions of the paper below.

Extended Burst-Mode Design Style: The extended burst-mode design style covers a wide spectrum of sequential circuits, which ranges from delay-insensitive to synchronous. It is significant *theoretically* because it is the first asynchronous design style that subsumes fully synchronous designs. It is significant *practically* because a wide range of practical circuits can be specified in a common specification language and synthesized using a single synthesis tool. For example, it can synthesize multiple-input change asynchronous finite state machines, including all burst-mode machines, and many circuits that fall in the gray area (hard to classify as synchronous or asynchronous) which are difficult or impossible to synthesize automatically. These include circuits that require clocking with multiple clocks, circuits that require clocking on both edges of a clock signal, and circuits that require selective clocking.

Hazard-Free Next-State Logic Synthesis Requirements: This paper presents two different hazard-free next-state logic synthesis methods: hazard-free two-level sums-of-products implementation and hazard-free generalized C-elements implementation. Existing theories for hazard-free combinational synthesis are extended to handle *nonmonotonic* input changes.

3D Automatic Synthesis Algorithm: This paper describes a complete set of automated sequential synthesis algorithms: hazard-free state assignment, hazard-free state minimization, and critical-race-free state encoding. Experimental data from a large set of examples are presented and compared to competing methods whenever possible.

Finally, the 3D synthesis tool described in this paper was used to design a significant portion of control circuitry for Intel's Asynchronous Instruction Length Decoder chip, a high-performance differential equation solver chip [27], a high-performance small computer systems interface (SCSI) con-

[1] Although the *directed don't care* was not a part of the formal burst-mode specification introduced in [21], a semantically equivalent feature called "long arc" was used in Postoffice controllers [26] and a part of the MEAT synthesis tool [23] developed at HP Laboratories.

troller [28], and a low-power infrared communication chip [29]. All the 3D controllers in the fabricated chips worked correctly in first silicon.

This paper is divided into two parts. Part I describes the specification formalism and the hazard-free implementations, and Part II presents automatic synthesis and experimental results. Section II of Part I describes a user-level specification formalism, called *extended burst-mode*. Section III describes a target implementation style, called *3D*, and an overview of how an asynchronous controller specified in extended burst-mode is transformed into a correct implementation. Section IV precisely characterizes every possible hazard that can arise in the 3D implementation of extended burst-mode machines. For every type of hazard, a necessary and sufficient condition or at least a sufficient condition for freedom from hazards is stated and proved.[2] This section presents the notion of generalized transition which is used for functional synthesis and for analysis of function hazards. This section also presents two different next-state logic synthesis methods: two-level sums-of-products implementation and generalized C-element implementation.

Automatic synthesis procedure and algorithms are presented in Part II. Part II describes the hazard-free state assignment algorithm and proves the existence of a hazard-free implementation for every legal extended burst-mode specification. It presents a state minimization heuristic and describes a critical-race-free state encoding algorithm. Finally, it reports the experimental results, after describing how the 3D synthesis uses Nowick and Dill's hazard-free combinational logic synthesis.

II. Specification

This section describes a user-level specification formalism, *extended burst-mode*, using an example, followed by a formal definition. The extended burst-mode is a powerful user interface for specifying a large class of controllers. It is intended for designing asynchronous state machines and the machines that fall in the gray area between asynchronous and synchronous, although it is theoretically possible to specify any synchronous Moore machine [30] and practically feasible to design small- to medium-size synchronous Moore machines.

We start by introducing the burst-mode specification. The first usage of "burst-mode" state machines can be traced back to a packet routing chip called *Postoffice* designed by Davis, Stevens, and Coates at HP Laboratories [26]. State machines in Postoffice were synthesized by an automatic synthesis tool called *MEAT* [23], [31]. However, the implementations were not guaranteed hazard-free: a verifier was used to detect any hazards, and if they occurred, resynthesis was performed. Nowick and Dill at Stanford later restricted and formalized the specification format used at HP Laboratories (and coined the term "burst-mode") and developed a set of synthesis tools and algorithms to guarantee a hazard-free implementation [21], [22].

[2]Note that not all hazard conditions are eliminated by the machine implementation. In our method, some sequential hazards are removed by constraining the response of the machine's environment, i.e., by imposing fundamental mode constraints.

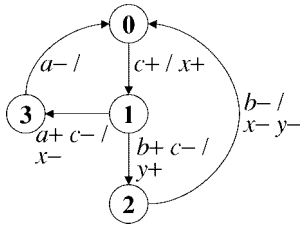

Fig. 1. A burst-mode specification.

A. Controller Specification

1) Burst-Mode Specification: The *burst-mode* is a specification formalism [21], [22] for asynchronous finite state machines allowing multiple-input changes. A burst-mode state machine is specified by a state diagram that contains a finite set of states, a number of labeled arcs connecting pairs of states, and a start state. Arcs are labeled with possible transitions from one state to another. Each state transition consists of a *nonempty* set of input edges (an *input burst*) and a set of output edges (an *output burst*). Every input burst must be nonempty; if no inputs change, the machine remains in the same state. Fig. 1 shows a burst-mode specification with three inputs, (a, b, c), two outputs, (x, y), and state 0 as the start state. $c+$ and $c-$ signify rising and falling edges of c, respectively. A slash $(/)$ is used to delimit each input burst. Thus $c+/x+$ means that output burst $x+$ is enabled by input burst $c+$.

In a given state, when all the inputs in some input burst have changed value, the machine generates the corresponding output burst and moves to a new state. Only the specified input changes may occur, but the order of arrivals may be arbitrary. In addition, there are two restrictions to the burst-mode specification introduced by Nowick and Dill, i.e., every legal burst-mode specification must adhere to the following properties.

a) *Maximal set property:* to prevent nondeterministic state transitions. No input burst in a given state can be a subset of another in the same state. If, for example, the specified input burst for transition from state 1 to 3 is $c-$ instead of $a+c-$ and the machine's environment lowers c first in state 1, then the machine would need to make an arbitrary decision on whether to proceed to state 3 or to wait for $b+$. The specification in Fig. 1 satisfies the maximal set property because $\{a+, c-\} \not\subseteq \{b+, c-\}$ and $\{b+, c-\} \not\subseteq \{a+, c-\}$.

b) *Unique entry condition:* to simplify hazard-free synthesis. Every state must have a unique entry point. For example, the entry point to state 0 is $abcxy = 00000$ whether the transition is from state 2 or from state 3.

2) Extended Burst-Mode Specification: Fig. 2 describes an extended burst-mode specification (*biu-fifo2dma*) with four inputs (ok, cntgt1, fain, dackn) and two outputs (frout, dreq). Signals not enclosed in angle brackets and ending with $+$ or $-$ are *terminating edge signals*. The signals enclosed in angle brackets are *conditionals*, which are level signals whose values are sampled when all of the terminating edges associated

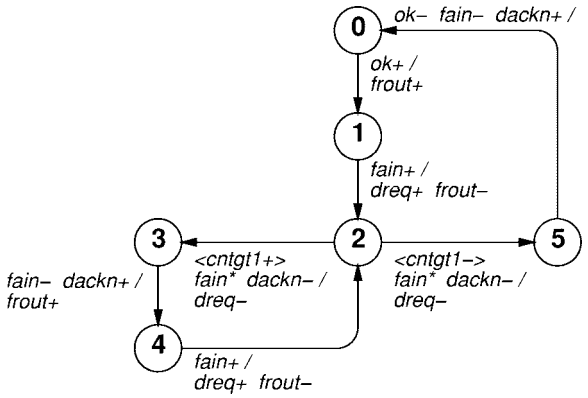

Fig. 2. Extended burst-mode specification (*biu-fifo2dma*).

with them have occurred. A conditional $\langle a+\rangle$ can be read "if a is high" and $\langle a-\rangle$ can be read "if a is low." A state transition occurs only if all of the conditions are met and all the terminating edges have appeared. A signal ending with an asterisk is a *directed don't care*. If a is a directed don't care, there must be a sequence of state transitions in the machine labeled with $a*$. If a state transition is labeled with $a*$, the following state transitions in the machine must be labeled with $a*$ or with $a+$ or $a-$ (the terminating edge for the directed don't care). Consider the state transitions out of state 2. The behavior of the machine at this point is: "if cntgt1 is low when dackn falls, then lower the output dreq and change the current state from 2 to 5; if cntgt1 is high when dackn falls, then lower dreq and change the state from 2 to 3."

A directed don't care may change at most once during a sequence of state transitions it labels, i.e., directed don't cares are *monotonic* signals, and, if it does not change during this sequence, it must change during the state transition its terminating edge labels. A terminating edge which is not immediately preceded by a directed don't care is called *compulsory*, since it *must* appear during the state transition it labels. In Fig. 2, fain is high when the machine enters state 2. It can fall at any point as the machine moves through state 3 or through state 5, depending on the level of cntgt1, but it must have fallen by the time the machine moves to states 4 or 0, because the terminating edge fain− appears between states 3 and 4 and between 5 and 0.

The input signals are globally partitioned into level signals (conditionals), which can never be used as edge signals, and edge signals (terminating or directed don't care), which can never be used as level signals. If a level signal is not mentioned on a particular state transition, it may change freely. If an edge signal is not mentioned, it is not allowed to change.

The following are some examples of labels on state transitions.

- $\langle c_1+\rangle\langle c_2-\rangle x+/z_1+z_2-$ means "if $c_1 = 1$ and $c_2 = 0$ when x rises, then the machine raises z_1 and lowers z_2."
- $x_1+x_2-/z+$ means "the machine raises z when x_1 rises and x_2 falls."

Summary

An extended burst-mode asynchronous finite state machine [32] is specified by a state diagram which consists of a finite number of states, a set of labeled state transitions connecting pairs of states, and a start state. Each state transition is labeled with a set of conditional signal levels and two sets of signal edges: an input burst and an output burst. An *output burst* is a set of output edges, and an *input burst* is a nonempty set of input edges (terminating or directed don't care), at least one of which must be *compulsory*.

In a given state, when all the specified conditional signals have stabilized and all the specified terminating edges in the input burst have appeared, the machine asserts the specified output changes and moves to a new state. Specified edges in the input burst may appear in arbitrary temporal order. Each signal specified as a directed don't care may change its value monotonically at any time, even while outputs are changing, unless it is already at the level specified by the next terminating edge. Output changes may be generated in any order.

The conditional signals must stabilize to correct levels before any compulsory edge in the input burst appears and must hold their values until after all of the terminating edges appear. The minimum delay from the conditional stabilizing to the first compulsory edge is called the *setup time*. Similarly, the minimum delay from the last terminating edge to the conditional change is called the *hold time*. Actual values of setup and hold times of conditional signals with respect to the first compulsory edge and the last terminating edge depend on the implementation. The period starting at the specified setup time before the first compulsory edge and ending at the specified hold time after the last terminating edge is called the *sampling period*. Conditional signal levels need not be stable outside of the specified sampling periods.

The next set of compulsory edges from the next input burst may not appear until the machine has stabilized. This requirement—the environment must wait until the circuit stabilizes before generating the next set of compulsory edges—is a variation of the *multiple-input change fundamental-mode environmental constraint*.

Restrictions

There is an additional restriction to extended burst-mode specifications (as to burst-mode), called *distinguishability constraint*, which prevents ambiguity among multiple input bursts emanating from a single state: For every pair of input bursts i and j from the same state, either the conditions are mutually exclusive or the set of compulsory edges in i is not a subset of the set of all possible input transitions in j. The second condition stipulates that the minimum set of input transitions in i must not be a subset of the maximum set of input transitions allowed in j (by the extended burst-mode semantics), which includes transitions on both directed don't cares and terminating signals.

For instance, the input bursts from state 0 in Fig. 3(a) are legal because $\langle c+\rangle$ and $\langle c-\rangle$ are mutually exclusive. However, the input bursts from state 0 in Fig. 3(c) are illegal because the conditions are *not mutually exclusive* and $\{b+\} \subset \{a+, b+\}$. Moreover, the input bursts from state 0 in Fig. 3(b) violate the distinguishability constraint because the set of all possible edges for the input burst $a+b*$ is $\{a+, b+\}$ and $\{b+\} \subset \{a+, b+\}$.

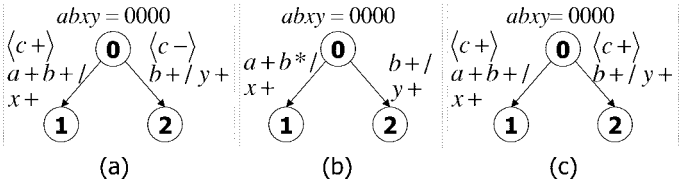

Fig. 3. Distinguishability constraints.

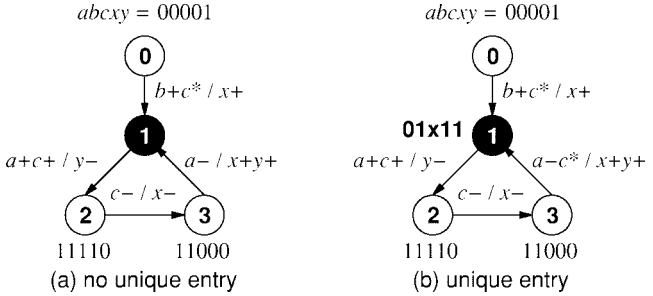

Fig. 4. Unique entry condition. (a) No unique entry. (b) Unique entry.

To simplify hazard-free synthesis, we assume that the *unique entry condition* is satisfied, again as in burst-mode. The set of possible *entry points* into a state (input and output values entering a state) from every predecessor state must be identical. This is a simplifying assumption that does not constrain the range of permissible behaviors since an extended burst-mode specification can always be transformed into an equivalent specification satisfying the unique entry condition by duplicating some states.

For example, the set of valid entry points to state 1 from state 0 in Fig. 4(a) is {01011, 01111}, but from state 3 to state 1 it is {01011}. Thus the unique entry condition is not met in this specification. A specification satisfying the unique entry condition is shown in Fig. 4(b).

B. Formal Definition of Extended Burst-Mode

The following formal definition of the extended burst-mode specification is adapted from the definition of the burst-mode specification in [22]. An extended burst-mode specification is a directed graph, $G = (V, E, C, I, O, v_0, \mathsf{cond}, \mathsf{in}, \mathsf{out})$, where V is a finite set of states; $E \subseteq V \times V$ is the set of state transitions; $C = \{c_1, \cdots, c_l\}$ is the set of conditional inputs; $I = \{x_1, \cdots, x_m\}$ is the set of edge inputs; $O = \{z_1, \cdots, z_n\}$ is the set of outputs; $v_0 \in V$ is the unique start state; cond labels each state transition with a set of conditional inputs; in and out are labeling functions used to define the unique entry cube of each state. The function cond: $E \rightarrow \{0, 1, *\}^l$ defines the values of the conditional inputs. The function in: $V \rightarrow \{0, 1, *\}^m$ defines the values of the edge inputs, and the function out: $V \rightarrow \{0, 1\}^n$ defines the values of the outputs upon entry to each state. Note that $C \cap I = \emptyset$, i.e., conditional inputs remain level signals throughout G and edge inputs remain edge signals throughout G as well.

Labeling functions $\mathsf{trans}_{\mathrm{IN}}$ and $\mathsf{trans}_{\mathrm{OUT}}$ are derived from graph G. $\mathsf{trans}_{\mathrm{IN}}$: $E \rightarrow \mathcal{P}(I)$ defines the set of edge input changes and $\mathsf{trans}_{\mathrm{OUT}}$: $E \rightarrow \mathcal{P}(O)$ defines the set of output changes. ($\mathcal{P}(I)$ and $\mathcal{P}(O)$ denote the power set of inputs and the power set of outputs, respectively.) Given a state transition, $(u, v) \in E$, $x_i \in \mathsf{trans}_{\mathrm{IN}}(u, v)$ iff $\mathsf{in}_i(u) \neq \mathsf{in}_i(v) \vee \mathsf{in}_i(v) = *$. That is, x_i+ is in the input burst iff $\mathsf{in}_i(v) = 1 \wedge \mathsf{in}_i(u) \neq 1$, x_i- is in the input burst iff $\mathsf{in}_i(v) = 0 \wedge \mathsf{in}_i(u) \neq 0$, and x_i* is in the input burst iff $\mathsf{in}_i(v) = *$. Similarly, $z_j \in \mathsf{trans}_{\mathrm{OUT}}(u, v)$ iff $\mathsf{out}_j(u) \neq \mathsf{out}_j(v)$. That is, z_j+ is in the output burst iff $\mathsf{out}_j(v) = 1 \wedge \mathsf{out}_j(u) = 0$, z_j- is in the output burst iff $\mathsf{out}_j(v) = 0 \wedge \mathsf{out}_j(u) = 1$. Finally, $\mathsf{ctrans}_{\mathrm{IN}}$ defines the set of compulsory edge input changes: $\mathsf{ctrans}_{\mathrm{IN}}(u, v) = \{x_i \in \mathsf{trans}_{\mathrm{IN}}(u, v) \mid \mathsf{in}_i(u) \neq * \wedge \mathsf{in}_i(v) \neq *\}$.

The unique entry condition is satisfied by the above definitions. The remaining requirements to ensure well-formed specifications are as follows.

- Every input burst must contain a compulsory edge. That is, for every state transition (u, v), there exists $x_i \in \mathsf{trans}_{\mathrm{IN}}(u, v)$ such that $\mathsf{in}_i(u) \neq * \wedge \mathsf{in}_i(v) \neq *$.
- Every pair of state transitions emanating from the same state must satisfy the distinguishability constraint. That is, for every pair, $(u, v), (u, w) \in E$, $\mathsf{ctrans}_{\mathrm{IN}}(u, v) \subseteq \mathsf{trans}_{\mathrm{IN}}(u, w)$ implies that either $v = w$ or $\mathsf{cond}(u, v)$ and $\mathsf{cond}(u, w)$ are mutually exclusive, that is, there exists k such that $\mathsf{cond}_k(u, v) \neq \mathsf{cond}_k(u, w) \wedge \mathsf{cond}_k(u, v) \neq * \wedge \mathsf{cond}_k(u, w) \neq *$.
- For every sequence of state transitions, $u \rightarrow v_1 \rightarrow \cdots \rightarrow v_n \rightarrow w$, with $n \geq 1$ and $\mathsf{in}_i(u) = \mathsf{in}_i(w) \neq *$, there exists $k \in 1, \cdots, n$ such that $\mathsf{in}_i(v_k) \neq *$. That is, a sequence of directed don't cares must be terminated with an edge that enables the signal to toggle.

III. Implementation Overview

In all sequential machines, the machine output depends not only on the inputs but also on the state of the machine, which keeps track of the history of input changes. All sequential machines use feedback to store the state of the machine. In Huffman-mode state machines [18], [19], the state of the machine is stored only in internal state variables—primary outputs do not store any state information. In our 3D machines, however, primary outputs are used to store the state of the machine whenever possible in order to minimize the number of internal state variables.

A 3D asynchronous finite state machine is formally defined as a 4-tuple (X, Y, Z, δ) where

- X is a nonempty set of primary input symbols;
- Y is a nonempty set of primary output symbols;
- Z is a (possibly empty) set of internal state variable symbols;
- $\delta : X \times Y \times Z \rightarrow Y \times Z$ is a *next-state function*.

The hardware implementation of a 3D state machine (see Fig. 5) is a hazard-free network which implements the next-state function, with the outputs of the network fed back as inputs to the network. A 3D implementation of an extended burst-mode specification is obtained from the *next-state table*, a three-dimensional tabular representation of δ. The next state of every *reachable* state must be specified in the next-state table; the remaining entries are don't cares.

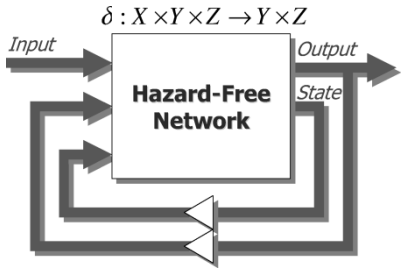

Fig. 5. 3D asynchronous state machine.

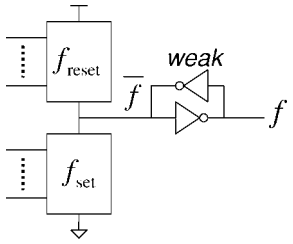

Fig. 6. A generalized C-element with a sustainer: f_{set} and f_{reset} are mutually exclusive but $f_{\text{set}} \neq \overline{f_{\text{reset}}}$.

Hazard-Free Network Implementations

Each output of the next-state function can be implemented in on-set logic or in set/reset logic. The on-set logic can be two-level sums of products (SOP's), two-level products of sums (POS's), or multilevel logic. The set and reset functions of the set/reset logic can be SOP, POS, or multilevel logic as well. Both on-set and set/reset logic can be implemented using basic gates or complex gates. In this paper, we describe two ways of implementing the next-state function: two-level SOP and generalized C-element (an efficient form of set/reset logic implemented as a pseudostatic asymmetric complex gate, as depicted in Fig. 6) [33], [4], [17], [34]. The underlying theory for hazard-free implementations is first developed for the two-level SOP and then applied to the generalized C-element.

A. Example I

A simple example is used to illustrate the synthesis and operation of a 3D machine (see Fig. 7). We describe the desired machine behavior according to an extended burst-mode specification and the next-state table entries needed to make the machine behavior conform to the specification. From a completed next-state table, we can extract the logic equations directly, because next-state tables describe the next values of outputs and state variables for every combination of inputs, outputs, and state variables.

In S_0 (the initial state), the machine waits for an input transition $s+$. Once s rises, the machine raises output r. Because d is an unspecified level signal, its value is undefined in S_0; hence, the output change $r+$ does not depend on it. Thus the next values of re for *sadRE* $=$ 00x00 and for *sadRE* $=$ 10xx0 are specified to be 00 and 10, respectively. When r stabilizes to 1, the machine is in S_1.

In S_1, the machine waits for $a+$. If $d = 0$ when a rises, then the machine lowers r and transitions to S_2. Thus the next re is specified to be 00 for *sadRE* $=$ 110x0. Once the machine stabilizes after r falls (when $d = 0$), i.e., after the hold time requirement for d is met, the machine is in S_2.

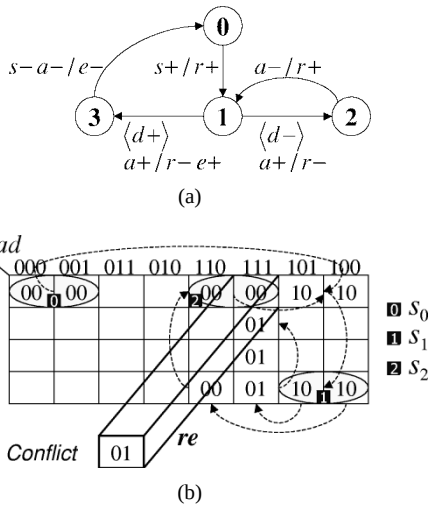

Fig. 7. Example I. (a) Specification and (b) conflict during state table construction.

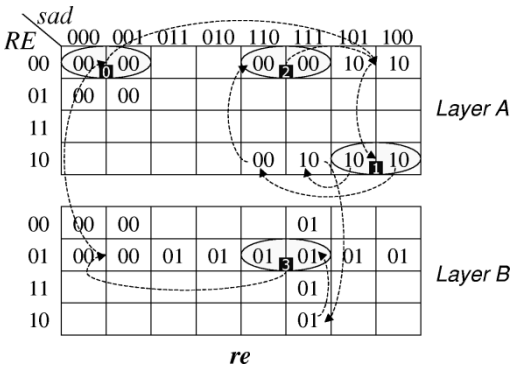

Fig. 8. Example I (next-state table before layer encoding).

After the machine enters S_2, the environment is allowed to change d. Thus the next re should be 00 before a falls, i.e., for both *sadRE* $=$ 11000 and 11100. When a falls, the machine raises r. Thus the next re for *sadRE* $=$ 10xx0 should be 10, which had been specified for $S_0 \rightarrow S_1$.

Now we are ready to specify the next values of re for $S_1 \rightarrow S_3$. Since the machine is to lower r and raise e and transitions to S_3, if $d = 1$ when a rises, the next re should be 01 for *sadRE* $=$ 111xx. This is to insure that the outputs change monotonically regardless of the order of fed-back variable changes $(R-E+)$. However, the next re for *sadRE* $=$ 11 100 had been specified to be 01 during S_2 to S_1.

Such conflicts can be avoided by adding state variables, which can be viewed as transitioning between *layers* of the next-state table (see Fig. 8). Conflicting entries can be placed in different layers. Our strategy, in this case, is to back up to the state following the input burst before the conflict (*sadRE* $=$ 11 110) and change the internal state variable before making output changes. Thus, after the input burst $\langle d+ \rangle a+$ in S_1, the machine transitions to layer B before changing r and e. Therefore, the next re for *sadRE* $=$ 11 110 is specified to be 10, i.e., no change, in layer A. In layer B, the next re for *sadRE* $=$ 111xx is 01.

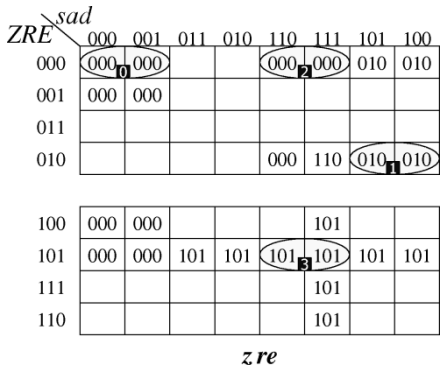

Fig. 9. Example I (next-state table after layer encoding).

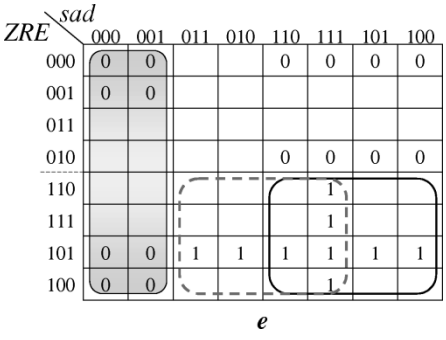

Fig. 10. Example I (Karnaugh map for e): $e = sZ + aZ$; $e_{\mathsf{set}} = sZ$; $e_{\mathsf{reset}} = \bar{s}\,\bar{a}$.

When the output burst $(r-e+)$ is complete, the machine is in S_3. Once in S_3, the machine awaits the input burst $(s-a-)$. During this input burst, sa changes from 11 to 00 via 10 or 01; the unspecified level signal d may change anytime. The next values of re for *sadRE* $=$ 11x01, 10x01, and 01x01 are specified to be 01, so that the outputs re remain unchanged until both s and a have fallen. After both s and a fall, the machine concurrently lowers e and transitions to layer A. Thus the next re for *sadRE* $=$ 00x0x is specified to be 00 on both layers A and B.

The resulting table (in Fig. 8) has two layers. Thus, just one state bit is needed to encode the layers. The code value of 0 is assigned to layer A and 1 to layer B. We can complete the construction of the next-state table by adding the resulting state bits to the next-state entries as shown in Fig. 9. At this point, all reachable entries of the next-state table are specified; next states of the remaining entries are *don't cares*.

We can then synthesize the logic directly from the next-state table. Each output of the next-state function can be implemented as a two-level AND-OR, which can be mapped to a hazard-free multilevel circuit [35], [36]. For example, the two-level AND-OR implementation $(e = sZ+aZ)$ in Fig. 11(a) is derived from a Karnaugh map of e shown in Fig. 10. Of course, care must be taken to avoid hazards in the logic when translating a Karnaugh map to logic.

Each output of the next-state function can also be implemented[3] as a generalized C-element. Fig. 11(b) shows a generalized C-element implementation of e, in which the

[3]Many alternative implementations for burst-mode circuits have been proposed elsewhere [37], [38].

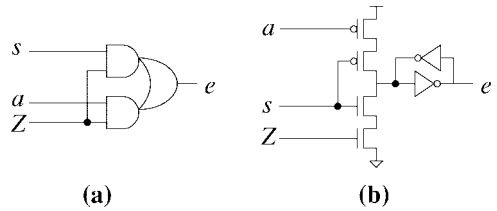

Fig. 11. (a) e in two-level AND-OR and (b) e in generalized C-element.

switch functions of N and P stacks are $e_{\mathsf{set}} = sZ$ and $e_{\mathsf{reset}} = \bar{s}\,\bar{a}$, respectively.

B. 3D Machine Operation

There are three types of machine cycles in a 3D state machine.

- *Type I)* an input burst followed by a concurrent output and state burst.
- *Type II)* an input burst followed by an output burst followed by a state burst.
- *Type III)* an input burst followed by a state burst followed by an output burst.

The selection of a machine cycle depends on the required level of concurrency and the next-state logic synthesis method used. Normally, Type I or II is selected. Type III is only used to avoid dynamic hazards that may be present in two-level AND-OR due to undirected don't cares (undefined conditionals) and should be used in lieu of Type I or II in those cases, which will be discussed in detail in Section IV of Part II. Both Type I and Type II offer shorter latencies (input to output delay) than Type III. The circuits implemented for Type II, in general, have smaller area but longer cycle times (input to circuit stabilization delay) than the circuits implemented for Type I.

At power-up[4] or after completion of the previous machine cycle, the machine waits for an input burst to arrive. In a Type I machine cycle, when the machine detects that all of the terminating edges of the input burst have appeared, it generates a concurrent output/state burst (which may be empty), completing a two-phase machine cycle. In a Type II machine cycle, when the machine detects that all of the terminating edges of the input burst have occurred, it generates an output burst (which may be empty). A state burst (which may also be empty) immediately follows the output burst, completing the three-phase cycle. Note that an output burst enables a state burst in the "burst-mode fashion"—the state variable changes are enabled only after all the changes of the output burst have fed back. In a Type III machine cycle, a state burst is enabled by the input burst and an output burst is enabled by the state burst. Note that the state assignment scheme in Example I produced two types of machine cycles: Type III for the transition from S_1 to S_3 and Type I for the transition from S_3 to S_1. However, state assignment schemes that generate a different combination of machine cycles can be used just as well.

[4]An explicit reset signal is used, when necessary, to ensure that all primary outputs and state variables are initialized to correct values.

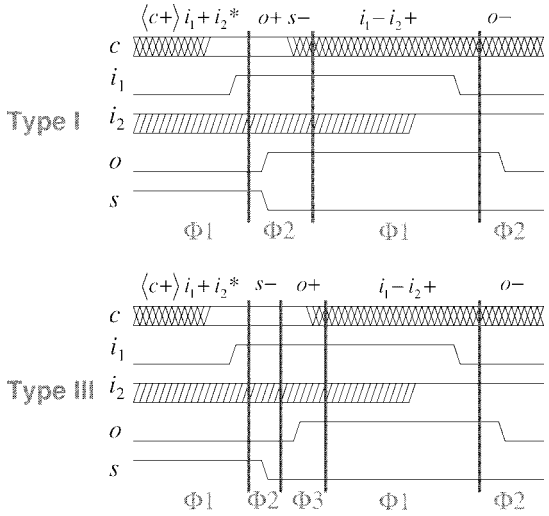

Fig. 12. 3D machine cycles (Types I and III).

Fig. 12 illustrates examples of two machine cycles (Type I and Type III). The first machine cycle begins with input burst (phase 1) $\langle c+\rangle i_1+i_2*$. The conditional signal c stabilizes to 1 before i_1+ fires. The directed don't care signal i_2 may remain at 0 or change to 1. In the Type I machine cycle, this input burst enables a concurrent output/state burst (phase 2) $o+s-$. In the Type III machine cycle, this input burst enables the state burst (phase 2) $s-$, which, in turn, enables the output burst (phase 3), $o+$. In the second machine cycle, an input burst i_1-i_2+ enables an output burst $o-$, and no state burst is required. Thus, for machine cycles not requiring state variable transitions, Type I and Type III are indistinguishable.

IV. Hazard-Free Implementations

There are many implementation styles that can be used to synthesize asynchronous controllers—each has advantages and disadvantages. This paper describes one particular implementation style called 3D, which is suitable for implementing extended burst-mode machines. It is similar to Huffman-mode machines [18], [19] in structure and similar to Mealy machines [30] in functionality.

The main problem in ensuring the correctness of asynchronous circuits is avoiding the possibility of hazards. A hazard is broadly construed as a potential for malfunction of the implementation. We review precise characterization of various kinds of hazards and describe how each is avoided. We show that the 3D machine synthesis problem reduces to one of synthesizing hazard-free next-state circuits and then show how the various sources of hazards are systematically eliminated.

Fig. 13 illustrates how the 3D machine can be viewed as a next-state logic function during each phase (Type II machine cycle is used in this example). Assume that no fed-back output change arrives at the network input until all of the specified changes of the output burst have appeared at the network output. The same assumption applies to the fed-back state variable changes and the state burst. These conditions will be met by inserting delays in the feedback paths as necessary. The machine then can be viewed as a next-state logic function

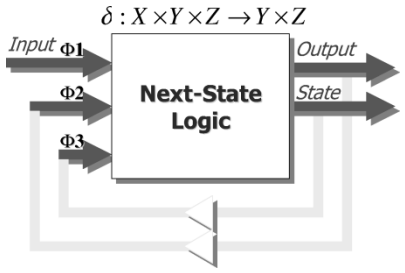

Fig. 13. Combinational view of the 3D state machine.

1) excited by the input changes during the input bursts (phase 1);
2) excited by the fed-back output changes during the output bursts (phase 2);
3) excited by the fed-back state variable changes during the state bursts (phase 3).

Note that the machine is stable at the beginning of each phase.

Therefore, the 3D machine synthesis procedure follows these steps:

1) specify a function-hazard-free next-state function that can be transformed into a hazard-free circuit;
2) implement a hazard-free circuit from the specified next-state function;
3) ensure that the sequential circuit created by connecting feedback paths are free of hazards.

The first step of the synthesis procedure is to correctly specify a next-state function that conforms to the specification. This step must ensure that the specified function is free of *function hazards*, that is, for every set of input changes and feedback signal changes with all the signals not specified to change set to correct values, both the static and dynamic behavior of every output is exactly as specified. In addition, this functional synthesis step must take measures to ensure that a hazard-free circuit exists for the specified function.

The second step of the synthesis procedure is to correctly implement a next-state circuit from the next-state function specified in the previous step. That is, this step must implement a circuit free of *logic hazards*.

The last step of the synthesis procedure is to complete the circuit construction by connecting outputs of the next-state network to the inputs, that is, creating feedback paths. This step must ensure that the circuit created by connecting feedback paths is free of *sequential hazards*, that is, the circuit behaves as specified as a sequential machine.

In the remainder of this section, we examine the sources of hazards (sequential hazards, function hazards, and logic hazards) in detail and provide remedies for each. The synthesis procedure itself and the algorithms are presented in Part II.

A. Sequential Hazard

The correct operation of the 3D machine relies on the assumption that all of the specified changes of the outputs of the next-state network excited by a set of changes at the network inputs are completed before the next set of changes arrives at the network inputs. A violation of this assumption may result in a *sequential hazard*, the hazard that

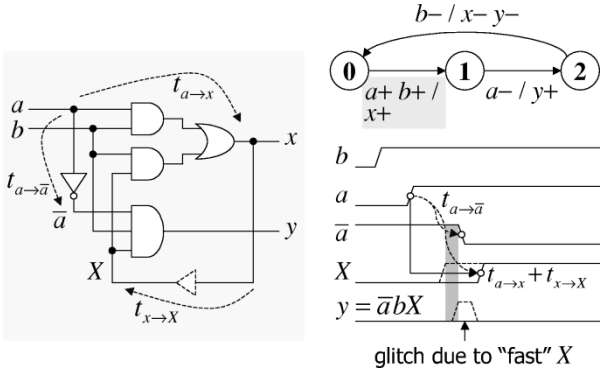

Fig. 14. Essential hazard in two-level AND-OR.

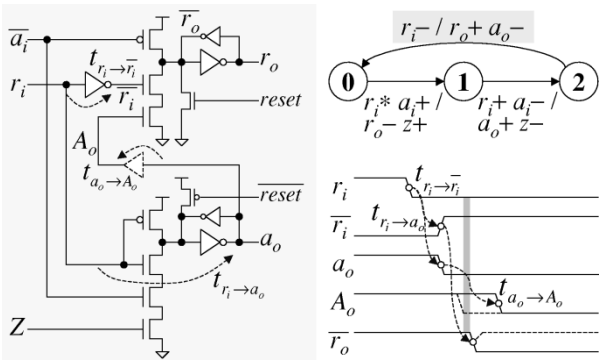

Fig. 15. Essential hazard in gC implementation of fifocellctrl.

exists regardless of the correctness of the underlying next-state circuit. Both the timing characteristics of the circuit itself and the environment of the circuit can cause sequential hazards.

A comprehensive treatment of this topic can be found in [39]. In this paper, only a sufficient condition to ensure freedom from sequential hazards is stated. A solution to synthesize practical circuits which are guaranteed to be free of sequential hazards under a bounded delay assumption can be found in [39].

1) Essential Hazard: We examine how the internal timing of the circuits can introduce sequential hazards. It has been assumed up to now that no change at the network output is fed back to the input of the next-state network until all the changes at the network outputs that are concurrently enabled have taken place. However, this assumption may be violated if feedback delays are short compared to the difference between the maximum and minimum feedforward delays. The hazard that arises due to the race between the arrivals of input edges and one or more fed-back output edges, enabled by the same input changes, at the network input is called *essential hazard* [19].

The possibility of an essential hazard during a $0 \to 0$ transition of an output in a burst-mode two-level AND-OR circuit is illustrated in Fig. 14. During the input/output burst $(a{+}b{+} \to x{+})$, y is specified to remain 0. However, if $x+$ is fed back to the network input before $\bar{a}$ goes low, then a 0-1-0 glitch may propagate to output y. Thus, we need to make sure that the feedback delay $(t_{x\to X})$ is sufficiently large to avoid essential hazards.

The possibility of an essential hazard during a $0 \to 1$ transition of an output in an extended burst-mode gC circuit is shown in Fig. 15. During the state transition from state 2 to 1, r_o and a_o are to rise and fall, respectively, triggered by $r_i{-}$. However, if a_o falls too fast and enables the fed-back output A_o to fall before the internal node $\overline{r_o}$ is pulled sufficiently low, then the gC gate may switch very slowly or may not even switch at all.

Essential hazards, in general, can be avoided simply by *inserting sufficient delays* in the feedback paths. However, the delays in the feedback paths increase the delay constraint between last output change and next compulsory input change that must be obeyed by the environment of the circuit. Hence, it is desirable to minimize feedback delays to improve system performance. Sometimes, it is possible to find tighter constraints, i.e., reduce feedback delays, if the details of the implementation technology are known [39].

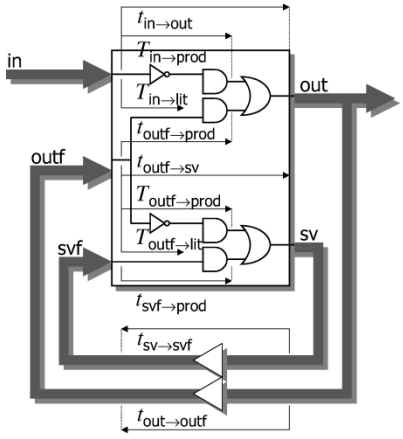

Fig. 16. Timing requirements for minimum feedback delay.

Sufficient Conditions for Freedom from Essential Hazards

If 3D machines are implemented in two-level AND-OR, a set of simple one-sided timing constraints can be used to characterize the minimum required feedback delay. We show below a set of timing constraints for Type II machine cycles. $t_{x\to y}$ denotes the minimum delay from a transition of type x to a transition of type y, while $T_{x\to y}$ denotes the maximum delay (see Fig. 16).

a) $t_{\text{in}\to\text{out}} + t_{\text{out}\to\text{outf}} > T_{\text{in}\to\text{lit}}$;
b) $t_{\text{in}\to\text{out}} + t_{\text{out}\to\text{outf}} + t_{\text{outf}\to\text{prod}} > T_{\text{in}\to\text{prod}}$;
c) $t_{\text{outf}\to\text{sv}} + t_{\text{sv}\to\text{svf}} > T_{\text{outf}\to\text{lit}}$;
d) $t_{\text{outf}\to\text{sv}} + t_{\text{sv}\to\text{svf}} + t_{\text{svf}\to\text{prod}} > T_{\text{outf}\to\text{prod}}$.

Usually, these inequalities are satisfied without adding delays, as should be clear by comparing the lengths of the paths followed on each side of the inequalities. Note that the requirements for a Type I machine cycle are simpler, because state variable changes are concurrent with output changes: only the first two inequalities are needed.

If the 3D machines are implemented in gC, the one-sided timing constraint shown below can be used to determine the

minimum required feedback delay

$$t_{\text{in}\rightarrow\text{out}} + t_{\text{out}\rightarrow\text{outf}} > T_{\text{in}\rightarrow\overline{\text{out}}}$$

where $\overline{\text{out}}$ represents the internal node of the gC circuit. We can, of course, perform a detailed post-synthesis timing analysis [39] to determine tighter bounds. However, for gC circuits in very high performance applications, even further reduction in feedback delay may be desirable. We, therefore, introduce a simple remedy, which works for most gC circuits, to eliminate feedback delay requirement altogether in Section IV.C.2.

2) Environmental Constraints: An inherent feature of the 3D implementation is that parts of the circuit may still be unstable after a change at the network output has taken place. In some sense, this feature can help improve the performance of the system by effectively making the stabilization of the circuit and the reaction of the environment concurrent [27], provided that the environment is slow to react to the changes in the circuit outputs. However, if the environment reacts so fast that the circuit detects the new input arrivals before the arrival of feedback variable changes, then the circuit may malfunction. Therefore, we must have the environment delay generating certain changes. This is called the *fundamental-mode* environmental constraint. In practice, this is usually not a problem, because of the delays in wires between the circuit and the environment and the time for the environment to react are generally longer than it takes for the circuits to stabilize. In addition, not all the input signals have to meet this constraint, because some signals are specified as don't cares in the extended burst-mode. Note that feedback signals cause all the primary outputs and state variables to "latch" their values, regardless of changes in don't care signals. Thus the fundamental-mode constraint under the presence of don't cares is with respect to this "latching" time.

Other forms of the environmental constraint required by the extended burst-mode 3D machine are the *setup time* and *hold time* requirements: all conditional signals specified to stabilize must stabilize for some interval before any compulsory (sampling) edge appears and must remain stable until the output/state burst has been completed. This requirement is similar to the setup and hold requirements on data signals with respect to clock of synchronous flip-flops.

3) Summary: The following are the timing requirements imposed by the synthesis method to guarantee correctness of the implementation.

a) *Feedback delay requirement*: feedback variable changes are not fed back until *all enabled* output and state variable changes have been completed.
b) *Fundamental-mode environmental constraint*: no compulsory edges of the next input burst may arrive until the machine has been stabilized.
c) *Setup and hold time requirements*: all conditional signals specified to be stable must be stabilized before any compulsory (sampling) edge appears and must remain stable until the output/state burst has been completed.

Assuming these timing constraints are met, we need only analyze the hazards in the next-state circuit that result from cutting feedback paths. Note that the next-state function is a *combinational* function, although it may be implemented with sequential circuit elements, such as generalized C-elements.

B. Function Hazard

A *function hazard* is a nonmonotonic change, i.e., more than one change, of a combinational function during a multiple-input change [40], [19]. Function hazards are problematic because they are present in *every* gate-level implementation of the function, if inputs to functions have arbitrary delay. Consequently, function hazards must be prevented before combinational synthesis. We consider function hazards during multiple-input changes in which some inputs are nonmonotonic, i.e., change more than once. We examine the implications of allowing certain input changes to be nonmonotonic, define what a function hazard is in this setting, and explain how function hazards are avoided in the 3D implementations.

1) Definitions: We summarize some definitions and concepts from [41]–[44] that are used in the following subsections.

A *logic function* f is a mapping from $\{0,1\}^n$ to $\{0,1,*\}$. A *minterm* of f is an n-tuple $[x_1, x_2, \cdots, x_n]$ where x_i, the value of the ith input of f, is 0 or 1.

The *on-set* of f is the set of minterms for which f is one; the *off-set* of f is the set of minterms for which f is zero; the *dc-set* of f is the set of minterms for which f is $*$.

A *cube* c, written as $[c_1, c_2, \cdots, c_n]$, is a vector in $\{0,1,*\}^n$. A minterm $[x_1, x_2, \cdots, x_n]$ is a cube such that for every $i \in 1, \cdots, n$, $x_i \neq *$.

A cube $[a_1, a_2, \cdots, a_n]$ is said to *contain* another cube $[b_1, b_2, \cdots, b_n]$ iff, for all i in $1, \cdots, n$, $a_i = b_i$ or $a_i = *$.

A cube $[a_1, a_2, \cdots, a_n]$ is said to *intersect* another cube $[b_1, b_2, \cdots, b_n]$ iff, for all i in $1, \cdots, n$, $a_i = b_i$ or $a_i = *$ or $b_i = *$.

A *literal* is a variable or its complement. A *product term* is a boolean product of literals, and a *sum of products* is a boolean sum of product terms. We consider only product terms satisfying the restriction that *no product term can have both a variable and its complement as inputs*. With this restriction, there is a one-to-one correspondence between product terms and cubes, so we use the terms *cube* and *product term* interchangeably. Thus a product term $x_1\overline{x_3}x_4$ is equivalent to a cube $[1, *, 0, 1, *, \cdots, *]$.

An *implicant* of f is a product term which contains no off-set minterms of f.

A *cover* C of a logic function f is a set of implicants of f such that every on-set minterm of f is contained in some cube of C but no off-set minterm. A cover is isomorphic to a sum-of-products implementation of f.

If $A = [a_1, a_2, \cdots, a_n]$ and $B = [b_1, b_2, \cdots, b_n]$, the *transition cube* $C = [c_1, c_2, \cdots, c_n]$ is determined so that, for $i = 1, \cdots, n$, $c_i = a_i = b_i$ if $a_i = b_i$, and $c_i = *$ if $a_i \neq b_i$. The transition cube C, denoted as $[A, B]$, is the smallest cube that contains both A and B.

A *trajectory* in $[A, B]$ is a vector of minterms contained in $[A, B]$, denoted as $[m_1, m_2, \cdots, m_p]$, such that, for every j in $1, \cdots, p-1$, the minterms m_j and m_{j+1} differ in just one bit position.

A combinational function has a function hazard if it changes more than once during a specified multiple-input change.

Assume, for now, that all input changes are monotonic (we will generalize it so that some input changes can be nonmonotonic in the following subsection). There is a corresponding transition cube for every multiple-input change. The transition cube contains all of the minterms in every possible trajectory of the specified input changes. If the function changes its value more than once along a certain trajectory, then there is a function hazard. The following "classical" definition of function hazard adapted from [42] captures this notion precisely.

Definition 1: A combinational function f contains a **function hazard** during a multiple-input change from A to B iff there exists a pair of minterms X and Y in $[A,B]$ ($A \neq X$ and $Y \neq B$) such that

1) $X \in [A,B]$ and $Y \in [X,B]$ and
2) $f(A) \neq f(X)$ and $f(Y) \neq f(B)$.

If $f(A) = f(B)$, it is a *static function hazard*, that is, a 1-0-1 or 0-1-0 function hazard. Otherwise, it is a *dynamic function hazard*, that is, a 1-0-1-0 or 0-1-0-1 function hazard.

2) Generalized Transition: If some inputs are allowed to change nonmonotonically during multiple-input changes, the classical definition of function hazard is inadequate. We develop a notion of generalized transition to remedy this deficiency and to provide a vehicle to discuss functional synthesis in analytical terms in Part II.

A *generalized transition* (T,A,B) defines a set of all *legal* trajectories in $[A,B]$, where A is a *start cube*, B is an *end cube*, and T is a mapping from a set of input signals to a set of *input types*. There are three types of inputs: *rising-edge*, *falling-edge*, and *level*. Edge inputs must change monotonically; therefore, edge inputs change at most once in a legal trajectory. Level inputs must be a constant (0 or 1) or a don't care, which implies that each level input must hold the same value in both A and B or be undefined in both A and B. Level inputs, if they are don't cares, may change nonmonotonically.

A generalized transition cube $[A,B]$ is the smallest cube that contains the start and end cubes A and B. Not all combinations of T and $[A,B]$ are legal. For example, if input i is 0 in A but 1 in B, then i can neither be a level type nor a falling-edge type. In summary:

1) if the value of i is the same in both A and B, i.e., both 0, both 1, or both $*$, then i can be of any type;
2) if $i = 0$ in A and 1 or $*$ in B, then i must be a rising-edge type;
3) if $i = 1$ in A and 0 or $*$ in B, then i must be a falling-edge type.

Open generalized transition cubes, $[A,B) = [A,B] - B$, $(A,B] = [A,B] - A$, and $(A,B) = [A,B) - A$, respectively. Note that $[A,B) = \emptyset$, if $A \subseteq B$. The *start subcube* A' is a maximal subcube of A such that:

1) the value of every rising-edge input i in A' is 0, if it is $*$ in A;
2) the value of every falling-edge input j in A' is 1, if it is $*$ in A.

The *end subcube* B' is a maximal subcube of B such that:

1) the value of every rising-edge input i in B' is 1, if it is $*$ in B;
2) the value of every falling-edge input j in B' is 0, if it is $*$ in B.

Intuitively, if edge signals have weight 1 and level signals have weight 0, the trajectories from A' to B' are the maximum-weight trajectories. If every don't care input is an edge signal in (T,A,B), $[A',B'] = [A,B]$ and A' and B' reduce to minterms.

Lemma 1: For every minterm X in $[A,B]$, all of the minterms in every legal trajectory from X to B is contained in $[X,B']$.

Proof: We prove by contradiction. Assume that there exists a legal trajectory such that one of the minterms in the trajectory is outside of $[X,B']$.

1) $[X,B'] = [X,B]$
   $[X,B]$ contains all of the minterms in every trajectory from X to B, which contradicts the assumption.
2) $[X,B'] \subset [X,B]$
   Then there exists Y contained in $[X,B] - [X,B']$ such that Y can be reached from X legally. Let $X = [x_1,\cdots,x_n]$, $Y = [y_1,\cdots,y_n]$, $B' = [b'_1,\cdots,b'_n]$, and $[X,B'] = [c_1,\cdots,c_n]$. For every i in $1,\cdots,n$, $c_i = *$ or $c_i = b'_i$. Since Y is not contained in $[X,B']$, there exists an edge signal j such that $y_j \neq * \wedge y_j \neq b'_j \wedge c_j \neq *$. $c_j \neq *$ implies that $x_j = b'_j$. This means that j has already reached the final value at point X, by the definition of B'. Therefore, Y cannot be reached legally from X, which is a contradiction. □

During a generalized transition (T,A,B), each output signal is assumed to change its value at most once. Furthermore, no output change is allowed in A and B. If not, a function hazard is said to be present. Below is the new definition of function hazard adapted for generalized transitions.

Definition 2: A combinational function f contains a **function hazard** in (T,A,B) iff

1) there exists a pair of minterms X,Y in A such that $f(X) \neq f(Y)$, *or*
2) there exists a pair of minterms X,Y in B such that $f(X) \neq f(Y)$, *or*
3) there exists a pair X,Y in (A,B) such that $Y \in [X,B')$ (or, equivalently, $X \in (A',Y]$) and $f(A) \neq f(X)$ and $f(Y) \neq f(B)$.

The last criterion states that there is a function hazard if there exist two minterms X and Y in a legal trajectory from A to B such that $f(A) \neq f(X)$ and $f(Y) \neq f(B)$.

A generalized transition (T,A,B) is a static transition for f iff $f(A) = f(B)$; it is a dynamic transition for f iff $f(A) \neq f(B)$. No change in level inputs can enable output changes directly, that is, at least one edge input must change from 0 to 1 or from 1 to 0 in a generalized dynamic transition.

Examples of generalized transitions are shown in Fig. 17. Fig 17(a), (b), and (c) shows rising-edge signals, and s is a level signal. Fig. 17(a) and (c) shows function-hazard-free static and dynamic transitions, respectively. Fig. 17(b) illustrates a 1-0-1 static function hazard, and Fig. 17(d) does a 0-1-0-1 dynamic function hazard on the trajectory, abc : $000 \rightarrow 100 \rightarrow 101 \rightarrow 111$.

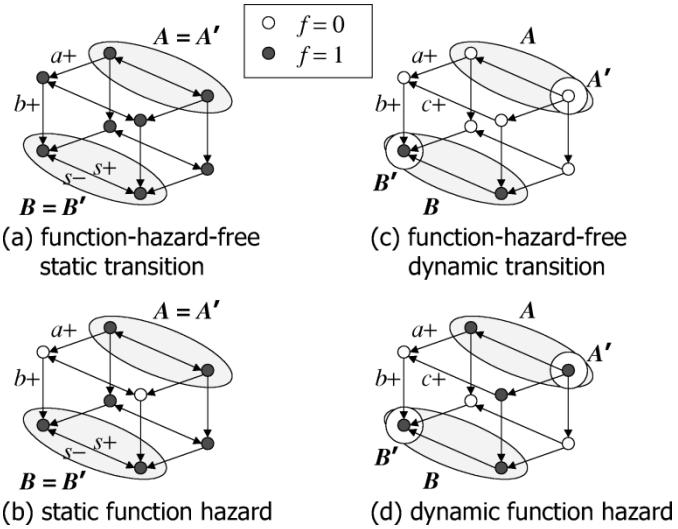

Fig. 17. Hazards in generalized transitions. s is a level signal; a, b, and c are rising-edge signals.

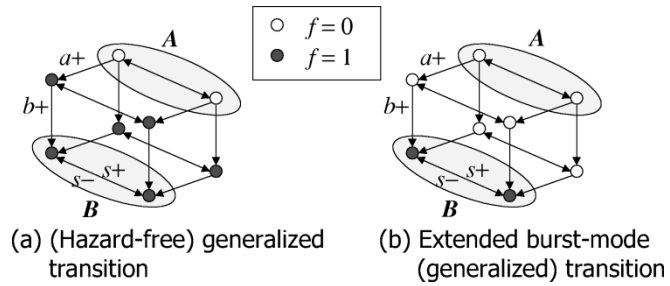

Fig. 18. (a) Generalized transition but not extended burst-mode transition: there exists X, Y in $A \cup [A, B)$ such that $f(X) \neq f(Y)$; (b) extended burst-mode (generalized) transition.

3) Extended Burst-Mode Transition: An extended burst-mode transition is a generalized transition with the following requirements.

a) For every pair of minterms X and Y in $A \cup [A, B)$, $f(X) = f(Y)$.
b) For every pair of minterms X and Y in B, $f(X) = f(Y)$.

Theorem 1: Every extended burst-mode transition is function-hazard-free.

Proof: Consider f during an extended burst-mode transition from A to B. Since $A \subseteq A \cup [A, B)$, for every pair of minterms X and Y in A, $f(X) = f(Y)$ by requirement 1 of the definition of extended burst-mode transition. This contradicts criterion 1 of Definition 2. For every pair of minterms X and Y in B, $f(X) = f(Y)$ by requirement 2, which contradicts criterion 2 of Definition 2. Finally, for all X in (A, B), $f(X) = f(A)$ by requirement 1, which contradicts criterion 3 of Definition 2. Therefore, f is free of function hazards. □

A (hazard-free) generalized transition (but not an extended burst-mode transition) and an extended burst-mode transition are shown in Fig. 18.

An edge signal that changes from 0 or $*$ to 1 or from 1 or $*$ to 0 during an extended burst-mode transition from A to B is a *terminating* signal in (T, A, B). An edge signal whose value is $*$ in B is a *directed don't care* in (T, A, B). A level signal whose value is $*$ in (T, A, B) is an *undirected don't care*. In a dynamic extended burst-mode transition, the output is enabled to change only after all of the terminating edges appear.

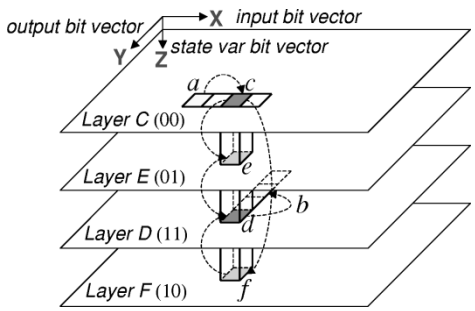

Fig. 19. Critical race during a Type III machine cycle.

Another way of describing terminating signals and don't cares is as follows: Let minterms $X = [\cdots, x_s, \cdots]$ and $X' = [\cdots, \overline{x_s}, \cdots]$, where x_s and $\overline{x_s}$ are the values of s in X and X'. s is a terminating signal iff $X \in B$ implies $X' \notin B$. s is a don't care (directed or undirected) iff $X \in B \Leftrightarrow X' \in B$ or, equivalently, $X \in [A, B) \Leftrightarrow X' \in [A, B)$.

A 3D machine cycle that requires no conditional signals to stabilize has transitions corresponding to an input burst and a concurrent output/state burst, if it is of Type I, or an input burst, an output burst, and a state burst, if it is of Type II or III. A 3D machine cycle that requires conditional signals to stabilize has an additional transition for setting up conditional signals. Each of these transitions by itself is free of function hazards, since these are all extended burst-mode transitions. However, as we have seen in Example I in Section III-A, a function-hazard-free next-state assignment requirement for one transition may conflict with another transition. The 3D state assignment algorithm avoids this type of conflict by adding state variables when necessary, as described in Part II.

4) Critical Race: If a transition between layers requires multiple state bit changes (see Fig. 19), the machine traverses intermediate layers (E or F) before reaching the final state (d) of the transition. In traditional asynchronous state machines [18], [19], a *critical race* is said to be present if reaching the final state depends on the order in which the state bits change. In 3D machines, *a **critical race** is said to be present if the transient states during a layer transition have different next values of outputs and state variables from those of the start-state of the transition*. Hence, in 3D machines, a critical race is simply a manifestation of a function hazard during a state burst. We insure that the machine is free of critical races by encoding layers so that the next states of the transient states during layer transitions are the same as those of the start-state of the transition.

C. Logic Hazards

Hazards in next-state circuits can also be introduced by the delay variations of physical gates and wires, even if the next-state functions are completely and correctly specified, i.e., function-hazard-free. In this section, we present two

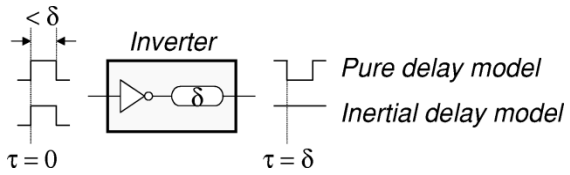

Fig. 20. Delay models.

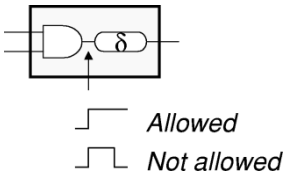

Fig. 21. Delay model used in 3D combinational synthesis.

different methods to implement hazard-free next-state logic: the two-level AND-OR implementation [43] and the generalized C-element implementation [34].

The existence of hazards depends on the delay assumptions in the circuit model used and on the models of the delay itself. Many delay models have been proposed [19], [45], [46]. Fig. 20 shows two commonly used examples: the *inertial* delay model which assumes that no input pulse of duration shorter than the gate delay is transported to the output of the gate, and the *pure* delay model which assumes that a pulse of any duration computed by the logic function of the gate is asserted on the gate output.

Our *combinational* synthesis method works for *all* delay models, because we use a strategy to avoid multiple input changes to a delay before output, as shown in Fig. 21, for all primary outputs of next-state circuits. In addition, we assume that both the gates and the wires connecting gates in the next-state network have finite but arbitrary delays.

1) Two-Level AND-OR Implementation: First, we consider the implementation of the next-state functions in two-level AND-OR logic. We develop a set of hazard-free covering requirements for the two-level AND-OR implementation of a logic function during an extended burst-mode transition. The hazard-free combinational logic synthesis for multiple *monotonic* input changes is described in [40]–[43] and [47]. The *new* results presented here are simple extensions of the theory in [43] to account for nonmonotonic input changes. We apply these results to the 3D machine combinational logic synthesis.

The following definitions of logic hazards are from [42] and [43].

Definition 3: A combinational network contains a **static logic hazard** during a function-hazard-free input change from A to B iff

1) $f(A) = f(B)$;
2) a momentary pulse may be present during the input change from A to B.

Definition 4: A combinational network contains a **dynamic logic hazard** during a function-hazard-free input change from A to B iff

1) $f(A) \neq f(B)$;
2) a momentary 0 and a momentary 1 output may appear during the input change from A to B.

Below, we state and prove necessary and sufficient conditions for hazard freedom for a two-level AND-OR circuit during an extended burst-mode transition. Note that the *product term* refers to an *on-set* cube for the remainder of this section.

Lemma 2: A product term that does not intersect the generalized transition cube $[A, B]$ remains 0 during a function-hazard-free transition (T, A, B).

Proof: Every product term that does not intersect $[A, B]$ has a literal whose value remains 0 during the input change. Thus a product term not intersecting $[A, B]$ remains 0. □

Lemma 3: A product term that contains A' (or B') changes monotonically during an extended burst-mode transition (T, A, B).

Proof: First, consider the case in which a product term p contains both A' and the start-point of a trajectory in the transition (T, A, B). The initial values of all the literals of p are 1. Level signals are either constants or don't cares in A'. If a level signal is a don't care in A', then it is a don't care in the cube that contains A'; therefore, it does not appear as a literal in the corresponding product term. Since all other input changes are monotonic, values of the literals change monotonically from 1 to 1 or from 1 to 0. Thus the output of p changes monotonically.

Now consider the case in which the product term p contains A' but not the start-point. By the definition of A', at least one monotonic change of an edge signal is needed to traverse from A' to a start-point in $A - A'$; no additional change of the same signal occurs in $[A, B]$. The value of the literal in p which corresponds to this input signal falls during a transition from A' to the start-point and remains 0. Thus the output of p remains 0 in $[A, B]$ if the start-point of the trajectory is not contained in p.

Thus the output of a product term that contains A' changes monotonically ($1 \rightarrow 1$, $1 \rightarrow 0$, or $0 \rightarrow 0$).

Using the same argument, the output of a product term that contains B' also changes monotonically ($0 \rightarrow 0$, $0 \rightarrow 1$, or $1 \rightarrow 1$). □

Theorem 2: The output of a two-level AND-OR circuit is hazard-free during a $0 \rightarrow 0$ extended burst-mode transition.

Proof: No product term intersects the transition cube since the transition is function-hazard-free. Thus all the product terms in the network remain 0 during the transition by Lemma 2. □

Theorem 3: The output of a two-level AND-OR circuit is hazard-free during a $1 \rightarrow 1$ extended burst-mode transition iff the circuit contains a product term which contains the transition cube $[A, B]$.

Proof: ($\Rightarrow$) Assume that the circuit does not contain a product term that contains $[A, B]$. In order for the transition to be function-hazard-free, $[A, B]$ is covered by more than one product term. During a transition from A to B, one or more product terms rise, one or more product terms fall, and the rest remain 0. If a falling edge of a product term precedes all rising edges, the output of the circuit may change from 1 to 0 to 1, which is a hazard, contradicting the hypothesis.

($\Leftarrow$) The output of a product term that contains $[A, B]$ remains 1 during a transition from any point in A to any point in B, because there is no literal in this product term that can change in $[A, B]$. Hence, the sum of products remains 1 throughout the trajectory. □

Theorem 4: The output of a two-level AND-OR circuit is hazard-free during a $1 \rightarrow 0$ extended burst-mode transition iff every product term intersecting the transition cube $[A, B]$ also contains the start subcube A'.

Proof: ($\Rightarrow$) Assume that there exists a product term p that intersects $[A, B]$ but does not contain A'. Consider a trajectory from a point in A' not contained in p to any point in B. The initial value of p is 0 since p does not contain the start-point. The final value of p is 0 because the final value of the output of the network must be 0. Because p intersects $[A, B]$, p changes from 0 to 1 to 0 on some trajectories from A' to B. All other product terms that contain A' fall during a transition from A' to B. Since the wire delay on p can be arbitrary, the output of the network may undergo a $1 \rightarrow 0 \rightarrow 1 \rightarrow 0$ transition. Thus the circuit is not hazard-free, which contradicts the hypothesis.

($\Leftarrow$) The final values of all the product terms are 0, because the final value of the output of the network must be 0. By Lemma 3, the product terms that contain A' change monotonically during a transition from A to B. Thus the product terms that intersect $[A, B]$ fall monotonically. The product terms that do not intersect $[A, B]$ remain 0, by Lemma 2. Thus the output of the network changes monotonically, i.e., hazard free. □

Theorem 5: The output of a two-level AND-OR circuit is hazard-free during a $0 \rightarrow 1$ extended burst-mode transition iff no product term intersects the transition cube $[A, B]$ unless it also contains the end subcube B'.

Proof: Exchange 0 and 1 and reverse trajectories in proof of Theorem 4. □

The hazard-free covering requirements for two-level AND-OR logic for extended burst-mode transitions can be summarized as below.

a) For every $1 \rightarrow 1$ transition:
   There exists a product term that contains $[A, B]$.
b) For every $1 \rightarrow 0$ $(0 \rightarrow 1)$ transition:
   Every product term that intersects $[A, B]$ must also contain A' (B').

Each maximal subcube of $[A, B]$ needed to satisfy the covering requirements above is called a *required cube* of $[A, B]$ [43], [22]. Just one cube is required for a $0 \rightarrow 1$ or $1 \rightarrow 1$ transition, whereas n cubes are required for a $1 \rightarrow 0$ transition enabled by n terminating input edges. Fig. 22 illustrates the hazard-free covering requirements for the example in Fig. 15.

Suppose a generalized transition cube $[A, B]$ for a $1 \rightarrow 0$ extended burst-mode transition is intersected by a required cube c_r (required for another transition). If c_r does not contain A' and cannot be expanded (by assigning 1 to don't care entries) to contain A', then the implementation has a dynamic logic hazard. Fig. 23 illustrates four examples of illegal intersections of transition cubes. In each of these cases,

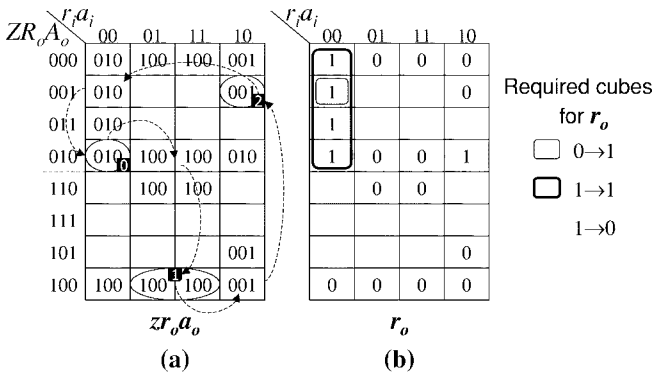

Fig. 22. Required cubes for two-level SOP implementation of r_o.

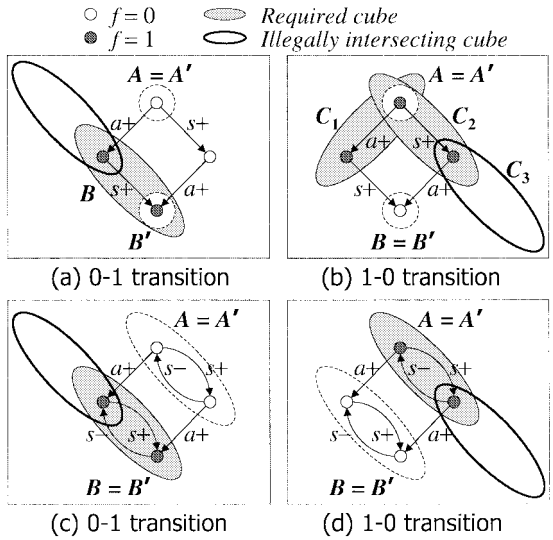

Fig. 23. Illegal intersection of privileged cube.

a dynamic logic hazard is present in the implementation of x. For instance, in Fig. 23(b), the output of c_3 may glitch (0-1-0) if s rises momentarily before a rises but the output of c_3 is slow to change, and this glitch may propagate to the output. This observation leads to the notion of privileged cube.

A generalized transition cube $[A, B]$ for a $1 \rightarrow 0$ extended burst-mode transition is said to be a *privileged cube* [43], [22] iff $[A, B)$ contains more than one minterm. Likewise, a generalized transition cube $[A, B]$ for a $0 \rightarrow 1$ extended burst-mode transition is said to be a *privileged cube* iff B contains more than one minterm. A cube that intersects a privileged $1 \rightarrow 0$ transition cube must also contain the start subcube, and a cube that intersects a privileged $0 \rightarrow 1$ transition cube must also contain the end subcube. Otherwise, the cube is said to intersect the privileged cube *illegally*.

To summarize, a cover C of a logic function f that implements an output or a state variable of the 3D machine is free of logic hazards iff it includes all of the required cubes and no cube in C intersects a privileged cube illegally.

2) Generalized C-Element Implementation: The synthesis method produces two-level AND-OR circuits for both set logic (f_{set}) and reset logic (f_{reset}). The N stack of the generalized C-element in Fig. 6 is simply the N stack of a fully complementary complex AND-OR-NOT gate that implements

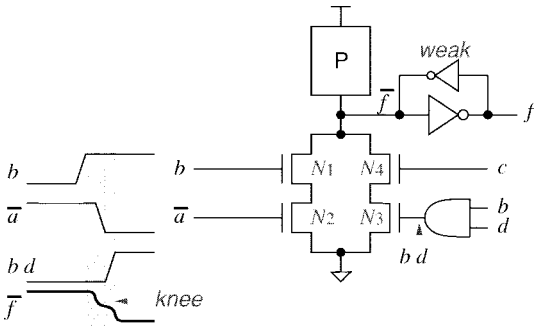

Fig. 24. Dynamic hazard in generalized C-elements.

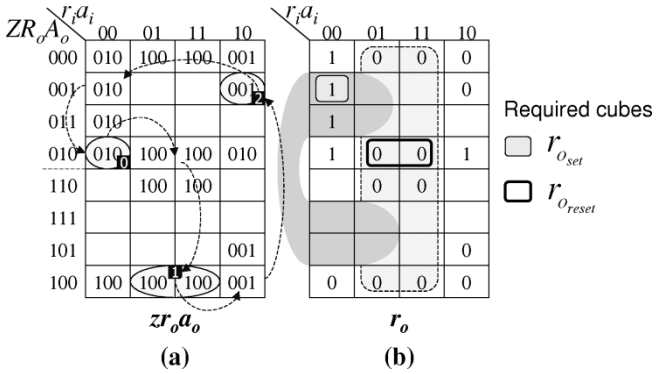

Fig. 25. (a) K-map for fifocellctrl example in Fig. 15. (b) Required cubes for gC implementation of r_o.

$\overline{f_{\mathsf{set}}}$; the P stack of the generalized C-element is the P stack of a full complementary complex AND-OR-NOT gate that implements f_{reset}. For example, a gC implementation of E for $E_{\mathsf{set}} = sZ_0$ and $E_{\mathsf{reset}} = \overline{s}\overline{a}$ is shown in Fig. 11(b).

The hazard avoidance techniques used for two-level AND-OR apply directly here, because of the way pull-down and pull-up stacks are implemented. As described below, the only difference is that no special precautions are necessary to make static transitions hazard-free. A similar technique was used to optimize complex CMOS gates in [38]. A detailed comparison to [38] can be found in [34].

Extended burst-mode circuits implemented with generalized C-elements as described above are hazard-free during $0 \rightarrow 0$ and $1 \rightarrow 1$ transitions. The output of a two-level AND-OR circuit is hazard-free during a $0 \rightarrow 0$ extended burst-mode transition, as shown in Theorem 2. Thus, when f undergoes a $1 \rightarrow 1$ transition, f_{reset} remains low, keeping the P stack turned off. The N stack, in the meantime, may be turned on or off, but the sustainer maintains the old value of the gC output. Likewise, f is hazard-free during $0 \rightarrow 0$ transitions. Note that static hazards are possible in fully complementary MOS gates if the N and P stacks are duals of each other: i.e., when the N stack is turned off, the P stack is on and vice versa.

On the other hand, special steps must be taken to avoid dynamic hazards. As in two-level AND-OR circuits, for a $0 \rightarrow 1$ transition to be hazard-free, all on-set minterms in each trajectory of the transition must be covered by a single cube, and every cube that intersects the trajectory must also include the endpoint of the trajectory, as shown in Theorem 5. Consider a transition $a*b+$ ($a = b = 0$ and $c = d = 1$ initially), in which f is supposed to rise monotonically when b rises, regardless of the behavior of a. Suppose that $\overline{a}$ and b change as shown in Fig. 24. $\overline{f}$ starts to discharge while b and $\overline{a}$ are both high, stops when $\overline{a}$ falls, and starts again when N_3 is fully turned on (after the AND output rises). Although it is very unlikely that complex gates exhibit glitches as illustrated in Fig. 24, it may be worthwhile to avoid any such possibilities. The synthesis algorithm described in Part II removes any possibilities of dynamic hazards.

To summarize, for the output of a generalized C-element to be hazard-free for a set of extended burst-mode transitions, the following requirements must be met.

a) There are no *reachable* states in which both P and N stacks are on.

b) N stack is hazard-free for all specified $0 \rightarrow 1$ transitions; P stack is hazard-free for all specified $1 \rightarrow 0$ transitions.

Requirement 1 is met by ensuring that the on-set of f_{set} (f_{reset}) is devoid of off-set (on-set) minterms of f. To satisfy requirement 2, we must ensure that, for every $0 \rightarrow 1$ ($1 \rightarrow 0$) transition of f, every product term of f_{set} (f_{reset}) that intersects $[A, B]$ must also contain B' (A'). Hazard-free logic minimization in conjunction with the state assignment step ensures that requirement 2 is met.

As in two-level synthesis, each maximal subcube of $[A, B]$ needed to satisfy the covering requirement (requirement 2) is called a *required cube* of $[A, B]$. Fig. 25(b) illustrates the hazard-free covering requirements for gC implementation of r_o from the fifocellctrl example in Fig. 15. Note the absence of required cubes for $1 \rightarrow 1$ ($0 \rightarrow 0$) transitions of $r_o\mathsf{set}$ ($r_o\mathsf{reset}$). This is because the sustainers maintain the logic level once the logic is set (or reset). Note also the overlapping of $r_o\mathsf{set}$ and $r_o\mathsf{reset}$. The overlapping region corresponds to unreachable states.

Removing Feedback Delay Requirement

As discussed in Section IV-A-1, it may be necessary to construct robust circuits without the aid of feedback delays. Unlike two-level AND-OR circuits, the essential hazard occurs only during dynamic transitions involving multiple output changes in gC circuits. For example, consider the state transition from state 2 to 1 in Fig. 15, during which r_o is to rise and a_o is to fall. In order to enable r_o+, A_o (the fed-back a_o) must be high. However, if a_o switches too fast, A_o may fall before r_o has switched. To prevent this, the product term to enable r_o+ must not include A_o as one of its literals.

A sufficient condition to guarantee this is to include the transition cubes associated with the output bursts as required cubes of f_{set}, if f is enabled to rise in the corresponding input bursts, and as required cubes of f_{reset}, if f is enabled to fall in the corresponding input bursts. For example, the *new* required cubes for r_o and the corresponding circuit implementation are shown in Fig. 26.

This requirement—*robust covering requirement for feedback delay removal in gC*—reduces the don't care space for logic minimization. The resulting circuit implementation may be less compact and may have longer latency but shorter cycle time.

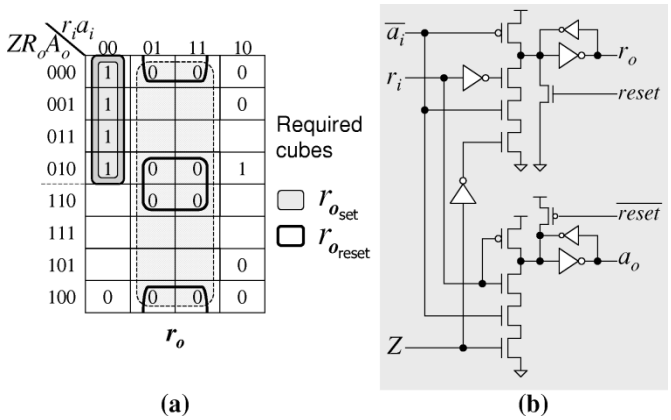

Fig. 26. (a) Required cubes for gC implementation of r_o which removes the feedback delay requirement. (b) The corresponding circuit implementation.

We modified the required cube generation part of the synthesis tool to study the impacts of this new covering requirement. We report the experimental results in Part II. Also note that this requirement makes state minimization less flexible in some rare cases, which is also discussed in Part II.

Signal Placement and Decomposition of Series Stacks

In extended burst-mode circuits, the order of signal arrivals is largely predetermined, so the signal placement can be optimized for performance. In our synthesis method, primary input signals that enable an output to change, i.e., *terminating* signals, are placed at the *top* of the stack (farthest from V_{DD}/Ground). Fed-back outputs and state variables are placed at the *bottom* of the stack (nearest to V_{DD}/Ground), because feedback signals do not enable outputs to change.

Although somewhat longer series stacks can be tolerated in extended burst-mode circuits than in conventional combinational circuits, as demonstrated in actual fabricated chips [27], larger circuits and deep submicron designs require a capability to decompose long stacks. The most straightforward way to decompose a long stack is to partition the signals and map every partition with more than one signal to a static AND/NAND followed by a transistor. This decomposition is hazard-free because each series stack corresponds to an AND gate in the AND-OR network that implements f_{set} or f_{reset} and decomposing AND gates recursively is hazard-free [19]. However, arbitrary partitioning is not allowed because it can lead to dc-path problems during dynamic transitions. The details of legal decomposition of gC burst-mode circuits can be found in [48].

3) Logic Minimization: For both static two-level AND-OR and pseudostatic gC implementations, we use exact algorithms for hazard-free logic, implemented in an automated logic minimizer [49], for hazard-free logic minimization.

V. Conclusion

We formally defined the extended burst-mode specification, presented an overview of the 3D synthesis, and discussed hazard elimination strategies. Because the most difficult problem in asynchronous circuit synthesis is avoiding hazards, we reviewed precise characterization of various kinds of hazards and described how each is avoided in the 3D machine implementation. We showed that the 3D machine synthesis problem reduces down to one of synthesizing hazard-free next-state logic and presented two approaches for next-state logic synthesis: two-level AND-OR implementation and generalized C-element implementation. We also presented an extension to existing theories for hazard-free combinational synthesis to handle nonmonotonic inputs. We showed that these methods require different constraints to guarantee that implementations are hazard-free. In Part II of the paper, we will show how the selection of the next-state logic synthesis method affects the state assignment. We will present an extensive set of experimental results and compare our results to competing methods whenever possible.

Acknowledgment

The authors would like to thank S. Nowick of Columbia University for insightful comments on hazard-free covering requirements and for many helpful discussions and the anonymous reviewers for their helpful suggestions.

References

[1] A. J. Martin, "Compiling communicating processes into delay-insensitive VLSI circuits," *Distributed Comput.*, vol. 1, no. 4, pp. 226–234, 1986.
[2] E. Brunvand, "Translating concurrent communicating programs into asynchronous circuits," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, PA, 1991.
[3] J. C. Ebergen, "A formal approach to designing delay-insensitive circuits," *Distributed Comput.*, vol. 5, no. 3, pp. 107–119, 1991.
[4] S. M. Burns, "Performance analysis and optimization of asynchronous circuits," Ph.D. dissertation, California Inst. Technol., Pasadena, CA, 1991.
[5] M. B. Josephs and J. T. Udding, "An overview of DI algebra," in *Proc. Hawaii Int. Conf. System Sciences*, Jan. 1993, vol. I, pp. 329–338.
[6] G. Gopalakrishnan, P. Kudva, and E. Brunvand, "Peephole optimization of asynchronous macromodule networks," in *Proc. Int. Conf. Computer Design (ICCD)*, Oct. 1994, pp. 442–446.
[7] 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," *IEEE J. Solid-State Circuits*, vol. 29, pp. 1429–1439, Dec. 1994.
[8] T.-A. Chu, "Synthesis of self-timed VLSI circuits from graph-theoretic specifications," Ph.D. dissertation, MIT Laboratory for Computer Science, Cambridge, MA, June 1987.
[9] T. H.-Y. Meng, R. W. Brodersen, and D. G. Messerschmitt, "Automatic synthesis of asynchronous circuits from high-level specifications," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 1185–1205, Nov. 1989.
[10] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli, "Synthesis of hazard-free asynchronous circuits with bounded wire delays," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 61–86, Jan. 1995.
[11] P. Vanbekbergen, "Synthesis of asynchronous control circuits from graph-theoretic specifications," Ph.D. dissertation, Catholic Univ. Leuven, Belgium, Sept. 1993.
[12] C. Ykman-Couvreur, B. Lin, and H. de Man, "Assassin: A synthesis system for asynchronous control circuits," Tech. Rep., IMEC, User and Tutorial manual, Sept. 1994.
[13] J. Cortadella, A Yakovlev, L. Lavagano, and P. Vanbekbergen, "Designing asynchronous circuits from behavioral specifications with internal conflicts," in *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, Nov. 1994, pp. 106–115.
[14] V. I. Varshavsky, Ed., *Self-Timed Control of Concurrent Processes: The Design of Aperiodic Logical Circuits in Computers and Discrete Systems*. Dordrecht, The Netherlands: Kluwer, 1990.
[15] P. A. Beerel, "CAD tools for the synthesis, verification, and testability of robust asynchronous circuits," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1994.
[16] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev, "Basic gate implementation of speed-independent circuits," in *Proc. ACM/IEEE Design Automation Conf.*, June 1994, pp. 56–62.

[17] C. J. Myers and T. H.-Y. Meng, "Synthesis of timed asynchronous circuits," *IEEE Trans. VLSI Syst.*, vol. 1, pp. 106–119, June 1993.
[18] D. A. Huffman, "The synthesis of sequential switching circuits," in *Sequential Machines: Selected Papers*, E. F. Moore, Ed. Reading, MA: Addison-Wesley, 1964.
[19] S. H. Unger, *Asynchronous Sequential Switching Circuits*. New York, Wiley-Interscience, 1969.
[20] B. Coates, A. Davis, and K. Stevens, "The post office experience: Designing a large asynchronous chip," *Integration, VLSI J.*, vol. 15, no. 3, pp. 341–366, Oct. 1993.
[21] S. M. Nowick and D. L. Dill, "Synthesis of asynchronous state machines using a local clock," in *Proc. Int. Conf. Computer Design (ICCD)*, Oct. 1991, pp. 192–197.
[22] S. M. Nowick, "Automatic synthesis of burst-mode asynchronous controllers," Ph.D. dissertation, Stanford Univ., Dept. Comput. Sci., Stanford, CA, 1993.
[23] A. Davis, B. Coates, and K. Stevens, "Automatic synthesis of fast compact asynchronous control circuits," in *Asynchronous Design Methodologies*, S. Furber and M. Edwards. Eds. Elsevier Science Publishers, vol. A-28 of *IFIP Transactions*, pp. 193–207, 1993.
[24] K. Y. Yun, "Synthesis of asynchronous controllers for heterogeneous systems," Ph.D. dissertation, Stanford Univ., Stanford, CA, Aug. 1994.
[25] S. M. Nowick and B. Coates, "UCLOCK: Automated design of high-performance asynchronous state machines," in *Proc. Int. Conf. Computer Design (ICCD)*, Oct. 1994, pp. 434–441.
[26] B. Coates, A. Davis, and K. Stevens, "The Post Office experience: Designing a large asynchronous chip," *Integration, VLSI J.*, vol. 15, no. 3, pp. 341–366, Oct. 1993.
[27] K. Y. Yun, P. A. Beerel, V. Vakilotojar, A. E. Dooply, and J. Arceo, "The design and verification of a high-performance low-control-overhead asynchronous differential equation solver," *IEEE Trans. VLSI Syst., vol. 6, pp. 643–655, Dec. 1998.*
[28] K. Y. Yun and D. L. Dill, "A high-performance asynchronous SCSI controller," in *Proc. Int. Conf. Computer Design (ICCD)*, 1995, pp. 44–49.
[29] A. Marshall, B. Coates, and P. Siegel, "Designing an asynchronous communications chip," *IEEE Design, Test Comput.*, vol. 11, no. 2, pp. 8–21, 1994.
[30] E. J. McCluskey, *Logic Design Principles with Emphasis on Testable Semicustom Circuits*. Englewood Cliffs, NJ: Prentice-Hall, 1986.
[31] K. S. Stevens, "Practical verification and synthesis of low latency asynchronous systems," Ph.D. dissertation, Dept. Comput. Sci., Univ. Calgary, Canada, Sept. 1994.
[32] K. Y. Yun and D. L. Dill, "Unifying synchronous/asynchronous state machine synthesis," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 1993, pp. 255–260.
[33] A. J. Martin, "Synthesis of asynchronous VLSI circuits," in *Formal Methods of VLSI Design*, J. Straunstrup, Ed. Amsterdam, The Netherlands: North Holland, 1990, ch. 6, pp. 237–283.
[34] K. Y. Yun, "Automatic synthesis of extended burst-mode circuits using generalized C-elements," in *Proc. European Design Automation Conf. (EURO-DAC)*, Sept. 1996, pp. 290–295.
[35] P. S. K. Siegel, "Automatic technology mapping for asynchronous designs," Ph.D. dissertation, Stanford Univ., Stanford, CA, Feb. 1995.
[36] P. A. Beerel, K. Y. Yun, and W. C. Chou, "Optimizing average-case delay in technology mapping of burst-mode circuits," in *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, Mar. 1996, pp. 244–260.
[37] K. Y. Yun, B. Lin, D. L. Dill, and S. Devadas, "Performance-driven synthesis of asynchronous controllers," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 1994, pp. 550–557.
[38] P. Kudva, G. Gopalakrishnan, H. Jacobson, and S. M. Nowick, "Synthesis of hazard-free customized CMOS complex-gate networks under multiple-input changes," in *Proc. ACM/IEEE Design Automation Conf.*, 1996, pp. 77–82.
[39] S. Chakraborty, D. L. Dill, K. Y. Yun, and K. Chang, "Timing analysis for extended burst-mode circuits," in *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, Apr. 1997, pp. 101–111.
[40] E. B. Eichelberger, "Hazard detection in combinational and sequential switching circuits," *IBM J. Res. Develop.*, vol. 9, pp. 90–99, Mar. 1965.
[41] J. Bredeson and P. Hulina, "Elimination of static and dynamic hazards for multiple input changes in combinational switching circuits," *Inform. Contr.*, vol. 20, no. 2, pp. 114–124, Mar. 1972.
[42] J. Bredeson, "Synthesis of multiple input change hazard-free combinational switching circuits without feedback," *Int. J. Electron. (GB)*, vol. 39, no. 6, pp. 615–624, Dec. 1975.
[43] 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.
[44] R. Rudell, "Logic synthesis for VLSI design," Tech. Rep. Memo. UCB/ERL M89/49, Univ. California, Berkeley, 1989.
[45] C.-J. Seger, "Models and algorithms for race analysis in asynchronous circuits," Research Report, Ph.D. dissertation, CS-88-22, Univ. Waterloo, Comput. Sci. Dept., Canada, May 1988.
[46] J. R. Burch, "Delay models for verifying speed-dependent asynchronous circuits," in *Proc. Int. Conf. Computer Design (ICCD)*, Oct. 1992, pp. 270–274.
[47] J. Beister, "A unified approach to combinational hazards," *IEEE Trans. Comput.*, vol. C-23, pp. 566–575, June 1974.
[48] K. W. James and K. Y. Yun, "Average-case optimized transistor-level technology mapping of extended burst-mode circuits," in *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, 1998, pp. 70–79.
[49] R. M. Fuhrer, B. Lin, and S. M. Nowick, "Symbolic hazard-free minimization and encoding of asynchronous finite state machines," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, 1995, pp. 604–611.

Kenneth Y. Yun (S'92–M'95) received the S.M. degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA.

He is currently an Assistant Professor in the Department of Electrical and Computer Engineering at University of California, San Diego. He had held design engineering positions at TRW and Hitachi for six years. His current research interests include the design, synthesis, analysis, and verification of mixed-timed VLSI circuits and systems: in particular, interface design methodologies and tools to facilitate ultra-high-speed communications between synchronous/asynchronous modules. He has been working with Intel Corp. as a primary consultant on the Asynchronous Instruction Decoder Project. He has organized ASYNC'98 as a program co-chair.

Dr. Yun is the recipient of a National Science Foundation CAREER award and a Hellman Faculty Fellowship. He has received the Charles E. Molnar Award for a paper that best bridges theory and practice of asynchronous circuits and systems at ASYNC'97 and a Best Paper Award at ICCD'98.

David L. Dill (M'90) received the S.B. degree in electrical engineering and computer science from the Massachusetts Institute of Technology (MIT), Cambridge, in 1979 and the M.S. and Ph.D. degrees from Carnegie-Mellon University, Pittsburgh, PA, in 1982 and 1987.

He is Associate Professor of Computer Science and, by courtesy, Electrical Engineering at Stanford University, Stanford, CA, where he has been on the faculty since 1987. His primary research interests relate to the theory and application of formal verification techniques to system designs, including hardware, protocols, and software. From July 1996 to September 1997 he was Chief Scientist of 0-In Design Automation.

Dr. Dill's Ph.D. dissertation, "Trace Theory for Automatic Hierarchical Verification of Speed Independent Circuits" was named as a Distinguished Dissertation by ACM and published as such by M.I.T. Press in 1988. He was the recipient of an Presidential Young Investigator Award from the National Science Foundation in 1988, and a Young Investigator Award from the Office of Naval Research in 1991. He has received Best Paper Awards at the International Conference on Computer Design in 1991 and the Design Automation Conference in 1993 and 1998.