Covering Conditions and Algorithms for the Synthesis of Speed-Independent Circuits

Peter A. Beerel Chris J. Myers Teresa H. Meng

Abstract— **This paper presents theory and algorithms for the synthesis of standard C-implementations of speed-independent circuits. These implementations are *block-level* circuits which may consist of atomic gates to perform complex functions in order to ensure hazard-freedom. First, we present boolean covering conditions that guarantee that the standard C-implementations operate correctly. Then, we present two algorithms that produce optimal solutions to the covering problem. The first algorithm is always applicable but does not complete on large circuits. The second algorithm, motivated by our observation that our covering problem can often be solved with a single cube, finds the optimal single-cube solution when such a solution exists. When applicable, the second algorithm is dramatically more efficient than the first, more general algorithm. We present results for benchmark specifications which indicate that our single-cube algorithm is applicable on most benchmark circuits and reduces run-times by over an order of magnitude. The block-level circuits generated by our algorithms are a good starting point for tools that perform technology mapping to obtain gate-level speed-independent circuits.**

Keywords— **Asynchronous circuits, speed-independence, automatic synthesis, standard C-implementation**

I. Introduction

As competitive asynchronous chips gain attention [16], [43], [45], asynchronous design is increasingly being considered as a practical and efficient design alternative. Asynchronous designs do not require a global clock for synchronization. Instead, synchronization is event-driven in that transitions on wires act to request the start of a computation and acknowledge its completion. By removing the global clock, asynchronous circuits have the advantages of absence of problems related to clock-skew, freedom from designing for worst-case delay, and automatic power-down of unused circuitry.

Speed-independent circuits are an attractive subclass of asynchronous circuits because they can tolerate delay variations resulting from variations in IC processing, temperature, and voltage. More precisely, these circuits work correctly regardless of the delays of individual gates, while assuming zero wire delays [36]. As a result, achieving speed-independence avoids the need for many timing assumptions and delay lines that can sometimes increase circuit area and delay and/or reduce circuit reliability. This insensitivity to variation in gate delays implies that speed-independent systems are also modular in that components within a speed-independent system can be replaced by faster components without needing to redesign any other part of the system. Moreover, speed-independent circuits can exhibit more concurrency than fundamental mode circuits [48], [39], [44] which require that inputs change only after the entire circuit is guaranteed to be stable. Speed-independent circuits can also be easily verified [3], [14] and have a testability advantage—they are self-checking with respect to a broad class of multiple output stuck-at-faults [2].

Traditionally, the synthesis of speed-independent circuits either required completion sensing networks or encoded inputs and outputs [1] which lead to slow and area inefficient designs. More recently, researchers proposed using complex-gates in which every specified output signal was implemented with a single, possibly very complicated, atomic gate [10], [34]. The reliability of such complex-gate circuits, however, can be low because unmodeled glitches (i.e., runt voltage pulses or *hazards*) within the complex-gates may cause circuit malfunction. This lack of reliability is especially problematic in standard-cell and programmable gate-array implementations in which complex-gates are usually implemented with a collection of standard logic cells. A more reliable approach is to synthesize gate-level implementations comprised of only basic limited-fanin gates that can be easily incorporated into standard-cell and gate-array libraries. Synthesizing limited-fanin basic-gate implementations, however, is challenging primarily because naive decompositions of complex or high-fanin gates can often introduce hazards into the circuits.

Martin and Burns faced similar problems when they use complex-gates to synthesize *quasi delay-insensitive circuits*, a family of circuits closely related to speed-independent circuits which are insensitive to delays on gates *and* an identified subset of the wires (referred to as *non-isochronic* forks). To address this problem, they add state variables to the specification in such a way as to simplify all complex-gates until all gates are small and exist in the gate library or can be reliably generated using a module generator [31]. Unfortunately, the addition of state variables often requires user intervention and some circuits require specialized gates which may not be suited for gate-array and standard-cell implementations. Nevertheless, this semi-automated method has been used to build many large custom designs including a sixteen-bit quasi delay-insensitive microprocessor [32], [43].

Kishinevsky and Varshavsky proved that a gate-level speed-independent implementation can always be found for a limited class of specifications (Chapter 5 of [47]). They considered *distributive* specifications of autonomous circuits [35] which have no inputs. They proved that all such

This research was supported in part by ARPA contract DABT63-91-K-0002, and by the Center for Integrated Systems, Stanford University.

Peter A. Beerel is affiliated with the EE-Systems Dept., University of Southern California, Los Angeles, CA 90089-2562.

Chris J. Myers is affiliated with the EE Dept., University of Utah, Salt Lake City, UT 84112

Teresa H.-Y. Meng is affiliated with the EE and CS Depts., Stanford University, Stanford, CA 94305

specifications can be implemented speed-independently using 2-input NAND gates. Their goal, however, was theoretical in nature and did not include practical considerations such as speed and area. As a result, their algorithm produces large, complex circuits because it unnecessarily adds many state variables to avoid hazards. In addition, since their circuits do not model inputs, their algorithm is restricted to circuits that do not exhibit conditional behavior modeled by input (environmental) choice.

Since then, there have been several works on synthesizing speed-independent circuits from specifications with choice. First, in a preliminary version of this work [4], we developed an algorithm to generate unlimited-fanin block-level speed-independent circuits from state graph (SG) specifications that can model input choice. Subsequently, K. Lin and C. Lin developed an algorithm transforming a free-choice signal transition graph (STG) [10] into an unlimited-fanin block-level speed-independent circuit [29]. In addition, Kondratyev et al. proposed a similar synthesis algorithm to our previous work [23]. The theory in [23] is ambiguous, allowing some hazardous circuits to be synthesized. Fortunately, this ambiguity can be resolved with an amendment [24], making their theory essentially equivalent to our results presented in [4]. Our original theory and algorithms provides the starting point for further extensions and improvements to both the synthesis of block-level implementations [21], [25] and the more recent works on the technology mapping of block-level implementations into gate-level realizations [12], [22].

This paper describes this underlying theory and presents algorithms for the block-level synthesis algorithm — the generation of a *standard C-implementation*. We show that this synthesis problem can be solved using a *binate covering algorithm*. The binate covering algorithm, however, is NP-complete. Consequently, straight-forward explicit-state implementations of the algorithm do not complete when applied to large circuits. This motivates the development of our *single-cube* binate covering algorithm which has significantly lower complexity than the general algorithm but targets only a subclass of circuits. We present run-time results of the resulting block-level circuits for numerous benchmark specifications, including those given in Berkeley's tool SIS [42]. The results show that the single-cube binate algorithm is applicable on most benchmark circuits and reduces run-times by over an order of magnitude on circuits. In fact, for some large circuits only the single-cube algorithm can successfully complete.

II. Background

This section describes our state graph (SG) specification model, the *standard C-implementation* block-level architecture, and our definition of a correct implementation of a given specification.

A. State graph

We specify circuits with a state graph (SG) which can be derived from one of many higher-level languages, including signal transition graphs [10] and CSP [38]. A SG is modeled by a tuple $\langle I, O, \Phi, \Gamma, s_0, \lambda \rangle$, where I is a set of input signals, O is a set of output signals, Φ is a set of states, $\Gamma \subseteq \Phi \times \Phi$ is a set of state transitions, s_0 is the initial state, and λ is a labeling function for states. When not ambiguous, we may refer to a state by its label. The union of input and output signals is denoted A_{Spec} and is called the set of *external signals*.

The labeling function λ labels each state $s \in \Phi$ with a bitvector over the external signals, i.e., $\lambda(s) \in \mathcal{B}^{A_{Spec}}$, where $\mathcal{B} = \{0, 1\}$. The value of a signal $u \in A_{Spec}$ in a state s, denoted $s(u)$, is the value of u in the label of s, i.e., $\lambda(s)(u)$. The function $bitcomp(s, u)$ returns the label formed from s by complementing the bit corresponding to u. For example, for the state s labeled [0000] in the SG in Figure 1, $bitcomp(s, a) = [1000]$.

A state graph must be strongly connected and each ordered pair $(s, s') \in \Gamma$ must differ in exactly one signal, i.e., $\lambda(s') = bitcomp(s, u)$ for some $u \in A_{Spec}$. When a state transition $(s, s') \in \Gamma$ and $s' = bitcomp(s, u)$, the notation $s \xrightarrow{u} s'$ may be used. A signal $u \in A_{Spec}$ is enabled in state s (denoted $enabled(u, s)$) if u can change in state s, that is, if there exists an $s' \in \Phi$ such that $s \xrightarrow{u} s'$ holds.

A SG has *complete state coding* (CSC) [10] if for every pair of states s and s' in Φ that have the same label ($\lambda(s) = \lambda(s')$), s and s' have the same output signals enabled, i.e.,

$$\forall u \in O \ \ [enabled(u, s) \Leftrightarrow enabled(u, s')]$$

Complete state coding is a necessary property for a SG to be implementable as a speed-independent circuit [10]. This property is necessary to ensure that derivation of the next-state logic for the output signals is possible. Adding state variables can transform an arbitrary SG into one that satisfies complete state coding [11], [19], [21], [27], [46]. We assume that such state assignments have been accomplished and deal only with state graphs that have the complete state coding property.

A signal u is disabled by a signal v, $v \neq u$, in a state s if u is enabled in state s and not enabled in s' where $s \xrightarrow{v} s'$. A SG is *determinate speed-independent* if in every state transition in the SG, output signals may not disable any signals (no output choice), i.e.,

$$\forall s, s' \in \Phi \ \ \forall v \in O \ \ \forall u \in I \cup O$$
$$\left[\left[s \xrightarrow{v} s' \ \wedge \ u \neq v \ \wedge enabled(u, s) \right] \Rightarrow enabled(u, s') \right],$$

and input signals may not disable output signals but may disable other input signals (input choice), i.e.,

$$\forall s, s' \in \Phi \ \ \forall v \in I \ \ \forall u \in O$$
$$\left[\left[s \xrightarrow{v} s' \ \wedge \ enabled(u, s) \right] \Rightarrow enabled(u, s') \right].$$

Notice that determinate speed-independent SGs can express a variety of behaviors, including OR-causality, but not arbitration (output choice).

This paper only deals with the synthesis of circuits from determinate speed-independent SGs. To handle speed-independent SGs that are not determinate (i.e., have output choice) we must design the logic associated with the

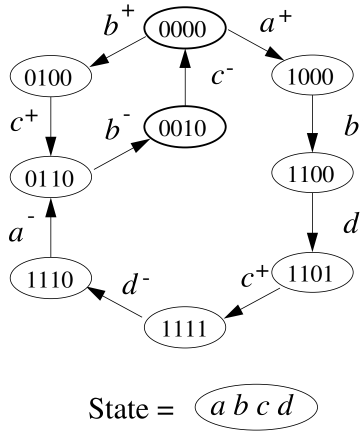

Fig. 1. A SG with input choice (motivated by the example in Figure 3(d) of [12]. The example has inputs $\{a, b, d\}$ and outputs $\{c\}$.

output signals exhibiting output choice manually (using some type of arbiter), re-label these output signals as inputs, and then, using the automated methodology described in this paper, synthesize the logic associated with the remaining output signals.

The SG in Figure 1 is determinate speed-independent. Formally, this SG is modeled by $\langle I, O, \Phi, \Gamma, s_0, \lambda\rangle$ where $I = \{a, b, d\}$, $O = \{c\}$, and $s_0 = [0000]$. There are 9 states in the set of states Φ including states [0000] and [0100]. There are 10 transitions in the set of transitions Γ including $[0000]\xrightarrow{a}[1000]$ and $[0000]\xrightarrow{b}[0100]$. These two state transitions illustrate input choice between a and b since when a fires, b is disabled and when b fires, a is disabled.

B. Folding the SG

A SG has *unique state coding* (USC) [10] if every state in the state graph has a unique label. Unlike complete state coding, unique state coding is not a necessary condition for synthesis. In this paper, however, we assume that the SG has unique state coding because this makes the concepts in this paper much easier to formalize and prove. Fortunately, a SG with complete state coding can be transformed into a SG with unique state coding by folding states with the same label into a single state. More precisely, to fold states, take every pair of states s and s' with the same label, remove s' from the set of states Φ, and replace s' with s in every place that s' appears in Γ. Also, if $s_0 = s'$, reassign s_0 to s. Notice that the SG depicted in Figure 1 already has unique state coding and thus does not require folding.

Note also that any specified sequence of state labels (or equivalently signal transitions) in the unfolded state graph is present in the folded state graph. Thus, intuitively, if any implementation operates properly for a folded state graph it will operate properly for the unfolded state graph. We formalize this notion of operate properly only for folded state graphs because the unique state coding property makes the formal definition significantly simpler (see Section II-D).

C. The Standard C-implementation

A circuit implementation is a tuple $\langle I, O, N, E, F\rangle$, where I is the set of input signals, O is the set of output signals, N is the set of internal signals, E is a set of connections between signals, and F is a set of gate functions. The union of inputs, outputs, and internal signals is denoted A_{Impl} and is also called the set of *circuit signals.* Each edge $e \in E$ represents a connection between circuit signals. An edge e is directed, connecting a source signal to a sink signal. The set of fanins of u, denoted $FI(u)$, is all sources of edges that have u as its sink. If u is an internal or output signal, $FI(u)$ is the set of inputs to the gate driving signal u. If u is an input, on the other hand, $FI(u) = \emptyset$.

We define an *implementation state* to model a snap-shot in time of all circuit signals. An *implementation state* is either a bitvector over A_{Impl}, i.e., $q \in \mathcal{B}^{A_{Impl}}$, or the special value, q_{fail}, that models the failure state of an implementation entered after the occurrence of a hazard. An implementation state of the circuit shown in Figure 2(a) is a Boolean vector that gives values to the state variables $[abcdegh]$. For example, in $q = [1110001]$, a, b, c, and h are at a logic high while d, e, and g are at a logic low. Each gate output signal u has an associated Boolean function $f \in F$ of arity $|FI(u)| + 1$. We refer to $f_u(q)$ as the *internal evaluation* of u in q. For $q \in \mathcal{B}^{A_{Impl}}$, it depends on the values of circuit signals in implementation state q and is defined to be $f_u(q(u), q(v_1), \ldots, q(v_r))$, where $FI(u) = \{v_1, \ldots, v_r\}$. For example, the function corresponding to signal g is $f_g = OR(e, d)$ and, in state $q = [1110001]$, $f_g(q) = 0$. For $q = q_{fail}$, on the other hand, we say $f_u(q)$ is unknown.

The theory and algorithms developed in this paper pertain to a restricted class of circuits whose structure is based on the standard C-implementation. In this framework, each output is driven by a *signal network* that consists of one two-input Muller C-element, two networks of combinational logic, as shown in Figure 2(b). The Muller C-element has a non-inverted input from the *set network* S_u and an inverted input from the *reset network* R_u. Its next state equation is $f_u(q) = (q(S_u) + \overline{q(R_u)}) \cdot q(u) + q(S_u) \cdot \overline{q(R_u)}$. In other words, when S_u is high and R_u is low, the signal u is driven high. When S_u is low and R_u is high, u is driven low. Otherwise, the signal u retains its old value.

To design these circuits, the state graph is first partitioned into a collection of excitation regions. An excitation region is a maximally connected set of states in which the output signal is both enabled and at a constant value. Excitation regions are divided into two types depending on the value of the output signal in the excitation region. If the value is 0, the excitation region is a *set region* since in all excitation region states the output signal is enabled to rise; otherwise the excitation region is a *reset region.* Both set and reset regions for a signal u are indexed with the variable k and the k^{th} set region of signal u is denoted $ER(u\uparrow, k)$. Similarly, the k^{th} reset region is denoted $ER(u\downarrow, k)$. For example, there are two set excitation regions for the signal c in Figure 1. The first, denoted $ER(c\uparrow, 1)$, is the set of states $\{[0100]\}$ and the second, denoted $ER(c\uparrow, 2)$, is the set of states $\{[1101]\}$.

For each excitation region $ER(u*, k)$, one region network is built which implements a *cover* of the excitation region denoted $C(u*, k)$. It is important to emphasize that in this paper we assume that the cover is implemented with an *atomic gate* which may be complex and have unlimited fanin. As illustrated in Figure 2(b), multiple set region networks are merged into the *set network* using a discrete OR gate. When only one set region network is needed, the OR gate is omitted. *Reset networks* are constructed in a similar fashion. The fact that some region networks may be implemented with a complex gate (i.e., an arbitrary "block" of logic that is assumed to be internally hazard-free) is the reason that we say our synthesis algorithm produces *block-level* circuits. The focus of this paper is to provide theory and algorithms to derive the cover of the region networks that ensures that the block-level circuits are hazard-free.

It is important to emphasize that the synthesized block-level circuits can be optimized. Further decomposition and logic optimization is typically done to obtain improved circuits that can be mapped into given gate libraries. The decomposition and optimization techniques involved, however, are outside of the scope of this paper (for more details see, e.g., [2], [9], [22]).

Important to finding the covers of region networks is the notion of a *quiescent region*. A maximally connected set of states in which an output signal u is not enabled is called a quiescent region of u. For each signal u in a determinate speed-independent SG, there exists at most one of its quiescent regions directly reachable from a given excitation region of u, but a quiescent region may be entered from multiple excitation regions of u. The quiescent region associated with the k^{th} excitation region is denoted $QR(u*, k)$. For example, the set regions $ER(c\uparrow, 1)$ and $ER(c\uparrow, 2)$ in Figure 1 share the same quiescent region $QR(c\uparrow, 1) = QR(c\uparrow, 2) = \{[1111], [1110], [0110]\}$.

Figure 2(a) depicts a standard C-implementation of the output c for the SG shown in Figure 1. The cover of the first set region $ER(c\uparrow, 1)$ is derived to be $\overline{a}\ b\ \overline{c}$ and is implemented with the complex gate *AND-N-1-3*(a, b, c). The cover of the second set region $ER(c\uparrow, 2)$ is derived to be d and implemented with a wire connected to the input d. The deriviation of region network covers is the focus of this paper. In particular, we develop a covering problem for each excitation region whose solutions constitute a hazard-free circuit (Section III) and provide efficient techniques for finding optimal covering solutions (Section IV).

Formally, the circuit is modeled by $\langle I, O, N, E, F\rangle$, where $I = \{a, b, d\}$, $O = \{c\}$. The internal signals $N = \{e, g, h\}$. The set of edges E include (a, e), (b, e), and (c, e) which correspond to the fanins of e. The functions $f \in F$ that we associate with each internal signal are $f_e = \textit{AND-N-1-3}(a, b, c)$, $f_g = \textit{OR}(d, e)$, $f_h = \textit{INV}(b)$, and $f_c = \textit{C-ELEMENT-N-1}(h, g)$. The circuit signals a, b, and d are input signals and thus have no associated gate function; their behavior is derived from the specification. Notice that the inverter bubbles are included in the complex gate function. For example, the complex gate *AND-N-1-3* is an AND gate whose first and third inputs are inverted.

D. Definition of Correctness

Informally, a correct speed-independent circuit is one whose behavior satisfies a given specification under all combinations of gate delays. We formalize this notion of satisfies with a definition of correctness of speed-independent circuits that is comprised of two parts: complex-gate equivalence which primarily deals with functional correctness and hazard-freedom which primarily deals with behavioral correctness, i.e., transient behavior.

D.1 Complex-gate equivalence

Intuitively, a circuit is complex-gate equivalent to its specification when *ignoring hazards* the circuit adheres to the specification. To model the notion of ignoring hazards, we analyze the implementation states in which all internal signals have settled and any transient hazards (glitches) have died down. We first define the notions of enabled, projection, and settled.

An internal signal is *enabled* in an implementation state q if u's value does not equal $f_u(q)$. For example, in state $q = [1110001]$, the internal signal g is enabled to fall because $q(g) = 1$ and $f_g(q) = 0$.

An implementation state q *projects* onto the specification state s, denoted $s = \textit{proj}(A_{Spec})(q)$, iff $s(u) = q(u)$ for all u in A_{Spec}. Because we restrict ourselves to specification SGs that satisfy USC, there exists at most one specification state which satisfies this property. Continuing with our example, implementation state [1110001] projects onto the specification state labeled [1110]. If no specification state satisfies this definition we say q projects onto a special specification state referred to as $s_{unknown}$. Such an implementation state can exist if, due to some bug in the circuit, an output signal fires when, according the specification, it is not supposed to fire. This is made more clear in the next section.

For each specification state $s \in \Phi$, there exists an implementation state $\textit{extend}(s)$, called an *implementation-state extension*, that projects onto s and in which no internal signals are enabled. More specifically, the value of signal u in $\textit{extend}(s)$ is called its *settled value* and is denoted $\textit{extend}(s)(u)$. The values of $\textit{extend}(s)(u)$ are unique and can be easily derived from the structure of the standard C-implementation. Consider first the settled value of the output u of a region network in $\textit{extend}(s)$. It equals one if and only if s is in the cover of the region network. The settled value of the output of the OR gate in a signal network in $\textit{extend}(s)$ equals the Boolean sum of the settled values of all region networks that are inputs to the OR gate. As an example, for the specification state [1110], $\textit{extend}(s) = [1110000]$ because [1110000] projects onto [1110] and because, in [1110000], all the internal signals e, g, and h are not enabled.

For each specification state s, the value that an output (or internal) signal is driven to in s is called the *external evaluation* of the signal in s, denoted $\textit{ext\_eval}(s)(u)$. The

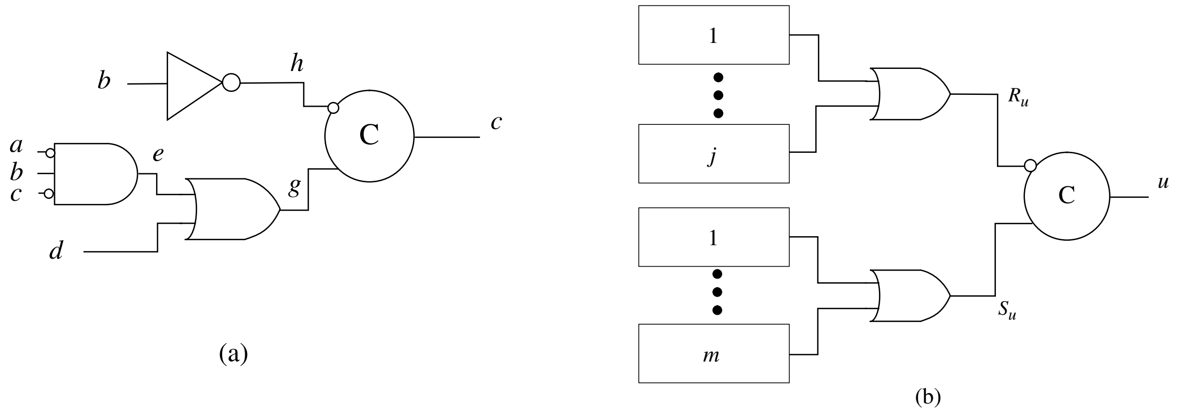

Fig. 2. (a) Standard C-implementation of output c for specification depicted in Figure 1. (b) Standard C-implementation framework for an output signal.

external evaluation of an output u in state s equals the local evaluation of u in the implementation-state extension of s. For example, the external evaluation of c in state [1110], $\mathit{ext\_eval}([1110])(c)$, equals $f_c(\mathit{extend}([1110])) = f_c([1110000]) = 1$.

A circuit is *complex-gate equivalent* to its specification when the external evaluation of all outputs agree with the specification, that is, if the external evaluation of each output differs from its current value in exactly those specification states in which it is enabled, i.e.,

$$\forall s \in \Phi \; \forall u \in O \, [[\mathit{ext\_eval}(s)(u) \neq s(u)] \Leftrightarrow \mathit{enabled}(u, s)] \, . \tag{1}$$

In our example circuit, the only specification states in which $s(c) \neq \mathit{ext\_eval}(s)(c)$ are [0010], [0110], and [1101]. Since these are exactly the states in which c is enabled, the circuit is complex-gate equivalent to the specification.

Note that complex-gate equivalence is similar to the notion *completeness with respect to specification* introduced by Ebergen [15] in that both ensure that the circuit can exhibit any specified behavior given the appropriate input choices and gate delays.

D.2 Hazard-freedom

If each output is built using a single atomic complex gate, then complex-gate equivalence is the only correctness criterion needed since under these conditions there are no hazards. However, this paper deals with block-level circuits which contain internal signals in which hazards can occur as a result of the added delay modeled within the circuit. Hence, the second part of our notion of correctness is *hazard-freedom.*

Hazard-freedom is a safety property of the actual behavior of a circuit implementation in a particular environment. The circuit and implementation's joint behavior is modeled using an *implementation state graph.* An implementation state graph is defined by $\langle Q, R, q_0 \rangle$, where Q is the set of reachable implementation states, R is a state transition relation, and q_0 is the initial state. As an example, the implementation state graph of our example circuit is depicted in Figure 3.

The initial state of the implementation q_0 is defined to be the implementation-state extension of the initial specification state s_0 (i.e., $q_0 = \mathit{extend}(s_0)$). For example, since $s_0 = [0000]$ in our example circuit, $q_0 = [0000001]$. This model is based on the assumption that after circuit power-up, the environment holds the external signals fixed until all internal signals have time to settle.

The transition relation R includes one transition for every enabled signal in every implementation state. In Section II-D.1, we defined that an internal signal is enabled if $f_u(q) \neq q(u)$. Here, we extend this definition to inputs and outputs. For outputs we use the same criterion, i.e., an output u is enabled in q if $f_u(q) \neq q(u)$. For an input u, on the other hand, u is enabled if u is enabled in $\mathit{proj}(q)(A_{Spec})$, the specification state on to which q projects. For example, in state $q = [1110010]$, a is enabled to fall since the input signal a is enabled to fall in [1110] (the specification state q projects onto). We also dictate that no signal is enabled in both the failure state q_{fail} and the special specification state $s_{unknown}$.

The destination states of these state transitions depend upon whether or not the transition is *hazard-free.* A transition associated with an enabled signal v in non-failure state q is defined to be hazard-free on u if the firing of v does not disable u. That is, $\mathit{hazard\text{-}free}(u, q, v)$ holds exactly when

$$[\mathit{enabled}(u, q) \Rightarrow \mathit{enabled}(u, \mathit{bitcomp}(v, q))] \, .$$

Note that we apply this definition only to internal and output signals, not to inputs which are allowed to disable other inputs. The state the circuit enters when an enabled signal fires depends on whether the transition causes a *hazard* on any gate output u (internal signal or output). If the firing of the signal causes a hazard at any gate output, then the state entered is defined to be the failure state q_{fail}. Otherwise, the destination state of the transition is $\mathit{bitcomp}(v, q)$. For example, there is a transition associated with g in state $q = [1110010]$ since g is enabled in this state. The transition is hazard-free on all signals since the only signal that does not maintain its enabled status during the transition is g itself.

Implementation state transitions in R are denoted by

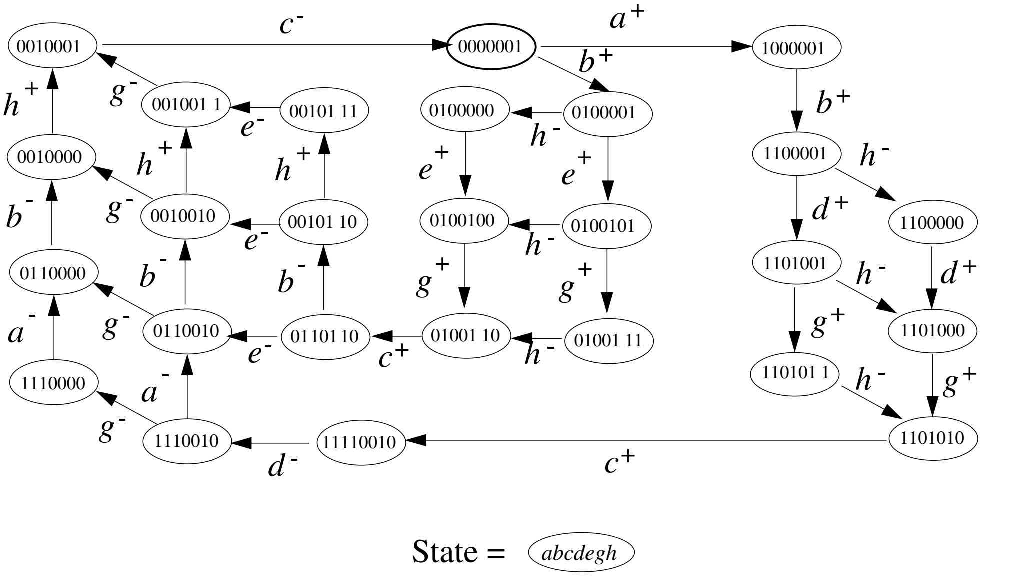

Fig. 3. The implementation SG describing the behavior of the circuit depicted in Figure 2(a) in the environment described by the specification SG depicted in Figure 1.

$q \xrightarrow{v} q'$, similar to specification state transitions. Thus, the transition described in the above paragraph is denoted $[1110010] \xrightarrow{g} [1110000]$. In addition, if the signal that changes, v, is not relevant, the notation $q \rightarrow q'$ may be used. This model assumes that any internal hazard in a speed-independent circuit can propagate to an output and hence cause a circuit malfunction. This assumption has been proven true for a large class of circuits [6]. For other circuits where this theory has not been proven, the assumption is conservative in that no hazardous circuit is considered hazard-free.

The set of reachable states of an implementation SG, denoted Q, can be recursively defined as follows:

$$extend(s_0) \in Q;$$

$$[\exists q \in Q \;\; [q \rightarrow q']] \;\Rightarrow\; [q' \in Q].$$

Intuitively, a circuit is hazard-free if no signal transition is ever disabled. We formalize this by saying that a circuit is hazard-free if the failure state is not reachable, i.e.,

$$q_{fail} \notin Q. \tag{2}$$

Since the implementation state graph of our example circuit does not contain q_{fail} it is hazard-free. Notice that hazard-freedom by itself is not a sufficient check for correctness because circuits that do not behave as specified may still be hazard-free.

III. Correct covers: theory

To ensure hazard-freedom of the block-level implementation, the covers of the region function must satisfy certain *correct cover* constraints. This section develops these constraints and proves that they guarantee our criteria for correctness: complex-gate equivalence and hazard-freedom.

A. Correct cover conditions

A cover is a *correct cover* if it satisfies two conditions. First, it must satisfy the *covering constraint* which says that the reachable states in the cover must include the entire excitation region but must not include any states outside of the union of the excitation and associated quiescent region, i.e.,

$$ER(u*, k) \subseteq [C(u*, k) \cap \Phi] \subseteq [ER(u*, k) \cup QR(u*, k)]. \tag{3}$$

Second, it must satisfy the *entrance constraint* which says that a correct cover must only be entered through excitation region states, i.e.,

$$[s \notin C(u*, k) \wedge s' \in C(u*, k) \wedge (s, s') \in \Gamma] \Rightarrow s' \in ER(u*, k). \tag{4}$$

The covering constraint guarantees that the circuit is complex-gate equivalent to the specification. Together the constraints guarantee that each region function is only allowed to turn on when it is actively trying to fire u. This guarantees that every transition of the region functions, the OR gates, and the C-element is hazard-free. It guarantees that no two inputs to the OR gates are simultaneously one, avoiding what has traditionally been called a delay hazard [1].

To illustrate the importance of the entrance constraint in correct covers, consider the cover and corresponding standard C-implementation for the output signal c shown in Figure 4. The cover {[0100], [0110], [0101], [0111]} (which includes unreachable states [0101] and [0111]) fails to satisfy the entrance constraint since the state [0110] (which is in the cover) can be reached from the state [1110] (which is not in the cover) and the state [0110] is not in the excitation region. As a result, the corresponding region function *AND-N-1*(a, b) can turn on and off without the AND gate firing. This can cause a glitch at the output of the AND

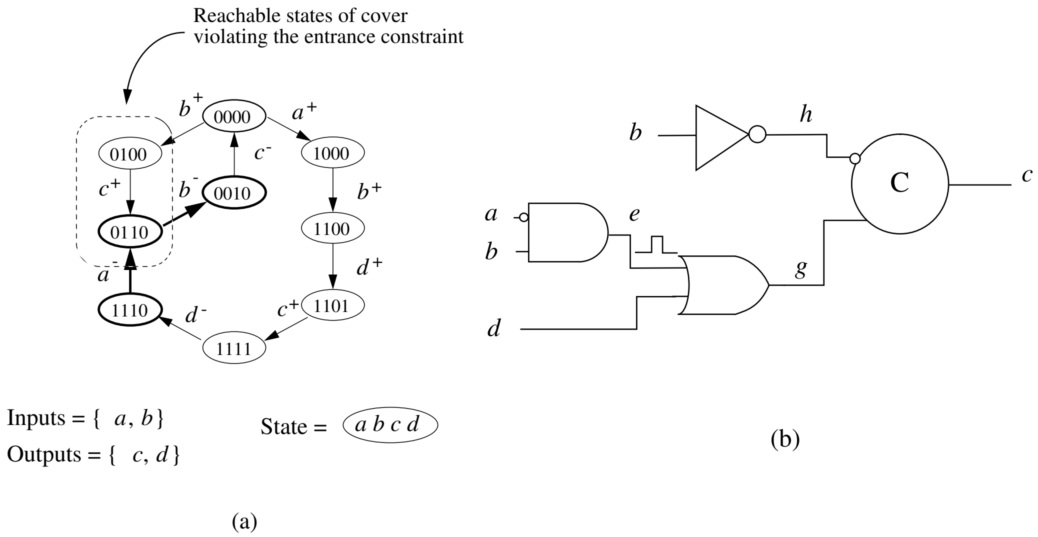

Fig. 4. (a) A cover violating the entrance constraint for a set excitation region of the signal c, and (b) the corresponding hazardous logic implementation.

gate which makes the circuit hazardous, as shown in Figure 4(b). Specifically, *AND-N-1*(a, b) can exhibit a runt positive pulse when the circuit goes through the sequence of states [1110] $\rightarrow$ [0110] $\rightarrow$ [0010] which is highlighted in Figure 4(a). Consequently, the circuit is hazardous and therefore not correct.

From a formal perspective, the existence of the hazard means that the implementation state graph describing the joint behavior of the circuit and its specification contains the failure state q_{fail}. Indeed, the portion of the implementation state graph depicted in Figure 5 contains many transitions to the failure state. For example, the transition b^- from state [0110000] is hazardous because it disables e^+, thereby creating the possibility of the runt pulse described in the above paragraph. Because of this hazard, the transition b^- from [0110000] leads to the failure state (as illustrated in Figure 5).

B. Proof that correct covers lead to correct circuits

This section presents a proof that our correct covers are sufficient to ensure that a standard C-implementation is a correct circuit. First, we prove that correct covers ensure standard C-implementations are complex-gate equivalent to their specification and then we prove hazard-freedom.

Lemma 1.1: If for all outputs $u \in O$ all region function covers $C(u*, k)$ are correct, then the standard C-implementation is complex-gate equivalent to its specification.

Proof: We prove the result using case analysis on the location of s. There are four possible cases.

Case 1: s is in a set region $ER(u\uparrow, k)$. Then, by the covering condition, s is in some set cover and is not in any reset cover. Thus, using the definition of settled value, we conclude that $extend(s)(S_u) = 1$ and $extend(s)(R_u) = 0$. From the next-state equation of the C-element, we conclude that $f_u(extend(s)) = 1$. Consequently, using the definition of external evaluation, we conclude that $ext\_eval(s)(u) = 1$. Since $s \in ER(u\uparrow, k)$ implies that $s(u) = 0$, we conclude that $ext\_eval(s)(u) \neq s(u)$ holds. Since $s \in ER(u\uparrow, k)$, u is enabled in s. Combining the last two conclusions, we have that $[ext\_eval(s)(u) \neq s(u)] \Leftrightarrow enabled(u, s)$ holds, and thus complex-gate equivalence is satisfied (see Eq. 1).

Case 2: s is in a set quiescent region $QR(u\uparrow, k)$. Then, by the covering condition, s is not in any reset cover. Thus, using the definition of settled value, $extend(s)(R_u) = 0$. Since $extend(s)(u) = s(u) = 1$, we conclude $f_u(extend(s)) = 1$. Since $f_u(extend(s)) = 1$, using the definition of external evaluation, we can also conclude that $ext\_eval(s)(u) = 1$. Putting the last two conclusions together we have that $ext\_eval(s)(u) \neq s(u)$ does not hold. Since $s \in QR(u\uparrow, k)$, u is not enabled in s. These last two conclusions mean that $[ext\_eval(s)(u) \neq s(u)] \Leftrightarrow enabled(u, s)$ holds, and thus that complex-gate equivalence is satisfied (see Eq. 1).

Case 3: s is in a reset region $ER(u\downarrow, k)$. Then, by the covering condition s is in some reset cover and is not in any set cover. Therefore, $extend(s)(S_u) = 0$ and $extend(s)(R_u) = 1$. Thus, $f_u(extend(s)) = 0$. Similar to case 1, we can conclude complex-gate equivalence is satisfied.

Case 4: s is in a reset quiescent region $QR(u\downarrow, k)$. Then, by the covering condition s is not in any set cover. Thus $extend(s)(S_u) = 0$. Since $extend(s)(u) = s(u) = 0$, we conclude $f_u(extend(s)) = 0$. Similar to case 2, we can conclude complex-gate equivalence is satisfied. ■

To show hazard-freedom, we first show that the covers in a set (reset) network are one-hot encoded.

Lemma 1.2: If all covers $C(u*, k)$ for the set (reset) regions of an output signal u are correct then their pair-wise intersections do not contain any states $s \in \Phi$.

Proof: (By contraposition) We show that if the pairwise intersections of two covers contain a state $s \in \Phi$ that the covers must not be not correct. Assume there exists two set (reset) covers $C(u*, i)$ and $C(u*, j)$ whose intersection contains a specification state s. Since excitation regions must be disjoint, we can conclude from the covering con-

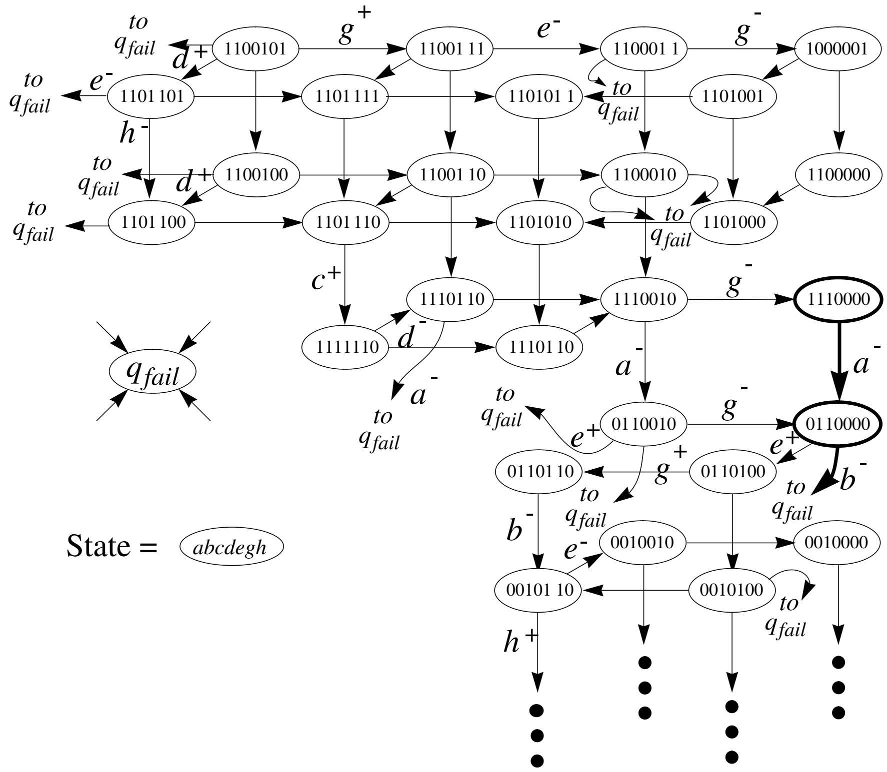

Fig. 5. A portion of the implementation state graph of the hazardous circuit depicted in Figure 4.

straint that $s \in QR(u*, i) \cap QR(u*, j)$. We do case analysis on s to show that in all cases the entrance constraint is violated and thus that the covers are not correct.

Case 1: There exists a path of states p contained in $C(u*, i)$ that originates from a state $s_i \in ER(u*, i)$ and ends at state s. Because of the covering constraint (Eq. 3), we know $ER(u*, i) \cap C(u*, j) = \emptyset$. Consequently, since $s_i \in ER(u*, i)$, we conclude that $s_i \notin C(u*, j)$. Summarizing, we know that the path p starts in the state s_i which is not in $C(u*, j)$ and ends in the state s which is in $C(u*, j)$. Thus, the path p must enter $C(u*, j)$. Let the state in which p enters $C(u*, j)$ be referred to as s'. We know s' cannot be in $ER(u*, j)$ because $s' \in p$, p is contained in $C(u*, i)$, and, by the covering constraint, $ER(u*, j) \cap C(u*, i) = \emptyset$. Consequently, by the covering constraint, we can conclude that s' must be in $QR(u*, j)$. This violates the entrance constraint (see Eq. 4).

Case 2: There does not exist a path of states p contained in $C(u*, i)$ that originates from a state $s_i \in ER(u*, i)$ and ends at state s. For this case, let L be the subset of states in $C(u*, i) \cap QR(u*, i)$ that are not reachable via paths that are contained in $C(u*, i)$ and originate from $ER(u*, i)$. Notice that L represents a subset of all quiescent region states through which the cover can be entered. In particular, it contains the subset of states that are not enterable through paths that originate from $ER(u*, i)$. Because the SG is strongly connected, there must exist some state $s' \in L$ that is directly reachable from a state s'' that is outside of L. Since $s'' \notin C(u*, i)$, s' violates the entrance constraint (see Eq. 4). ■

Traditionally, hazard-freedom (sometimes called speed-independence) is guaranteed when the transition of an output signal *acknowledges* that the circuit is stable and capable of accepting new inputs [35], [30]. To formalize this notion, we introduce the notions of a request path and its acknowledgement. Let $p = q_1, q_2, \ldots, q_n$ be a path of implementation states. Path p is called a *request path for signal v* if $f_v(q_j) = f_v(q_k) = b$ for all $n \geq j, k \geq 1$. In all these states v is being driven to the value b. We say the request path p is a *maximal* request path of v if p can be entered from and exited to states with a different internal evaluation of v. Thus, for the path p to be maximal, there must exist state transitions $q_0 \rightarrow q_1$ and $q_n \rightarrow q_{n+1}$ for which $f_v(q_0) \neq f_v(q_1)$ and $f_v(q_n) \neq f_v(q_{n+1})$. We say a request path p for signal v is *acknowledged* by a transition of an output signal u if it contains a state q_i from which there exists a transition $q_i \xrightarrow{u} q_j$ such that $q_j(v) = f_v(q_i)$. Thus, if a request path for v is acknowledged, the signal v is guaranteed to reach its internal evaluation and thus not be disabled. For example, in the implementation state graph of the hazard-free circuit, the path $p = [0010000] \xrightarrow{h} [0010001] \xrightarrow{c} [0000001]$ is a maximal request path for h for the following reasons. First, for all $q \in p$, $f_h(q) = 1$ and thus p is a request path for h to rise. Second, by letting $q_0 = [0110000]$ and $q_1 = [0010000]$, we have $q_0 \rightarrow q_1$ and $f_h(q_0) = 0 \neq f_h(q_1)$. Third, by letting $q_n = [0000001]$ and $q_{n+1} = [0100001]$, we have $q_n \rightarrow q_{n+1}$ and $f_h(q_{n+1}) = 0 \neq f_h(q_n)$. The last two facts mean that the request path is maximal. Moreover, this maximal request path is acknowledged by the output signal c since $[0010001] \xrightarrow{c} [0000001]$ and in $[0010001]$ $h = 1$.

The proof that our correct cover conditions lead to hazard-free implementations can be reduced to showing that all maximal request paths are acknowledged. An important part of this proof relies on the definition of the reachable implementation states Q and, in particular, the initial state q_0. Recall that q_0 is defined such that in it, no internal signals are enabled. This is an important assumption since the choice of the initial state can introduce hazards in an otherwise hazard-free circuit. For example, consider a standard C-implementation that satisfies the correct cover conditions. Consider an alternative definition of Q which would initialize the circuit in an implementation state q_0' in which in an OR gate of a signal network is enabled to rise because the output of a region network is one but enabled to fall. Then, there is a race between the region network falling and the OR gate rising. If the region network falls first, the OR gate is disabled. We say such a state is not *externally aligned* since in it an internal signal is at its external evaluation and also enabled. It can be shown that if any reachable state is not externally aligned, the circuit is hazardous [2]. Our definition of q_0 ensures that q_0 is externally aligned (because in it no internal signals are enabled).

Lemma 1.3: If for all outputs $u \in O$, all region network covers $C(u*, k)$ are correct, then all maximal request paths for all signals in the signal network of u are acknowledged by u and all the implementation's reachable states are externally aligned.

Proof: (Sketch, by induction on the set of reachable states)

Base Case: q_0 is hazard-free because it does not equal q_{fail}. It is externally aligned because, by definition, in it no internal signals are enabled.

Inductive Hypothesis: Consider the set of reachable states Q' reachable from the initial state q_0 in at most N state transitions. Any maximal request path for a signal v in this set is acknowledged by the firing of some output u. In addition, all these reachable states are externally aligned.

Inductive Step: We first show that any state transition from the states in Q' is hazard-free and that every new state reached is also externally aligned.

Consider a state $q \in Q'$. We show that every internal and output signal v is not disabled in any state transition from q using case analysis.

Case 1: Let v be the output of the k^{th} set (reset) region network for a signal u and assume v is enabled to fall in q. If q is not the last state in a maximal request path p for v, it cannot be disabled. If, on the other hand, q is the last state of a maximal request path p for v, then, by the definition of maximal, a transition from q to q' must be possible where $f_v(q') = 1$. This means that $s' \in C(u\uparrow, k)$ ($s' \in C(u\downarrow, k)$), where $s' = proj(A_{Spec})(q')$. Using the entrance constraint (Eq. 4), we can deduce that $s' \in ER(u\uparrow, k)$ ($s' \in ER(u\downarrow, k)$). Before the excitation region is entered, however, the output signal u must fall (rise). The only way u can fall (rise) is if v falls and the OR gate in the set (reset) network falls. After falling v is not enabled until the circuit enters state s'. Thus, v cannot be enabled in q, a contradiction. Consequently, v cannot be disabled in any state transition from q.

Case 2: Let v be the output of the OR gate in a set (reset) signal network that is enabled to fall in q. If q is not the last state in a maximal request path for v, it cannot be disabled. Consider next the case that q is the last state of a maximal request path p for v. In this case, because all states in p are externally aligned (by the inductive hypothesis), all states in the path project onto specification states not contained in any cover because, otherwise, the OR gate would be enabled and at its external evaluation (logic 1), violating external alignment. Consequently, the path must contain a transition in which u rises (falls). As in case 1, this means that v cannot be enabled in q and thus cannot be disabled.

Case 3: Let v be the output of the k^{th} set (reset) region network for a signal u and let v be enabled to rise. Then q must be part of a maximal request path for v to rise. Request paths for the network to rise project onto states $s \in C(u*, k)$. The request path must extend until reaching a state that projects onto $s' \notin C(u*, k)$. Because of the covering constraint (Eq. 3), this means that v is enabled to rise until after u rises (falls). Lemma 1.2 guarantees that in all states in $C(u*, j)$ the region network is the only network enabled high. Since u cannot rise (fall) unless one set (reset) region network rises and the associated OR gate rises, we conclude that u firing acknowledges the rising request paths of the region network. Consequently v cannot be enabled in q and thus cannot be disabled.

Case 4: Let v be the output of the OR gate in a set (reset) signal network that is enabled to rise. As in case 2, q must be part of a request path containing only externally aligned states. All such paths project onto states in the cover $C(u*, j)$ for some j. The region network will be enabled high until the circuit leaves the cover which, because of the covering constraint (Eq. 3), can only happen after u rises (falls). Since u cannot rise (fall) unless the OR gate rises, we conclude that u firing acknowledges the rising request paths of the region network, the OR gate is acknowledged. Consequently, u cannot be enabled in q and thus cannot be disabled.

Case 5: Let $v = u$ and let v be enabled to rise (fall) in q. Similar to cases 2 and 4, q must be part of maximal request path in which u rises and (falls) and this path must contain u firing. Consequently, u acknowledges the request path and cannot be disabled.

We now show that the next state entered is externally aligned by doing case analysis on the internal signals of the circuit.

Case 1. Region network outputs: Because no region network output is disabled, we can conclude that changes of inputs change a region network's internal evaluation only when the region network is settled. In addition, region networks only fire in the direction of their settled value. Thus, region networks in the next state entered must be externally aligned.

Case 2: OR gate outputs: Because the inputs to the OR gate are one-hot encoded and are hazard-free, they fire only when the OR gate is settled. In addition, the OR gate fires only in the direction of its settled value. Thus, the OR gate output in the next state is always externally aligned. ■

It may be useful to note that the hazardous state graph has many maximal request paths that are not acknowledged. For example, a (short) maximal request path for e is [0110000]. It can be entered by a falling from [1110000] enabling e to rise. It can be exited by b falling, thereby disabling e and driving the circuit into the failure state q_{fail}.

From the above results, we now prove our final theorem.

Theorem 1: If for all outputs $u \in O$ all region network covers $C(u*, k)$ are correct, then the standard C-implementation is correct.

Proof: We have proven complex-gate equivalence in Lemma 1.1 and hazard-freedom in Lemma 1.3. ■

C. Completeness of the theory

It is important to realize that the cover that includes only excitation region states is always a correct cover, meaning that a correct cover always exists. More formally,

Theorem 2: For all excitation regions $ER(u*, k)$ in a determinate SG satisfying USC a correct cover exists.

Proof: The cover $C(u*, k) = ER(u*, k)$ satisfies both the covering and entrance constraints. ■

Thus, for example, the cover $\overline{a}\ b\ \overline{c}\ \overline{d}$ is a correct cover for the excitation region $ER(c\uparrow, 1)$ because it includes only the states in $ER(c\uparrow, 1)$, i.e., the one state [0100]. The goal of our synthesis algorithms described in the next section is to find correct covers which have the lowest cost such as defined below.

IV. Algorithms

This section presents algorithms to solve the above covering problem to obtain an optimal region function for each excitation region. In general, a cover is implemented with a set of *cubes*. A cube is a set of *literals* which are either an external signal or its complement. First, we present a general algorithm that finds an implementation for each region function composed of the minimal number of cubes. It is often the case, however, that a region function can be implemented using only a single cube. For this case, we have developed a substantially more efficient algorithm which finds a single-cube implementation for each region function composed of the minimal number of literals.

While standard logic minimization techniques exist to find optimal covers [7], they do not guarantee hazard-free logic. In particular, they are not suited to solve our more constrained covering problem. To guarantee hazard-free logic, we must include the notion of an entrance constraint which requires that a correct cover can be entered only through excitation region states. The entrance constraint ensures that if a state in the quiescent region is covered then each of its predecessor states must also be covered. This implication leads to a *binate covering problem* [18].

A. General algorithm

The goal of the general algorithm is to find an optimal sum-of-products function for each region function that satisfies our definition of a correct cover. The sum-of-products cover consists of a disjunction of *implicants.* An implicant of an excitation region is a cube that may be part of a correct cover. In other words, a cube c is an implicant of an excitation region $ER(u*, k)$ if the set of reachable states covered by c is a subset of the states in the union of the excitation region and associated quiescent region, i.e.,

$$[c \cap \Phi] \subseteq [ER(u*, k) \cup QR(u*, k)].$$

A *prime implicant* of an excitation region is an implicant which is not contained by any other implicant of the excitation region. A sum-of-products cover is optimal if there exist no other cover with fewer implicants.

To capture the entrance constraint, each implicant c is said to have a corresponding set of *implied states* (denoted $IS(c)$). An implied state of a cube c is a state that is not covered by the implicant but due to the entrance constraint must be covered if the implicant is to be part of the cover. More precisely, a state s is an implied state of an implicant c for the excitation region $ER(u*, k)$ if it is not covered by c, and s is a predecessor of a state that is both covered by c and not in the excitation region, i.e.,

$$\begin{aligned} IS(c) = \{s \ \mid s \notin c \wedge \\ \exists s' \, [(s' \in c) \wedge (s, s') \in \Gamma) \wedge (s' \notin ER(u*, k))]\} \end{aligned}$$

It is important to note that an implicant may have implied states that are outside the excitation and quiescent regions and cannot be covered by any correct cover. If this implicant is the only prime implicant which covers some excitation region state, then the covering problem would need to be solved using some non-prime implicant.

For this reason, we introduce the notion of *candidate implicants.* An implicant is a candidate implicant if there exists no other implicant which properly contains it and has a subset of the implied states. In other words, c is a candidate implicant if there *does not exist* an implicant c' that satisfies the following two conditions:

$$\begin{aligned} c' &\supset c \\ IS(c') &\subseteq IS(c). \end{aligned}$$

Notice that prime implicants are always candidate implicants, but that a candidate implicant need not be prime.

As an example, consider the Karnaugh map depicted in Figure 6 describing the covering problem for $ER(c\uparrow, 1)$. The figure identifies two implicants $\overline{a}b$ and $\overline{a}b\overline{c}$, the former of which is prime. Because the implicant $\overline{a}b\overline{c}$ contains no quiescent region states it has no implied states. Because $\overline{a}b$ contains the quiescent region state [0110] which can be entered from [1110], it has [1110] as an implied state. $\overline{a}b\overline{c}$ is a candidate implicant because the only implicant that is larger than it is $\overline{a}b$ and $\overline{a}b$ does not have a subset of implied states.

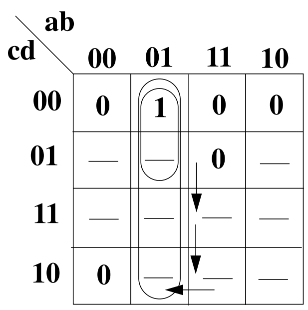

Fig. 6. A Karnaugh map illustration of the covering problem for $ER(c\uparrow, 1)$. The sole excitation region state [0100] is labeled "1". The unreachable states [0001], [0010], [0101], [0111], [1001], [1010] and the quiescent region states [1111], [1110], [0110] are labeled "-" . All other states are labeled with "0". In addition, the Karnaugh map is annotated with arrows that describe possible transitions into the quiescent region states. Two candidate implicants are illustrated, $\overline{a}b$ and $\overline{a}b\overline{c}$, the former of which is prime.

To find an optimal cover, we now prove it is sufficient to examine covers that consist of only candidate implicants.

Theorem 3: An optimal correct cover of a region function always exists that consists of only candidate implicants.

Proof: Consider the set of optimal covers that contain non-candidate implicants. If this set of covers is empty, then since some cover always exists, the set of all covers of the region function, which must include the optimal cover, must consist of only candidate implicants (thereby, proving the theorem statement). Otherwise, let C be the cover in this set that has the least number of literals. Let c be a non-candidate implicant in C. By definition of candidate implicants, there must exist some other implicant c' which properly contains c and has a subset of implied state. Let C' be the cover formed from C in which c' replaces c. C' is a correct cover because C is a correct cover, $c' \supset c$, and c' has a subset of the implied states of c. Since C' has less literals then C and C has the least number of literals of all covers containing a non-candidate implicant, C' must consist of only candidate implicants. ∎

Our covering problem is then formulated by creating a binary function in conjunctive (product-of-sums) form of candidate implicants to be satisfied with minimum cost. The binary function is defined over a set of Boolean variables l_i, one for each candidate implicant c_i. The variable l_i is *TRUE* if the cube c_i is included in the cover and *FALSE* otherwise. A conjunctive function over these variables is constructed up of two types of disjunctive clauses. This function is *TRUE* when the included cubes make up a correct cover.

First, a *covering clause* is included for each state s in the excitation region. Each clause consists of a disjunction of candidate implicants that cover s, i.e.,

$$\bigvee_{i:s\in c_i} l_i.$$

To satisfy the covering clause for each state s in $ER(u*, k)$, at least one l_i must be set to *TRUE*. This means that one cube that covers s must be included in the cover. It follows that the set of covering clauses for an excitation region guarantee all excitation region states are covered. Since all candidate implicants are guaranteed not to include states outside of the excitation and associated quiescent region, the cover is guaranteed to satisfy the covering constraint.

Second, for each candidate implicant c_i, a *closure clause* is included for each of its implied states $s \in IS(c_i)$. Each closure clause represents an implication that states if the Boolean variable associated with the cube c_i is true then the implied state s must be covered. To fit into a conjunctive form the implication is translated to the equivalent disjunction, i.e.,

$$\overline{l_i} \vee \bigvee_{j:s\in c_j} l_j.$$

A closure clause guarantees that if c_i is in the cover, some other cube must also be selected that covers the implied state s. These conditions together ensure that the cover satisfies the entrance constraint.

When both parts of the conjunctive function are satisfied, the corresponding cover is correct. Our goal is to find an assignment of Boolean variables that satisfies the function with the minimum cost. The cost function we minimize is the number of implicants, though, the number of literals can also be used. Since the implication introduces negated variables into the satisfiability product-of-sums framework, our optimization problem is a *binate covering problem.*

We now present an algorithm to find a cover using the minimum number of candidate implicants. First, the algorithm finds the prime implicants for each region function. Second, it uses this set to find all the candidate implicants. Then, it solves the binate covering problem represented here as a *covering and closure table* (or *CC table*) [17], using traditional reduction and branching techniques.

In order to find the set of prime implicants, our algorithm partitions the Boolean space into three sets, the on-set, the off-set, and the don't care-set. The on-set is composed of every state in the excitation region. The don't-care set is composed of every state in the associated quiescent region as well as every unreachable state. The off-set is composed of every other reachable state. The prime implicants are found using standard techniques [7]. For the $ER(c\uparrow, 1)$ region , six prime implicants are found: $\overline{a}b$, ac, bc, $\overline{a}d$, $\overline{b}d$, and cd.

Next, the algorithm expands the set of prime implicants to include all candidate implicants as described in [17]. The algorithm seeds the list of candidate implicants with the prime implicants, sorted by the number of literals in the implicant. Beginning with the candidate prime with the fewest number of literals, the algorithm considers all implicants extended with a literal not already used in the prime. If any new implicant satisfies the conditions given above then the algorithm inserts it into the list. Each subsequent implicant is considered in order until no new candidate implicants can be added. For the $ER(c\uparrow, 1)$ example, two new candidate implicants are found: $\overline{a}b\overline{c}$ is found by extending $\overline{a}b$ with the literal $\overline{c}$ and $a\overline{b}c$ is found by extending ac with the literal $\overline{b}$.

To solve the binate covering problem, a CC table is constructed to represent the conjunctive function described above. The table has one row for each candidate implicant and one column for each clause. The columns are divided into a *covering section* and a *closure section*, corresponding to covering and closure clauses. In the covering section, for each excitation region state s, a column exists containing a cross ($\times$) in every row corresponding to a candidate implicant that covers s. In the closure section, for each implied state s of each candidate implicant c_i, a column exists containing a dot ($\circ$) in the row corresponding to c_i and a cross in each row corresponding to a candidate implicant c_j that covers the implied state s.

As an example, the CC table for the excitation region $ER(c\uparrow, 1)$ in our example is depicted in Table I. The first column in the closure section is labeled with the state transition $[1110]\xrightarrow{a}[0110]$. Since $[0110]$ is an implied state of the candidate implicant $\overline{a}b$, the row corresponding to $\overline{a}b$ contains a circle. In addition, the column has crosses in the rows corresponding to the two candidate implicants that cover the implied states, ac and bc. Notice also that the table has three columns associated with the transition $[1101]\xrightarrow{c}[1111]$ corresponding to the three candidate implicants for which $[1101]$ is an implied state. These columns have no crosses in them because no candidate implicant exists which covers $[1101]$.

The CC table is solved using the reduction rules described in [17], which are listed here for convenience:

Rule 1: (Select essential rows) If a column contains only a single cross and blanks elsewhere, then the row with the cross must be selected. The row is deleted together with all columns in which it has crosses.

Rule 2: (Remove columns with only dots) If a column has only a single dot and blanks elsewhere, the row with the dot must be deleted together with all columns in which it has dots.

Rule 3: (Remove dominating columns) A column C_j dominates a column C_i if it has all the crosses and dots of C_i. If C_j dominates C_i, then C_j is deleted.

Rule 4: (Remove dominated rows) A row R_i dominates a row R_j if it (a) has all the crosses of R_j, and (b) for every column C_p in which R_i has a dot, either R_j has a dot in C_p or there exists a column C_q in which R_j has a dot, such that, disregarding the entries in rows R_i and R_j, C_p dominates C_q. If R_i dominates R_j, then R_j is deleted together with all columns in which it has dots.

Rule 5: (Remove rows with only dots) If a row only has dots, then the row is deleted together with all columns in which it has dots.

It is important to note that when applying Rule 4, two rows may mutually dominate each other. To break this tie, our algorithm removes the row corresponding to the implicant composed of the larger number of literals.

The table is completely solved when all columns are eliminated, and the resulting cover is the set of essential rows selected by Rule 1. In our limited experience, these reduction rules are usually sufficient to solve the table. For some cases, however, the reduction rules do not reduce the table completely, leaving a *cyclic table*. To solve the cyclic table, we use traditional branching techniques [33] in which case splitting is recursively performed on the inclusion of one of the remaining candidate implicants. The first time case splitting is applied, it replaces the original table with two new tables, one corresponding to including the chosen implicant in the cover and one corresponding to not including the chosen implicant. Both tables are reduced using the above reduction rules and case splitting is recursively applied on any remaining cyclic tables. In the worst case, this process generates an exponential number of tables, each of which may correspond to a possible covering solution. The process terminates by choosing the solution with the lowest cost. Since this exact procedure can sometimes be computationally impractical, our implementation includes a heuristic alternative in which it terminates after finding one solution.

TABLE I

THE CC TABLE FOR GENERAL COVERING OF $ER(c\uparrow, 1)$.

	CC Table				
	Covering	Closure			
	[0100]	$[1110]\xrightarrow{a}$ [0110]	$[1101]\xrightarrow{c}$ [1111]	$[1101]\xrightarrow{c}$ [1111]	$[1101]\xrightarrow{c}$ [1111]
$\overline{a}b$	$\times$	$\circ$			
$\overline{a}b\overline{c}$	$\times$				
ac		$\times$	$\circ$		
$a\overline{b}c$					
bc		$\times$		$\circ$	
$\overline{a}d$					
$\overline{b}d$					
cd					$\circ$

The reduction steps solve the table depicted in Table I as follows. First, the rows ac, bc, and cd along with the three columns associated with the implied state [1111] can be removed by Rule 2. Then, $\overline{a}b$, $a\overline{b}c$, $\overline{a}d$, and $\overline{b}d$ are dominated by row $\overline{a}b\overline{c}$ and can be removed along with the column ($\overline{a}b$, [0110]) by Rule 4. The remaining candidate implicant, $\overline{a}b\overline{c}$ is essential and is picked by Rule 1, solving the table. Note that in this case the table can only be solved by selecting an implicant that is not prime.

This example motivates one optimization. Prime implicants that cover only unreachable states need not be considered in the generation of the candidate implicants, since such candidate implicants are never part of an optimal cover. This optimization can make the initial CC table significantly smaller. For example, the prime implicants $\overline{b}d$ and $\overline{a}d$ only cover unreachable states. Since these implicants or any implicants contained in these implicants do not cover any excitation or quiescent region state, the rows in the table corresponding to these implicants have no crosses. Thus, these implicants cannot be an effective part of a cover and can instead be ignored (i.e., never generated).

B. Single-cube algorithm

The above binate covering formulation is often more general than needed since many region functions can be implemented with a single-cube cover. In this section we present a more efficient algorithm which finds an optimal single-cube cover, if one exists. Here, a single-cube cover is optimal if it has the least number of literals among all single-cube covers. This algorithm is derived from an algorithm used to synthesize complex-gate timed circuits [37] by adding the necessary closure constraints needed to handle gate-level hazards.

For a single-cube cover to hazard-freely implement a region function, all literals in the cube must correspond to signals that are *persistent*, i.e., constant throughout the excitation region (this is a slightly more general definition than the one in [10]). Otherwise, the single-cube cover would not cover all excitation region states. When a single-cube cover exists, an excitation region $ER(u*,k)$ can be sufficiently approximated using an *enabled cube* which is the supercube of the states in the excitation region, denoted $EC(u*,k)$, defined on each signal v as follows:

$$EC(u*,k)(v) \equiv \begin{cases} 0 & \text{if } \forall s \in ER(u*,k)\ [s(v)=0] \\ 1 & \text{if } \forall s \in ER(u*,k)\ [s(v)=1] \\ X & \text{otherwise} \end{cases}$$

If a signal is 0 or 1 in the enabled cube, it can be used in the cube implementing the region. A cube, such as the enabled cube, implicitly represents a set of states in the obvious way. The set of states implicitly represented by the enabled cube is always a superset of the set of excitation region states.

Each single-cube cover in the implementation is composed of *trigger signals* and *context signals*. For a given excitation region, a trigger signal is a signal whose firing can cause the circuit to enter the excitation region while any non-trigger signal which is stable in the excitation region can be a context signal. The set of trigger signals for an excitation region $ER(u*,k)$ can also be represented with a cube called a *trigger cube*, denoted $TC(u*,k)$, defined as follows for each signal v:

$$TC(u*,k)(v) \equiv \begin{cases} s'(v) & \text{If } \exists s,s' \left[(s \xrightarrow{v} s') \wedge \right. \\ & (s \notin ER(u*,k)) \wedge \\ & \left. (s' \in ER(u*,k)) \right] \\ X & \text{otherwise} \end{cases}$$

The intuition behind the single cube algorithm is that we try to get away with finding trigger cubes, introducing the minimal context signals necessary to ensure that the cube satisfies the covering and entrance constraints.

It is easy to show that in order for a single-cube cover to satisfy the covering constraint it must contain all its trigger signals. Since only persistent signals can be included in a single-cube cover, a necessary condition for a single-cube cover to exist is that all trigger signals be persistent. In other words, for a given excitation region $ER(u*,k)$, the trigger cube should contain the enabled cube (i.e., $TC(u*,k) \supseteq EC(u*,k)$).

TABLE II

Enabled cubes and trigger cubes for our example, where cube vector is $\langle a,b,c,d \rangle$

u,k	$EC(u*,k)$	$TC(u*,k)$
$c\uparrow,0$	1101	XXX1
$c\uparrow,1$	0100	X1XX
$c\downarrow,0$	0010	X0XX
$d\uparrow,0$	1100	X1XX
$d\downarrow,1$	1111	XX1X

The enabled cubes and trigger cubes are easily found with a single pass through the state graph. The enabled cubes and trigger cubes corresponding to all the excitation regions in our example are shown in Table II. Notice that every trigger signal is persistent and our algorithm proceeds to find the optimal single-cube cover.

The goal of the single-cube algorithm is to find a cube $C(u*,k)$ where $EC(u*,k) \subseteq C(u*,k) \subseteq TC(u*,k)$ such that it satisfies the covering and entrance constraints and is maximal. Our algorithm starts with a cube consists only of the trigger signals. If this cover contains no *violations*, i.e., states that violate either the covering or entrance constraint, we are done. This, however, is often not the case, and context signals must be added to the cube to remove any violating states. For each violation detected, the procedure determines the choices of context signals which would exclude the violating state. Finding the smallest set of context signals to resolve all violations is a covering problem. Due to the implication in the entrance constraint, inclusion of certain context signals may introduce additional violations which must be resolved. Therefore, the covering problem is again binate.

To solve this binate covering problem, we again create a CC table [17] for each region. There is a row in the CC table for each context signal, and there is a column for each violation and each violation that could potentially arise from the choice of a context signal. An entry in the table contains a cross ($\times$) if the context signal resolves the violation. An entry in the table contains a dot ($\circ$) if the inclusion of the context signal would require the violation to be resolved.

To construct the table for a given excitation region $ER(u*,k)$, the algorithm first finds all states in the initial cover which violate the covering constraint. In other words, a violation exists if a state s is (implicitly) contained by $TC(u*,k)$ but is not in the excitation or associated quiescent region. If a violation exists, the algorithm adds a new column to the table with a cross in each row corresponding to a context signal v that would exclude the violating state (i.e., $EC(u*,k)(v) = \overline{s(v)}$).

The next step in the table construction is to find all state transitions which violate the entrance constraint in the initial cover or may violate it due to a context signal choice. For any state transition $s \xrightarrow{v} s'$, this is possible when s is not in the excitation region (i.e., $s \notin EC(u*,k)$),

s' is in the quiescent region (i.e., $s' \in QR(u*,k)$), s' is in the initial cover (i.e., $s' \in TC(u*,k)$), and v excludes s (i.e., $EC(u*,k)(v) = \overline{s(v)}$). For each entrance violation detected, the algorithm adds a new column to the table again with a cross in each row corresponding to a context signal that would exclude the violating state. If the signal v in the state transition is a context signal, the state s' only needs to be excluded if v is included in the cover. This implication is represented with a dot being placed in the row corresponding to the signal v.

In a single pass through the state graph, all the CC tables can be constructed. Returning to our example, the CC table for the excitation region $ER(c\uparrow,1)$ is given in Table III. For this excitation region, the enabled cube is [0100] and b is its only trigger signal. The covering section includes states [1100] and [1101] because all other states are either in the excitation or quiescent region or are excluded by the trigger signal b. There are two closure columns. The first, corresponding to the transition $[1110]\xrightarrow{a}[0110]$, indicates that if a is included then state [0110] must be excluded. The only context signal that excludes this state is c. The second closure column corresponds to the transition $[1111]\xrightarrow{d}[1110]$ and is formed similarly. Note that the transition $[1101]\xrightarrow{c}[1111]$ does not have a column since $EC(c\uparrow,1)(c) = 0$ which does not exclude state [1101].

TABLE III
THE CC TABLE FOR SINGLE-CUBE COVERING OF $ER(c\uparrow,1)$.

	CC Table			
	Covering Section		Closure Section	
	[1100]	[1101]	$[1110]\xrightarrow{a}[0110]$	$[1111]\xrightarrow{d}[1110]$
a	×	×	∘	×
c			×	×
d		×		∘

When the construction of the CC table is successful, the table is solved using essentially the same reduction algorithm used in the general case outlined above. In this case, however, ties that occur in Rule 4 are resolved by picking the rule that provides symmetry between different regions of the same signal. This symmetry can often be exploited later during logic optimizations. Returning to our example, the table is solved as follows. First, row a is picked since it is an essential row (Rule 1), removing it as well as columns [1100], [1101], and $[1111]\xrightarrow{d}[1110]$ from the table. Since this removes a dot in column $[1110]\xrightarrow{a}[0110]$, this column is covered next. To accomplish this, row c is picked since it is an essential row (Rule 1), removing the column $[1110]\xrightarrow{a}[0110]$ solving the table. The resulting correct cover consisting of the single cube $\overline{a}b\overline{c}$. Notice that, as expected, this is the same result found by the general algorithm.

When a trigger signal is not persistent or when the CC table construction fails, we can use the more general algorithm described above to find a multi-cube cover. Alternatively, we can change the specification by constraining concurrency [34] or by adding state variables [23], [47], [5] such that a single-cube cover can be found. We note that these alternatives may not be possible *without changing the interface behavior of the circuit* (i.e., without constraining an input signal).

C. Complexity comparison

Although both the single-cube and general algorithm have exponential complexity with respect to the size of their tables, the complexity of the single-cube algorithm is much less than that of the general algorithm for two reasons.

First, the general algorithm must compute all prime and candidate prime implicants which are not needed in the single-cube algorithm. In particular, the number of prime implicants can be as many as $3^n/n$ [13] where n is the number of signals. To find the candidate implicants, it is necessary to expand each "don't care" with a '0' and a '1' and check to see if it is a new candidate implicant. The check requires that the potential candidate implicant is checked against each larger candidate implicant. The complexity of this test, therefore, is $O((3^n/n)^2)$.

Second, the sizes of the binate covering tables which must be solved are substantially larger in the general algorithm than in the single-cube algorithm. For the general algorithm, there needs to be one row for each candidate implicant (i.e., $O(3^n/n)$ rows) and one column for each excitation region state and for each implied state of a candidate implicant (i.e., $O(|\Phi|+|\Phi|\times 3^n/n)$ columns). For the single-cube algorithm, there needs to be one row only for each potential context signal (i.e., $O(n)$ rows) and a column for each violating state and state transition (i.e., $O(|\Phi|+|\Gamma|)$ columns). Thus, the CC tables for the general algorithm can be exponentially larger than in the single-cube algorithm. This can lead to dramatic differences in run-time since the worst-case complexity of solving the binate covering problem is exponential in the size of the table.

D. Run-time comparison

Both the general and single-cube covering algorithms described in this paper have been automated within the CAD tool `ATACS` using the well-known reduction and branching techniques [17]. The algorithms were tested on a large benchmark of circuits from academia and industry [26], [41]. The runtime results for both algorithms are shown in Table IV. The experiments were performed on a SPARC-station 20 with 128 MBytes of physical memory and 256 MBytes of virtual memory.

When applicable, the single-cube algorithm is consistently an order of magnitude faster. In two examples, the general algorithm took several hours to find the candidate primes, and exhausted the memory when it attempted to build the CC tables. In a third case, we terminated the general algorithm after it ran for more than 24 hours. There is no single-cube solution in 4 of the 27 circuits. For each of these circuits, the single-cube algorithm determined in a matter of microseconds that no single-cube cover exists. Fortunately, in these four cases, the general algorithm could be used to find a cover. Thus, during synthesis we always attempt to run the single-cube algorithm first. Only when it fails, do we apply the more general algorithm.

The literal count in all but one example is the same for the two algorithms. This one discrepancy is due to the fact that the reduction rules for the general algorithm optimized for number of cubes and not the number of literals. Note that we could easily extend the general algorithm to optimize the number of literals by casting it as a *weighted* binate covering problem at the cost of additional complexity. Since a difference in literal count occurred only in one example, our experimental results suggest that this extension is not critical and that the added complexity may not be justified.

We may be able to speedup solving of the binary covering problems by employing newer, more efficient algorithms [8], [20], [28], [40]. But since these algorithms do not change the inherent differences in the complexity of the covering problems, we expect that similar differences in run-time would exist.

TABLE IV
EXPERIMENTAL RESULTS FOR SPEED-INDEPENDENT BENCHMARKS. TIME VALUES ARE GIVEN IN SECONDS.

Examples	$\lvert\Phi\rvert$	$\lvert\Gamma\rvert$	Single-cube Lits	Single-cube Time	General Lits	General Time	CPU ratio
2demux	3200	12178	60	13.0	space		n/a
ebergen	18	22	18	0.05	18	0.77	15
etlatch	93	206	infeasible		21	3.04	n/a
false	12	16	infeasible		7	0.38	n/a
5fifo	2704	8304	70	29.0	time		n/a
full	16	24	8	0.04	8	0.38	10
hazard	12	14	10	0.04	10	0.36	9
hybridf	80	168	16	0.12	16	2.58	22
master-read	2108	7103	35	7.23	space		n/a
mp-forward-pkt	22	28	18	0.05	18	0.97	19
nak-pa	58	120	22	0.12	22	5.67	47
nowick	20	24	21	0.04	21	0.86	22
pe-rcv-ifc	54	76	78	0.21	78	6.96	33
pe-send-ifc	110	213	93	0.25	95	17.49	70
ram-read-sbuf	39	58	23	0.08	23	2.05	26
rlm	12	13	9	0.04	9	0.55	14
rpdft	22	22	19	0.04	19	0.54	14
sbuf-ram-write	64	114	24	0.14	24	5.97	43
sbuf-read-ctl	19	22	15	0.05	15	0.90	18
sbuf-send-ctl	27	32	33	0.06	33	1.79	30
sbuf-send-pkt2	26	34	27	0.06	27	1.24	21
trimos-send	336	888	infeasible		36	147.2	n/a
vbe4a	20	28	8	0.04	8	0.57	14
vbe5b	24	38	12	0.04	12	0.61	15
vbe5c	24	38	10	0.04	10	0.62	16
vbe10b	256	736	32	0.43	32	3.08	7
xyz	8	10	infeasible		10	0.50	n/a

V. Conclusion

We have presented new covering conditions and algorithms needed in the synthesis of standard-C implementations of speed-independent circuits. We have developed correctness conditions based on the ideas of complex-gate equivalence and hazard-freedom. We have proven that our covering conditions guarantee that the circuits produced are both complex-gate equivalent and hazard-free. We formulated our synthesis problem as a binate covering problem, and we described a general algorithm to solve this covering problem. Finally, we developed an efficient covering algorithm to find single-cube covers. We demonstrated that this algorithm is applicable in most of the standard benchmarks, and it can yield synthesis results over one order of magnitude faster. In addition, our results showed that the single cube algorithm could complete on a number of circuits that were too large for the general algorithm to handle.

References

[1] D. B. Armstrong, A. D. Friedman, and P. R. Menon. Design of asynchronous circuits assuming unbounded gate delays. *IEEE Transactions on Computers*, C-18(12):1110–1120, December 1969.

[2] P. A. Beerel. *CAD Tools for the Synthesis, Verification, and Testability of Robust Asynchronous Circuits*. PhD thesis, Stanford University, August 1994.

[3] P. A. Beerel, J. R. Burch, and T. H.-Y. Meng. Sufficient conditions for correct gate-level speed-independent circuits. In *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*, November 1994.

[4] P. A. Beerel and T. H.-Y. Meng. Automatic gate-level synthesis of speed-independent circuits. In *IEEE ICCAD* Digest of Technical Papers, pages 581–586, 1992.

[5] P. A. Beerel and T. H.-Y. Meng. Gate-level synthesis of speed-independent asynchronous control circuits, 1992. In collection of papers of the *ACM International Workshop on Timing Issues in the Specification of and Synthesis of Digital Systems.*

[6] P. A. Beerel and T. H.-Y. Meng. Semi-modularity and testability of speed-independent circuits. *INTEGRATION, the VLSI journal*, 13(3):301–322, September 1992.

[7] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic, 1984.

[8] R. K. Brayton and F. Somenzi. An exact minimizer for boolean relations. In *International Conference on Computer-Aided Design*, pages 316–320. IEEE Computer Society Press, 1989.

[9] S. M. Burns. General conditions for the decomposition of state holding elements. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems.* IEEE Computer Society Press, March 1996.

[10] T.-A. Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-theoretic Specifications.* PhD thesis, Massachusetts Institute of Technology, 1987.

[11] T.-A. Chu. Synthesis of hazard-free control circuits from asynchronous finite state machine specifications. *Journal of VLSI Signal Processing*, 7(1/2):61–84, February 1994.

[12] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Technology mapping of speed-independent circuits based on combinational decomposition and resynthesis. In *Proc. European Design and Test Conference*, 1997.

[13] G. De Micheli. *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, Inc., New York, New York, 1994.

[14] D. L. Dill. Trace theory for automatic hierarchical verification of speed-independent circuits. ACM Distinguished Dissertations, 1989.

[15] J. C. Ebergen. A verifier for network decompositions of command-based specifications. In *Proc. of the 26th Annual HICSS*, pages 310–318. IEEE Computer Society Press, 1993.

[16] S. B. Furber, P. Day, J. D. Garside, N. C. Paver, and J. V. Woods. A micropipelined ARM. In *VLSI '93*, 1993.

[17] A. Grasselli and F. Luccio. A method for minimizing the number of internal states in incompletely specified sequential networks. *IEEE Transactions on Electronic Computers*, pages 350–359, June 1965.

[18] A. Grasselli and F. Luccio. Some covering problems in switching theory. In G. Biorci, editor, *Network and Switching Theory*. Academic Press, 1966.

[19] J. Gu and R. Puri. Asynchronous circuit synthesis with boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, 14(8):961–973, August 1995.

[20] S. Jeong and F. Somenzi. A new algorithm for the binate covering problem and its application to the minimization of boolean relations. In *IEEE ICCAD Digest of Technical Papers*, pages 417–420, 1992.

[21] S. T. Jung, U. S. Park, J. S. Kim, and C. S. Jhon. Automatic synthesis of gate-level speed-independent control circuits from signal transition graphs. In *Proc. International Symposium on Circuits and Systems*, pages 1411–1414, 1995.

[22] A. Kondratyev, M. Kishinevsky, J. Cortadella, L. Lavagno, and A. Yakovlev. Technology mapping for speed-independent circuits: decomposition and resynthesis. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society Press, April 1997.

[23] A. Kondratyev, M. Kishinevsky, B. Lin, P.Vanbekbergen, and A. Yakovlev. Basic gate implementation of speed-independendent circuits. In *Proc. ACM/IEEE Design Automation Conference*, pages 56–62, June 1994.

[24] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev. *Private communication*, July 1993.

[25] A. Kondratyev, M. Kishinevsky, and A. Yakovlev. On hazard-free implementation of speed-independent circuits. In *Proceedings of the Asian South Pacific Design Automation Conference*, pages 241–248, 1995.

[26] L. Lavagno. *Synthesis and Testing of Bounded Wire Delay Asynchronous Circuits from Signal Transition Graphs*. PhD thesis, University of California, Berkeley, 1992.

[27] L. Lavagno, C. Moon, R. Brayton, and A. Sangiovanni-Vincentelli. Solving the state assignment problem for signal transition graphs. In *Proc. ACM/IEEE Design Automation Conference*, pages 568–572. IEEE Computer Society Press, June 1992.

[28] B. Lin, O. Coudert, and J. C. Madre. Symbolic prime generation for multiple-value functions. In *Design Automation Conference*, pages 40–44. ACM/IEEE, June 1990.

[29] K.-J. Lin, J.-W. Kuo, and C.-S. Lin. Direct synthesis of hazard-free asynchronous circuits from STGs based on lock relation and MG-decomposition approach. In *Proc. European Design and Test Conference (EDAC-ETC-EuroASIC)*, pages 178–183. IEEE Computer Society Press, 1994.

[30] A. J. Martin. Programming in VLSI: from communicating processes to delay-insensitive VLSI circuits. In C.A.R. Hoare, editor, *UT Year of Programming Institute on Concurrent Programming*. Addison-Wesley, 1990.

[31] A. J. Martin. *Private communication*, October 1994. A. J. Martin is a professor of computer science at Caltech.

[32] A. J. Martin, S. M. Burns, T. K. Lee, D. Borković, and P. J. Hazewindus. The design of an asynchronous microprocessor. In *Decennial Caltech Conference on VLSI*, pages 226–234, 1989.

[33] E. J. McCluskey. *Logic Design Principles with Emphasis on Testable Semicustom Circuits*. Prentice-Hall, Englewood Cliffs, NJ, 1986.

[34] T. H.-Y. Meng, R. W. Brodersen, and D. G. Messershmitt. Automatic synthesis of asynchronous circuits from high-level specifications. *IEEE Transactions on Computer-Aided Design*, 8(11):1185–1205, November 1989.

[35] R. E. Miller. *Switching Theory, Volume II: Sequential Circuits and Machines*. Wiley, New York, 1965.

[36] D. E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *Proceedings of an International Symposium of the Theory of Switching*, pages 204–243, 1959.

[37] C. J. Myers and T. H.-Y. Meng. Synthesis of timed asynchronous circuits. *IEEE Transactions on VLSI Systems*, 1(2):106–119, June 1993.

[38] C. J. Myers. *Computer-Aided Synthesis and Verification of Gate-Level Timed Circuits*. PhD thesis, Dept. of Elec. Eng., Stanford University, October 1995.

[39] S. M. Nowick. *Automatic Synthesis of Burst-Mode Asynchronous Controllers*. PhD thesis, Stanford University, Department of Computer Science, 1993.

[40] J. Rho, G. Hachtel, F. Somenzi, and R. Jacoby. Exact and heuristic algorithms for the minimization of incompletely specified state machines. *IEEE Transactions on Computer-Aided Design*, pages 167–177, February 1994.

[41] O. Roig, J. Cortadella, and E. Pastor. Hierarchical gate-level verification of speed-independent circuits. In *Asynchronous Design Methodologies*, pages 129–137. IEEE Computer Society Press, May 1995.

[42] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical Report UCB/ERL M92/41, University of California, Berkeley, May 1992.

[43] J. A. Tierno, A. J. Martin, D. Borković, and T. K. Lee. A 100-MIPS GaAs asynchronous microprocessor. *IEEE Design & Test of Computers*, 11(2):43–49, 1994.

[44] S. H. Unger. *Asynchronous Sequential Switching Circuits*. New York: Wiley-Interscience, 1969. (re-issued by R. E. Krieger, Malabar, 1983).

[45] C.H. (Kees) van Berkel, R. Burgess, J. Kessels, A. Peeters, M. Roncken, and F. Saeijs. A fully-asynchronous low-power error corrector for the digital compact cassette player. In *IEEE International Solid-State Circuits Conference*, 1994.

[46] P. Vanbekbergen, B. Lin, G. Goossens, and H. de Man. A generalized state assignment theory for transformations on signal transition graphs. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 112–117. IEEE Computer Society Press, November 1992.

[47] V. I. Varshavky, editor. *Self-Timed Control of Concurrent Processes*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1990.

[48] K. Y. Yun and D. L. Dill. Automatic synthesis of 3D asynchronous state machines. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 576–580. IEEE Computer Society Press, November 1992.

Peter A. Beerel received his B.S.E. degree in Electrical Engineering from Princeton University in 1989 and his M.S. and Ph.D. degrees in Electrical Engineering from Stanford University in 1991 and 1994, respectively. Since 1994, he has been an assistant professor in the EE-Systems Department of the University of Southern California. He was a member of the technical program committee of the Second Working Conference on Asynchronous Design Methodologies, the 1995 ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU'95), and the third International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC'97). He is now the Program Co-chair for ASYNC'98. His research interests include computer-aided-design of asynchronous and mixed asynchronous/synchronous VLSI systems as well as formal verification of communication protocols. Dr. Beerel is a co-winner of the Charles E. Molnar award for two papers published in ASYNC'97 that best bridged theory and practice of asynchronous system design. He is a recipient of an NSF Career Award and a 1995 Zumberge Fellow. He has also been a primary consultant for the Intel Corporation on their Asynchronous Instruction Decoder Project.

Chris J. Myers received his B.S. degree in Electrical Engineering and Chinese History in 1991 from the California Institute of Technology and his M.S.E.E. and Ph.D. degrees from Stanford University, in 1993 and 1995, respectively. He has been an Assistant Professor in the Department of Electrical Engineering at the University of Utah, Salt Lake City, UT, since 1995. In 1996, Dr. Myers received an NSF CAREER award. Recently, he was awarded a Center for Asynchronous Circuit and System Design by the State of Utah for which he serves as

Director. His current research interests are innovative architectures for high performance and low power, algorithms for the computer-aided analysis and design of real-time concurrent systems, formal verification, and asynchronous circuit design.

Teresa H. Meng joined the faculty of the Electrical Engineering Department at Stanford University in 1988, where she is an Associate Professor. Her current research activities include low-power circuit design, wireless communication, and portable DSP systems. Among the awards she has received are the IEEE Signal Processing Society's Paper Award in 1989, the 1989 NSF Presidential Young Investigator Award, the 1989 ONR Young Investigator Award, a 1989 IBM Faculty Development Award, and the 1988 Eli Jury Award from U.C. Berkeley for recognition of excellence in systems research. She received a B.S. degree from National Taiwan University in 1983 and an M.S. and Ph.D. from U.C. Berkeley in 1984 and 1988 respectively. She was co-program chair of the 1992 Application Specific Array Processor Conference and of the 1993 HOTCHIP Symposium. Dr. Meng also served as General Chair of the 1996 IEEE Workshop on VLSI Signal Processing.