Synthesis of Hazard-Free Multi-level Logic under Multiple-Input Changes from Binary Decision Diagrams

Bill Lin[a,*] Srinivas Devadas[b,†]

[a]IMEC Laboratory, B-3001 Leuven, Belgium, Email:billlin@imec.be
[b]Department of EECS, MIT, Cambridge, MA, Email:devadas@mit.edu

Abstract — *We describe a new method for directly synthesizing a hazard-free multilevel logic implementation from a given logic specification. The method is based on free/ordered Binary Decision Diagrams (BDD's), and is naturally applicable to multiple-output logic functions. Given an incompletely-specified (multiple-output) Boolean function, the method produces a multilevel logic network that is hazard-free for a specified set of multiple-input changes. We assume an arbitrary (unbounded) gate and wire delay model under a pure delay (PD) assumption, we permit multiple-input changes, and we consider both static and dynamic hazards. This problem is generally regarded as a difficult problem and it has important applications in the field of asynchronous design. The method has been automated and applied to a number of examples. The results we have obtained are very promising.*

1 Introduction

The design of correct asynchronous circuitry is a difficult task since an asynchronous circuit can malfunction (i.e. produce unexpected behavior) during execution if it is not free of *hazards*, which correspond to *undesired* glitches in a circuit. This is in contrast with synchronous design styles where the problem is avoided by the use of a global clocking scheme that coordinates and synchronizes all collective activities.

In this paper, we focus on a particular class of hazards — namely hazards in combinational logic. Hazard-free combinational logic is critical to the correctness of most asynchronous designs. Our goal in this work is to develop a method that can synthesize combinational logic that avoids *all* combinational hazards under a *specified* set of multiple-input changes. This is a general combinational synthesis problem which arises in many asynchronous sequential applications. For example, the problem arises in the current synthesis trajectories for asynchronous finite state machines [13, 17]. In this work, we assume that gates and wires can have arbitrary delays, which means we do not require bounded delay assumptions for correct operation or the use of delay elements to fix or filter out glitches. We also assume a *pure delay* (PD) model, which means we do not assume the presence of slow inertial delays to insure correctness.

The *two-level minimization version* of the problem has been addressed by a number of researchers in the past [15, 7, 2, 3]. More recently, Nowick [14] has developed an exact two-level minimizer that combines a number of previous ideas on this problem. A limitation of the two-level implementation approach is that it is not always possible to find a two-level cover that can insure freedom from *all* static and dynamic hazards even though a hazard-free multilevel implementation may exist.

In this paper, we describe a new framework based on Binary Decision Diagrams (BDD's) for *synthesizing a hazard-free multilevel logic implementation directly from a logic description.* A Binary decision diagram is a *directed acyclic graph* representation of Boolean function. BDD's have gained widespread use in the areas of formal verification and logic synthesis due to the canonical and easily manipulable nature of a class of BDD's [4]. Our framework is based on the use of both *free* as well as *ordered* BDD's and is naturally applicable to multiple-output logic functions. We permit multiple-input changes, and we consider both static and dynamic hazards, which means the resulting framework is general and powerful. In particular, we show that a multiplexor logic network derived from a *reduced* free or ordered BDD by replacing each node in the BDD by a two-input multiplexor is free of all static logic hazards. For dynamic logic hazards, we have developed the *Trigger Signal Ordering Requirement* (or *TSO-Requirement* for short) on the BDD variable ordering that, if satisfied, will lead to a multiplexor logic network that is also free of all dynamic logic hazards for the given set of allowable input transitions. The resulting multiplexor logic network is proved to be fully hazard-free under arbitrary gate and wire delays. While it is not always possible to generate hazard-free implementations using our technique, even if an implementation theoretically exists, in many cases we are able to generate hazard-free multilevel implementations when hazard-free two-level implementations cannot be found.

We have also developed *safe replacement strategies* that can replace a multiplexor by a functional equivalent *sum-of-products* representation which preserves the hazard-free properties. We provide a characterization on when such replacements are possible. The part of the network that can be safely replaced by AND- and OR- gates can be further optimized using *non-hazard-increasing* logic transformations, such as the ones discussed in [9].

Our combinational logic synthesis method can be applied directly to the synthesis of hazard-free logic for asynchronous state machines that operate under the fundamental mode assumption [13, 17]. Further, it can be generalized to the extended burst-mode state machine case [18]. We have automated our method and have applied it to a number of examples. The results we have obtained are very promising.

2 Background

2.1 Basic Definitions

A **transition cube** is a cube with a **start point** and an **end point**. Given input states A and B, the transition cube $[A, B]$ from A to B has start point A and end point B and contains all minterms that can be reached during a transition from A to B. It can be represented by the **smallest** cube that contains both A and B. The **open transition cube** (A, B) from A to B is defined as $[A, B] - B$. A **multiple-input change** or **input transition** from input state A to B is described by transition cube $[A, B]$. We will use the notation $A \Rightarrow B$ to denote the the input transition from A to B. Input variables are assumed to change simultaneously. Equivalently, since inputs may be skewed arbitrarily by wire delays, inputs can be assumed to change monotonically in any order and at any time. Once a multiple-input change occurs, no further input changes may

*Supported in part by the European Commission under the ESPRIT Project No. 6143 EXACT.

†Supported in part by the Defense Advanced Research Projects Agency under contract N00014-91-J-1698 and in part by a NSF Young Investigator Award with matching funds from Mitsubishi and IBM Corporation.

occur until the circuit has stabilized. An input transition from state A to B for a Boolean function f is a **static transition** if $f(A) = f(B)$; it is a **dynamic transition** if $f(A) \neq f(B)$. In the case of an incompletely specified function, we assume that f is fully defined for every specified static and dynamic transition; that is, for every $X \in [A, B]$, $f(X) \in \{0, 1\}$.

2.2 Circuit Delays

We assume gates and wires in a combinational circuit can have arbitrary finite delays. A pure delay model is assumed.

2.3 Function Hazards

A function f which does not change monotonically during an input transition is said to have a *function hazard* in the transition.

Definition 2.1 *A Boolean function f contains a* **static function hazard** *for input transition from A to C iff (1) $f(A) = f(C)$, and (2) there exists some state $B \in [A, C]$ such that $f(A) \neq f(B)$.*

Definition 2.2 *A Boolean function f contains a* **dynamic function hazard** *for input transition from A to D iff (1) $f(A) \neq f(D)$, and (2) there exists a pair of states B and C ($A \neq B, C \neq D$) such that $B \in [A, D]$ and $C \in [B, D]$ and $f(B) = f(D)$ and $f(A) = f(C)$.*

If a transition has a function hazard, *no* implementation of the function can avoid a glitch on the transition, assuming arbitrary gate and wire delays [7, 3].

2.4 Logic Hazards

Definition 2.3 *A combinational circuit for a function f contains a* **static logic hazard** *for the input transition from A to B iff (1) $f(A) = f(B)$, and (2) for some delay assignment, the circuit's output changes momentarily during the transition interval.*

Definition 2.4 *A combinational circuit for a function f contains a* **dynamic logic hazard** *for the input transition from A to B iff (1) $f(A) \neq f(B)$, and (2) for some delay assignment, the circuit's output is not monotonic during the transition interval.*

3 BDDs and Multiplexor Networks

3.1 Binary Decision Diagrams

In this section, we will restate from [4] the definitions for free Binary Decision Diagrams and reduced ordered Binary Decision Diagrams.

Definition 3.1 (BDD) *A* **Binary Decision Diagram** *is a rooted, directed acyclic graph with vertex set V containing two types of vertices. A* **non-terminal** *vertex v has as attributes an argument index $index(v) \in \{1, \ldots, n\}$ and two children $low(v), high(v) \in V$. A* **terminal** *vertex v has as attribute a value $value(v) \in \{0, 1\}$.*

The correspondence between BDD's and Boolean functions is defined as follows:

Definition 3.2 *A Binary Decision Diagram G having root vertex v denotes a function f_v defined recursively as: If v is a terminal vertex, then $f_v = value(v)$, which may be 0 or 1. If v is a non-terminal vertex with $index(v) = i$, then*

$$f_v(x_1, \ldots, x_n) = \overline{x_i} \cdot f_{low(v)}(x_1, \ldots, x_n) + x_i \cdot f_{high(v)}(x_1, \ldots, x_n)$$

x_i is called the **decision variable** *for vertex v.*

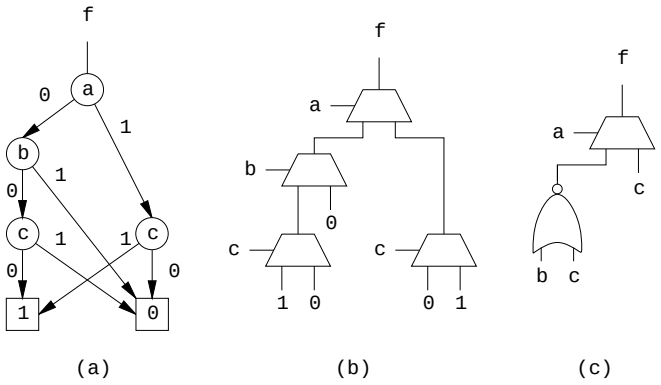

Figure 1: (a) A BDD. (b) The derived multiplexor multilevel network. (c) Simplification of multiplexors by constant propagation.

We require the following additional properties in Binary Decision Diagrams: (1) When traversing any path from a terminal vertex to the root vertex we can encounter each decision variable at most once. (2) A **reduced** BDD is one in which $low(v) \neq high(v)$ for any vertex v and no two subgraphs in the BDD are identical.

From Definition 3.1, a canonical form called a **reduced ordered Binary Decision Diagram** [4] (or simply ordered BDD) can be derived if the following restrictions are imposed: for any non-terminal vertex v, if $low(v)$ is also a non-terminal, then we must have $index(v) < index(low(v))$; and if $high(v)$ is also a non-terminal, then we must have $index(v) < index(high(v))$.

A **reduced free Binary Decision Diagram** (or simply free BDD) is a BDD where we require that we encounter each variable at most once in any path in the BDD and that the BDD is reduced, but do not require a strict variable ordering restrictions on BDD's. That is, different paths may have a different variable ordering as long as each variable is encountered at most once along any path.

3.2 Deriving a Multiplexor Network

A multilevel logic network can be derived directly from a BDD by replacing each BDD vertex with a *two-input* MUX-ELEMENT. An example is shown in Figure 1. A BDD and its corresponding derived multiplexor multilevel network are shown in Figure 1(a) and (b), respectively. The multiplexor network can be simplified by means of *constant propagation*. That is, the MUX-ELEMENTs can be replaced by simpler gates if one or more of its inputs is a constant. This propagation can be carried out topologically from inputs to outputs. The simplified network is shown in Figure 1(c).

4 Static Hazard-Free Synthesis

The hazard-free synthesis problem can be stated as follows. Given a (possibly incompletely specified) Boolean function f, and a set, T of *specified* function-hazard-free (both static and dynamic) input transitions of f, find a multilevel logic implementation that is free of logic hazards for every input transition $t \in T$.

In this paper, we propose synthesis procedures from BDD's that can produce hazard-free multilevel logic implementations. Let us first consider a simple procedure that transform an incompletely specified function f to a multiplexor network. If the function is incompletely specified, then some preprocessing is required as follows: in the case of an incompletely specified function, the *don't-care* minterms contained inside some specified transition $t \in T$ must be assigned properly so that no

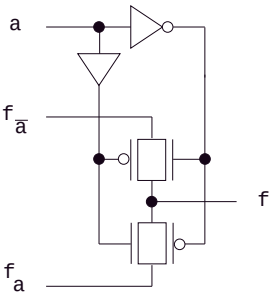

Figure 2: A transistor-level implementation of a hazard-free multiplexor.

functional hazards can occur. The other don't care minterms can be used for optimization, for example using techniques described in [6] (cf. the *restrict* and the *generalized cofactor* operators). So for all practical purposes, we only need to consider completely specified functions. Once this preprocessing step is performed, the synthesis procedure is as follows:

1. Construct a BDD G for the Boolean function f. The BDD here is meant to be either an *ordered* or a *free* reduced BDD, where each variable can appear at most once along any path.
2. Generate a multilevel circuit C by replacing each BDD node with a two-input MUX-ELEMENT.

For the hazard analysis in this section, we will first assume that the MUX-ELEMENT is an *atomic gate* with no internal hazards, and that the MUX-ELEMENT and the wires connecting them can have arbitrary delays. An implementation of a hazard-free multiplexor is shown in Figure 2. The only constraint on the layout of the gate is that the *difference* in the delays of the two paths from the control input a that pass through the buffer and the inverter should be smaller than the inertial delay corresponding to a transistor turning on or off.

The logical function implemented by the gate is $f = a \cdot f_a + \overline{a} \cdot f_{\overline{a}}$. This function is free of all dynamic hazards, but has a potential static logic hazard on the $0 \rightarrow 1$ transition on a with f_a and $f_{\overline{a}}$ constant at 1. However, if the path balancing criterion stated above is met, then the implementation of the MUX-ELEMENT will not have a static hazard.

We will first analyze *static hazard properties* of such networks assuming the MUX-ELEMENT as a basic hazard-free element. We will defer to Section 6 the discussion regarding the replacement of MUX-ELEMENTs with basic gates, the constant propagation issue, and possible simplification and resynthesis steps.

Theorem 4.1 (Static logic hazard-freeness) *C is free of all possible static hazards under any multiple-input change that does not correspond to a function hazard.*

Proof: Without loss of generality we will assume a single specified static transition $A \Rightarrow B$. The circuit C implements the Boolean function f which is free of function hazards for the specified input transition $A \Rightarrow B$. Further the circuit C has been derived using the synthesis procedure outlined.

Assume that the multiplexor driving the output of C has a as its control variable. The data inputs to the multiplexor corresponds to functions f_a and $f_{\overline{a}}$, the Shannon cofactors of f with respect to a and $\overline{a}$.

Assume that f is to make a static $1 \rightarrow 1$ transition, *i.e.* $f(A) = 1$ and $f(B) = 1$. We will first consider the case when the input a is at a constant 1. Clearly, if a is a 1, f will be free of static hazards if f_a remains at a constant 1 and is free of hazards. We know that $f_a(A) = 1$ and $f_a(B) = 1$. Further we know that $\forall m \in [A, B]$, $f_a(m) = 1$. Otherwise, it implies that there is a function hazard associated with f. Since f_a can only make a static transition in $A \Rightarrow B$, clearly f will be free of static hazards if f_a is free of static hazards. One can recursively apply the analysis above to f_a to show that it is free of static hazards. We will finally reach the base case where the control variable to the muliplexor is x and both the data inputs are constants. If both data inputs are the same, then this multiplexor will not exist in the BDD or the circuit C by the reduction rules of BDDs. Otherwise, this multiplexor reduces to either the literal function x or its negation $\overline{x}$. Then the input x is assumed to remain at constant 1 in the case of x and at constant 0 in the case of $\overline{x}$. Otherwise, there is a function hazard associated with f.

In the case when the input a is at a constant 0, then f will be free of static hazards if $f_{\overline{a}}$ is free of hazards. This follows from similar arguments as above.

Next consider the case when the input a makes a $0 \rightarrow 1$ transition or a $1 \rightarrow 0$ transition corresponding to $A \Rightarrow B$. Clearly f will be free of static hazards if both $f_{\overline{a}}$ and f_a are free of hazards. We claim that both $f_{\overline{a}}(A) = f_{\overline{a}}(B) = 1$ and $f_a(A) = f_a(B) = 1$. Further we claim that $\forall m \in [A, B]$, $f_{\overline{a}}(m) = f_a(m) = 1$. Therefore, both $f_{\overline{a}}$ and f_a can only make a static transition in $A \Rightarrow B$. Thus, it is sufficient to show that they are free of static hazards. Again, this argument can be recursively applied to $f_{\overline{a}}$ and f_a with the same base case as above. Since both $f_{\overline{a}}$ and f_a remain at constant 1 and are hazard free, only the control variable a can change at the multiplexor associated at the output of f. By the assumption that the multiplexor is an atomic gate and is internally hazard-free, then f is also free of static hazards for the static transition $[A, B]$.

The proof for the case when f makes a static $0 \rightarrow 0$ transition follows similarly. □

Theorem 4.1 states that the derived circuit is free of static hazards for *any input transition that does not cause a function hazard.* So we now say that a multiplexor implementation from either a free or an ordered BDD is free of all function hazards (by definition) and free of all static logic hazards. An important corollary is as follows.[1]

Corollary 4.1 *The static hazard-freeness of C is independent of the variable ordering chosen for the BDD G. Further, the BDD G can be a free BDD with different orderings along different paths.*

This means that there are *no* restrictions on the variable ordering for static hazards. This is however *not* always the case for dynamic logic hazards, as will be described next.

5 Dynamic Hazard-Free Synthesis

While a multiplexor implementation derived from a reduced BDD is guaranteed to be free of static logic hazards, it is not necessarily free of dynamic logic hazards. In this section, we will first characterize the problem. Then we will present a method that will ensure the non-existence of dynamic hazards as well.

5.1 The Problem

Let us consider an example shown in Figure 3. Let us consider the *dynamic* input transition

$$0 * 0 * 0 \Rightarrow 110, \text{ where } f(000) = 1 \text{ and } f(110) = 0.$$

[1] Due to space limitations, the proofs to some theorems and corollaries have been omitted. They can be found in [11].

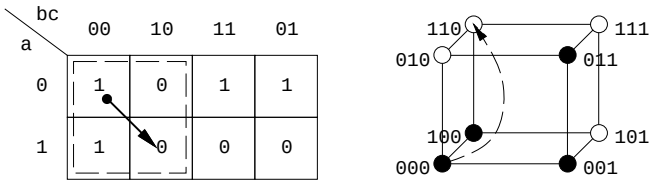

Figure 3: Dynamic-hazard example.

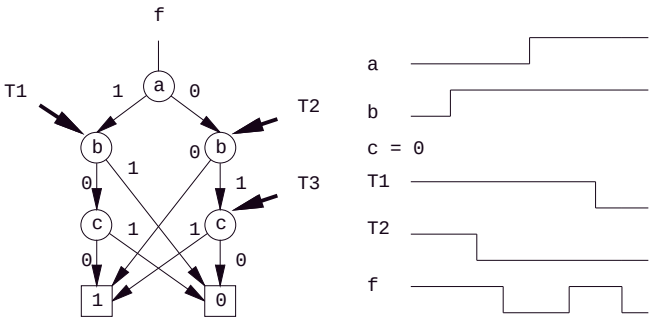

Figure 4: BDD implementation with ordering $a < b < c$.

We will use " $*$ " to indicate that the corresponding signal is *excited* to change. In this case, the signals a and b are *excited* to make the *transitions* $a+$ and $b+$. The corresponding *transition cube* is

$$[000, 110] = - - 0.$$

Now suppose we implement an ordered BDD with the variable ordering $a < b < c$. The corresponding BDD is shown in Figure 4.

Let us consider a multiplexor implementation translated from this BDD. This multiplexor implementation can exhibit a dynamic hazard as follows:

1. Initially, $a = 0$, $b = 0$, $c = 0$. This implies $T1 = 1$, $T2 = 1$, $T3 = 0$, and $F = 1$, where the Ti's are the output of the multiplexors and F is the output of the circuit.
2. In the transition $000 \Rightarrow 110$, both an $a+$ and a $b+$ can occur concurrently. Recall that under the unbounded gate/wire delay assumption, either $a+$ can occur first or $b+$ can occur first, but we must consider both transition orderings. Let us assume $b+$ occurs first and makes a $0 \rightarrow 1$ transition.
3. Then $T2$ makes a $1 \rightarrow 0$ transition, but $T1$ is *slow* to change. F makes a $1 \rightarrow 0$ transition.
4. Then let $a+$ happen, making a $0 \rightarrow 1$ transition, but $T1$ is still slow to change to 0, meaning it is still at value 1. This will cause F to change $0 \rightarrow 1$ back to 1.
5. Finally, $T1$ changes from $1 \rightarrow 0$. This causes F to change $1 \rightarrow 0$ back to 0. Thus, the transition sequence $1 \rightarrow 0 \rightarrow 1 \rightarrow 0$ has occurred on F, a dynamic hazard has been manifested.

However, when $a+$ occurs first, the dynamic transition takes place without any dynamic logic hazards. This is because when $a+$ occurs first, nothing changes. Then when $b+$ occurs, $T1$ will change, which will cause F to change, but F only changes once.

Now consider instead an alternative BDD implementation using variable orderings $b < a < c$ or $c < b < a$, shown in Figure 5 (a) and (b), respectively.

Let us first consider a multiplexor implementation translated from the BDD shown in Figure 5 (a). This multiplexor

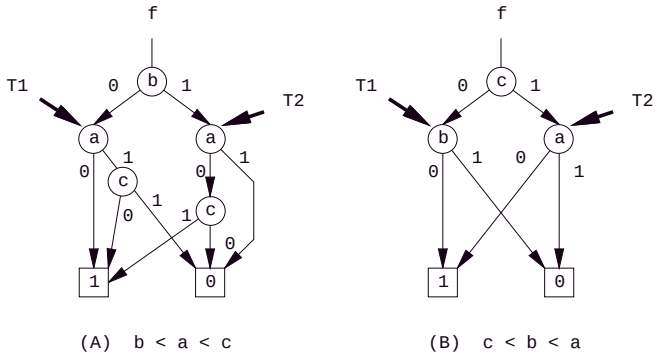

Figure 5: BDD implementation with orderings (a) $b < a < c$ and (b) $c < b < a$.

implementation is free of dynamic hazards under the transition $000 \Rightarrow 110$. The analysis is as follows.

1. Initially, $a = 0$, $b = 0$, $c = 0$. This implies $T1 = 1$, $T2 = 0$, and $F = 1$.
2. If $b+$ happens first, then F will change $1 \rightarrow 0$. Then when $a+$ occurs, nothing else changes. Hence there is no dynamic hazard.
3. If $a+$ happens first, nothing happens. Then when $b+$ occurs, F changes from $1 \rightarrow 0$. Again no dynamic hazard occurs.

Let us now consider a multiplexor implementation translated from the BDD shown in Figure 5 (b). This multiplexor implementation is free of dynamic hazards under the transition $000 \Rightarrow 110$. The analysis is as follows.

1. Initially, $a = 0$, $b = 0$, $c = 0$. This implies $T1 = 1$, $T2 = 1$, and $F = 1$.
2. If $b+$ happens first, then F will change $1 \rightarrow 0$. Then when $a+$ occurs, nothing else changes. Hence no dynamic hazard occurs.
3. If $a+$ happens first, nothing happens. Then when $b+$ occurs, F changes from $1 \rightarrow 0$. Again no dynamic hazard occurs.

From this informal introduction, we will show that the *variable ordering* in fact plays a very important role in guaranteeing freedom from dynamic hazards. Recall that we have already stated that BDD implementations are free of static hazards. The removal of dynamic hazards is addressed next.

5.2 Conditions for Dynamic Hazard-Freeness

In this section, we will consider the requirements on the BDD synthesis procedure in order to produce a multiplexor implementation free of dynamic logic hazards. We will first consider this requirement with respect to an *ordered* BDD implementation. We will defer to Section 5.4 the discussion regarding the employment of *free* or unordered BDD's to satisfy the same requirement.

The key to the analysis is the concept of **trigger signals**.

Definition 5.1 (Context signal) *Given an input transition $A \Rightarrow B$, a signal q is said to be a* **context** *signal if it changes its value across A and B. If it remains at a constant value in A and B, then it is said to be a* **non-context** *signal.*

By the definition of input transition (cf. Section 2.1), a context signal can only monotonically change once during a $A \Rightarrow B$ transition.

Definition 5.2 (Excited signal) *Given a state $X \in [A, B]$ in the input transition $A \Rightarrow B$, a signal q is said to be* **excited** *in X if and only if its value in X is equal to its value in A.*

Definition 5.3 (Quiescent signal) *Given a state $X \in [A,B]$ in the input transition $A \Rightarrow B$, a context signal q is said to be a* **quiescent signal** *in X if its value in X is equal to its value in B.*

In the example shown in Figure 3, signals a and b are "context" signals in the transition $000 \Rightarrow 110$ bcause both are enabled to change values, whereas c is a "non-context" signal in this transition. Signals a and b are "excited" in state 000 because both signals can change. In the state 100, only b is "excited"; the signal a is "quiescent" in state 100.

Definition 5.4 (Trigger state and signal) *A state $X \in [A,B]$ in a dynamic input transition $A \Rightarrow B$ is said to be a* **trigger state** *for $A \Rightarrow B$ if and only if there is an excited signal q ($q+$ or $q-$) such that the state $Y \in [A,B]$ reached by changing q has a different output value from X: i.e., $f(X) \neq f(Y)$.*

The signal q is called a **trigger signal** *of X in $A \Rightarrow B$, and the corresponding transition, either $q+$ or $q-$, is called a* **trigger transition** *of X in $A \Rightarrow B$. In a given trigger state, an excited signal that will not cause the output to change is referred to as a* **non-trigger signal**. *Its corresponding transition is referred to as a* **non-trigger transition**.

The set of maximally connected trigger states in a dynamic input transition is called a **trigger region**.

Referring again to Figure 3, states 000 and states 100 are "trigger states" since $f(000)$ and $f(100)$ are both equal to "1", but there exists a signal transition from either state that will cause the output to change to "0".

In the case of 000, both a and b are "excited" to change. Changing b will cause the output to change to "0". In this case, b is a "trigger signal" and $b+$ is a "trigger transition". Changing a first will not cause the output to change (it requires changing b also). In this case, a is a "non-trigger signal" and $a+$ is a "non-trigger transition".

Informally, the basic idea here is to construct a BDD such that "trigger signals" are **ordered before** "non-trigger signals". That is, for every trigger state for a *given* dynamic input transition $A \Rightarrow B$, the BDD variable ordering must be such that the trigger signals appear in the variable ordering before the non-trigger signals. This is formalized in the following requirement.

Definition 5.5 (Trigger signal ordering) *Given a function f, an ordered BDD G for f is said to satisfy the* **Trigger Signal Ordering (TSO-) Requirement** *for a dynamic input transition $A \Rightarrow B$ in T if and only if the following two conditions hold: (1) for every trigger state $X \in [A,B]$, the trigger signal variables in X appear in the variable ordering* **before** *the non-trigger signal variables; and (2) for every trigger state $X \in [A,B]$ with multiple trigger signals, the trigger signal variables in X all appear before each of the quiescent signal variables,* **or** *all appear after each of the quiescent signal variables. The BDD G is said to satisfy the TSO-requirement* **globally** *if and only if its variable ordering satisfies the TSO-requirement for every specified dynamic input transition.*

The second condition ensures that there is no quiescent signal "in between" any trigger signals during any specified transition.

If a strict variable ordering can be found that can satisfy the TSO-requirement globally, then the derived multiplexor network is also free of dynamic hazards.

Theorem 5.1 (Dynamic logic hazard-freedom) *C is free of dynamic hazards for all specified dynamic transitions.*

Proof: Without loss of generality we will assume a single specified dynamic transition $A \Rightarrow B$. The circuit C implements the Boolean function f which is free of function hazards for the specified input transition $A \Rightarrow B$. Further the circuit C has been derived using the synthesis procedure outlined.

Assume that the multiplexor driving the output of C has a as its control variable. The data inputs to the multiplexor correspond to functions f_a and $f_{\overline{a}}$, the Shannon cofactors of f with respect to a and $\overline{a}$.

Assume that f is to make a $0 \rightarrow 1$ transition, *i.e.* $f(A) = 0$ and $f(B) = 1$.

1. We will first consider the case when the input a is at a constant 1. Clearly, if a is a 1, f will be free of dynamic hazards if f_a is free of dynamic hazards.
2. If a is a constant 0, f will be free of dynamic hazards if $f_{\overline{a}}$ is free of dynamic hazards.
3. Next consider the case when the input a makes a $0 \rightarrow 1$ transition corresponding to $A \Rightarrow B$.

   (a) Consider the case when $f_a(A) = 0$ and $f_a(B) = 1$. We claim that $f_{\overline{a}}(A) = f_{\overline{a}}(B) = 0$. Suppose $f_{\overline{a}}(A) = 1$. Then, clearly, $f(A) \neq 0$. Therefore, $f_{\overline{a}}(A) = 0$. Suppose $f_{\overline{a}}(B) = 1$. There exists a cube $m \in [A,B]$ such that $f_{\overline{a}}(m) = 1$. Clearly the cube m does not contain the literal a or $\overline{a}$ since the cofactor $f_{\overline{a}}$ is not dependent on a. There are two possibilities. In the first case $f_a(m) = 0$. If there is such a cube, then we have a function hazard on f, on the path in the transition cube $[A,B]$ corresponding to $A \Rightarrow \overline{a} \cdot m \Rightarrow a \cdot m \Rightarrow B$, because $f(A) = 0$, $f(\overline{a} \cdot m) = 1$, $f(a \cdot m) = 0$, and $f(B) = 1$.
   The second case corresponds to $f_a(m) = 1$. Consider the path in the transition cube $A \Rightarrow \overline{a} \cdot m \Rightarrow a \cdot m \Rightarrow B$. We claim that a is a non-trigger signal in state A. If a is a trigger signal in state A, then when a goes $0 \rightarrow 1$ so does f. This means that $f_a(A) = 1$. Obviously since $f(B) = 1$ and a is making a $0 \rightarrow 1$ transition, $f_a(B) = 1$. Case b below corresponds to $f_a(A) = f_a(B) = 1$.
   Therefore, in state A the signal a is a non-trigger signal. If only a single signal, call it s, changes from A to $\overline{a} \cdot m$, then s is clearly a trigger signal in A. We clearly have a violation of Condition 1 of the TSO-requirement with a non-trigger signal a being before the trigger signal s in the ordering.
   Multiple signals could change from A to $\overline{a} \cdot m$. Without loss of generality consider the case where two signals s_1 and s_2 change from A to $\overline{a} \cdot m$. We have two paths corresponding to s_1 changing first and s_2 changing first. Denote these paths $A \Rightarrow \overline{a} \cdot m_1 \Rightarrow \overline{a} \cdot m$ and $A \Rightarrow \overline{a} \cdot m_2 \Rightarrow \overline{a} \cdot m$. If $f(\overline{a} \cdot m_1) = 1$ ($f(\overline{a} \cdot m_2) = 1$) then s_1 (s_2) is a trigger signal in state A. Since a is a non-trigger signal in state A we have violated Condition 1 of the TSO-requirement.
   Therefore, we require $f(\overline{a} \cdot m_1) = f(\overline{a} \cdot m_2) = 0$. If a is a non-trigger signal in either state $\overline{a} \cdot m_1$ or state $\overline{a} \cdot m_2$, then we again have a violation of Condition 1 of the TSO-requirement, since s_2 and s_1 are, respectively, trigger signals in states $\overline{a} \cdot m_1$ and $\overline{a} \cdot m_2$.
   Therefore, a is a trigger signal in both states $\overline{a} \cdot m_1$ and $\overline{a} \cdot m_2$. Now, in state $\overline{a} \cdot m_1$, we have two trigger signals, namely a and s_2 and a quiescent signal s_1. Similarly, in state $\overline{a} \cdot m_2$, we have two trigger signals, namely a and s_1 and a quiescent signal s_2. The orderings $a < s_1 < s_2$ or $a < s_2 < s_1$ will both cause a violation of Condition 2 of the TSO-requirement. (A quiescent signal appears in between two trigger signals.)
   In all cases, the ordering requirement imposed in the construction of C has been violated. Therefore $f_{\overline{a}}(A) =$

$f_{\overline{a}}(B) = 0$. Since $f_{\overline{a}}$ is itself a circuit derived from a BDD, by Theorem 4.1, $f_{\overline{a}}$ is free of static hazards and will stay at a steady 0 throughout $A \Rightarrow B$. $f_{\overline{a}}$ cannot have function hazards since that would imply a dynamic function hazard in f on $A \Rightarrow B$.

(b) Consider the case when $f_a(A) = f_a(B) = 1$. Since f_a itself is a circuit obtained from a BDD, it is free of static hazards by Theorem 4.1. Further, if f_a has a function hazard on $A \Rightarrow B$, then f would have a function hazard. Therefore f_a is function hazard-free on $A \Rightarrow B$. If $f_{\overline{a}}$ is constant at a 0, then by the above argument $f_{\overline{a}}$ would be free of static hazards as well. This means f would be free of dynamic and static hazards. If $f_{\overline{a}}$ makes a $0 \rightarrow 1$ transition on $A \Rightarrow B$ then f will be dynamic hazard-free if $f_{\overline{a}}$ is dynamic hazard-free. If either a makes a $0 \rightarrow 1$ transition or if $f_{\overline{a}}$ makes a $0 \rightarrow 1$ transition then f follows and stays at a 1.

Therefore in all cases, if either f_a or $f_{\overline{a}}$ is free of dynamic hazards in its $0 \rightarrow 1$ transition then f will be free of dynamic hazards.

4. A similar argument can be made for the $1 \rightarrow 0$ transition on a in $A \Rightarrow B$ to show that f is dynamic hazard-free if $f_{\overline{a}}$ or f_a is dynamic hazard-free.

For each of the four possibilities corresponding to a in $A \Rightarrow B$, we are guaranteed that if at most one of f_a or $f_{\overline{a}}$ is free of dynamic hazards then so is f. We also know that in each case the particular f_a or $f_{\overline{a}}$ will be free of function hazards on $A \Rightarrow B$. Further the trigger signal ordering requirement is a global imposition on C, and the change in f is caused by the particular f_a or $f_{\overline{a}}$ corresponding to each case. Therefore, one can apply the arguments above at any level in C. We will finally reach the primary inputs which are dynamic hazard-free by definition. Therefore, f is dynamic hazard-free.

The proof for the case when f makes a $1 \rightarrow 0$ transition follows similarly. □

As we have shown already that a BDD implementation is free of static logic hazards and is free of function hazards by the problem definition, then C derived using the above procedure is *fully hazard-free* for all hazards under the specified input transitions.

Theorem 5.2 (Complete hazard-freedom) *C is free of static and dynamic hazards, both function and logical, for all specified input transitions.*

Corollary 5.1 *A BDD-based circuit C under any ordering is free of hazards under all single-input changes.*

Corollary 5.2 *A Circuit C derived from a BDD G is hazard-free to all multiple-input dynamic transitions $A \Rightarrow B$ as long as either $\forall X \in (A, B]$, $f(X)$ is a constant or $\forall X \in [A, B)$, $f(X)$ is a constant.*

Note that this corollary shows that there are no ordering constraints generated on "burst-mode" transitions.

A systematic procedure for finding a variable ordering, if one exists, that satisfies the TSO-requirement is given in [11].

5.3 An Example

To illustrate the ideas, we have an example from Nowick [12] that was used to illustrate his two-level minimizer. The Karnaugh map of the example is shown in Figure 6. For this example, this is a set of four *specified* input transitions $T = \{t_1, t_2, t_3, t_4\}$. These transitions are:

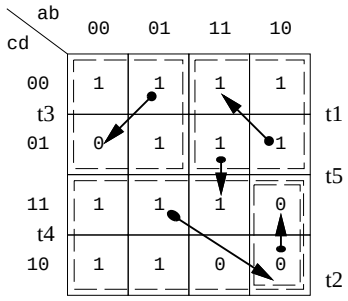

Figure 6: Another dynamic hazard example.

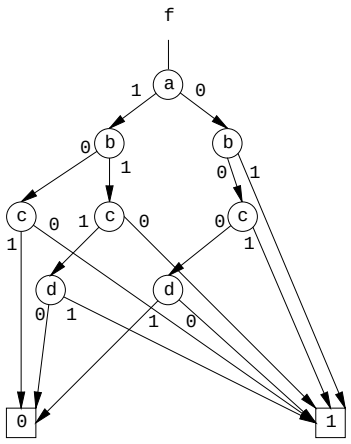

Figure 7: Dynamic hazard free BDD implementation with ordering $a < b < c < d$.

t_1	$10^*01^* \Rightarrow 1100$	1-1 static $\{b+, d-\}$ transition
t_2	$1010^* \Rightarrow 1011$	0-0 static $\{d+\}$ transition
t_3	$01^*00^* \Rightarrow 0001$	1-0 dynamic $\{b-, d+\}$ transition
t_4	$0^*1^*11^* \Rightarrow 1010$	1-0 dynamic $\{a+, b-, d-\}$ transition

The input transitions are indicated in Figure 6. The starting point of each transition is described by a dot, and its transition cube is described by a dotted circle.

From Theorem 4.1, the BDD implementation is free of static hazards, so transitions t_1 and t_2 will not cause a problem.

For dynamic transitions t_3 and t_4, we need to analyze the variable ordering requirements to guarantee that the BDD implementation is dynamic hazard-free for these two specified transitions. Indeed, the variable ordering $a < b < c < d$ will ensure the satisfaction of the TSO-requirements for dynamic transitions t_3 and t_4. Therefore, the resulting BDD implementation is also free of dynamic hazards. The BDD is shown in Figure 7.

It has been shown that it is not always possible to produce a hazard-free two-level SOP cover. For example, if we add the following specified transition to the above example (shown in Figure 6), then it can be shown that *no hazard-free two-level SOP cover exists* [14].

$$t_5 \quad 110^*1 \Rightarrow 1111 \quad \text{1-1 static } \{c+\} \text{ transition}$$

However, with the BDD approach, the above BDD implementation is also free of hazards for this transition (since it is a static transition), as well as the other specified transitions.

5.4 Synthesis from Free BDD's

Although in the example shown in Section 5.3 we can find a variable ordering that satisfied all ordering requirements, it is not always possible to find such a *strict* variable ordering in the general case.

It is possible to have *"cyclic"* ordering constraints that cannot be satisfied using a strict variable ordering. For example, it could be that in one specified dynamic input transition, x is required to appear before y; but in another specified dynamic input transition, y is required to appear before x. In this case, we have a cyclic constraint.

Cyclic constraints can frequently be resolved by using *free* BDD's where *different* variable orderings may be used along different paths. (Note that a free BDD still has the constraint of a variable appearing at most once along any path.)

Recall that each "path" in a reduced BDD corresponds to a "cube" in a disjoint cover. In this sense, a "path" **covers** a set of states contained in the cube. Intuitively, we can derive "local" ordering requirements for each specified dynamic input transition $A \Rightarrow B$ separately. Then we derive a free BDD where the ordering constraints are respected for each dynamic input transition $A \Rightarrow B$.

A procedure for finding a free BDD, if one exists, that satisfies the TSO-requirement is given in [11].

6 Replacement and Resynthesis

6.1 Replacement Circuits

It is worthwhile to replace the multiplexors with primitive gates so non-hazard-increasing logic transformations (e.g., [9]) can be applied on the network to further reduce the area or improve the performance.

Each MUX-ELEMENT $f = a \cdot f_a + \overline{a} \cdot f_{\overline{a}}$ in the synthesized circuit C will have the following conditions at its inputs by Theorem 5.1.

1. If the control input is constant at 1, f_a and $f_{\overline{a}}$ can *both* change $0 \rightarrow 1$ or $1 \rightarrow 0$.
2. If the control input is constant at 0, f_a and $f_{\overline{a}}$ can *both* change $0 \rightarrow 1$ or $1 \rightarrow 0$.
3. If a makes a transition, we have at most one of f_a or $f_{\overline{a}}$ making a transition.

We consider the characteristics of three primitive gate replacement circuits for the MUX-ELEMENT.

A: $f = a \cdot f_a + \overline{a} \cdot f_{\overline{a}}$
This circuit has a static logic hazard on the $0 \rightarrow 1$ transition on a with f_a and $f_{\overline{a}}$ constant at 1. It is free of all dynamic hazards.

B: $f = (a + f_{\overline{a}}) \cdot f_a + \overline{a} \cdot f_{\overline{a}}$
This is free of all static hazards but has dynamic hazards for the case where $a = 0$, and f_a and $f_{\overline{a}}$ make opposite polarity transitions.

C: $f = a \cdot f_a + (\overline{a} + f_a) \cdot f_{\overline{a}}$
This is free of all static hazards but has dynamic hazards for the case where $a = 1$, and f_a and $f_{\overline{a}}$ make opposite polarity transitions.

Our replacement strategy is as follows:

- If the MUX-ELEMENT has the conditions 1) and/or 2) at its inputs, but not 3), we use replacement circuit **A**.
- If the MUX-ELEMENT has the conditions 1) and/or 3) at its inputs, but not 2), we use replacement circuit **B**.
- If the MUX-ELEMENT has the conditions 2) and/or 3) at its inputs, but not 1), we use replacement circuit **C**.
- If the MUX-ELEMENT has the conditions 1), 2) and 3) at its inputs, we do not replace it.

In the majority of the cases, the multiplexor circuit can be transformed into one consisting entirely of primitive gates. The transformation depends on the input conditions at each multiplexor in the network. Note that if we are successful in replacing all multiplexors with the primitive gate implementation **A**, then we can convert the network into a disjoint two-level cover[2] that also satisfies the hazard freedom requirements. However, the two-level network may be considerably larger than the multiplexor-based network.

6.2 Constant Propagation

Since some of the MUX-ELEMENTs are connected to constant 0 and 1 values, they can be simplified. This simplification does not change the hazard characteristics of the circuit. After replacement, the primitive gate circuits can be simplified if they have constant inputs. For example the primitive gate circuit

$$f = (a + f_{\overline{a}}) \cdot f_a + \overline{a} \cdot f_{\overline{a}}$$

simplifies to

$$f = \overline{a} \cdot f_{\overline{a}}$$

if f_a is connected to logical 0.

6.3 Handling Cyclic Constraints Using Replacement

In some cases, when cyclic ordering constraints exist that cannot be satisfied even using a free BDD implementation it may be possible to simplify certain multiplexors in the circuit to produce a hazard-free realization.

Assume that the TSO ordering requirement produces a cyclic ordering graph. We discard a minimal number of constraints so as to produce an acyclic ordering graph. In particular, we discard constraints generated by Condition 2 of the TSO-requirement. Then, we generate a BDD and a multiplexor-based network using an ordering that satisfies the acyclic set of constraints. Of course, given that we have violated the TSO ordering requirement, the resulting network is not necessarily hazard-free. (It is possible that the resulting network obtained under the smaller set of constraints is hazard-free because the TSO-requirement is a sufficient, and not necessary, condition for hazard freedom.)

In some cases, it is possible to simplify multiplexors to make the network hazard-free. In particular, we check for the following cases:

1. A dynamic hazard is caused at a multiplexor because its control input a, and data inputs f_a and $f_{\overline{a}}$ all make $0 \rightarrow 1$ transitions. We check if the logical functions f_a and $f_{\overline{a}}$ are such that $f_{\overline{a}} = 1 \Rightarrow f_a = 1$. If so, we replace the multiplexor $f = a \cdot f_a + \overline{a} \cdot f_{\overline{a}}$ with $f = a \cdot f_a + f_{\overline{a}}$. This eliminates the dynamic hazard, since when $f_{\overline{a}}$ goes $0 \rightarrow 1$ the output f goes $0 \rightarrow 1$, and both a and f_a have to go $0 \rightarrow 1$ for the output to go $0 \rightarrow 1$.
2. A dynamic hazard is caused at a multiplexor because its control input a makes a $1 \rightarrow 0$ transition, and data inputs f_a and $f_{\overline{a}}$ make $0 \rightarrow 1$ transitions. We check if $f_a = 1 \Rightarrow f_{\overline{a}} = 1$. If so, we replace the multiplexor with $f = f_a + \overline{a} \cdot f_{\overline{a}}$. Again, this eliminates the dynamic hazard.
3. Same as Case 1 except that a, f_a and $f_{\overline{a}}$ make $1 \rightarrow 0$ transitions.
4. Same as Case 2 except that a makes a $0 \rightarrow 1$ transition, and f_a and $f_{\overline{a}}$ make $1 \rightarrow 0$ transitions.

7 Experimental Results

We have implemented the techniques described in this paper and have tested them on a number of examples.

[2] A disjoint cover is one in which each cube in the cover does not intersect any other cube.

name	in/out	two-level literals	multilevel literals
vanbekbergen	4/3	15	15
chu-opt	4/3	16	13
dme	5/3	16	12
dme-opt	5/3	20	14
dme-fast	5/3	15	16
dme-fast-opt	5/3	26	27
pe-send-ifc	7/3	87	65
scsi-ctrl	12/5	395	512
q42	5/3	27	24
binary-counter	8/7	110	80
binary-counter-co	9/8	122	88
cache-ctrl	36/20	1016	1379
tsend	16/9	504	583
tsend-bm	12/6	130	88
isend-bm	13/7	256	103
abcs	18/9	292	278
stetson-p1	30/17	542	754
stetson-p2	24/16	232	319

Table 1: Comparison between two-level and multilevel realizations when hazard-freedom is required under a specified set of multiple-input static and dynamic transitions.

We present two sets of results. The results in Table 1 correspond to a direct comparison with the two-level hazard-free synthesis procedure of [14]. Hazard-free two-level and BDD-based circuits were synthesized using the specified set of static and dynamic transitions for the benchmark examples given in [14]. For the BDD-based circuits, a hazard-free MUX-ELEMENT requiring four literals was assumed. The initial two-level circuits contained some very large fanin gates which were decomposed into two-input gates to report the literal counts. For example, the initial two-level circuit corresponding to **scsi-ctrl** has a 19-input OR gate. This decomposition did not alter the hazard properties of the circuits. The literal counts for the BDD-based realizations are comparable to the two-level realizations in most examples.

Our BDD-based realizations under any ordering are guaranteed to be hazard-free for *all* static transitions by Theorem 4.1. However, this is *not* true of the two-level realizations of Table 1. In order to make a two-level circuit hazard-free for static transitions, all the prime implicants of the logic function have to be included in the realization. This results in a substantially greater number of literals. A comparison between two-level and multilevel realizations when hazard-freedom is required for all static multiple-input change transitions is presented in Table 2. The last three examples in Table 2 are four-bit, eight-bit and sixteen-bit adder circuits. The BDD-based realization for circuits such as adders and comparators is very efficient and is exponentially smaller than the two-level realization. While there are other asynchronous synthesis trajectories, notably of the self-timed variety, that can implement circuits such as adders efficiently, we include Table 2 to indicate an extreme case in the comparison between BDD-based circuits and two-level circuits.

In other design scenarios, hazard-freedom is required under all single-input changes. By Corollary 5.1, BDD-based realizations (under any ordering) are hazard-free under *all* single-input dynamic transitions. In order to ensure hazard-freedom under all single-input dynamic transitions a significant number of additional prime implicants have to be added to the larger circuits of Table 1.

The CPU times required for BDD-based synthesis was on the order of a few seconds on a DECstation 5000.

name	in/out	primes	two-level literals	multilevel literals
vanbekbergen	4/3	6	15	15
chu-opt	4/3	4	16	13
dme	5/3	4	16	12
dme-opt	5/3	4	20	14
dme-fast	5/3	5	15	16
dme-fast-opt	5/3	8	26	27
pe-send-ifc	7/3	19	122	65
scsi-ctrl	12/5	147	1125	512
adder4	9/5	135	684	68
adder8	17/9	2519	21692	136
adder16	33/17	*	*	272

Table 2: Comparison between two-level and multilevel realizations when hazard-freedom is required under a specified set of multiple-input dynamic transitions and all multiple-input static transitions.

Acknowledgements — We would like to thank Alex Kondratyev, Ken Yun, and Dave Dill for providing detailed and constructive comments to an earlier draft of this work. We also would like to thank Steve Nowick and Peter Vanbekbergen for various helpful discussions on hazard-free synthesis problems.

References

[1] P.A. Beerel and T. Meng. Automatic gate-level synthesis of speed-independent circuits. In *ICCAD-1992*.

[2] J. Beister. A unified approach to combinational hazards. *IEEE Transactions on Computers*, C-23(6), 1974.

[3] J.G. Bredeson. Synthesis of multiple input-change hazard-free combinational switching circuits without feedback. *Int. J. Electronics*, 39(6):615–624, 1975.

[4] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[5] T.-A. Chu. Synthesis of self-timed VLSI circuits from graph-theoretic specifications. Technical Report MIT-LCS-TR-393, Massachusetts Institute of Technology, 1987.

[6] O. Coudert and J.C. Madre. A unified framework for the formal verification of sequential circuits. In *ICCAD-90*, pages 126–129, November 1990.

[7] E.B. Eichelberger. Hazard detection in combinational and sequential switching circuits. *IBM J. Res. Develop.*, 9(2):90–99, 1965.

[8] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev. On the conditions for gate-level speed-independence of asynchronous circuits. In *TAU-1993*.

[9] D.S. Kung. Hazard-non-increasing gate-level optimization algorithms. In *ICCAD-1992*.

[10] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli. Algorithms for synthesis of hazard-free asynchronous circuits. In *DAC-91*.

[11] B. Lin and S. Devadas. Synthesis of Hazard-Free Multi-level Logic Implementations under Multiple-Input Changes from Binary Decision Diagrams". In *IMEC Technical Report TR-VSDM-93-11.*, November, 1993. Revised May, 1994.

[12] S. Nowick, 1993. Private communication.

[13] S.M. Nowick and D.L. Dill. Automatic synthesis of locally-clocked asynchronous state machines. In *ICCAD-1991*.

[14] S.M. Nowick and D.L. Dill. Exact two-level minimization of hazard-free logic with multiple-input changes. In *ICCAD-1992*.

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

[16] P. Vanbekbergen, B. Lin, G. Goossens, and H. De Man. A generalized state assignment theory for transformations on signal transition graphs. In *ICCAD-1992*.

[17] K. Y. Yun and D. L. Dill. Unifying Asynchronous/Synchronous State Machine Synthesis. In *ICCAD-93*, pages 255–260, November 1993.

[18] K. Y. Yun, B. Lin, D. L. Dill, and S. Devadas. Performance-Driven Synthesis of Asynchronous Controllers. In *ICCAD-94*, November 1994.