Synthesis of Hazard-Free Multilevel Logic Under Multiple-Input Changes from Binary Decision Diagrams

Bill Lin, *Member, IEEE*, Srinivas Devadas, *Member, IEEE*

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 under the fundamental-mode assumption. Our framework is thus general and powerful. While it is not always possible to generate hazard-free implementations using our technique, we show that in some cases hazard-free multilevel implementations can be generated when hazard-free two-level representations cannot be found. 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.**

I. Introduction

Asynchronous design styles are becoming increasingly popular because they offer the potential benefits of improved system performance, avoidance of clocking problems, low-power operation, and modular design [8], [17], [18], [14], [26], [19], [21], [28], [2], [12], [15], [6], [27], [1]. However, 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 [21], [28]. 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 [25], [16], [10], [5], [3], [4], [11]. More recently, Nowick [22] 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 [7]. 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 [13].

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 as-

Manuscript received December 8, 1993; February 17, 1995. This paper was recommended by Associate Editor K. Keutzer.

B. Lin is with IMEC, B-3001 Leuven, Belgium.

S. Devadas is with the Department of EECS, MIT, Cambridge, MA 02139 USA.

IEEE Log Number 9412310.

sumption [21], [28]. Further, it can be generalized to the extended burst-mode state machine case [29]. We have automated our method and have applied it to a number of examples. The results we have obtained are very promising.

II. Background

A. Basic Definitions

To simplify the discussion, we will consider single-output functions only with binary input and output variables. Extension to multiple-output functions is straightforward.

Let $\{0,1\}^n$ be a Boolean space. Each $A \in \{0,1\}^n$, corresponding to a point in the Boolean space, is referred to as a **minterm**. It will also be referred to as an **input state** or simply **state**.

A **Boolean function**, f, of n variables, $x_1, x_2, \ldots, x_n$, is defined as a mapping: $f : \{0,1\}^n \rightarrow \{0,1,*\}$. The **ON-set** of a function is the set of minterms for which the function has value 1. The **OFF-set** is the set of minterms for which the function has value 0. The **DC-set** (don't-care set) is the set of minterms for which the function has the value $*$.

A cube of a Boolean function f is written as $c = [c_1, \cdots, c_n]$. For $1 \leq i \leq n$, c_i is 0 if variable x_i appears complemented in c, c_i is 1 if variable x_i appears uncomplemented in c, and c_i is $-$ if x_i does not appear in c. Thus, a cube is a set of minterms.

We will write $c \in d$, if cube c is such that for each position in c that has a 0 the corresponding position in cube d has a 0 or a $-$, and for each position in c that has a 1 the corresponding position in d has a 1 or a $-$.

The **intersection** of two cubes c and d is empty if there is a position i where $c_i = 0$ and $d_i = 1$ or vice versa. If the intersection is not empty, then it can be computed as a new cube $e = c \cap d$, where $e_i = 1$ if either $c_i = 1$ or $d_i = 1$, $e_i = 0$ if either $c_i = 0$ or $d_i = 0$, and $e_i = -$ otherwise.

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 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 $X \in [A, B]$, $f(X) \in \{0,1\}$.

B. Modeling Delays

We assume gates and wires in a combinational circuit can have arbitrary finite delays. Each gate is modelled as an instantaneous Boolean operator with a delay element attached to its output wire. This delay element describes the total gate delay. Each wire is modelled as a connection with an attached delay element. This delay element describes the total wire delay. The delays may have arbitrary but finite values. Since delay elements are attached only to wires, this model has been called the *unbounded wire delay model*. We assume a *pure delay* model, which means a pulse of any length can propagate. A *delay assignment* is an assignment of fixed finite delay values to every gate and wire in a circuit.

C. 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 1 (Static function hazard) *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 (Dynamic function hazard) *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
(a) $B \in [A, D]$ and $C \in [B, D]$ and
(b) $f(B) = f(D)$ 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 [10], [5]. Therefore, we consider only transitions which are free of function hazards (see [10], [4], [3]).

D. Logic Hazards

If f is free of function hazards for a transition from input A to B, it may still have hazards due to possible delays in the actual logic realization.

Definition 3 (Static logic hazards) *A combinational circuit for a function f contains a* **static logic hazard** *for the input transition from A to B iff:*
1. $f(A) = f(B)$
2. For some delay assignment, the circuit's output changes momentarily during the transition interval.

This means that we have one or more $0 \rightarrow 1 \rightarrow 0$ (or $1 \rightarrow 0 \rightarrow 1$) transitions while the specified behavior is a static 0 (or a static 1).

Definition 4 (Dynamic logic hazards) *A combinational circuit for a function f contains a* **dynamic logic hazard** *for the input transition from A to B iff:*
1. $f(A) \neq f(B)$

2. For some delay assignment, the circuit's output is not monotonic during the transition interval.

This means that we have a $0 \to 1 \to 0 \to 1$ (or $1 \to 0 \to 1 \to 0$) transitions while the specified behavior is a single $0 \to 1$ transition (or $1 \to 0$ transition).

III. Binary Decision Diagrams and Derived Multiplexor Networks

A. Binary Decision Diagrams

In this section, we will restate from [7] the definitions for free Binary Decision Diagrams and reduced ordered Binary Decision Diagrams. We will then indicate how a multiplexor-based multilevel logic network can be derived from them.

Given a Boolean function, the function resulting when some argument x_i of the function f is replaced by a constant $b \in \{0,1\}$ is called a **cofactor** of the function with respect to $x_i = b$, and this is denoted as $f|_{x_i=b}$. That is, for any arguments $x_1, \ldots, x_n$,

$$f|_{x_i=b}(x_1,\ldots,x_n) = f(x_1,\ldots,x_{i-1},b,x_{i+1},\ldots,x_n)$$

Using this notation, the **Shannon expansion** of a function with respect to a variable x_i is given by:

$$f = \overline{x_i} \cdot f|_{x_i=0} + x_i \cdot f|_{x_i=1}$$

Definition 5 (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 6 *A Binary Decision Diagram G having root vertex v denotes a function f_v defined recursively as:*
1. If v is a terminal vertex:
(a) If $value(v) = 1$, then $f_v = 1$.
(b) If $value(v) = 0$, then $f_v = 0$.
2. If v is a non-terminal vertex with $index(v) = i$, then f_v is the function:

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

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

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 5, a canonical form called a **reduced ordered Binary Decision Diagram** [7] (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.

B. Deriving a Multilevel Multiplexor Logic 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).

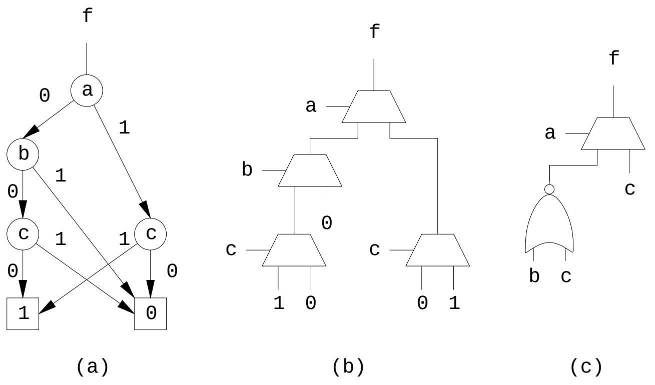

Fig. 1. (a) A BDD. (b) The derived multiplexor multilevel network. (c) Simplification of multiplexors by constant propagation.

IV. Static Hazard-Free Synthesis from BDD's

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 functional hazards can occur. The other don't care minterms can be used for optimization, for example using techniques described in [9], [24] (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.

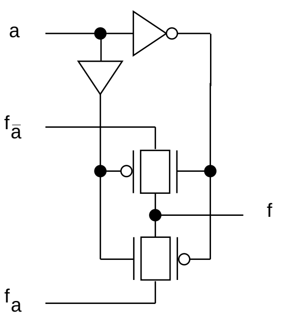

Fig. 2. A transistor-level implementation of a hazard-free multiplexor.

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 VI the discussion regarding the replacement of MUX-ELEMENTs with basic gates, the constant propagation issue, and possible simplification and resynthesis steps.

Theorem 1 (Static logic hazard-freeness) *Let C be a circuit derived from a BDD G by replacing each node in G with a hazard-free multiplexor. 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 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.

Corollary 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.*

Proof: Follows from Theorem 1. ■

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.

V. Dynamic Hazard-Free Synthesis from BDD's

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.

A. The Problem

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

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

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.

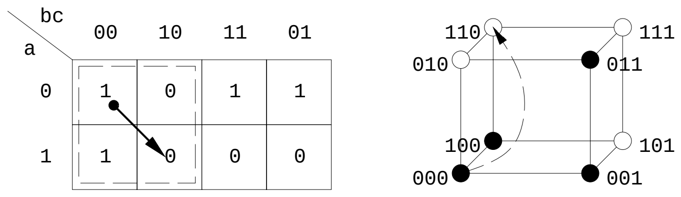

Fig. 3. Dynamic-hazard example.

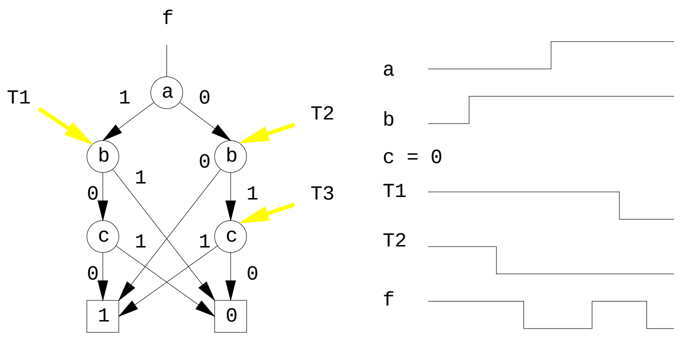

Fig. 4. BDD implementation with ordering $a < b < c$.

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.

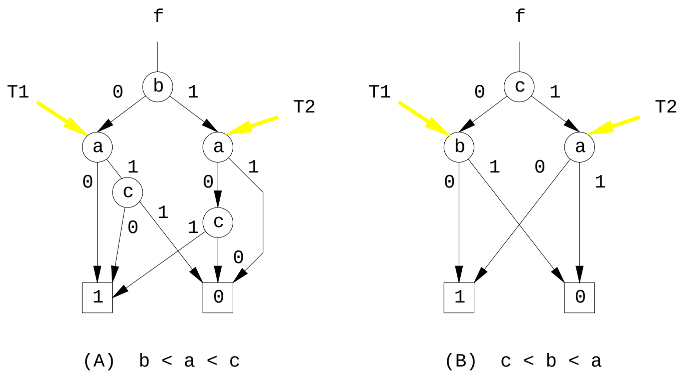

Fig. 5. BDD implementation with orderings (a) $b < a < c$ and (b) $c < b < a$.

Let us first consider a multiplexor implementation translated from the BDD shown in Figure 5 (a). 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 $T1=1, T2=0$, 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.

B. 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 V-E 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 7 (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 II-A), a context signal can only monotonically change once during a $A \Rightarrow B$ transition.

Definition 8 (Excited signal) *Given a state $X \in [A, B]$ in the input transition $A \Rightarrow B$, a context 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 9 (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 10 (Trigger state, signal, transition) *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***.*

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 11 (Trigger signal ordering (TSO-) requirement) *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.*

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 2 (Dynamic logic hazard-freedom) *Let C be a circuit derived from a BDD G by replacing each node in G with a hazard-free multiplexor. C is free of dynamic hazards for all specified dynamic transitions, if the TSO-requirement is satisfied globally.*

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 $0 \to 1$ 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 $0 \to 1$ 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 $0 \to 1$ so does f. This means that $f_a(A) = 1$. Obviously since $f(B) = 1$ and a is making a $0 \to 1$ 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 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 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 $0 \to 1$ 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 $0 \to 1$ transition or if $f_{\overline{a}}$ makes a $0 \to 1$ 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 $0 \to 1$ transition then f will be free of dynamic hazards.

4. A similar argument can be made for the $1 \to 0$ 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 $1 \to 0$ 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 3 (Complete hazard-freedom) *C is free of static and dynamic hazards, both function and logical, for all specified input transitions.*

Proof: Function hazard-freedom is immediate from the problem definition; static logic hazard-freedom is guaranteed by Theorem 1; and dynamic logic hazard-freedom is guaranteed by Theorem 2. ■

Corollary 2 (Hazard-freedom under single-input changes) *A BDD-based circuit C under any ordering is free of hazards under all single-input changes.*

Proof: Function hazard-freedom is immediate from the problem definition; static logic hazard-freedom is guaranteed by Theorem 1; and dynamic logic hazard-freedom under single-input changes is guaranteed by Theorem 2 since no ordering constraints can exist for single-input changes. ■

Corollary 3 (Hazard-freedom under multiple-input dynamic transitions) *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.*

Proof: Recall that $(A, B]$ corresponds to $[A, B] - A$ and $[A, B)$ corresponds to $[A, B] - B$. For both cases, i.e., $\forall X \in (A, B]$ $f(X)$ is constant or $\forall X \in [A, B)$ $f(X)$ is a constant, for each trigger state we have only two possibilities: (1) all the excited signals in the trigger state are trigger signals or (2) the single excited signal in a trigger state is a trigger signal. Hence, in either case, there are no ordering constraints on G imposed by Condition 1 of the TSO requirement. In case (1), the trigger state must be A. Hence, there is no quiescent signal. In case (2), there is only a single trigger signal and all other signals are quiescent. Hence, in either case, there are no ordering constraints on G imposed by Condition 2 of the TSO requirement. Therefore, dynamic logic hazard-freedom is guaranteed by Theorem 2. ■

Note that this corollary shows that there will no ordering constraints generated on “burst-mode” transitions.

C. Finding a Variable Order

We now give a systematic procedure for finding a variable ordering, if one exists, that satisfies the TSO-requirement:

1. Let $\Gamma = (V, E)$ be a *directed graph* where the each vertex $v \in V$ represents a unique variable, and each edge $x \leftarrow y \in E$ means that signal x must appear **before** the signal y in the variable order (i.e., $index(x) < index(y)$). If there is a *path* $x \xleftarrow{p} y$ in Γ, then it also means x must appear **before** y in the variable order.
2. Initialize Γ with V as the set of variables and $E = \emptyset$.

3. For each dynamic input transition $A \Rightarrow B$, add the following constraints to satisfy condition 1 of the TSO-requirement (cf. Definition 11):
(a) For each trigger state $X \in [A, B]$ (cf. Definition 10), determine the set of trigger signals S_T and the set of non-trigger signals S_{NT}.
(b) For each $x \in S_T$ and $y \in S_{NT}$, add the directed edge $x \leftarrow y$ to E.
4. Check if the graph Γ is still acyclic. If yes, proceed; otherwise, abort because no solution exists.
5. For each dynamic input transition $A \Rightarrow B$, add the following constraints to satisfy condition 2 of the TSO-requirement:
(a) For each trigger state $X \in [A, B]$, determine the set of trigger signals S_T and the set of quiescent signals S_Q in that state.
(b) For each $y \in S_Q$, check if there already exists a path $x_i \xleftarrow{p} y$ from y to some $x_i \in S_T$ in Γ and if there already exists a path $y \xleftarrow{p} x_j$ from some $x_j \in S_T$ to y in Γ.
i. If no such path exists, then either **for all** $x_k \in S_T$, add the directed edges $x_k \leftarrow y$, $\forall y$ to E, or **for all** $x_k \in S_T$, add the directed edges $y \leftarrow x_k$, $\forall y$ to E.
ii. If there are only paths $x_i \xleftarrow{p} y$ from some y to some $x_i \in S_T$ in Γ, then **for all** $x_k \in S_T$, add the directed edges $x_k \leftarrow y$, $\forall y$ to E.
iii. If there are only paths $y \xleftarrow{p} x_j$ from some $x_j \in S_T$ to some y in Γ, then **for all** $x_k \in S_T$, add the directed edges $y \leftarrow x_k$, $\forall y$ to E.
iv. Otherwise, backtrack to the most recently made decision at step (i) and change the set of directed edges added to E. If backtracking is not possible, then abort because no solution exists.
6. Find a BDD variable ordering, such that $\forall x \leftarrow y \in E$: $index(x) < index(y)$. If there choices select one arbitrarily.
7. Construct the BDD with the chosen variable ordering. Derive a multiplexor network C.

Theorem 4 (Strict variable ordering) *A strict variable ordering can be derived if and only if an acyclic graph can be derived.*

Proof:
Necessity: If Γ contains a cycle between x and y, it means we require both $index(x) < index(y)$ and $index(y) < index(x)$. This is not possible with a strict BDD variable ordering.
Sufficiency: If Γ does not contain a cycle between x and y, it means either we have only $x \xleftarrow{p} y$, which can be satisfied by $index(x) < index(y)$, or we have only $y \xleftarrow{p} x$, which can be satisfied by $index(y) < index(x)$. ■
In essence, the resulting directed acylic graph (DAG) Γ represents a *partial order*. Any strict BDD variable ordering that satisfies this partial order can be used. One can be chosen to minimize the resulting BDD size. When an acyclic graph cannot be found, then it means no strict BDD variable ordering exists. In this case, we can make use of free BDD's. This discussion will be deferred to Section V-E.

We now illustrate the ideas with an example.

D. An Example

To illustrate the ideas, we have an example from Nowick [20] 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:

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

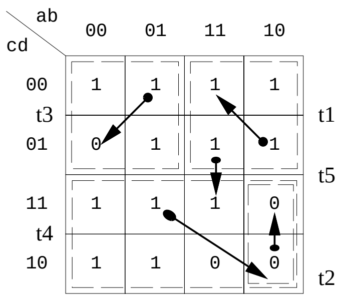

Fig. 6. Another dynamic hazard example.

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 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.

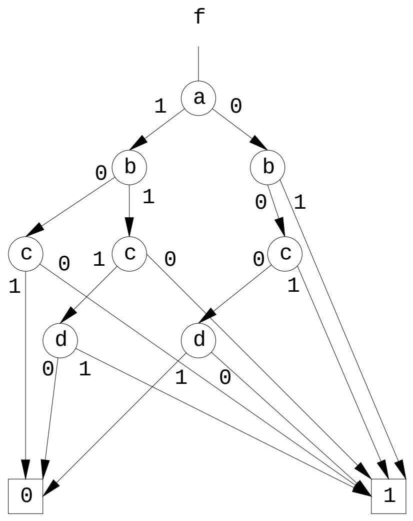

Fig. 7. Dynamic hazard free BDD implementation with ordering $a < b < c < d$.

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**.

The reason why no SOP solution exists is the following: in order to eliminate static logic hazards for the static transition $t_5 : 110{*}1 \Rightarrow 1111$, the SOP solution must contain at least one cube, called a *required-cube*, that contains the transition cube $11-1$. However, this required-cube conflicts with requirements for eliminating dynamic logic hazards for the dynamic transitions $t_4 : 0*1*11* \Rightarrow 1011$. Specifically, the transition cube $t5 = 11-1$ intersects the transition cube of $t4 = [0111, 1011]$, but it does not contain the start state 0111, which is a dynamic hazard violation. See [11], [4], [22] for more details.

t_5	$110*1 \Rightarrow 1111$	1-1 static transition	input changes $\{c+\}$

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.

E. Synthesis from Free BDD's

Although in the example shown in Section V-D 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 $A \Rightarrow B$.

The augmented procedure is as follows.
1. For each specified dynamic transition $T_i \in G$, obtain $\Gamma_{T_i}(V, E)$ as described earlier.
2. Attempt to construct a global ordering that satisfies $\Gamma_{T_i}(V, E)$ for all $T_i \in G$. If this is possible go to Step 7. If this is not possible, identify **conflict** tuples $\{x, y, T_j, T_k\}$, where for transition T_j x appears before y, and for transition T_k y appears before x. (Note that there may be multiple conflicts for the same pair of variables x and y resulting in multiple conflict tuples.)
3. In the transition cube $[A_i, B_i]$ for each T_i find the set of variables S_i that are constant at 0 or 1. Call this assignment of values to S_i the cube c_i.
4. Choose a conflict tuple. Given a conflict tuple that relates T_j to T_k, check if $c_j \cap c_k = \phi$. If so, we find a variable z in the c_j and c_k cubes that has different values and further does not appear as a y in any edge $x \xleftarrow{p} y$ in $\Gamma_{T_i}(V, E)$. If we cannot find such a variable, we cannot resolve the conflict using free BDD's with the given set of $\Gamma_{T_i}(V, E)$'s and we exit the procedure.
5. We choose $z = 0$, and group the specified transitions for which $z = 0$ in a set G_0. G_0 will contain T_j. Similarly, we group the specified transitions for which $z = 1$ in a set G_1. G_1 will contain T_k. If z changes in a T_i then the T_i can be grouped in either set.
6. Go to Step 1, recurring with $G = G_{z=0}$ and $G = G_{z=1}$.
7. Generate the ordering for G, or for $G_{z=0}$ and $G_{z=1}$. Add z to the top of both these orderings with the appropriate value. Exit the procedure.

If the procedure completes successfully, we will obtain a set of orderings of the input variables each tagged with an input combination. For example, we may obtain for the function $f(a, b, c, d, e)$, the orderings:

$$a = 0, c, b, d, e$$

$$a = 1,\ b = 0, d, c, e$$

$$a = 1,\ b = 1, e, c, d$$

This implies that in the free BDD, a is the variable closest to the output. The $a = 0$ input of the root multiplexor is connected to an ordered BDD with ordering c, b, d, e. The $a = 1$ input is connected to a multiplexor with b as its control variable. The $b = 0$ input is connected to an ordered BDD with ordering d, c, e and the $b = 1$ input is connected to an ordered BDD with ordering e, c, d.

To obtain a circuit C_f corresponding to the above ordering, cofactor the given function f with respect to $a = 0$ and $a = 1$. Create an ordered BDD under the specified ordering of c, b, d, e for $f_{\overline{a}}$. Cofactor f_a with respect to $b = 0$ and $b = 1$ to obtain $f_{a \cdot \overline{b}}$ and $f_{a \cdot b}$. Create ordered BDD's under the specified ordering for the cofactors.

Theorem 5 (Dynamic logic hazard-freedom in a free BDD) *C_f is free of dynamic hazards for all specified dynamic transitions.*

Proof: For any given input transition T_i, C_f does not violate the ordering requirement. Therefore, by Theorem 2 C_f is dynamic hazard-free for T_i. ∎

VI. Replacement Strategies and Resynthesis

A. Replacement Circuits

It is worthwhile to replace the multiplexors with primitive gates so non-hazard-increasing logic transformations (e.g., [13]) 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 2.
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 $0 \to 1$ 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[1] that also satisfies the hazard freedom requirements. However, the two-level network may be considerably larger than the multiplexor-based network.

B. 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.

C. 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 acylcic 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.)

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

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 $0 \to 1$ 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 $0 \to 1$ the output f goes $0 \to 1$, and both a and f_a have to go $0 \to 1$ for the output to go $0 \to 1$.
2. A dynamic hazard is caused at a multiplexor because its control input a makes a $1 \to 0$ transition, and data inputs f_a and $f_{\overline{a}}$ make $0 \to 1$ 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 $1 \to 0$ transitions.
4. Same as Case 2 except that a makes a $0 \to 1$ transition, and f_a and $f_{\overline{a}}$ make $1 \to 0$ transitions.

VII. Experimental Results

We have implemented the techniques described in this paper and have tested them on a number of examples. The software has been implemented using the BDD package in SIS [23].

We present a set of results using benchmarks from the asynchronous design benchmark set. The results in Table I correspond to a direct comparison with the two-level hazard-free synthesis procedure of [22]. 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 [22]. For the BDD-based circuits, a hazard-free MUX-ELEMENT requiring four literals was assumed. The two-level circuits contain some very large fanin gates, however, we have reported the literal counts prior to decomposition (which would increase literal count). Note that hazard nonincreasing transformations can be applied to *both* the BDD-based and the two-level circuits improving their area characteristics. The literal counts for the BDD-based realizations compare favorably to the two-level realizations in most examples.

More substantial improvements over two-level solutions can be obtained using the BDD-based method in a sequential synthesis trajectory such as that described in [29]. The BDD solution has less restrictions on state assignment and state minimization, so better results can be obtained. For instance, fewer state variables are required in many cases, resulting in fewer inputs and outputs in the hazard-free combinational logic. The interested reader is referred to [29] for a more comprehensive set of results.

Our BDD-based realizations under any ordering are guaranteed to be hazard-free for *all* static transitions by Theorem 1. However, this is *not* true of the two-level realizations of Table I. In order to make a two-level circuit hazard-free for static tran-

TABLE I
COMPARISON BETWEEN TWO-LEVEL AND MULTILEVEL REALIZATIONS WHEN HAZARD-FREEDOM IS REQUIRED UNDER A SPECIFIED SET OF MULTIPLE-INPUT STATIC AND DYNAMIC TRANSITIONS.

	Specification				**Total literals**	
	States / Transitions		Primary In	Out	*2-level*	*BDD*
q42	4	4	2	2	27	15
ring-counter	8	8	1	2	45	68
binary-counter	32	32	1	4	94	70
binary-counter-co	32	32	1	5	104	80
pe-send-ifc	11	14	5	3	90	88
cache-ctrl	38	49	16	19	704	1231
tsend	22	30	7	4	328	511
tsend-bm	11	14	6	4	96	90
isend-bm	12	15	6	4	177	88
abcs	23	33	9	7	199	271
stetson-p1	31	38	13	14	376	455
stetson-p2	25	27	8	12	178	195
biu-fifo2dma	11	13	5	2	125	119
fifocellctrl	3	3	2	2	16	14
scsi-targ-send	7	8	4	2	53	57
scsi-init-send	7	8	4	2	31	43
scsi-init-rcv-sync	4	5	3	1	20	21

sitions, all the prime implicants of the logic function have to be included in the realization. This can result in a substantially greater number of literals.

By Corollary 2, 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 two-level circuits of Table I.

VIII. CONCLUSIONS

In this paper, we have described a new synthesis method based on ordered and free Binary Decision Diagrams for synthesizing hazard-free multilevel logic implementations. Methods for producing hazard-free logic implemenations have important applications in the field of asynchronous design. We have given automated synthesis procedures that can produce circuits that are free of both static and dynamic hazards for a given set of multiple-input changes. The circuits produced using our method remain hazard-free under any arbitrary gate or wire delays. Our method is based on pure delay model assumption, which means we do not need to rely on potentially slow inertial delays to filter out unexpected spurious transitions. The method has been implemented and its effectiveness has been shown on a number of examples.

We believe our framework is quite general and powerful. As indicated in Section VII, it has been used to handle the synthesis problems arising in extended burst-mode asynchronous circuits [29].

ACKNOWLEDGEMENTS

The authors would like to thank Steve Nowick of Columbia University and Peter Vanbekbergen of Synopsys for a number of very helpful discussions concerning the hazard-free synthesis problem and for the benchmark examples. Thanks to Kenneth Yun and Alex Kondratyev for providing detailed and constructive comments, and helping with the proof of Theorem 5.1. Thanks to Ken Yun and Dave Dill for pointing out the applicability of BDD-based networks to extended burst-mode FSMs.

REFERENCES

[1] V. Akella and G. Gopalakrishnan. Shilpa: a high-level synthesis system for self-timed circuits. In *ICCAD-1992*.

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

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

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

[5] J.G. Bredeson and P.T. Hulina. Elimination of static and dynamic hazards for multiple input changes in combinational switching circuits. *Information and Control*, 20:114–224, 1972.

[6] E. Brunvand and R. F. Sproull. Translating concurrent programs into delay-insensitive circuits. In *ICCAD-1989*.

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

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

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

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

[11] J. Frackowiak. Methoden der analyse und synthese von hasardarmen schaltnetzen mit minimalen kosten I. *Elektronische Informationsverarbeitung und Kybernetik*, 10(2/3):149–187, 1974.

[12] 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*.

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

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

[15] Alain J. Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed Computing*, 1:226–234, 1986.

[16] E.J. McCluskey. *Introduction to the Theory of Switching Circuits*. McGraw-Hill, 1965.

[17] Teresa H.-Y. Meng, Robert W. Brodersen, and David G. Messerschmitt. Automatic synthesis of asynchronous circuits from high-level specifications. *IEEE Transactions on CAD*, 8(11):1185–1205, November 1989.

[18] C.W. Moon, P.R. Stephan, and R.K. Brayton. Synthesis of hazard-free asynchronous circuits from graphical specifications. In *ICCAD-1991*.

[19] C. Myers and T. Meng. Synthesis of timed asynchronous circuits. In *ICCD-1992*.

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

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

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

[23] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli. Sequential Circuit Design Using Synthesis and Optimization. In *Proceedings of the Int'l Conference on Computer Design: VLSI in Computers and Processors*, pages 328–333, October 1992.

[24] H. J. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit state enumeration of finite state machines using BDD's. In *ICCAD-90*, pages 130–133, November 1990.

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

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

[27] M.L. Yu and P.A. Subrahmanyam. A path-oriented approach for reducing hazards in asynchronous designs. In *DAC-1992*.

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

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

Bill Lin received the B.Sc., the M.S., and the Ph.D degrees in Electrical Engineering and Computer Sciences from the University of California, Berkeley, in 1985, 1988, and 1991, respectively. Since graduating from Berkeley, he has been with the VLSI Systems Design Methodologies division of the Interuniversity Micro-Electronics Center (IMEC) in Leuven, Belgium. He is currently heading the System Control and Communications group at IMEC, which is working on various aspects of system-level design technology. His current research interests include hardware/software co-design, design of telecom applications, and synthesis of asynchronous circuits and interface modules. He has previously been with the Hewlett Packard Corporation, the Hughes Aircraft Company, and the Western Digital Corporation. Dr. Lin received the Best Paper Award at the 24th Design Automation Conference in Miami, FL, in 1997. In 1989 and 1990, respectively, he received a distinguished paper citation at the IFIP VLSI conference in Munich, Germany, and a distinguished paper citation at the ICCAD conference in Santa Clara, CA. In 1994, he received a best paper nomination at the European Design and Test Conference in Paris, France, and a best paper nomination at the ACM Design Automation Conference in San Diego, CA.

Srinivas Devadas (S'87-M'88), for a photograph and biography, see p. 95 of the January 1995 issue of this TRANSACTIONS.