BDD-Based Synthesis of Extended Burst-Mode Controllers

Kenneth Y. Yun, *Member, IEEE*, Bill Lin, *Member, IEEE*,
David L. Dill, *Member, IEEE*, and Srinivas Devadas, *Fellow, IEEE*

Abstract— **We examine the implications of a new hazard-free combinational logic synthesis method [1], which generates multiplexor-based networks from** *binary decision diagrams* **(BDDs) — representations of logic functions factored recursively with respect to input variables — on extended burst-mode asynchronous synthesis. First, this method guarantees that there exists a hazard-free BDD-based implementation for every legal extended burst-mode specification. Second, it reduces the constraints on state minimization and assignment, which reduces the number of additional state variables required in many cases. Third, in cases where conditional signals are sampled, it eliminates the need for state variable changes preceding output changes, which reduces overall input to output latency. Finally, we describe a circuit that exemplifies how the BDD variable ordering affects the path delay.**

I. Introduction

There have been many recent advances in asynchronous circuits and systems, both in tool design [2], [3], [4], [5], [6], [7], [8], [9], [10] and actual systems design [11], [12], [13], [14], [15], [16], [17], [18]. However, for maximum acceptability, it is imperative to be able to synthesize circuits that work with existing systems, which are largely made out of synchronous components. One particularly promising design style is the extended burst-mode [19], [10].

This paper describes a new synthesis algorithm for asynchronous controllers specified in extended burst-mode [19], [10]. This algorithm assumes the target implementation to be a combinational circuit with both primary outputs and state variables fed back. The combinational circuit is derived from a Binary Decision Diagram [20] using a recently developed hazard-free combinational synthesis method [1]. Finally, this algorithm guarantees that there exists a hazard-free BDD-based implementation for every legal extended burst-mode specification.

This new approach has definite advantages over other synthesis methods [21], [7], [19], [22], which implement the combinational logic as two-level AND-OR circuits, for a *subclass* of extended burst-mode specifications, although the results appear to be mixed in general. In particular, the circuits synthesized using this new method have considerably lower output latencies than the circuits synthesized by the method in [19], for the specifications with conditional input bursts. Furthermore, this method in conjunction with BDD variable ordering exploration *can be used* to further minimize the delay on user-specified input/output path, which can be very important for achieving high performance in systems that use asynchronous components. We describe a circuit that exemplifies this point.

This work was supported in part by the Semiconductor Research Corporation, Contract no. 93-DJ-205 and by the European Commission under the ESPRIT (6143) project "EXACT".

K. Yun and B. Lin are with Dept. of ECE, UC San Diego; D. Dill is with Computer Science Dept., Stanford University; S. Devadas is with Dept. of EECS, MIT.

II. Background Review

In this section, we review extended burst-mode design style and 3D synthesis method [10]. We point out a limitation associated with the previous target implementation (two-level AND-OR). Finally, we provide a brief review of how a multiplexor network, the target implementation of the 3D machine in this paper, is derived from a binary decision diagram.

A. Extended Burst-Mode Specification

Fig. 1 describes an extended burst-mode state machine (*biu-dma2fifo*) with 4 inputs (ok, $cntgt1$, $frin$, $dackn$) and 2 outputs ($faout$, $dreq$). Labeled edges represent the specified input behavior and the response of the machine during state transitions. For example, "$ok-frin- \,/\, faout-$" means that in state 6 the machine waits for ok and $frin$ to fall. After both have fallen, it lowers $faout$ and transitions to state 0.

Signals not enclosed in angle brackets, such as ok, $frin$, and $dackn$ are *edge signals*. Edge signals ending with $+$ or $-$ are *terminating signals*; the ones ending with $*$ are *directed don't cares*. If a state transition is labeled with a directed don't care $a*$, then the following state transition must be labeled with $a*$ or $a+$ or $a-$. A terminating signal $a+$ denotes a $0 \rightarrow 1$ transition of a if a was initially 0, and no transition at all if a was initially 1. A sequence of state transitions labeled with $a*$ and terminated with $a+$ represents a *single* $0 \rightarrow 1$ transition of a at any point in the sequence. A terminating signal not immediately preceded by a directed don't care represents a *compulsory* transition. Signals enclosed in angle brackets, such as $cntgt1$, represent *conditional* or *level signals*. $\langle cntgt1+ \rangle$ and $\langle cntgt1- \rangle$ denote conditional clauses "if $cntgt1$ is high" and "if $cntgt1$ is low."

An input burst is a non-empty set of input edges (*terminating*

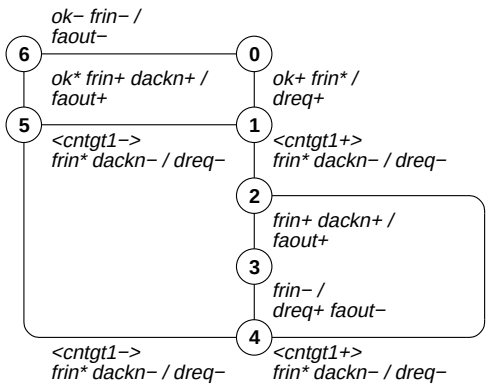


Fig. 1. Extended burst-mode specification.

or *directed don't care*) at least one of which must be a compulsory transition. An output burst consists of a possibly empty set of output edges. If a state transition is not labeled with a level signal, the signal may change freely during the transition. However, if an edge signal is not mentioned in a transition, it is not allowed to change.

In a given state, when all the specified conditional signals have correct values and when all the specified terminating signals in the input burst have changed, the machine generates the corresponding output burst and moves to a new state. Specified edges in the input burst may appear in arbitrary temporal order. However, the conditional signals must stabilize to correct levels some *setup time* before any compulsory edge in the input burst appears and must retain their values until some *hold time* after all of the terminating edges appear. The setup and hold time requirements between conditionals and compulsory edges are similar to those for synchronous flip-flops. Outputs may be generated in any order, but the next set of compulsory edges from the next input burst may not appear until the machine has stabilized.

The following formal definition of the extended burst-mode specification is from [10]. An extended burst-mode specification is a directed graph, $G = (V, E, C, I, O, v_0, \mathsf{cond}, \mathsf{in}, \mathsf{out})$, where V is a finite set of states; $E \subseteq V \times V$ is the set of state transitions; $C = \{c_1, \ldots, c_l\}$ is the set of conditional inputs; $I = \{x_1, \ldots, x_m\}$ is the set of edge inputs; $O = \{z_1, \ldots, z_n\}$ is the set of outputs; $v_0 \in V$ is the unique start state; $\mathsf{cond} : E \rightarrow \{0, 1, *\}^l$ defines the values of the conditional inputs; $\mathsf{in} : V \rightarrow \{0, 1, *\}^m$ defines the values of the edge inputs; $\mathsf{out} : V \rightarrow \{0, 1\}^n$ defines the values of the outputs upon entry to each state.

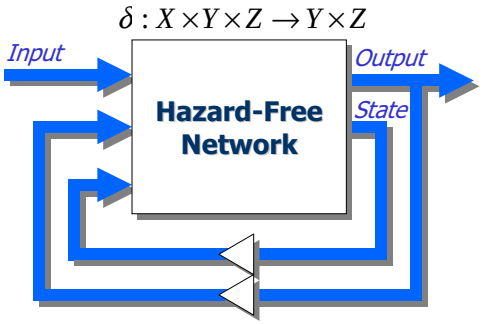


Fig. 2. 3D asynchronous state machine.

B. 3D Implementation

A 3D asynchronous finite state machine is a 4-tuple (X, Y, Z, δ) where X is a non-empty set of primary input symbols, Y a non-empty set of primary output symbols, Z a possibly empty set of internal state variable symbols, and $\delta : X \times Y \times Z \rightarrow Y \times Z$ is a *next-state function*. The hardware implementation of a 3D state machine (see Fig. 2) is a combinational network, which implements the next-state function, with the outputs of the network fed back as inputs to the network.

A 3D implementation of an extended burst-mode specification is obtained from the *next-state table*, a 3-dimensional tabular representation of δ. The next state of every *reachable* state must be specified in the next-state table; the remaining entries are don't cares.

A Type I machine cycle[1] [10] consists of an input burst followed by a concurrent output and state burst. Initially or after completion of the previous output and state burst, the machine waits for an input burst to arrive. When the machine detects that all of the terminating edges of the input burst have appeared, it generates a concurrent output/state burst (it may not include any output or state variable transition at all in some cases).

In the 3D implementation of extended burst-mode machines, no fed-back output or state variable change arrives at the network input until all of the specified edges in the output and state burst have appeared at the network output. These conditions are met by inserting delays in the feedback paths as necessary. A 3D machine can then be viewed as a combinational network alternately excited by a set of input edges (during an input burst) and by a set of fed-back output and state variable edges (during an output/state burst). Thus each burst is a generalized transition of inputs to the combinational network, as described below.

Generalized transition.

A *generalized transition* is a triple (T, A, B) where T is a mapping from a set of inputs to a set of *input types*, A a *start-cube*, and B an *end-cube*. There are three types of inputs: *rising edge*, *falling edge*, and *level* signals. Edge inputs can only change monotonically. Level inputs must remain constant or undefined (don't care), which implies that each level input must hold the same value in both A and B or be undefined in both A and B. Level inputs, if they are undefined, may change non-monotonically,

A generalized transition cube $[A, B]$ is the smallest cube that contains the start- and end-cubes A and B. It represents the set of all minterms that can be reached during a *legal* transition from a point in start-cube A to a point in end-cube B, assuming that the inputs can change in arbitrary order. *Open generalized transition cubes*, $[A, B)$, $(A, B]$, and (A, B), denote $[A, B] - B$, $[A, B] - A$, and $[A, B) - A$ respectively. Note that $[A, B) = \emptyset$, if $A = B$. The *start-subcube* A' is a maximal subcube of A such that the value of every rising edge input i in A' is 0, if it is $*$ in A, and the value of every falling edge input j in A' is 1, if it is $*$ in A. The *end-subcube* B' is a maximal subcube of B such that the value of every rising edge input i in B' is 1, if it is $*$ in B, and the value of every falling edge input j in B' is 0, if it is $*$ in B. Intuitively, the longest transitions, disregarding non-monotonic signals, are those that lead from A' to B'.

A generalized transition (T, A, B) is a static transition for f iff $f(A) = f(B)$; it is a dynamic transition for f iff $f(A) \neq f(B)$. No change in level inputs can enable output changes directly, that is, at least one edge input must change from 0 to 1 or from 1 to 0 in a generalized dynamic transition. During a generalized transition (T, A, B), each output signal is assumed to change its value at most once. If not, a function hazard is said to be present.

An extended burst-mode transition is a generalized transition with the following requirements:

[1] Type II machine cycle — an input burst followed by an output burst followed by a state burst — can also be used.

1. For every pair of minterms X and Y in $[A, B)$, $f(X) = f(Y)$.
2. For every pair of minterms X and Y in B, $f(X) = f(Y)$.

Every extended burst-mode transition is function-hazard-free [10].

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

Delay model.

In order to simplify the synthesis, we use the unbounded wire delay model for the circuit with feedback wires cut. In other words, for each burst, the combinational circuit generated by cutting feedback wires functions correctly, *for any value of gate/wire delay*. However, once the feedback wires are connected together, we must assume *bounded wire delay*. Likewise, the setup/hold time constraints are calculated based on bounded gate/wire delay.

Limitations of two-level implementation.

In order for a 2-level AND-OR implementation of an output or a state variable function to be hazard-free, a set of covering requirements [19], [7] must be satisfied for each burst, i.e., extended burst-mode transition. It was shown in [19] that it is not always possible to satisfy the covering requirements for all of the specified bursts under the presence of non-monotonically changing (undefined) conditionals, if a *single transition time* (STT) state assignment [23], [24] is used. The approach taken in [19] was to insert a state burst between a conditional input burst and the corresponding output burst in order to guarantee that the covering requirements can be satisfied for all of the specified bursts. Unfortunately, the early state burst between the input burst and output burst increased the input/output latency significantly. Section III-C provides a specific example which illustrates this point.

C. Multiplexer Networks Derived from BDDs

The following definition of a Binary Decision Diagram is from [20].

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

The correspondence between a BDD and a Boolean function is defined as below:

Definition 2 *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 $\mathsf{value}(v) = 1$, *then $f_v = 1$;*
(b) If $\mathsf{value}(v) = 0$, *then $f_v = 0$.*

2. If v is a non-terminal vertex with $\mathsf{index}(v) = i$, *then*

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

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

In addition,

1. Each decision variable occurs at most once on every path from a terminal vertex to the root vertex,
*2. A **reduced** BDD is a BDD in which $low(v) \neq high(v)$ for any vertex v and no two subgraphs are identical.*

A *reduced ordered BDD* (ROBDD) is a canonical form with the following restriction: for any non-terminal vertex v, if $low(v)$ is a non-terminal, then $\mathsf{index}(v) < \mathsf{index}(low(v))$, and if $high(v)$ is a non-terminal, then $\mathsf{index}(v) < \mathsf{index}(high(v))$.

A *reduced free BDD* (free BDD) is a BDD which does not require a strict variable ordering (unlike in an OBDD) but still requires that each decision variable is encountered at most once when traversing a path from a terminal vertex to the root vertex.

Every path from the root vertex to a terminal vertex corresponds to a cube. For example, the path from the root to the terminal vertex 1 via decision variables a, c, and b corresponds to an on-set cube, $ac\overline{b}$, in Fig. 3a; the path from the root to the terminal vertex 0 via decision variables a, b, and q corresponds to an off-set cube, $\overline{a}b\overline{q}$.

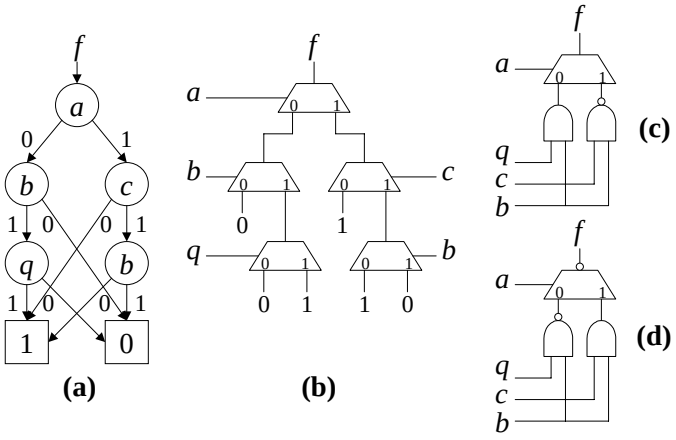


Fig. 3. (a) BDD; (b) MUX network derived from BDD; (c) Simplified network (by constant propagation); (d) After "bubble shuffling" is applied.

A multi-level network can be derived directly from a BDD[2] by replacing each vertex with a two-input MUX with the decision variables as the *select* inputs of the MUXes. Fig. 3b shows a MUX network derived from the BDD in Fig. 3a. If one or more input of a MUX is constant, the MUX can be replaced with a simpler gate, such as a NAND or a NOR. This *constant propagation* is carried out topologically from inputs to outputs. Figs. 3cd show equivalent networks after constant propagation and after "bubble shuffling".

III. BDD-Based Synthesis

In this paper, we use a new BDD based combinational synthesis technique from [1]. This approach imposes a different set

[2]BDDs have been used to generate multi-level synchronous circuits [25], [26].

of requirements to guarantee freedom from all hazards, but we will show that it is always possible to meet these requirements *without* the multiple transition time state assignment that was required in the method of [19], resulting in greatly reduced latency in many cases.

Combinational networks that describe next-state functions are constructed from a BDD description. The basic gates that comprise combinational networks are inverters, NANDs, NORs, ANDs, ORs, inverting MUXes, and MUXes. We assume that every basic gate is *atomic*, i.e., a single transition of a gate input cannot cause a multiple transitions at the output. Fig. 4 shows a CMOS pass transistor implementation of an inverting multiplexor. This implementation is atomic, if the delays t_1 and t_2 are closely matched. Note that a hazard-free MUX implemented in two-level SOP requires 3 two-input NAND gates and 1 three-input NAND gate.

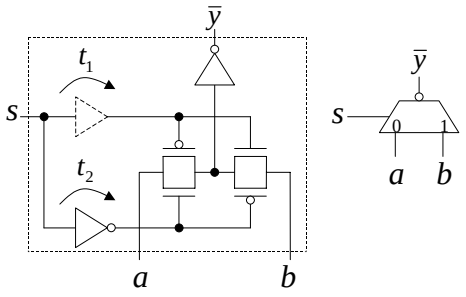


Fig. 4. A CMOS inverting multiplexor.

A. Hazard-free Combinational Synthesis

We use the approach from [1] to synthesize hazard-free combinational circuits under extended burst-mode transitions. This method is based on building a BDD for a specified function and deriving a multi-level circuit from it. To ensure that the resulting multi-level circuit is hazard-free, a requirement called the *trigger signal ordering* (TSO) must be satisfied. This requirement imposes constraints on the variable ordering of the BDD. It was shown in [1] that if this variable ordering is satisfied, then the resulting multi-level circuit is free of logic hazards for a set of specified transitions. Note that every input change in [1] was assumed to be monotonic during each transition. We will prove that the resulting circuit is free of logic hazards for a set of specified extended burst-mode transitions, in which some inputs may change non-monotonically, as long as the TSO requirement is satisfied.

A *trigger state* is a state in which an input change enables the output to change. A *trigger signal* is an input signal whose transition in a trigger state enables the output to change; a *non-trigger signal* is an input signal which is enabled to change but cannot by itself enable the output to change. The TSO requirement states that *trigger signals in a trigger state must appear before the non-trigger signals of the same trigger state in the variable ordering*.

In the generalized transition cube that corresponds to an extended burst-mode dynamic transition, all *terminating* signals are trigger signals in one or more minterms, because terminating edges can appear in any temporal order and the last one that appears is a trigger signal. Note that no terminating signal can be a non-trigger signal, because no output change can be enabled until *all* terminating edges appear. Furthermore, all *don't care* signals (directed or undirected) are non-trigger signals in one or more minterms, because their values may change anywhere, including in the trigger states, in the generalized transition cube. Clearly, no don't care signal can be a trigger signal in any minterm in the generalized transition cube, because don't care signals can never enable outputs to change. Therefore, we can impose a set of ordering requirements, which do not conflict, as a sufficient condition for hazard freedom *per generalized transition cube*, although the TSO requirement in [1] is an imposition on each trigger state in the transition cube.

Now we can state the variable ordering requirements for the extended burst-mode transitions as follows: *Along every path from root to terminal of the BDD whose corresponding cube intersects the generalized transition cube, no don't care signal of a dynamic transition appears before a terminating signal of the same.*

Here, we prove that the combinational network derived from a reduced free BDD description is hazard-free during an extended burst-mode transition as long as the BDD satisfies the variable ordering requirement for the transition stated above.

Lemma 1 *If (T, A, B) is an extended burst-mode transition for f, then $f_{\overline{s}}(X) = f_s(X)$ for every don't care signal s in (T, A, B) and for every minterm X in $[A, B]$, where $f_{\overline{s}}$ and f_s are the Shannon cofactors of f with respect to $\overline{s}$ and s respectively.*

Proof: Suppose that s is a don't care in (T, A, B) and a minterm X is in $[A, B]$. $X = [\ldots, x_s, \ldots]$ and $X' = [\ldots, \overline{x_s}, \ldots]$, where x_s and $\overline{x_s}$ are the values of s in X and X' and all other components are the same. Because s is a don't care, $X \in B$ implies $X' \in B$ and $X \in [A, B)$ implies $X' \in [A, B)$. Thus $f(X) = f(X')$. If $x_s = 1$, $f(X) = f_s(X)$ and $f(X') = f_{\overline{s}}(X')$. $f_{\overline{s}}(X') = f_{\overline{s}}(X)$ because $f_{\overline{s}}$ is independent of s. Thus $f_s(X) = f_{\overline{s}}(X)$. On the other hand, if $x_s = 0$, $f(X) = f_{\overline{s}}(X)$ and $f(X') = f_s(X') = f_s(X)$. Therefore, $f_s(X) = f_{\overline{s}}(X)$. ■

Example. Fig. 5 illustrates a generalized transition cube for an extended burst-mode transition $s* a+ b+$ enabling $f+$ (a and b are terminating signals and s is a directed don't care). $f(X) = 1$ for all $X \in B$ and $f(X) = 0$ for all $X \in [A, B)$. Because s is a don't care, f is independent of s. Thus $f_s(X) = f_{\overline{s}}(X)$ for all $X \in [A, B]$.

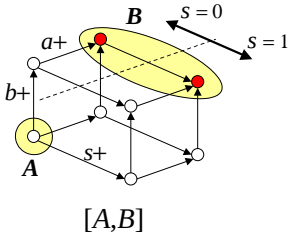


Fig. 5. Generalized transition cube $[A, B]$ for extended burst-mode transition $s* a+ b+ / f+$. Signals s, a, and b are of rising-edge type in (T, A, B).

Definition 3 *Subtransitions:*
1. If s is not a constant 0 in (T, A, B), (T, A_s, B_s) is a subtransition of (T, A, B) with the value of s fixed to 1.
2. If s is not a constant 1 in (T, A, B), $(T, A_{\overline{s}}, B_{\overline{s}})$ is a subtransition of (T, A, B) with the value of s fixed to 0.

Note that $(T, A_s, B_s) = (T, A, B)$, if s is a constant 1 in (T, A, B), and $(T, A_{\overline{s}}, B_{\overline{s}}) = (T, A, B)$, if s is a constant 0 in (T, A, B). $B_s \subset B$, i.e., B_s is a subcube of B, if $X \in B_s$ implies $X \in B$ but $B_s \neq B$. Since $f(X) = f(Y)$ for any pair of minterms X and Y in B for extended burst-mode transition (T, A, B), we use the notation $f(B)$ to describe the value of f for all X in B. Likewise, $f(X) = f(Y)$ for any pair of minterms X and Y in $[A, B)$; thus $f([A, B))$ denotes the value of f for all X in $[A, B)$.

Lemma 2 *If (T, A, B) is an extended burst-mode transition for f, then*
1. (T, A_s, B_s) is an extended burst-mode transition for f_s, if s is not a constant 0;
2. $(T, A_{\overline{s}}, B_{\overline{s}})$ is an extended burst-mode transition for $f_{\overline{s}}$, if s is not a constant 1.

Proof: First, we will prove that (T, A_s, B_s) is an extended burst-mode transition for f_s, if s is not a constant 0.
1. s is a constant 1 in (T, A, B).
Then $f_s = f$ in $[A, B]$ and $(T, A_s, B_s) = (T, A, B)$. Thus (T, A_s, B_s) is an extended burst-mode transition for f_s.
2. s is a don't care in (T, A, B).
Then s is a don't care in B. Thus $B_s \subset B$, so $f(B_s) = f(B)$. $[A_s, B_s) = [A_s, B_s] - B_s = [A_s, B_s] - B \subset [A, B)$, so $f([A_s, B_s)) = f([A, B))$. Thus (T, A_s, B_s) is an extended burst-mode transition for f_s.
3. s is a rising terminating signal in (T, A, B).
Then $s = 1$ in B, which implies $B_s = B$ and $f(B_s) = f(B)$. $[A_s, B_s) = [A_s, B_s] - B \subset [A, B)$, so $f([A_s, B_s)) = f([A, B))$. Thus (T, A_s, B_s) is an extended burst-mode transition for f_s.
4. s is a falling terminating signal in (T, A, B).
Then $s = 0$ in B. Thus $[A_s, B_s] \cap B = \emptyset$, which implies $[A_s, B_s] \subset [A, B)$. Therefore, $f([A_s, B_s]) = f([A, B))$, which means that (T, A_s, B_s) is an extended burst-mode transition for f_s.
Similarly, $(T, A_{\overline{s}}, B_{\overline{s}})$ is an extended burst-mode transition for $f_{\overline{s}}$, if s is not a constant 1. ■

Corollary 1 *If (T, A, B) is an extended burst-mode dynamic transition for f and s is a don't care in (T, A, B), then (T, A_s, B_s) is an extended burst-mode dynamic transition for f_s and $(T, A_{\overline{s}}, B_{\overline{s}})$ for $f_{\overline{s}}$.*

Corollary 2 *Static transitions of cofactors:*
1. (T, A_s, B_s) is a static transition for f_s if (T, A, B) is an extended burst-mode transition for f and s is a falling terminating signal.
2. $(T, A_{\overline{s}}, B_{\overline{s}})$ is a static transition for $f_{\overline{s}}$ if (T, A, B) is an extended burst-mode transition for f and s is a rising terminating signal.

Theorem 1 *The combinational network C derived from a reduced BDD (ordered or free) description of f is hazard-free during an extended burst-mode transition if it satisfies the variable ordering requirement for the transition:* ***no don't care signal appears before a terminating signal.***

Proof: We prove by induction on the number of variables.
Base case: The sole input of the network is connected to the select input of the multiplexor. The other input terminals are connected to a constant 1 or 0. Since the multiplexor is atomic and only the select input can change, f is hazard-free.
Inductive hypothesis: Now assume that a combinational network derived from a reduced BDD representation of an n-input function ($n \geq 1$), which satisfies the variable ordering requirements for an extended burst-mode transition, is hazard-free during the extended burst-mode transition.
Now consider the network C derived from a reduced BDD representation of function f with $n + 1$ input variables and an extended burst-mode transition (T, A, B) for f. Assume that the select input of the multiplexor driving the output of C is s and the data inputs are $f_{\overline{s}}$ and f_s. Then (T, A_s, B_s) is an extended burst-mode transition for f_s if s is not a constant 0, and $(T, A_{\overline{s}}, B_{\overline{s}})$ is for $f_{\overline{s}}$ if s is not a constant 1, by Lemma 2. Since f satisfies the variable ordering requirements, so do f_s and $f_{\overline{s}}$. Therefore, f_s is hazard-free if s is not a constant 0, and $f_{\overline{s}}$ is hazard-free if s is not a constant 1, by the inductive hypothesis. We will consider 3 cases: s is a constant, s is a don't care, and s is a terminating signal.
1. s is a constant:
The multiplexor is a wire as long as s remains constant. Thus, if $s = 0$, $f = f_{\overline{s}}$ is hazard-free since s is not a constant 1. Likewise, f is hazard-free, if $s = 1$.
2. s is a don't care:
First we prove by contradiction that (T, A, B) must be a static transition for f if s is a don't care. Assume that (T, A, B) is a dynamic transition for f. By Corollary 1, (T, A_s, B_s) is an extended burst-mode dynamic transition for f_s. Suppose s remains at 1 while f_s changes. Then the change in f_s propagates to f, which means that there is a terminating signal that enables f to change, regardless of s, violating the variable ordering requirement.
By Lemma 1, $f(X) = f_s(X) = f_{\overline{s}}(X)$ for every X in $[A, B]$. Therefore, (T, A_s, B_s) and $(T, A_{\overline{s}}, B_{\overline{s}})$ are static transitions of same type, that is, both $0 \rightarrow 0$ or both $1 \rightarrow 1$, for f_s and $f_{\overline{s}}$ respectively. By the inductive hypothesis, f_s and $f_{\overline{s}}$ are hazard-free, therefore constant. Since the multiplexor is atomic, f is hazard-free.
3. s is a terminating signal:
Without loss of generality, consider only the case in which s rises. By Corollary 2, $f_{\overline{s}}$ undergoes a static transition, i.e., does not change. By the inductive hypothesis, both f_s and $f_{\overline{s}}$ are hazard-free. Consider the case in which $f_{\overline{s}} = 0$. We prove by contradiction that f_s must rise or remain a constant. Assume that f_s is initially 1 and falls to 0, but s rises first. Since $f_{\overline{s}} = 0$ and $f_s = 1$ initially, f is enabled to rise as s rises. This is a static function hazard, since f is assumed to be 0 at the end of the transition. Thus f_s must rise or remain a constant; in both cases, f is hazard-free. Similarly, we can prove that f is hazard-free

when $f_{\overline{s}} = 1$, by proving that f_s must fall or remain a constant. ■

Corollary 3 *The combinational network C derived from a reduced BDD (ordered or free) description of f is hazard-free during an extended-burst-mode static transition.*

B. Sequential Synthesis Procedure

The sequential synthesis procedure consists of the following three steps: (1) hazard-free state assignment (2) hazard-free layer minimization (3) layer encoding.

B.1 Hazard-free State Assignment

We use an algorithm for assigning states which always results in a hazard-free single-transition-time (STT) state assignment. In contrast, the previous algorithm for extended burst-mode [19] use a multiple-transition-time assignment: a state variable change was required before an output change, increasing latency significantly. Indeed, it can be shown that multiple transitions are necessary for extended burst-mode when implemented with 2-level AND-OR logic, so the use of BDDs has an inherent performance benefit.

This algorithm builds a *primitive next-state table* — a 3-dimensional table with X-axis representing the input bit vector, Y-axis the output bit vector, and Z-axis the specification states. The algorithm assigns, according to the extended burst-mode semantics, a next state, which consists of two components (next outputs and next specification state), to entries in the table. We use the state assignment for Type I machine cycle described in [10]. An XY-plane of the primitive next-state table is called a *layer*. Initially, each specification state is assigned to a unique layer. The algorithm then collapses the primitive next-state table into a *reduced next-state table* by merging compatible specification states without violating TSO requirements.

We show that Type I next state assignment is free of logic hazards for a BDD-based implementation, if each specification state is assigned to a unique layer and if the layers can be encoded so that every transition crossing the layer boundary is critical-race-free. The BDD-based implementation is hazard-free during an extended-burst-mode static transition and, if the variable ordering requirement is satisfied, hazard-free during an extended-burst-mode dynamic transition as well. This ordering requirement can be satisfied trivially for each transition individually; however, we need to check whether it is possible to satisfy the requirements for every transition simultaneously.

Our strategy is to build an ROBDD using a *global* variable ordering, if such an ordering can be found, or to build a free BDD. If no global order exists, we must find a variable that can appear first. This variable partitions the function into a left and right BDD. The left and right BDDs need not have the same variable order, so they can be constructed recursively using the same method.

Assume that each specification state is assigned to a unique layer. In a Type I machine cycle, only the input bursts can be dynamic transitions. Therefore, it suffices to check whether there are *conflicting* ordering requirements among the input bursts from the same specification state.

Lemma 3 *There always exists a BDD that satisfies the variable ordering requirements for any two dynamic input burst transitions from a specification state.*

Proof: If there are no conflicting ordering requirements among the input bursts, then the variable ordering requirements are trivially satisfied in an ordered BDD.

Assume that the input bursts from state transitions (u, v) and (u, w) have conflicting ordering requirements. By the distinguishability constraint [10], either the conditions are mutually exclusive, or the set of compulsory edges in the input burst of (u, v) is not a subset of the set of all possible edges in the input burst of (u, w).

If the conditions of two input bursts are mutually exclusive, then there exists a conditional signal such that it is a constant in both input bursts but its value in one input burst is different from that in the other input burst. If this conditional variable appears before any variable involved in the ordering, the variable ordering for each input burst is satisfied in the left or right partition created by this conditional variable.

If the conditions of two input bursts are not mutually exclusive, then there exist compulsory signals i and j in the input bursts of (u, v) and (u, w) respectively such that i is a constant in the input burst of (u, w) and j is a constant in the input burst of (u, v). Suppose that we select a variable ordering such that i appears before j and also before all other variables involved in the orderings. Let the output of the multiplexor with i as its select signal be g. Without loss of generality, assume that i *rises* during the input burst of (u, v). For the input burst of (u, w), g_i is irrelevant, because i is a constant 0 in the input burst of (u, w). Therefore, the variable ordering requirement for the input burst of (u, w) is satisfied by selecting an appropriate variable ordering in the sub-BDD $g_{\overline{i}}$.

Let (T, A, B) be the input burst transition of (u, v). Since (T, A, B) is an extended burst-mode transition for g, $(T, A_{\overline{i}}, B_{\overline{i}})$ is an extended burst-mode transition for $g_{\overline{i}}$, by Lemma 2. By Corollary 2, $(T, A_{\overline{i}}, B_{\overline{i}})$ is a static transition for $g_{\overline{i}}$. Thus, there is no ordering requirement in the sub-BDD $g_{\overline{i}}$ for $(T, A_{\overline{i}}, B_{\overline{i}})$. We can select an appropriate variable ordering in the sub-BDD g_i to satisfy the requirement for the input burst of (u, v). By symmetry, a free BDD with j appearing before i can also satisfy the variable ordering requirement.

■

Example. Consider two input bursts, $a+\, b*\, c+$ and $a*\, b+\, d+$, which correspond to (T_1, A, B) and (T_2, A, C) respectively with $A = 0000$, $B = 1\text{x}10$, and $C = \text{x}101$, as shown in Fig. 6a. (T_1, A, B) requires that $a < b$ and $c < b$, because b is a directed don't care and a and c are terminating signals in (T_1, A, B). (T_2, A, C) requires that $b < a$ and $d < a$, because a is a directed don't care and b and d are terminating signals in (T_2, A, C). Obviously, we cannot satisfy $a < b$ and $b < a$ globally. Since c and d are compulsory edges in (T_1, A, B) and (T_2, A, C) respectively and $c = 0$ in (T_2, A, C) and $d = 0$ in (T_1, A, B), free BDD implementations that satisfy $a < b$ and $b < a$ "locally", such as the ones in Fig. 6bc, exist.

We can use an inductive argument to show that the result of Lemma 3 applies to the general case with multiple dynamic tran-

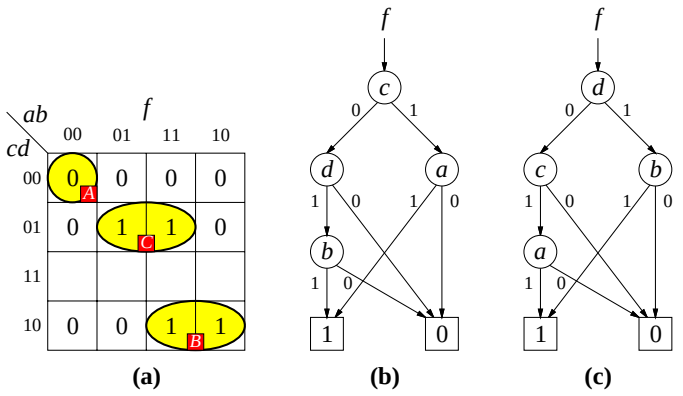


Fig. 6. Satisfying variable ordering locally.

sitions from a specification state. In short, either the conditions are mutually exclusive, or, in each input burst, there exists a *unique* compulsory input that does not change its value in the other input bursts from the same specification state. Therefore, a combination of conditionals and unique compulsory inputs can be used to partition the BDD so that the variable ordering requirement for each dynamic transition is satisfied locally in each partition.

In the 3D implementation of extended burst-mode machines, only input bursts can be dynamic transitions. If a unique code is assigned to each specification state, we can always use fed-back state variables as partitioning variables, so that the variable ordering requirements of each specification state are satisfied locally in each partition. Therefore, ***there exists a hazard-free free BDD implementation for every legal extended burst-mode specification.*** Layer minimization must also be constrained to avoid creating variable ordering conflicts, as shown below.

B.2 Hazard-free Layer Minimization

In the next step, the algorithm transforms the primitive next-state table into a reduced next-state table by merging layers. Specification states can be merged into a common layer, iff they are compatible.

In order to define *compatibility* of specification states precisely, we formally define *primitive next-state table* as below. A *primitive next-state table*, $T = (V, W, X, Y, \delta, \lambda)$, is a 6-tuple, where V is the set of specification states; W is the set of conditional input bit vectors, $\{(c_1, \ldots, c_l) \mid c_i \in \{0,1\}, i \in 1, \ldots, l\}$; X is the set of edge-input bit vectors, $\{(x_1, \ldots, x_m) \mid x_j \in \{0,1\}, j \in 1, \ldots, m\}$; Y is the set of output bit vectors, $\{(y_1, \ldots, y_n) \mid y_k \in \{0,1\}, k \in 1, \ldots, n\}$; $\delta : V \times W \times X \times Y \rightarrow V \cup \{*\}$ and $\lambda : V \times W \times X \times Y \rightarrow \{0,1,*\}^n$ define the *next specification state function* and the *next output function* respectively. Note that don't care values ($*$) are assigned to unreachable entries in the primitive next-state table for further state minimization.

For two specification states, u and v, to be compatible, we must ensure that the TSO order be preserved when u and v are merged, in order to guarantee that the corresponding BDD-based implementation is free of dynamic hazards.

Definition 4 *u and v in V are **BDD-dhf-compatible** (BDD dynamic-hazard-free compatible) iff for every pair of state transitions (u, w_u) and (v, w_v),*
1. there exists $k \in 1, \ldots, n$ such that $\mathsf{out}_k(u) \neq \mathsf{out}_k(v)$ or
2. i is a terminating signal in (u, w_u) and j is a don't care in (u, w_u) $\Rightarrow$ i is a terminating signal in (v, w_v) or j is a don't care in (v, w_v), that is, $\mathsf{in}_i(w_u) \neq \mathsf{in}_i(u) \wedge \mathsf{in}_i(w_u) \neq * \wedge \mathsf{in}_j(w_u) = *$ implies $\mathsf{in}_i(w_v) \neq \mathsf{in}_i(v) \wedge \mathsf{in}_i(w_v) \neq *$ or $\mathsf{in}_j(w_v) = *$.

This criterion states that no input burst from u has conflicting ordering requirements with an input burst in v that has identical values of fed-back outputs, which is only a sufficient condition for dynamic hazard freedom. Note that the dhf-compatibility is implementation-dependent; hence the BDD-dhf-compatibility criterion applies to BDD-based implementations only.

Example. Fig. 7 shows an example of merging two specification states resulting in conflicting variable ordering requirements. Merging i and j would require $a < b$, $c < d$, $b < a$, and $d < c$, which are impossible to satisfy simultaneously. Therefore, i and j are not BDD-dhf-compatible.

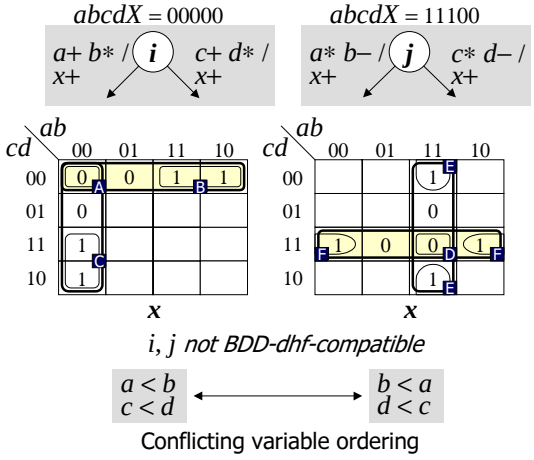


Fig. 7. Output-compatible but not BDD-dhf-compatible.

Definition 5 *u and v in V are **compatible** ($u \sim v$) iff u and v are BDD-dhf-compatible and, for every s in $W \times X \times Y$,*
1. $\lambda(u, s) = * \vee \lambda(v, s) = * \vee \lambda(u, s) = \lambda(v, s)$ and
2. $\delta(u, s) = * \vee \delta(v, s) = * \vee \delta(u, s) \sim \delta(v, s)$.

The layer minimization and encoding steps to complete the sequential synthesis are identical to the two-level 3D synthesis described in [10]. The only difference in sequential synthesis is the state compatibility criterion: in particular, dhf-compatibility.

C. Example: Comparison to Two-Level AND-OR

Fig. 8 shows the specification, the next-state table, and a BDD-based implementation of the *mode follower* circuit [19]. If the mode bit d sampled at the rising edge of the clock ϕ is 1, the output x follows the clock for that cycle and the output y remains 0. Otherwise, y follows the clock and x remains 0.

All three states are compatible and thus merged in a single layer. The resulting next-state table is shown in Fig. 8b. During transitions from states 1 and 2 back to state 0, d is a don't care.

Thus the BDDs for outputs x and y must satisfy the variable ordering, $d < \phi$. A BDD for output x that satisfies the variable ordering is shown in Fig. 8c. In the resulting circuit after constant propagation shown in Fig. 8c, the latency from $\phi+$ to $x+$ is the delay through two gates: an inverter plus a NOR gate.

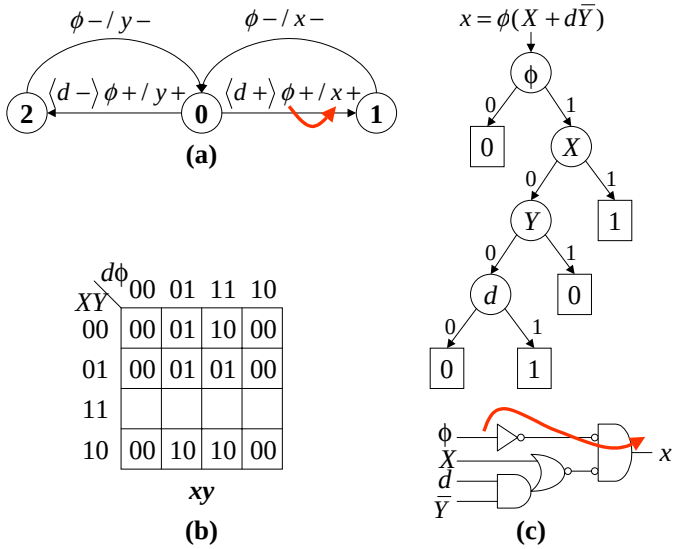


Fig. 8. *Mode follower* in BDD-based implementation. (a) Specification; (b) Next-state table; (c) A BDD for output x and the corresponding implementation of x after constant propagation.

Fig. 9 shows the specification of the mode follower with state variable transitions "backannotated" and a two-level SOP implementation. This implementation executes a 3-phase operation for the state transition enabled by conditional input bursts: input burst followed by a state burst followed by an output burst. For example, the transition from state 0 to 1 is done in 3 phases: $\phi+ \rightarrow p+ \rightarrow x+$ after d stabilizes to 1. Hence the latency from $\phi+$ to $x+$ is $t_1 + t_2$ — four gate delays: 3 NAND gates plus an inverter. Clearly, the two-level AND-OR solutions incur additional delays for state variable transitions inserted between input and output bursts.

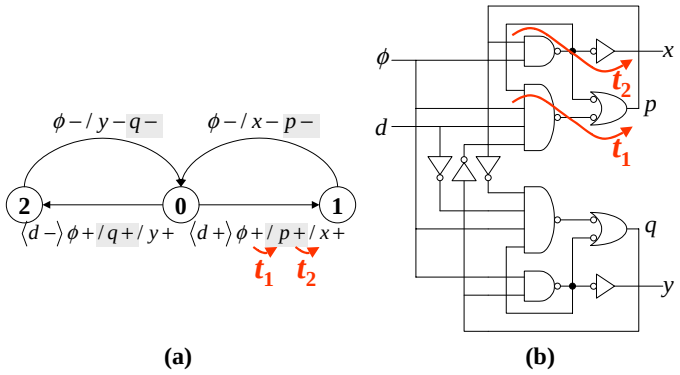


Fig. 9. *Mode follower* in two-level SOP implementation. (a) Specification with state variable transitions "backannotated"; (b) Implementation.

IV. Effects of Variable Ordering on Path Delays

In synchronous designs, one of the important design objectives is to carefully balance the computation blocks so that no part of the circuits is idle while other parts are busy because the clock period is determined by the worst-case delay of all the computation blocks. However, asynchronous designs can proceed immediately upon receipt of a completion signal, so it is often desirable to create *highly unbalanced* asynchronous circuits, where one path has been optimized, possibly at the expense of others, to optimize overall system performance.

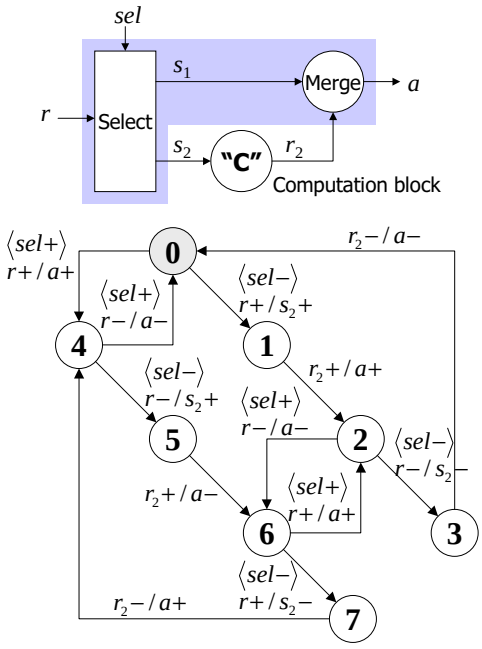


Fig. 10. Select/merge example.

To illustrate this point and also provide a nice circuit example, we consider a hypothetical problem posed by Ivan Sutherland [27]. When the circuit in Fig. 10 detects a transition on r (request), if sel is high, it signals on s_2 to start an expensive operation "C", then signals completion on a when it receives r_2 from the operation; otherwise, it does nothing except signaling completion on a as quickly as possible. It is quite likely in this situation that the designer would want the minimum possible latency from r to a in this case to maximize overall system performance. The implementation shown inside the shaded box in Fig. 10 (select and merge elements in cascade) does not do a very good job of minimizing this latency, because there would be a significant delay from r to s_1 for most select implementations along with an additional delay through the merge element, which is an XOR gate.

This controller uses the *2-phase* signaling, i.e., every transition of a signal is considered as a request or acknowledge. We have included the extended burst-mode specification in Fig. 10. If sel is sampled high when r (request) toggles, the controller toggles s_2, signaling the block C to begin a computation. When C finishes the computation, it toggles r_2; the controller then toggles a (acknowledge). However, if sel is sampled low when r toggles, then the controller toggles a directly.

The result of applying our synthesis method from [19] turned out to be remarkably similar to the naive design at the top of Fig. 10, which we found disappointing. Our hand designs were better, but also unsatisfactory. However, using this new BDD-

based approach, we were able to produce an extremely good result: the latency from r to a was just the delay from the select input to the output of a single multiplexor. The solution was generated by building a BDD so that the decision variable r is placed as close to the output a as possible, while satisfying other requirements to keep the circuit hazard-free. The final implementation is shown in Fig. 11.

It would be possible to generalize this idea by allowing the user to specify a set of particular paths to optimize, in order of priority [28], [29]. The variable ordering in the BDD could be chosen to put the high-priority inputs as close as possible to the output, subject to the correctness constraints imposed by TSO, etc.

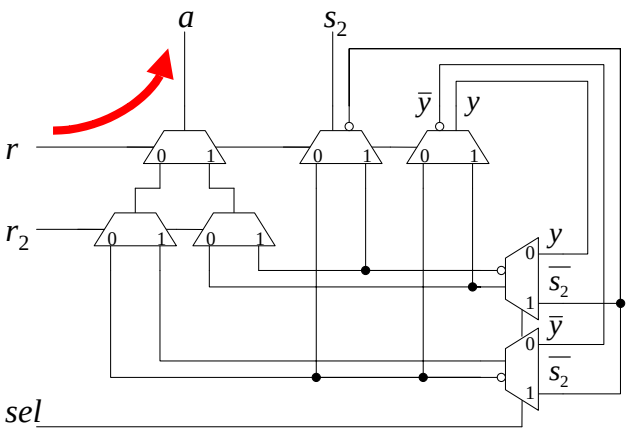


Fig. 11. Select/merge implementation.

V. Experiments

We modified the 3D synthesis tool described in [19], in particular, the hazard-free state assignment and combinational synthesis steps. We also extended the combinational synthesis tool described in [1] to handle extended burst-mode transitions, in particular, the generation of variable ordering constraints. We used these two modified tools in conjunction to perform experiments (see Table I) on an extensive set of benchmarks previously synthesized by the method described in [19]. The benchmarks are divided into four sets, corresponding to the four sections in Table I. The first set of benchmarks was taken from an academic SCSI controller design developed at Stanford [22]. The second set of benchmarks was taken from an asynchronous communications chip design developed at Hewlett Packard [13]. The third set of benchmarks came from an industrial SCSI controller design undertaken at AMD [30]. Finally, the last set of benchmarks corresponds to various other academic examples. We purposely included a large set of benchmarks to fully test the efficiency of the new method.

For the two-level circuits, we used the exact hazard-free logic minimizer described in [7]. The literal counts reported are prior to decomposition.[3] For the BDD-based circuits, a hazard-free inverting MUX requiring four literals was assumed (as shown in Fig. 4). We report the results after constant propagation has been applied. To satisfy the requirements imposed by Theorem 1, we used a random search algorithm to generate the variable orderings. It is worth noting that, although our synthesis method requires the usage of free BDDs[4] in some cases, none of the benchmark examples required a free BDD implementation. We believe that the area results will be further improved with the development of heuristic variable ordering algorithms tuned to our application. The CPU times for the BDD results are in the order of a few minutes.

Out of 39 examples synthesized, the new BDD-based solutions required less area than the previous method in 26 cases, primarily because of the reduction in the number of state variables due to simpler state assignment. In 12 cases, the area increased. In one case, the results are the same. It is interesting to note that on the benchmarks taken from practical designs, benchmarks corresponding to the first three sets in Table I [22], [13], [30], the BDD-based solutions were better in 11 cases and worse in 9 cases.

	Specification				State vars added		Total literals	
	States / Trans.		Prim. In	Prim. Out	*2L*	*BDD*	*2L*	*BDD*
tsend*	22	30	7	4	5	5	328	511
isend*	24	32	7	4	5	7	490	829
trcv*	16	22	7	4	3	2	175	111
ircv*	16	22	7	4	3	2	188	113
tsend-bm*	11	14	6	4	2	2	96	90
trcv-bm*	8	10	6	4	3	1	77	92
isend-bm*	12	15	6	4	3	3	177	88
ircv-bm*	8	10	6	4	3	1	80	61
tsend-csm*	11	14	6	4	4	3	92	66
trcv-csm*	8	10	6	4	3	2	70	70
isend-csm*	12	15	6	4	3	4	142	68
ircv-csm*	8	10	6	4	3	2	80	74
abcs	23	33	9	7	3	3	199	271
stetson-p1	31	38	13	14	3	3	376	455
stetson-p2	25	27	8	12	4	4	178	195
stetson-p3	8	11	4	2	1	0	16	9
biu-fifo2dma*	11	13	5	2	5	5	125	119
fifocellctrl	3	3	2	2	1	1	16	14
scsi-targ-send*	7	8	4	2	3	3	53	57
scsi-init-send*	7	8	4	2	2	2	31	43
scsi-init-rcv-sync	4	5	3	1	1	1	20	21
iccad93ex*	3	4	2	2	2	0	20	9
edac93ex*	4	5	3	2	2	1	32	17
condtest*	4	5	3	2	2	1	30	24
dff1*	4	6	2	2	2	0	28	17
dff2*	4	6	2	2	2	0	28	17
sr2*	8	12	2	3	3	2	82	37
sr2x2*	8	20	3	3	4	2	131	58
q42	4	4	2	2	1	1	27	15
select2ph*	4	8	2	2	2	0	42	32
selmerge2ph*	8	12	3	2	2	1	89	20
sin	13	17	3	4	3	4	71	77
ring-counter	8	8	1	2	1	1	45	68
binary-counter	32	32	1	4	3	3	94	70
binary-counter-co	32	32	1	5	3	3	104	80
pe-send-ifc	11	14	5	3	2	2	90	88
pe-rcv-ifc	12	15	4	4	3	2	84	68
dramc	12	14	7	6	1	0	71	54
cache-ctrl	38	49	16	19	1	1	704	1231

TABLE I
Experimental Results.

[3] In a number of cases, the two-level circuit contains some very large fanin gates. After decomposition, their literal counts should increase.

[4] The usage of free BDDs often severely limits the opportunity for subgraph sharing.

Output latency is also an important issue. Out of 39 examples evaluated, 24 of them (the names with *) previously required state variable changes before output changes for some of the specified transitions. In these cases, using BDD-based synthesis instead of two-level synthesis eliminated the need for the state burst to proceed sequentially before the output burst, which can help to reduce the output latency significantly.

Overall, the results indicate that both the BDD method and the two-level method are required to produce the best results. Since both methods have been implemented in the 3D synthesis tool, the user can evaluate both options.

VI. Conclusion

In this paper, we have presented a new synthesis method based on Binary Decision Diagrams for synthesizing extended burst-mode controllers. In contrast to earlier work on extended burst-mode synthesis [19] that aimed at two-level implementations, this new method has two significant advantages: it reduces the constraints on state minimization and assignment, which reduces the number of additional state variables required in many cases. Second, it eliminates the need for the state burst to precede the output burst, which reduces overall input to output latency. The method presented has been implemented and its effectiveness has been shown on a number of examples.

References

[1] B. Lin and S. Devadas, "Synthesis of hazard-free multilevel logic under multi-input changes from binary decision diagrams," *IEEE Transactions on Computer-Aided Design*, vol. 14, no. 8, pp. 974–985, Aug. 1995.

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

[3] T.-A. Chu, *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*, Ph.D. thesis, MIT Laboratory for Computer Science, June 1987.

[4] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E80-D, no. 3, pp. 315–325, Mar. 1997.

[5] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli, "Synthesis of hazard-free asynchronous circuits with bounded wire delays," *IEEE Transactions on Computer-Aided Design*, vol. 14, no. 1, pp. 61–86, Jan. 1995.

[6] C. J. Myers and T. H.-Y. Meng, "Synthesis of timed asynchronous circuits," *IEEE Transactions on VLSI Systems*, vol. 1, no. 2, pp. 106–119, June 1993.

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

[8] M. H. Sawasaki, C. Ykman-Couvreur, and B. Lin, "Externally hazard-free implementations of asynchronous control circuits," *IEEE Transactions on Computer-Aided Design*, vol. 16, pp. 835–848, Aug. 1997.

[9] P. Vanbekbergen, B. Lin, G. Goossens, and H. de Man, "A generalized state assignment theory for transformations on signal transition graphs," *Journal of VLSI Signal Processing*, vol. 7, no. 1/2, pp. 101–115, Feb. 1994.

[10] K. Y. Yun, *Synthesis of Asynchronous Controllers for Heterogeneous Systems*, Ph.D. thesis, Stanford University, Aug. 1994.

[11] B. Coates, A. Davis, and K. Stevens, "The Post Office experience: Designing a large asynchronous chip," *Integration, the VLSI journal*, vol. 15, no. 3, pp. 341–366, Oct. 1993.

[12] S. B. Furber, J. D. Garside, S. Temple, J. Liu, P. Day, and N.C. Paver, "AMULET2e: An asynchronous embedded controller," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Apr. 1997, pp. 290–299.

[13] A. Marshall, B. Coates, and P. Siegel, "Designing an asynchronous communications chip," *IEEE Design & Test of Computers*, vol. 11, no. 2, pp. 8–21, 1994.

[14] R. F. Sproull, I. E. Sutherland, and C. E. Molnar, "The counterflow pipeline processor architecture," *IEEE Design & Test of Computers*, vol. 11, no. 3, pp. 48–59, Fall 1994.

[15] J. A. Tierno, A. J. Martin, D. Borkovic, and T. K. Lee, "A 100-MIPS GaAs asynchronous microprocessor," *IEEE Design & Test of Computers*, vol. 11, no. 2, pp. 43–49, 1994.

[16] K. van Berkel, R. Burgess, J. Kessels, A. Peeters, M. Roncken, and F. Schalij, "A fully-asynchronous low-power error corrector for the DCC player," *IEEE Journal of Solid-State Circuits*, vol. 29, no. 12, pp. 1429–1439, Dec. 1994.

[17] T. E. Williams and M. A. Horowitz, "A zero-overhead self-timed 160ns 54b CMOS divider," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 11, pp. 1651–1661, Nov. 1991.

[18] K. Y. Yun, P. A. Beerel, V. Vakilotojar, A. E. Dooply, and J. Arceo, "The design and verification of a high-performance low-control-overhead asynchronous differential equation solver," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Apr. 1997, pp. 140–153.

[19] K. Y. Yun and D. L. Dill, "Unifying synchronous/asynchronous state machine synthesis," in *Proc. International Conf. Computer-Aided Design (ICCAD)*, Nov. 1993, pp. 255–260.

[20] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, Aug. 1986.

[21] S. M. Nowick and B. Coates, "UCLOCK: Automated design of high-performance asychronous state machines," in *Proc. International Conf. Computer Design (ICCD)*, Oct. 1994, pp. 434–441.

[22] K. Y. Yun and D. L. Dill, "Automatic synthesis of 3D asynchronous state machines," in *Proc. International Conf. Computer-Aided Design (ICCAD)*, Nov. 1992, pp. 576–580.

[23] S. H. Unger, *Asynchronous Sequential Switching Circuits*, Wiley-Interscience, John Wiley & Sons, Inc., New York, 1969.

[24] R. M. Fuhrer, B. Lin, and S. M. Nowick, "Symbolic hazard-free minimization and encoding of asynchronous finite state machines," in *Proc. International Conf. Computer-Aided Design (ICCAD)*, 1995, pp. 604–611.

[25] P. Ashar, S. Devadas, and K. Keutzer, "Gate-delay-fault testability properties of multiplexor-based networks," *Formal Methods in System Design*, vol. 2, no. 1, pp. 93–112, Feb. 1993.

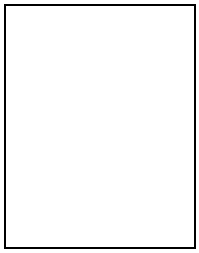
[26] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, *Logic Synthesis for Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1996.

[27] I. E. Sutherland, "Select/merge circuit," 1994, Private Communication.

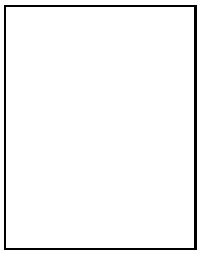
[28] P. A. Beerel, K. Y. Yun, and W.-C. Chou, "Optimizing average-case delay in technology mapping of burst-mode circuits," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Mar. 1996, pp. 244–260.

[29] K. W. James and K. Y. Yun, "Average-case optimized transistor-level technology mapping of extended burst-mode circuits," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Mar. 1998, pp. 70–79.

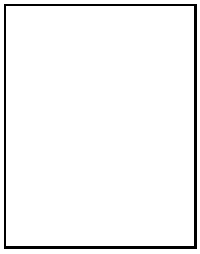
[30] K. Y. Yun and D. L. Dill, "A high-performance asynchronous SCSI controller," in *Proc. International Conf. Computer Design (ICCD)*, Oct. 1995, pp. 44–49.



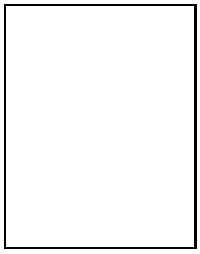
Kenneth Y. Yun is currently an assistant professor in the Dept. of Electrical and Computer Engineering at University of California, San Diego. He has a Ph.D. in Electrical Engineering from Stanford University and an S.M. in Electrical Engineering and Computer Science from MIT. He had held design engineering positions at TRW and Hitachi for 6 years. His current research interests include the design, synthesis, analysis, and verification of mixed-timed VLSI circuits and systems: in particular, interface design methodologies and tools to facilitate ultra-high-speed communications between synchronous/asynchronous modules. He has been working with Intel Corp. as a primary consultant on the Asynchronous Instruction Decoder Project. He has organized ASYNC'98 as a program co-chair. Dr. Yun is a recipient of 1996-99 National Science Foundation CAREER award and 1996 Hellman Faculty Fellowship, and a co-recipient of the Charles E. Molnar award for a paper that best bridges theory and practice of asynchronous circuits and systems at ASYNC'97.



Bill Lin has a BS, a MS, and a Ph.D in Electrical Engineering and Computer Sciences from the University of California, Berkeley. He is currently with the Department of Electrical and Computer Engineering at the University of California, San Diego, where he is affiliated with the Center for Wireless Communications. His current research interests are primarily in the areas of novel IC architectures for telecom applications and electronic design automation for designing them. Prior to joining the faculty at UCSD, he was heading a research team at IMEC, Belgium, working on various aspects of design automation and implementation techniques for embedded systems. His research has led to over 80 journal and conference publications. He has received a number of publication awards, including a best paper award at the 1987 Design Automation Conference, distinguished paper citations at the 1989 IFIP VLSI Conference and the 1990 International Conference on Computer-Aided Design, a best paper nomination at the 1998 Conference on Design Automation and Test in Europe , and the 1995 IEEE Transactions on VLSI Systems best paper award. He has served on panels and given invited presentations at several major conferences, including the International Conference on Computer-Aided Design. He has served on program committees of several conferences and workshops, including the Design Automation Conference and the European Design and Test Conference.



David L. Dill is Associate Professor of Computer Science and, by courtesy, Electrical Engineering at Stanford University. He has been on the faculty at Stanford since 1987. He has an S.B. in Electrical Engineering and Computer Science from the Massachusetts Institute of Technology (1979), and an M.S and Ph.D. from Carnegie-Mellon University (1982 and 1987). His primary research interests relate to the theory and application of formal verification techniques to system designs, including hardware, protocols, and software. Prof. Dill's Ph.D. thesis, "Trace Theory for Automatic Hierarchical Verification of Speed Independent Circuits" was named as a Distinguished Dissertation by ACM and published as such by M.I.T. Press in 1988. He was the recipient of an Presidential Young Investigator award from the National Science Foundation in 1988, and a Young Investigator award from the Office of Naval Research in 1991. He has received Best Paper awards at International Conference on Computer Design in 1991 and the Design Automation Conference in 1993 and 1998. From July 1996 to September 1997 he was Chief Scientist of 0-In Design Automation.



Srinivas Devadas received a B. Tech in Electrical Engineering from the Indian Institute of Technology, Madras in 1985 and a MS and Ph.D in Electrical Engineering from the University of California, Berkeley, in 1986 and 1988 respectively. Since August 1988, he has been at the Massachusetts Institute of Technology, Cambridge, and is currently an Associate Professor of Electrical Engineering and Computer Science. He held the Analog Devices Career Development Chair of Electrical Engineering from 1989 to 1991. In 1992, he received a NSF Young Investigator Award. Prof. Devadas' research interests span all aspects of synthesis of VLSI circuits, with emphasis on optimization techniques for synthesis at the logic, layout and architectural levels, design for low power, testing of VLSI circuits, formal verification, hardware/software co-design, design-for-testability methods and interactions between synthesis and testability of VLSI systems. Prof. Devadas has authored or co-authored over 150 technical papers in journals and conferences, and has co-authored four books. He has received six Best Paper awards at CAD conferences and journals, including the 1990 IEEE Transactions on CAD and the 1996 IEEE Transactions on VLSI Systems Best Paper awards. He has served on the technical program committees of several conferences and workshops including the Int'l Conference on Computer Design, and the Int'l Conference on Computer-Aided Design. He serves on the Editorial Board of ACM TODAES, Formal Methods in VLSI Design, and Design Automation of Embedded Systems journals. Prof. Devadas is a member of ACM.