Performance-driven Synthesis of Asynchronous Controllers

Kenneth Y. Yun*

Department of ECE, University of California, San Diego, CA 92093

Bill Lin†

IMEC, Kapeldreef 75, B-3001 Leuven, Belgium

David L. Dill

Computer Systems Laboratory, Stanford University, Stanford, CA 94305

Srinivas Devadas‡

Department of EECS, MIT, Cambridge, MA 02139

Abstract

We examine the implications of a new hazard-free combinational logic synthesis method [10], 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, the use of the BDD-based synthesis reduces the constraints on state minimization and assignment, which reduces the number of additional state variables required in many cases. Second, 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. Third, selection variables can easily be ordered to minimize the latency on a user-specified path, which is important for optimizing the performance of systems that use asynchronous components. We present extensive evaluations showing that, with only minimal optimization, the BDD-based synthesis gives results comparable in area to previous exact two-level synthesis method. We also give a detailed example of the specified path optimization.

1 Introduction

There have been many recent advances in asynchronous circuits and systems, both in tool design [1, 2, 4, 6, 9, 11, 13, 15, 16, 17, 18, 23, 24, 25] and actual systems design [3, 7, 8, 12, 13, 14, 19, 20]. 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 [26, 27].

This paper describes a new synthesis algorithm for asynchronous controllers specified in extended-burst-mode [26, 27]. This algorithm assumes the target implementation to be a combinational circuit with both

*Supported by the Semiconductor Research Corporation, Contract no. 93-DJ-205.

†Supported by the EXACT project under the program ESPRIT 6143 of the European Commission

‡Supported by the NSF Young Investigator Award

primary outputs and state variables fed back. The combinational circuit is derived from a Binary Decision Diagram [5] using a recently developed hazard-free combinational synthesis method [10]. This algorithm is designed, first and foremost, to minimize output latency.

This new approach has many advantages over other synthesis methods [18, 17, 26, 25], which implement the combinational logic as two-level AND-OR circuits. In many cases, the circuits synthesized using this new method have considerably lower output latencies than the circuits synthesized by the method in [26]. Furthermore, this method provides a platform 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.

2 Controller Specification and Implementation

In this section, we summarize a design style, called extended burst-mode, and an implementation style, called 3D, to specify and implement asynchronous controllers for heterogeneous systems — systems with both synchronous and asynchronous components. These topics have been discussed in more detail in previous publications [26, 25].

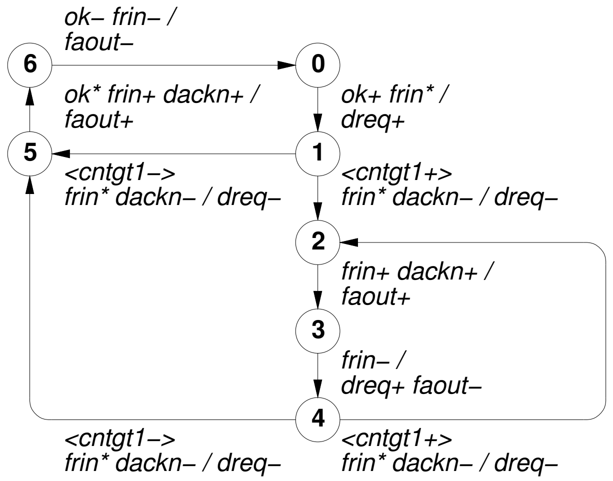

Figure 1: Extended-burst-mode specification.

Figure 1 shows an example of an extended burst-mode specification. Signals not enclosed in angle brackets and ending with + or − are *terminating signals*. These are edge signals. The signals enclosed in angle brackets are *conditionals*, which are level signals whose values are sampled when all of the terminating edges associated with them have occurred. A conditional $\langle a^+\rangle$ can be read "if a is high" and $\langle a^-\rangle$ can be read "if a is low." A state transition occurs only if all of the conditions are met and all the terminating edges have appeared. A signal ending with an asterisk is a *directed don't care.* If a is a directed don't care, there must be a sequence of state transitions in the machine labeled with a^*. If a state transition is labeled with a^*, the following state transitions in the machine must be labeled with a^* or with a^+ or a^- (the terminating edge for the directed don't care). This specification describes a state machine having a conditional input (cntgt1), 3 edge inputs (ok, frin, dackn), and 2 outputs (dreq, faout). Consider the state transitions out of state 4. The behavior of the machine at this point is: "if cntgt1 is low when dackn falls, change the current state from 4 to 5 and lower the output dreq; if cntgt1 is high when dackn falls, change the current state from 4 to 2 and lower output dreq."

A directed don't care may change at most once during a sequence of state transitions it labels, i.e., directed don't cares are *monotonic* signals, and, if it doesn't change during this sequence, it must change during the state transition its terminating edge labels. A terminating edge which is not immediately preceded by a directed don't care is called *compulsory*, since it *must* appear during the state transition it labels. frin is low when the specification is in state 4. It can rise at any point as the machine moves through states 5 and 6 or through state 2, depending on the level of cntgt1, and it must have risen by the time the machine moves to states 6 or 2, because the terminating edge frin$^+$ appears between states 5 and 6 and between 4 and 2.

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

More generally, an extended-burst-mode asynchronous finite state machine [26, 27] is specified by a state diagram which consists of a finite number of states, a set of labeled state transitions connecting pairs of states, and a start state. Each state transition is labeled with a set of conditional signal levels and two sets of signal edges: an input burst and an output burst. An *output burst* is a set of output edges, and an *input burst* is a non-empty set of input edges (terminating or directed don't care), at least one of which must be *compulsory*.

In a given state, when all the specified conditional signals have correct values and when all the specified terminating edges in the input burst have appeared, 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 before any compulsory edge in the input burst appears and must hold their values until after all of the terminating edges appear. The minimum delay from the conditional stabilizing to the first compulsory edge is called the *setup time*. Similarly, the minimum delay from the last terminating edge to the conditional changing is called the *hold time*. Actual values of *setup* and *hold* time of conditional signals with respect to "sampling" edges depend on the implementation. Conditional signal levels need not be stable outside of the specified sampling periods. Each signal specified as a directed don't care may change its value monotonically at any time including during output bursts, unless it is already at the level specified by the next terminating edge. 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. This requirement — the environment must wait generating the next set of compulsory edges until the circuit stabilizes — called the *fundamental-mode environmental constraint*.

There is an additional restriction to extended-burst-mode specifications, called the *distinguishability constraint*, which prevents ambiguity among multiple input bursts emanating from a single state: For every pair of input bursts i and j from the same state, either the conditions are mutually exclusive, or the set of compulsory edges in i is not a subset of the set of all possible edges in j.

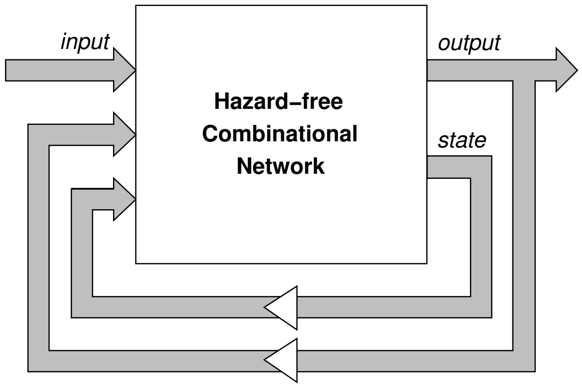

Figure 2: 3D asynchronous state machine.

2.1 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 $\delta : X \times Y \times Z \to Y \times Z$ is a *next-state function*. The hardware implementation of a 3D state machine (see figure 2) is a combinational network, which implements the next-state function, with the outputs of the network fed back as inputs to the network. There are no explicit storage elements such as latches, flip-flops or C-elements in a 3D machine.

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 machine cycle 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, which may be empty.

2.2 2-level Synthesis

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

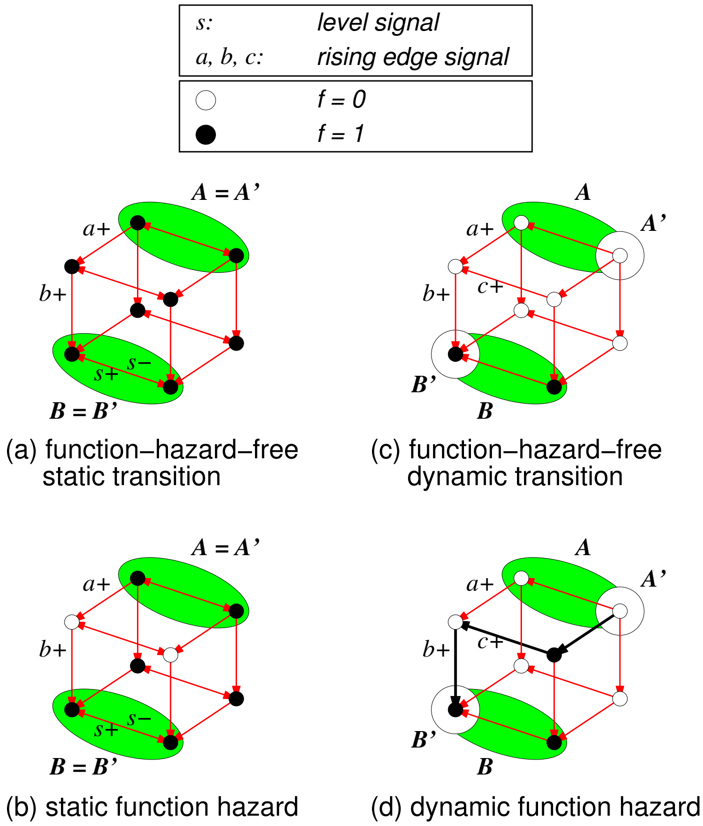

Figure 3: Generalized transition cube $[A, B]$.

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. Below is the definition of function hazard adapted for generalized transitions:

DEFINITION 2.1 *A combinational function f contains a* ***function hazard*** *during a generalized transition (T, A, B) iff*

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

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

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, $f(X) = f(Y)$.

THEOREM 2.1 *Every extended-burst-mode transition is function-hazard-free.*

Proof: Consider f during an extended-burst-mode transition from A to B. Since $A \subseteq A \cup [A, B)$, for every pair of minterms X and Y in A, $f(X) = f(Y)$ by requirement 1 of the definition of extended-burst-mode transition. This contradicts criterion 1 of Definition 2.1. For every pair of minterms X and Y in B, $f(X) = f(Y)$ by requirement 2, which contradicts criterion 2 of Definition 2.1. Finally, for all X in (A, B), $f(X) = f(A)$ by requirement 1, which contradicts criterion 3 of Definition 2.1. Therefore, f is free of function hazards. ■

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.

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 [26, 17] must be satisfied for each burst, i.e., extended-burst-mode transition. It was shown in [26] 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 the *single transition time* (STT) state assignment [22] is used. The approach taken in [26] 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, and also tended to increase the circuit area.

3 BDD-based Combinational Synthesis

In this paper, we use a new BDD based combinational synthesis technique from [10]. This approach imposes a different set 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 [26], resulting in greatly reduced latency in many cases.

Combinational networks that describe next-state functions are constructed from a BDD (binary decision diagram) description. The basic gates that comprise combinational networks are ANDs, ORs, NANDs, NORs, inverters, 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. We do not rely on inertial delays, that is, we assume a *pure gate delay* model, and allow for arbitrary wire delays.

3.1 Multiplexer Networks derived from BDDs

The following definition of a Binary Decision Diagram is from [5].

DEFINITION 3.1 *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* $\mathsf{low}(v), \mathsf{high}(v) \in V$*. 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 3.2 *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*
   $f_v(x_1, \ldots, x_n) = \overline{x_i} \cdot f_{\mathsf{low}(v)}(x_1, \ldots, x_n) + x_i \cdot f_{\mathsf{high}(v)}(x_1, \ldots, x_n)$.
   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* $\mathsf{low}(v) \neq \mathsf{high}(v)$ *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 $\mathsf{low}(v)$ is a non-terminal, then $\mathsf{index}(v) < \mathsf{index}(\mathsf{low}(v))$, and if $\mathsf{high}(v)$ is a non-terminal, then $\mathsf{index}(v) < \mathsf{index}(\mathsf{high}(v))$.

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.

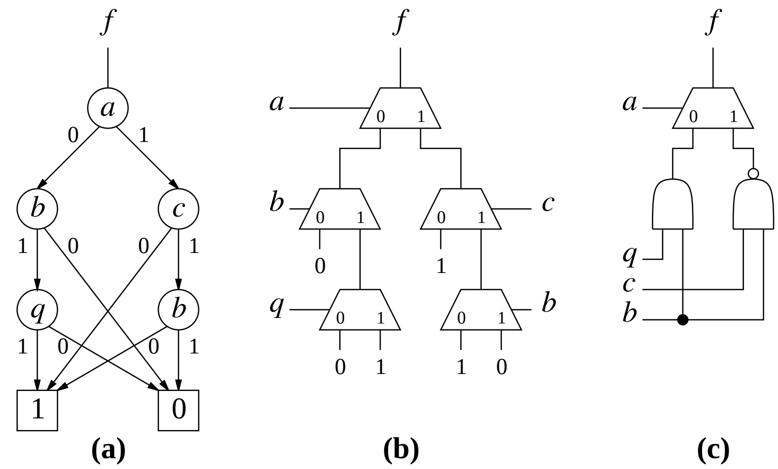

Figure 4: (a) BDD (b) MUX network derived from BDD (c) Simplified network (by constant propagation).

A multi-level network can be derived directly from a BDD by replacing each vertex with a two-input MUX with the decision variables as the *select* inputs of the MUXes. Figure 4b shows a MUX network derived from the BDD in figure 4a. 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. Figure 4c shows an equivalent network after the constant propagation.

3.2 Hazard-free Combinational Synthesis

We use the approach from [10] 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 [10] 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 [10] 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 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 [10] 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 3.1 *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 $[A, B]$.*

> *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 in (T, A, B), $X \in B$ 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, $f_s(X) = f_{\overline{s}}(X)$. ∎

DEFINITION 3.3 *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).

LEMMA 3.2 *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 3.1 *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 3.2 *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 3.1 *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, the Shannon cofactors of f with respect to $\overline{s}$ and 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 3.2. 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 3.1, (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 3.1, $f(X) = f_s(X) = f_{\overline{s}}(X)$ 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 $0 \to 0$ or both $1 \to 1$, 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 3.2, $f_{\overline{s}}$ undergoes a static transition. 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.

■

LEMMA 3.3 *There always exists a BDD that satisfies the variable ordering requirements for 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, 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 before any variable 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. 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 $g_{\bar{i}}$.

Let (T, A, B) be the input burst transition of (u, v). Since (T, A, B) is an extended-burst-mode transition for g, $(T, A_{\bar{i}}, B_{\bar{i}})$ is an extended-burst-mode transition for $g_{\bar{i}}$, by Lemma 3.2. By Corollary 3.2, $(T, A_{\bar{i}}, B_{\bar{i}})$ is a static transition for $g_{\bar{i}}$. Thus, there is no ordering requirement in the sub-BDD $g_{\bar{i}}$ for $(T, A_{\bar{i}}, B_{\bar{i}})$. 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. ■

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, there is always a variable that can appear first. This variable partitions the function into a left and right BDD. The left and right BDDs do not have to have the same variable order, so they can be constructed recursively using the same method.

In this 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. State minimization is also constrained to avoid creating variable ordering conflicts.

4 Sequential Synthesis Procedure

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

4.1 Hazard-free State Assignment

We describe an algorithm for simultaneously minimizing and assigning states which always results in a hazard-free single-transition-time (STT) state assignment. In contrast, the previous algorithm for extended-burst-mode [26] 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. 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.

In order to describe the primitive next-state table construction formally, a formal definition of the extended-burst-mode specification, adapted from the definition of the burst-mode specification in [17], is needed. 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; cond labels each state transition with a set of conditional inputs; in and out are labeling functions used to define the unique entry cube of each state. The function $\mathsf{cond} : E \rightarrow \{0, 1, *\}^l$ defines the values of the conditional inputs. The function $\mathsf{in} : V \rightarrow \{0, 1, *\}^m$ defines the values of the edge inputs and the function $\mathsf{out} : V \rightarrow \{0, 1\}^n$ defines the values of the outputs upon entry to each state.

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

The unique entry condition is satisfied by the above definition. The remaining requirements to ensure well-formed specifications are:

- Every input burst must contain a compulsory edge. That is, for every state transition (u, v), there exists $x_i \in \mathsf{trans}_{IN}(u, v)$ such that $\mathsf{in}_i(u) \neq * \wedge \mathsf{in}_i(v) \neq *$.

- Every pair of transitions emanating from the same state must satisfy the distinguishability constraint. That is, for every pair, $(u,v),(u,w) \in E$, $\mathsf{ctrans}_{IN}(u,v) \subseteq \mathsf{trans}_{IN}(u,w)$ implies that either $v = w$ or $\mathsf{cond}(u,v)$ and $\mathsf{cond}(u,w)$ are mutually exclusive, that is, there exists k such that $\mathsf{cond}_k(u,v) \neq \mathsf{cond}_k(u,w) \wedge \mathsf{cond}_k(u,v) \neq * \wedge \mathsf{cond}_k(u,w) \neq *$.

- For every sequence of state transitions, $u \to v_1 \to \cdots \to v_n \to w$, with $n \geq 1$ and $\mathsf{in}_i(u) = \mathsf{in}_i(w) \neq *$, there exists $k \in 1,\ldots,n$ such that $\mathsf{in}_i(v_k) \neq *$.

Given an extended-burst-mode specification, G, as described above, let W be the set of conditional input bit vectors $\{(c_1,\ldots,c_l) \mid c_i \in \{0,1\},\, i \in 1,\ldots,l\}$, X be the set of edge input bit vectors $\{(x_1,\ldots,x_m) \mid x_j \in \{0,1\},\, j \in 1,\ldots,m\}$, and Y be the set of output bit vectors $\{(y_1,\ldots,y_n) \mid y_k \in \{0,1\},\, k \in 1,\ldots,n\}$. A *primitive next-state table* is defined as $T = (V,W,X,Y,\delta,\lambda)$, where V is the set of specification states, and $\delta : V \times W \times X \times Y \to V \cup \{*\}$ and $\lambda : V \times W \times X \times Y \to \{0,1,*\}^n$ define the *next specification state function* and the *next output function* respectively.

Below, we describe the next-state assignment. $L(u)$ is a symbolic code assigned to state u. $[\mathsf{cond}(u,v),\mathsf{in}(v),\mathsf{out}(u),L(u)]$ denotes a cube in $\{0,1,*\}^{l+m+n+r}$, where r is the length of state variable bit vectors. Of course, the actual value of r will not be determined until the state encoding is done. $*$ in the place of conditional inputs is a shorthand notation, meaning that all conditional inputs are $*$. in' is defined as: for all $j \in 1,\ldots,m$,

$$\mathsf{in'}_j(u) = \begin{cases} \mathsf{in}_j(u) & \text{if } \exists (u,v) : \mathsf{in}_j(v) \neq * \\ * & \text{otherwise} \end{cases}.$$

The next-state assignment is as follows: For all $k \in 1,\ldots,n$ and for every state transition, (u,v)

- Conditional input burst setup:
  for all M in $[A,B]$, $\lambda_k(M) = \mathsf{out}_k(u)$ and $\delta(M) = u$, where
  $A = [*,\, \mathsf{in}(u),\, \mathsf{out}(u),\, L(u)]$,
  $B = [*,\, \mathsf{in}'(u),\, \mathsf{out}(u),\, L(u)]$.

- Input burst:
  for all M in $[A,B)$, $\lambda_k(M) = \mathsf{out}_k(u)$ and $\delta(M) = u$, where
  $A = [\mathsf{cond}(u,v),\, \mathsf{in}(u),\, \mathsf{out}(u),\, L(u)]$,
  $B = [\mathsf{cond}(u,v),\, \mathsf{in}(v),\, \mathsf{out}(u),\, L(u)]$.

- Output/state burst:
  for all M in $[A,B]$, $\lambda_k(M) = \mathsf{out}_k(v)$ and $\delta(M) = v$, where
  $A = [\mathsf{cond}(u,v),\, \mathsf{in}(v),\, \mathsf{out}(u),\, L(u)]$,
  $B = [\mathsf{cond}(u,v),\, \mathsf{in}(v),\, \mathsf{out}(v),\, L(v)]$.

4.2 Hazard-free State 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, as described below.

DEFINITION 4.1 *States u and v in V are **compatible** (denoted as $u \sim v$) iff 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. $\delta(u, s) = * \vee \delta(v, s) = * \vee \delta(u, s) \sim \delta(v, s)$ and

3. *if* $\mathsf{out}_k(u) = \mathsf{out}_k(v)$ *for all* $k \in 1, \ldots, n$, *then for every pair of state transitions,* (u, w_u) *and* (v, w_v), i *is a terminating signal in* (u, w_u) *and* j *is a don't care in* (u, w_u) *imply that* 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* $\mathsf{in}_j(w_v) = *$.

Informally, the third 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.

The state minimization and encoding to complete the sequential synthesis are described in [25].

5 Path Optimization Example

In synchronous designs, one of the important design objectives is to carefully balance the computation blocks so that no part of the circuits are 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.

To illustrate this point and also provide a nice circuit example, we consider a hypothetical problem posed by Ivan Sutherland [21]. When the circuit in figure 5 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 signal 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 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 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 figure 5. 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 [26] turned out to be remarkably similar to the naive design at the top of figure 5, 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 figure 6.

It would be very easy to generalize this idea by allowing the user to specify a set of particular paths to optimize, in order of priority. 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.

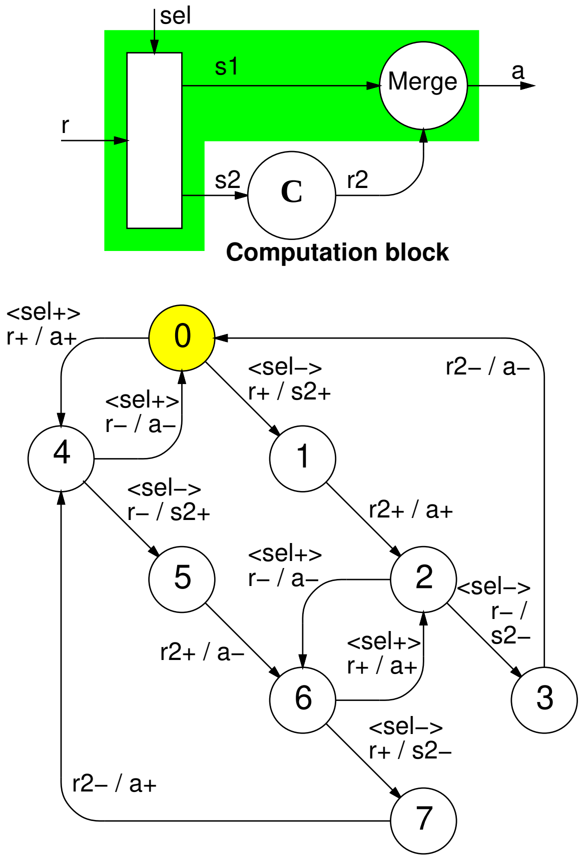

Figure 5: Select/merge example.

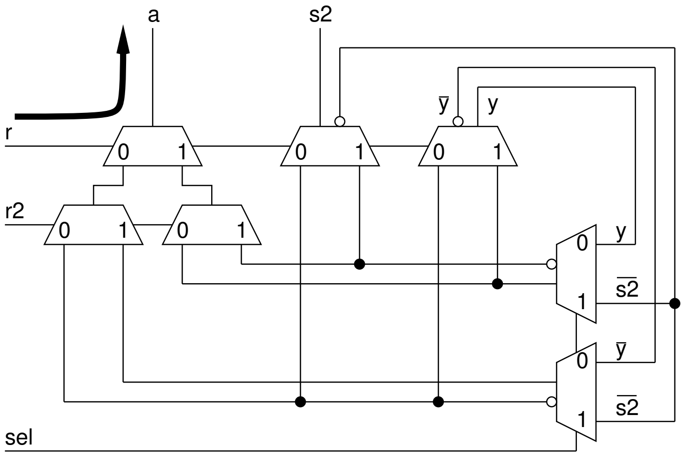

Figure 6: Select/merge implementation.

	Specification				State vars added		Total literals	
	States / Transitions		Primary In	Primary Out	*2-level*	*BDD*	*2-level*	*BDD*
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
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

Table 1: Experimental Results.

6 Experiments

We modified the 3D synthesis tool described in [26], in particular, the hazard-free state assignment and combinational synthesis steps. We used this modified tool in conjunction with the combinational synthesis tool [10] to perform experiments (see table 1) on many examples previously synthesized by the method described in [26]. With very modest efforts to find the optimal variable order (we tried a few random orderings and picked the best result), most of the examples required less area than the previous method, primarily because of the reduction in the number of state variables due to simpler state assignment. For a minority of the examples, area increased somewhat. We believe that the area results will be further improved with the development of heuristic variable ordering algorithms tuned to our application.

A more important issue is output latency. Out of 39 examples synthesized, 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 rather than 2-level synthesis improved the output latency by 100% or more.

References

[1] V. Akella and G. Gopalakrishnan. SHILPA: A high-level synthesis system for self-timed circuits. In *Proceedings of the 1992 IEEE/ACM International Conference on Computer Aided Design*, pages 587–591, November 1992.

[2] P. Beerel and T. H.-Y. Meng. Automatic gate-level synthesis of speed-independent circuits. In *Proceedings of the 1992 IEEE/ACM International Conference on Computer Aided Design*, November 1992.

[3] E. Brunvand. The NSR processor. In *Proceedings of the Twenty-Sixth Annual Hawaii International Conference on System Sciences*, volume I, pages 428–435. IEEE Computer Society Press, January 1993.

[4] E. Brunvand and R. F. Sproull. Translating concurrent programs into delay-insensitive circuits. In *Proceedings of the 1989 IEEE International Conference on Computer Aided Design*. IEEE Computer Society Press, 1989.

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

[6] T.-A. Chu. *Synthesis of Self-timed VLSI Circuits from Graph-Theoretic Specifications*. PhD thesis, MIT, 1987. Technical Report MIT-LCS-TR-393.

[7] W. S. Coates, A. L. Davis, and K. S. Stevens. The Post Office experience: Designing a large asynchronous chip. *INTEGRATION, the VLSI Journal*, 15(4):341–366, 1993.

[8] S. B. Furber. AMULET1 – an asynchronous ARM processor. In *Symposium Record of Hot Chips V*, Stanford, CA, August 1993.

[9] L. Lavagno and A. Sangiovanni-Vincentelli. *Algorithms for Synthesis and Testing of Asynchronous Circuits*. Kluwer Academic Publishers, 1993.

[10] B. Lin and S. Devadas. Synthesis of hazard-free multi-level logic implementations under multiple-input changes from binary decision diagrams. In *Proceedings of the 1994 IEEE/ACM International Conference on Computer Aided Design*. IEEE Computer Society Press, November 1994.

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

[12] A. J. Martin, S. M. Burns, T.K. Lee, D. Borković, and P.J. Hazewindus. The design of an asynchronous microprocessor. In C. L. Seitz, editor, *Proceedings of the Decennial Caltech Conference on Very Large Scale Integration*, pages 351–373. MIT Press, 1989.

[13] T. H. Meng. *Synchronization Design for Digital Systems*. Kluwer Academic Publishers, 1990.

[14] C. E. Molnar, T.-P. Fang, and F. U. Rosenberger. Synthesis of delay-insensitive modules. In Henry Fuchs, editor, *1985 Chapel Hill Conference on Very Large Scale Integration*, pages 67–86. Computer Science Press, Inc., 1985.

[15] C. W. Moon. *Synthesis and Verification of Asynchronous Circuits from Graph Specifications*. PhD thesis, University of California, Berkeley, 1992.

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

[17] S. M. Nowick. *Automatic Synthesis of Burst-mode Asynchronous Controllers*. PhD thesis, Stanford University, 1993.

[18] S. M. Nowick and B. Coates. Automated design of high-performance unclocked state machines. In *Proceedings of the 1994 IEEE International Conference on Computer Design: VLSI in Computers and Processors*. IEEE Computer Society Press, October 1994.

[19] S. M. Nowick, M. E. Dean, D. L. Dill, and M. Horowitz. The design of a high-performance cache controller: a case study in asynchronous synthesis. *Integration, The VLSI Journal*, 15(3):241–262, October 1993.

[20] S. M. Nowick, K. Y. Yun, and D. L. Dill. Practical asynchronous controller design. In *Proceedings of the 1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 341–345. IEEE Computer Society Press, October 1992.

[21] I. E. Sutherland, 1994. Private Communication.

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

[23] P. Vanbekbergen. *Synthesis of asynchronous controllers from graph-theoretic specifications*. PhD thesis, Interuniversitair Micro-Elektronica Centrum, 1993.

[24] C. Ykman-Couvreur, B. Lin, G. Goossens, and H. De Man. Synthesis and optimization of asynchronous controllers based on extended lock graph theory. In *Proceedings of The European Conference on Design*

Automation with The European Event in ASIC Design, pages 512–517. IEEE Computer Society Press, February 1993.

[25] K. Y. Yun and D. L. Dill. Automatic synthesis of 3D asynchronous finite-state machines. In *Proceedings of the 1992 IEEE/ACM International Conference on Computer Aided Design*, pages 576–580. IEEE Computer Society Press, November 1992.

[26] K. Y. Yun and D. L. Dill. Unifying synchronous/asynchronous state machine synthesis. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer Aided Design*, pages 255–260. IEEE Computer Society Press, November 1993.

[27] Kenneth Y. Yun. *Synthesis of Asynchronous Controllers for Heterogeneous Systems*. PhD thesis, Stanford University, 1994. Technical Report CSL-TR-94-644.