# Average-Case Technology Mapping of Asynchronous Burst-Mode Circuits

Wei-Chun Chou, *Student Member, IEEE,* Peter A. Beerel, *Member, IEEE,* and Kenneth Y. Yun, *Member, IEEE*

*Abstract*— **This paper presents a technology mapper that optimizes the average performance of asynchronous burst-mode control circuits. More specifically, the mapper can be directed to minimize either the average latency or the average cycle time of the circuit. The input to the mapper is a burst-mode specification and its NAND-decomposed unmapped network. The mapper pre-processes the circuit's specification using stochastic techniques to determine the relative frequency of occurrence of each state transition. Then, it maps the NAND-decomposed network using a given library of gates. Of many possible mappings, the mapper selects a solution that minimizes the sum of the delays (latency or cycle time) of all state transitions, weighted by their relative frequencies, thereby optimizing for average performance. We present experimental results on a large set of benchmark circuits, which demonstrate that our mapped circuits have significantly lower average latency and cycle time than comparable circuits mapped with a leading conventional mapping technique which minimizes the worst-case delay. Moreover, these performance improvements can be achieved with manageable run-times and significantly smaller area.**

*Keywords*— **Technology mapping, burst-mode asynchronous circuits, average-case optimization.**

## I. Introduction

Asynchronous circuits are event-driven. That is, they can respond immediately to the completion of a computation without waiting for a clock edge. Since the computation delays are, in general, data-dependent, the performance of asynchronous circuits should be characterized by their average-case behavior, not their worst-case behavior, which is the gage used for synchronous circuits. Therefore, the performance goal of technology mapping techniques for asynchronous circuits should be to maximize the average-case speed, unlike synchronous circuits which are mapped to minimize the worst-case delays [7], [13], [18], [28], [33], [34].

Furthermore, most existing technology mappers for asynchronous circuits focus on ensuring hazard-freedom [17], [21], [31], [30] without regards to performance. One reason for this apparent lack of research is that an explicit technology mapping step is not needed in all asynchronous circuit design methodologies. Specifically, many asynchronous design methodologies rely on the existence of specialized cell libraries which make the technology mapping step trivial [10], [19], [32], [4]. For control-oriented synthesis methodologies, such as burst-mode designs, however, the technology mapping using existing standard-cell libraries is often an important part of the design process. In fact, the motivation for this work arises from the Asynchronous Instruction Length Decoder Project at Intel Corporation [26]. The lack of tools resulted in manual mapping of control circuits using back-of-the-envelope techniques, which was both labor-intensive and sub-optimal.

This paper proposes the first known technique to optimize the average-case performance of a popular form of asynchronous controllers called burst-mode control circuits [22]. The control circuits are specified in extended burst-mode (XBM) [39] and implemented using a modified Huffman architecture [22], [36], [39]. Each circuit consists of a combinational logic block with some outputs fed-back to the inputs through delay elements. The XBM specification and an implementation of SCSI-INIT-SEND controller are shown in Fig. 1 as an example.

In each state, the circuit waits for a set of specified input transitions, referred to as an *input burst*; upon detecting specified input transitions, it toggles a set of output signals, referred to as an *output burst*. In addition, depending on the implementation style, the circuit may internally toggle a number of state signals either concurrently with output signals or in a separate *state burst* before or after the output burst [36]. For the machine to operate properly, the environment must obey the generalized *fundamental-mode constraint* which essentially states that the next input burst must arrive *after* the fed-back signals have propagated deeply enough into the combinational logic block, in order to ensure that there are no unexpected hazards [6], [36].

There are two parameters often used to characterize the performance of burst-mode circuits: *latency* and *cycle time*. The latency of a state transition is the *maximum* delay from the transitions on primary inputs to the transitions on primary outputs. If the state signals toggle concurrently with outputs or after outputs, the latency is determined by the delay of the combinational logic. If the state signals toggle before the outputs, however, the latency includes the delay through the feedback delay elements. The cycle time of a state transition is the *minimum* delay necessary between the end of the input burst and the beginning of the input burst of the next state transition. Note that the cycle time of a state transition is the sum of its latency and the associated fundamental-mode constraint.

For many applications, the optimization of latency is more important than that of cycle-time. Consider a situation in which controllers are responsible for controlling the sequence of various datapath operations [38]. In these applications, the environmental response time corresponds to the delay of the datapath units and thus is relatively long. Consequently, the fundamental-mode constraint is typically easily met and the system performance is determined by the sum of the datapath and controller latencies. Thus, in these applications, the burst-mode cycle time

This research was funded in part by a Charles Lee Powell Foundation Research Equipment Grant for Young Faculty of the USC School of Engineering, a Zumberge Fellowship for Assistant Professors at USC, NSF CAREER Award MIP-9502386, NSF Grant No. CCR-9812164, and a gift from Intel Corporation. The work by K. Y. Yun was supported in part by a gift from Intel Corporation and NSF CAREER Award MIP-9625034.

W-C. Chou and P. A. Beerel are with the Electrical Engineering-Systems Dept., University of Southern California, Los Angeles, CA 90089, USA; Email: {wchou,pabeerel}@eiger.usc.edu.

K. Y. Yun is with the Electrical and Computer Engineering Dept., University of California, San Diego, La Jolla, CA 92093-0407, USA; Email: kyy@ucsd.edu.

does not adversely affect system performance.

In control-dominated applications, on the other hand, the environmental response time of a controller may just be the latency through another controller and thus can be small. In those cases, the fundamental-mode constraints are harder to meet, and, in some cases, delays must be added to increase the environmental response time. Ideally, these delays should be placed inside the communicating controller such that they delay only the response time of those state transitions that violate their fundamental-mode constraints. Otherwise, it may be necessary to add the delays to the output signals to meet the worst-case fundamental-mode constraint, which unfortunately slows down the environment's response time *uniformly* for every state transition. In this paper, we assume that the environment is ideal and thus our goal is to minimize the controller's average cycle time. Extensions to minimize the worst-case fundamental-mode constraint are straightforward but not discussed here.

Because an output signal is often specified to change in multiple state transitions, its logic may contain different paths that are excited in different state transitions. To optimize for average performance, the delay along these paths should be prioritized according to the relative frequencies of the associated state transitions. In order to achieve this, we assume that conditional probabilities of all state transitions are either provided by the user or estimated through a behavioral simulation of the circuit in its environment. In the SCSI-INIT-SEND circuit, the conditional probabilities depend on the packet size of SCSI transfers. For example, using a reasonable packet size of 1K bytes, the conditional probabilities from state 3 to state 4 and to state 6 would be 0.999 and 0.001 respectively. Our mapper pre-processes these conditional probabilities, using a Markov chain analysis, to obtain the relative priority of each state transition.

Our technology mapper maps only the combinational logic portion of the controller. However, the mapping choices, and thus the structure of the combinational logic, affect the amount of feedback delays required and the fundamental-mode constraint. To account for this, we use a combination of techniques used in [6] and event-driven simulation to estimate the required feedback delay and fundamental-mode constraint during the mapping procedure. Consequently, our mapper can optimize both the latency and the cycle time (the choice is user-specified).

Our mapper performs two steps: decomposition and covering. In the decomposition step, optimized circuit equations are decomposed into a set of available base functions, such as 2-input NAND's and inverters. This decomposition is necessary to ensure that every node of the network can be implemented by at least one library gate. It also increases the granularity of the network, which provides more mapping options, yielding better final circuits [28]. A unique feature of our decomposition algorithm is that it incorporates an efficient heuristic that optimizes the decomposition for average performance. In the covering step, the optimized decomposed network is covered with available library gates with the objective of optimizing average performance under a given area constraint.

Our covering algorithm is motivated by Chaudhary and Pedram's covering algorithm [7] which evaluates area-delay tradeoffs based on static (pessimistic) timing analysis. Novel aspects of our algorithm are as follows.

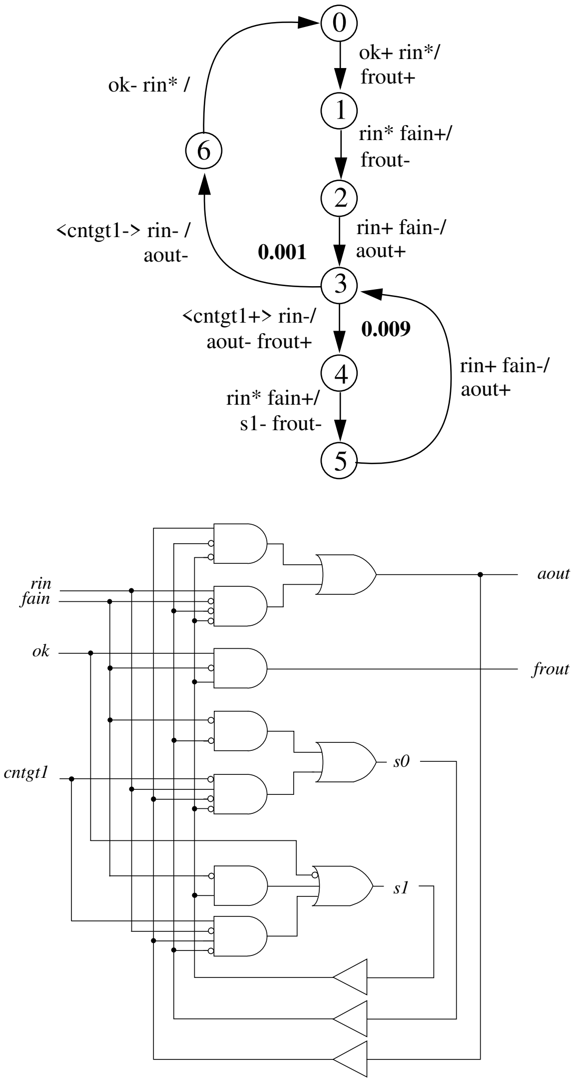


Fig. 1. Scsi-init-send example.

- Generation of a multi-dimensional area-performance tradeoff surface that extends the notion of a two-dimensional area-delay tradeoff curve proposed in [7].
- Adaptation of an input-pattern-dependent method to consider delays of only *true* critical paths associated with each state transition [15], thereby avoiding the *false path problem*.
- Inclusion of a new heuristic to remove likely-non-optimal mappings from consideration to reduce both CPU run-times and memory usage.

We tested our algorithm on a large set of benchmark circuits. To evaluate the benefit of our algorithm, we compared the average performance of our mapped circuits to the average performance of comparable circuits mapped by an algorithm that minimizes the worst-case delay. The results indicate that our circuits have significantly better average performance and are, on average, significantly smaller, with comparable CPU run-times.

The remainder of the paper is organized as follows. Section II provides a necessary background on technology mapping, burst-mode circuits, and timing analysis of synchronous circuits. Section III formalizes our notions of average latency and cycle time for burst-mode circuits. Sections IV and V describe our average-case decomposition step and covering step, respectively. Section VI reports our experimental results and Section VII presents our conclusions.

## II. Background

In this section, we describe previous work on technology mapping for both synchronous and asynchronous circuits, review burst-mode circuits, and review worst-case timing analysis of synchronous circuits.

### *A. Previous work on technology mapping*

#### A.1 Technology mapping of synchronous circuits

For synchronous circuits, technology mapping is often reduced to a problem of covering a directed acyclic graph (DAG) which is then approximated by a sequence of optimal tree coverings [13], [28]. First, a set of base functions is chosen, such as a 2-input NAND gate and an INVERTER. Second, the set of equations for the target circuit (typically obtained from a technology independent optimization) are decomposed into a graph, in which each node is a base function. Last, the decomposed graph is covered using available library gates so that a selected cost function is minimized. Initially, algorithms were developed to minimize the area [13] and the worst-case delay [28] of the mapped circuit. Later, both Chaudhary and Pedram [7] and Touati et al. [33] proposed more sophisticated algorithms to obtain the covering that yields the minimum area under given delay constraints.

#### A.2 Technology mapping of asynchronous circuits

Different asynchronous design styles make different assumptions about the delays within the circuit and how the circuit interacts with the environment. These characteristics have a dramatic impact on their technology mapping. This section briefly characterizes these differences.

This paper focuses on burst-mode circuits, a class of asynchronous circuits that rely on the *fundamental-mode assumption* [35], which states that the circuit is allowed to settle between input transitions that arrive at the circuit in bursts. Other asynchronous design styles, such as speed-independent circuits and timed circuits, however, operate in the *input/output mode*. That is, the environment may introduce new input changes concurrently with the circuit generating new outputs as long as the input behavior satisfies a given specification.

Siegel and De Micheli [30] studied the technology mapping problem for burst-mode circuits. They found that, with just small modifications, synchronous technology mapping solutions can be *safely* used. More specifically, Siegel and De Micheli used results from Unger [35] to demonstrate that the standard tree decomposition of the equations into base functions does not introduce hazards. This result was formalized by Kung who proved that the tree decomposition is a hazard-freedom-preserving logic transformation for fundamental-mode circuits [16]. Moreover, Siegel and De Micheli presented an algorithm that analyzes implementations of library gates to determine if they might in some cases cause a hazard when used in the covering phase. Their results demonstrated that most library gates can be used safely, but some MUXes and AOIs, for example, might cause hazards depending on how they are used. When such gates match to a particular node in the graph, further analysis is required to determine if the match causes a hazard in that instance [30]. The key shortcoming of their work is that the underlying synchronous technology mapper they used is limited to optimizing the worst-case performance, not the average-case performance. It is this limitation that our approach addresses.

We must emphasize that the hazard-freedom preserving feature of decomposition in burst-mode circuits does not hold for input/output-mode circuits. Intuitively, this is because, in input/output-mode circuits, extra transitions on internal signals newly generated by the decomposition are not guaranteed to settle before the circuit inputs change and consequently may unexpectedly interact with changing inputs, thereby causing hazards. Consequently, technology mapping techniques for speed-independent and timed circuits have to pay extra attention to the hazard-preserving nature of their mapping techniques [1], [5], [14], [9], [21], [31].

We also note that the additional challenges associated with the decomposition of input/output-mode circuits may partially explain why these efforts have not yet addressed optimizing for average-case performance. Moreover, we believe that the ease of decomposition for fundamental-mode circuits makes burst-mode circuits a good initial target for average-case optimization research.

### *B. Burst-mode control circuits*

This section reviews the specification, implementation, and operation of burst-mode circuits.

#### B.1 Extended burst-mode (XBM) specifications

An extended burst-mode (XBM) specification [36], [39] consists of a finite number of states, a set of labeled state transitions connecting pairs of states, and a start state. Fig. 1a shows the XBM specification of the SCSI-INIT-SEND circuit, which has a conditional input *cntgt1*, 3 edge inputs (*ok*, *rin*, *fain*), and 2 outputs (*aout*, *frout*). Signals not enclosed in angle brackets and ending with $+$ or $-$ are *terminating edge signals*. For example, in the state transition out of state 0, $ok$ is a terminating edge signal. 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. For example, in the state transitions out of state 3, $cntgt1$ is a conditional signal. $\langle cntgt1^+\rangle$ denotes "if $cntgt1$ is high" and $\langle cntgt1^-\rangle$ denotes "if $cntgt1$ is low." A state transition occurs when all of the conditions are met and all the terminating edges have occurred. In the state transitions out of state 3, the circuit behavior is interpreted as: "if *cntgt1* is low when *rin* falls, change the current state from 3 to 6 and lower *aout*; if *cntgt1* is high when *rin* falls, change the current state from 3 to 4, lower *aout* and raise *frout*."

A signal ending with an asterisk (such as $rin$ in state 0) is a *directed don't care* [36], [39]. If a state transition is labeled with $a^*$, the following state transitions in the specification must be labeled with $a^*$ or with $a^+$ or $a^-$ ($a^+$ and $a^-$ terminate the sequence of don't cares, which is the reason these are called terminating edge signals). In addition, a directed don't care may change at most once during a sequence of state transitions it labels, *i.e.,* changes *monotonically*, and, if it does not change during this sequence, it must change in 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 in the state transition it labels [36], [39]. In Fig. 1a, $ok^+$ is a compulsory edge because it must appear in the transition from 0 to 1. $rin^+$ in the transition from state 5 to 3, on the other hand, is a terminating edge, but not a compulsory edge, because *rin* can rise at any point as the circuit transitions from state 4 through state 5 but it must have risen by the time the circuit enters state 3.

### B.2 3D implementation style

In this paper, we assume that XBM specifications are implemented as *3D* machines [39]. A 3D machine is formally represented as a 4-tuple $\langle X, Y, Z, \delta \rangle$, where $X$ is a set of primary inputs, $Y$ is a set of primary outputs, $Z$ is a possibly empty set of state variables, and $\delta : X \times Y \times Z \rightarrow Y \times Z$ is a *next-state function*.

The hardware implementation of a 3D machine is a combinational network in which all state variables and some primary outputs are fed back as inputs to the network through delay elements. For example, the 3D implementation of the SCSI-INIT-SEND controller is shown in Fig. 1b. Note that the state variables, $s0$ and $s1$, and the primary output $aout$ are fed back as inputs to the network. There are no explicit storage elements such as latches, flip-flops, or C-elements in 3D implementations.[1]

Let the fed-back primary outputs be $Y'$ and the fed-back state variables $Z'$. Let $H$ represent the set of internal nodes in the circuit. Then the combinational next-state logic of the circuit can be represented by a DAG, $\langle N, E, G \rangle$, where $N$ is a set of nodes $X \cup Y \cup Z \cup Y' \cup Z' \cup H$, and $E$ is a set of edges ($E \subseteq N \times N$), and $G$ is a set of gates that label nodes. Edge $(m, n) \in E$ connects node $m$ to node $n$. $m$ is called a *fanin* of $n$ and the set of fanins of $n$ is denoted $FI(n)$. Gate $g$ associated with node $n$ represents the logic function of that node and is referred to as $g(n)$. If $m \in FI(n)$, we say that $m$ is a fanin of gate $g(n)$.

For convenience, the set of nodes in $X \cup Y' \cup Z'$ is referred to as *DAG inputs* (DI's). Similarly, the set of nodes in $Y \cup Z$ is referred to as *DAG outputs* (DO's).

### B.3 State transitions in 3D machines

The state transitions in 3D machines can be designed in two ways: *two-phase* or *three-phase*. In a two-phase state transition, the machine waits for an input burst and then generates a concurrent output and (possibly empty) state burst. In a three-phase state transition, the machine waits for an input burst and then generates a state burst followed by an output burst [36]. Alternatively, three-phase state transitions can generate the output burst before the state burst. To simplify the notation, we focus on the former three-phase design; extensions to handle the latter three-phase design are straightforward.

[1] *3D-gc machines* which are implemented using generalized C-elements [40] are not considered in this paper and the technology mapping of these circuits is left as future work.

### C. Worst-case timing analysis for synchronous circuits

To put our asynchronous timing analysis techniques into perspective, we first briefly discuss several worst-case delay analysis algorithms, including *static timing analysis*, *static sensitization analysis*, and *dynamic sensitization analysis*. To simplify the exposition, we restrict ourselves to circuits composed of simple gates, such as NAND, NOR, AND, and OR, etc.

For a synchronous circuit, the worst-case delay of a combinational block sets a lower bound on the clock cycle time of the circuit. Traditional *static timing analysis* estimates the worst-case delay by computing the topological delay of the longest path [11]. This estimation is pessimistic since the longest path may not be exercised by any input pattern, i.e., no event can propagate along the path. Such a path is said to be *non-sensitizable* and is often called a *false path* [8], [3], [20], [24].

Many researchers have worked on the path sensitization problem to eliminate false paths. *Static sensitization analysis*, which is based on the well-known D-algorithm [27], ignores the timing of each input signal and consequently may underestimate the delay of the circuit [3]. *Dynamic sensitization analysis*, on the other hand, considers the timing of input signals and thus finds the true critical path but requires a computationally intensive task of analyzing all possible input patterns [8]. More efficient input-pattern-independent methods have been proposed to approximate dynamic sensitization analysis, providing an upper bound on the worst-case delay [20], [24].

To the best of our knowledge, these advanced techniques have not been incorporated into a technology mapping procedure. Instead, most technology mappers use more pessimistic, static timing analysis techniques. Consequently these technology mappers can result in non-optimal circuits because of the false path problem.

It is also important to note that all the above techniques adopt the *floating-mode* assumption; i.e., the initial values on circuit nodes are unknown at the time the input pattern is applied. This is because these techniques target combinational circuits in which the sequence of input patterns applied is assumed to be arbitrary.

## III. AVERAGE PERFORMANCE OF BURST-MODE CIRCUITS

This section formalizes our technology mapping objective functions: average latency and cycle time. It first defines the delay through the combinational blocks and the required feedback delays and fundamental-mode constraints. It then formalizes our definitions for average latency and cycle time using Markovian analysis.

### A. Determining the delay through a combinational logic block

This section describes and extends Kung's technique [15] to compute the delay through the combinational logic blocks of burst-mode circuits. Kung observed that in such circuits the sequence of possible input patterns is well-defined and the internal signals are guaranteed to stabilize to known values before the application of each new input pattern. Consequently, he adopted the *single step transition mode* model rather than of the less accurate floating-mode model mentioned above [15].

For our application, we associate one input pattern for each burst that generates transitions at the DO's. Thus, for both two-phase and three-phase state transition, we create a *cld-pattern* modeling the combinational logic delay of the input burst. Three-phase state transitions, however, have an additional cld-pattern modeling the combinational logic delay of the state variable burst.

The values in the cld-pattern correspond to the values of all DI's immediately after the last terminating transition of the burst arrives at the circuit. For example, in Fig. 1a, the input pattern for the input burst $\langle cntgt, rin, fain \rangle$ of state transition $3 \rightarrow 4$ is $\langle$ 1, F, 0 $\rangle$. *fain* is 0 because it falls during state transition $2 \rightarrow 3$. *cntgt* is 1 because we assume that all conditional signals are stable at their sampled value before the compulsory transitions arrive. Note that all state variables introduced by the synthesis process should also be included in the pattern. Moreover, we assume that the user indicates when directed don't care signals change. This latter assumption can be easily generalized, assuming that the user specifies the probability of the directed don't care firing.

The first step of Kung's delay analysis is to perform an extended 8-valued logic simulation to obtain the logic value at the output of each circuit node for each input pattern [15]. In this logic, 1 and 0 denote *stationary* values, R and F denote *transitional* rising and falling values, S0 and S1 denote *static* 0 and *static* 1 *hazardous* values, and DR and DF denote *dynamic* rising and falling *hazardous* values.

To perform the 8-valued simulation, the DI's associated with an input pattern are first set to either a stationary or transitional value depending on whether the input pattern indicates that they have changed compared to their known initial value. These values are then propagated through the circuit using 8-valued truth tables of all gates until the DO's are reached. For example, Table I shows the 8-valued truth table of a node associated with a 2-input NAND gate.

| NAND | 1 | 0 | R | F | S1 | S0 | DR | DF |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | F | R | S0 | S1 | DF | DR |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| R | F | 1 | F | S1 | DF | S1 | DF | S1 |
| F | R | 1 | S1 | R | DR | S1 | S1 | DR |
| S1 | S0 | 1 | DF | DR | S0 | S1 | DF | DR |
| S0 | S1 | 1 | S1 | S1 | S1 | S1 | S1 | S1 |
| DR | DF | 1 | DF | S1 | DF | S1 | DF | S1 |
| DF | DR | 1 | S1 | DR | DR | S1 | S1 | DR |

TABLE I
8-VALUED TRUTH TABLE OF 2-INPUT NAND GATE.

Since the circuit is hazard-free, the DO's must have either stationary or transitional values. Internal signals, however, may have hazardous values. Kung observed that because the DO's are hazard-free under the unbounded gate and wire delay model, the delay at any DO cannot be influenced by the delay of internal nodes with hazards. In addition, he realized that any node with a stationary value cannot affect the delay of any DO. Kung thus argued that the delay analysis for a given pattern $i$ can be performed on a reduced circuit in which all nodes with hazardous values, stationary values, and any value which would be eventually suppressed by a stationary 0 value are removed. In practice, this is implemented by associating each input pattern $i$ with the set of nodes $N(i)$, referred to as *sensitized* nodes for pattern $i$, which have only transitional values that affect DO's. We thus compute the delay of a cld-pattern for only sensitized nodes, thereby savings both run-time and memory. Because we adopt a fixed-delay model, the delay of a node having a transitional value for a given pattern is fixed and it is called *pattern arrival time* of the node.

To define the pattern arrival time, we first review the notions of controlling and dominance [15]. A sensitized fanin of a simple gate (whose output is also sensitized) is controlling if it has a logic value that can independently determine the timing of the output transitional value. For example, a sensitized fanin of a NAND (NOR) gate with a value F (R) is controlling. A sensitized fanin is non-controlling otherwise. Node $m \in FI(n)$ *dominates* $n$ for pattern $i$ if $m$'s pattern arrival time plus the pin-to-pin delay of $g(n)$ from $m$ to $n$ is minimum (maximum) among all controlling (non-controlling) sensitized fanins.[2].

Using these definitions, we define the $i^{th}$ *pattern arrival time* of $n$, denoted $pat(n, g(n), i)$, recursively on the structure of the circuit as follows:

$$\begin{aligned} pat(n, g(n), i) \quad &= \quad pat(m, g(m), i) + \\ & \quad\quad p2p\text{-}delay(n, m, g(n), v(i)), \qquad (1) \end{aligned}$$

where $m$ dominates $n$ for cld-pattern $i$ and $p2p\text{-}delay(n, m, g(n), v(i))$ is the pin-to-pin delay of gate $g(n)$ from $m$ to $n$ for $v(i)$, the transitional value at $n$ for $i$. For example, consider a 2-input NAND node $n$, with sensitized fanins $a$ and $b$. Assume that the pin-to-pin delays from $a$ to $n$ and from $b$ to $n$ are 1 and 2 respectively, and the pattern arrival times at $a$ and $b$ for pattern $i$ are 3 and 4 respectively. If the logic values at $a$ and $b$ are $F$ (transitional falling) for this pattern, then $a$ is the dominant fanin of $n$ because both fanins are non-controlling and $min(1 + 3, 2 + 4)$ is 4.

Note that the pattern arrival time for DI's can be set by the user. When processing cld-patterns, however, their default value is 0. Once set, repeated application of the above definition yields the pattern arrival times at the DO's, i.e., the delay through the combinational logic.

### B. Determining the minimum required feedback delay and settling times

In addition to the delay through the combinational logic, the performance of a burst-mode circuit is determined by the minimum required feedback delay and the settling times. This subsection describes how these values are determined.

Consider a circuit in which the combinational logic has been mapped but the feedback delay has not been added. To compute the minimum feedback delay to add, our algorithm first determines which gates may exhibit an essential hazard, referred to as *essential-hazard problem gates* [6]. For each essential-hazard problem gate, our algorithm determines the minimum amount of feedback delay that guarantees that the essential hazard does not occur. It then conservatively take the maximum of these delays as the feedback delay. Once the feedback delay is determined, it

[2]This definition of *dominant fanin* extends Kung's definition [15] to account for pin-to-pin delays.

can compute the settling time required for a given pair of state transitions. It does this by first determining which gates may exhibit hazards due to a violation of the fundamental-mode assumption, referred to as a *fundamental-mode problem gate*. For each gate, it determines the minimum amount of delay the environment must wait before injecting the new input burst. Our algorithm then conservatively take the maximum of these delays as the required fundamental-mode constraint for this state transition.

To identify the essential-hazard and fundamental-mode problem gates our algorithm uses the techniques described in [6]. The essential-hazard problem gates are found by performing an extended-logic simulation of the circuit with a new pattern, called a *fbd-pattern*, in which the feedback state variables change simultaneously with the input burst. Gates which have hazards for this pattern but do not exhibit hazards during the input and state patterns are the essential-hazard problem gates [6]. Similarly, the fundamental-mode problem gates are found by performing a extended-logic simulation with a new *fmc-pattern*, in which the last burst of the current state transition change simultaneously with the input burst of subsequent state transition. Gates which have hazards for this pattern but do not exhibit hazards during the last burst and next input burst separately are the fundamental-mode problem gates [6].

Our approaches to compute the minimum feedback delay and settling time for a given problem gate are also similar. Consider computing the minimum feedback delay required for a given problem gate $g$. Our algorithm first derives the *propagation delays* to the fanins of gate $g$ excluding the feedback delays (which at this stage of the design process are unknown). Consider the case where the essential hazard at $g$ is caused by a fanin $f_1$ changing in response to the changing feedback state variable occurs sooner than a second fanin $f_2$ changing directly in response to the input burst. The minimum feedback delay necessary to avoid an essential hazard at this gate is obtained by subtracting the propagation delay to $f_2$ plus the corresponding pin-to-pin delay with the propagation delay to $f_1$ plus the corresponding pin-to-pin delay.

When calculating the propagation delay to the fanin of problem gates (such as $f_1$ and $f_2$ described above), we need to consider the delay of nodes that exhibit hazards. Consequently, the delay calculation used for cld-patterns, which ignores hazards, is not applicable. The basic issue to be resolved is that hazardous nodes may transition multiple times in response to an fbd- or fmc-pattern and the delays of these transitions must be considered. To do this, our algorithm uses two different approaches.

The first approach is to use standard event-driven simulation in which we associate an arrival time with each transition edge of a hazard caused by a fbd- or fmc-pattern. At fanin of a gate, each transition edge representing an input event may cause an output event whose associated arrival time is computed as the arrival time associated with the input event plus the corresponding pin-to-pin delay of the associated gate. For the feedback delay computation described above, the propagation delay of $f_2$ ($f_1$) is set to be the arrival time associated with the latest (earliest) event of $f_2$ ($f_1$). This ensures that the minimum feedback delay is not underestimated.

One disadvantage of this approach is that an exponential number of arrival times is possible for a single node. Consequently, although computationally feasible in our decomposition routine, as is made more clear in Section V, it becomes too costly to use during our covering routine. Thus, during covering, for each node, our algorithm computes a single arrival time that is an upper bound of the time needed for the node to settle. Specifically, our algorithm computes the latest possible arrival time when the last event (edge) of a hazard can occur. For example, consider a 2-input AND gate driving node $n$, with fanins $a$ and $b$ that exhibits a static 0-hazard due to $a$ rising and $b$ falling. Because the last edge of the static 0-hazard can occur only after $b$ falls, the latest arrival time of node $n$ is the arrival time of $b$ falling plus the corresponding pin-to-pin delay of the gate. To simplify notation, we refer to this computed arrival time as the pattern arrival time for fbd- and fmc-patterns.

Note that more advanced techniques in which the delay of the hazard is captured using both an upper and lower bound is also possible (e.g., see [6]). However, we have found that using a single upper bound is adequate and has the benefit of reducing time and space requirements significantly.

### *C. Formalizing our average performance objective functions*

This subsection describes how to obtain the probability for each state transition. Using these probabilities, it then formalizes the two objective functions we optimize for: *average cycle time* and *average latency*.

#### C.1 The probabilities of state transitions

First, we model the execution of the circuit with a stochastic process $\{B_n, n = 0, 1, 2, \ldots\}$. $B_n$ (the $n^{th}$ state of the process) equals state transition $t$ if the $n^{th}$ state transition executed by the process is state transition $t$. Consider the circuit depicted in Fig. 1. If from the initial state (state 0) of the circuit, the machine executes state transitions $0 \to 1$ followed by state transitions $1 \to 2$ and $2 \to 3$, then $B_1 = 0 \to 1$, $B_2 = 1 \to 2$ and $B_3 = 2 \to 3$.

Second, we let a *trace* $T$ of the circuit be a sequence of specified state transitions, starting with the initial state. We denote by $T^k$ all possible traces of length $k$. For each such trace $T \in T^k$ we associate a probability which reflects how common this trace is with respect to other traces of the same length. These probabilities are subject to the constraint that for a given length $k$ the probabilities of the traces of length $k$ add up to 1, *i.e.,* $\sum_{T \in T_k} Pr_{XBM}(T) = 1$.

The long term proportion of a transition $t$, denoted $\Pi_t$ is the long term proportion of states that the stochastic process is in transition $t$:

$$\Pi_t = \lim_{k \to \infty} \frac{\sum_{T \in T^k} (\text{\# of } t \text{ in } T) \cdot Pr_{XBM}(T)}{k} \quad (2)$$

Markov chain theory is then used to obtain the long-term proportion of state transitions. A Markov chain is a stochastic process $X_n$ in which the conditional distribution of $X_{n+1}$ is independent of past states and only depends on the value of $B_n$. If we assume there is no correlation between subsequent environmental choices, then this condition holds. A Markov chain is irreducible if every state can be reached from every other state. If the XBM specification is strongly-connected, this condition holds.

Fortunately, in practice, these assumptions are usually satisfied or represent a reasonable approximation of reality. In addition, for XBM specifications that are not strongly-connected, simple extensions to this analysis apply.

For an irreducible Markov chain it can be shown that the set of $\Pi_t$'s for each transition $t$ are the unique non-negative solutions to the following set of equations [25]:

$$\Pi_t = \sum_{t' : sink(t') = source(t)} \Pi_{t'} \cdot Pr_{XBM}(t) \quad (3)$$

$$\sum_{t \in \delta} \Pi_t = 1 \quad (4)$$

where $Pr_{XBM}(t)$ is the conditional probability of state transition $t$.

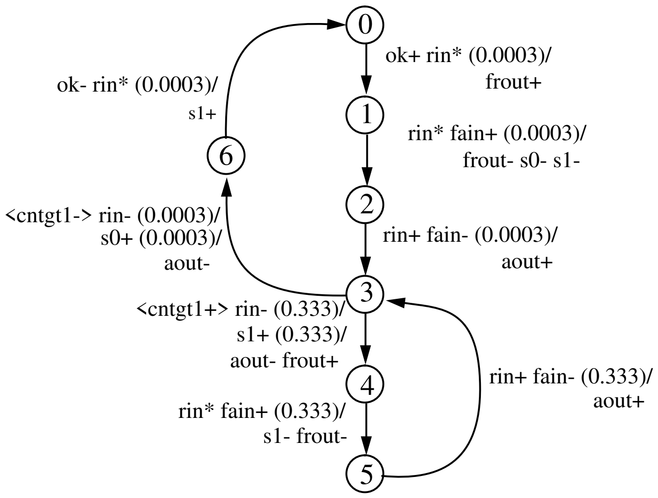


Fig. 2. Specification after state encoding annotated with results of our stochastic analysis.

Consider the SCSI-INIT-SEND controller described earlier. Given the conditional probabilities discussed in the introduction, these equations show that $\Pi_t$ for each state transition $t$ has one of two weights. All state transitions in the reset cycle have weight 0.0003 and all state transitions in the main cycle have weight 0.333. To verify these numbers, first, notice that the sum of weights of all state transitions adds up to essentially 1. (The difference is caused by the limited precision of the weights.) Next, consider the equations dictating the weight of state transition $3 \to 4$

$$\begin{aligned} \Pi_{3\to4} &= \Pi_{5\to3} \cdot 0.999 + \Pi_{2\to3} \cdot 0.999 \\ &= 0.333 \cdot 0.999 + 0.0003 \cdot 0.999 \\ &= 0.333, \end{aligned}$$

as expected. The equations dictating the weight of the other state transitions can be similarly verified.

### C.2 Definition of average latency

As mentioned earlier, latency is the critical performance metric of burst-mode circuits in which the environment is slow. Informally speaking, latency is the time after which the input burst has arrived and the output burst is generated and, if not hidden by the concurrent operation of other datapath components (e.g., multiplexing) [38], is the performance overhead associated with sequencing datapath computations.

Each burst-mode state transition has a latency. The latency of two-phase state transitions is simply the maximum pattern arrival time over all outputs for the transition's input burst, i.e.,

$$lat(t) = \max_{o \in Y}(pat(o, g(o), \textit{i-burst}(t))) \quad (5)$$

The latency of a three phase state transitions, on the other hand, is the sum of the maximum pattern arrival time over all state variables for the transition's input burst, the delay through the feedback buffers, and the maximum pattern arrival time over all output variables for the transition's state variable burst. Note that for the state variable burst, the pattern arrival time of the fed-back inputs is initialized to 0.

$$\begin{aligned} lat(t) &= \max_{o \in Z}(pat(s, g(s), \textit{i-burst}(t))) + \textit{fb-delay} + \\ &\quad \max_{o \in Y}(pat(o, g(o), \textit{s-burst}(t))) \end{aligned} \quad (6)$$

The average latency is defined as the sum of the latencies weighted by the relative probability of the state transitions.

### C.3 Definition of average cycle time

As mentioned earlier, the cycle time of the circuit becomes a critical performance metric when the environment is very fast. *Cycle time*, defined over a pair of sequential state transitions $(t, t')$, is the *minimum* time between the end of the input burst of transition $t$ and the beginning of the input burst of the next state transition $t'$. It equals the latency of the state transition $t$ plus the required fundamental-mode constraint needed in preparation of $t'$'s input burst. This fundamental-time constraint includes the feedback delay plus the settling time required for the fed-back signals to propagate into the combinational logic deeply enough to ensure that no unexpected hazards occur when $t'$'s input burst arrives. Thus,

$$cyc(t, t') = lat(t) + \textit{fb-delay} + \textit{settling-time}(t, t'). \quad (7)$$

The average cycle time is defined as the weighted sum of the cycle times of all state transition pairs. The weight of each state transition pair $(t, t')$ equals the long-term probability of being in state transition $t$, $\Pi_t$, times the conditional probability that the next transition is $t'$ given the current state transition is $t$.

## IV. PHASE 1: DECOMPOSITION FOR AVERAGE LATENCY AND CYCLE TIME

The goal of decomposition is to transform a circuit into an equivalent network that consists of only a set of base functions. These base functions form a "fine-grain" representation which provides the subsequent covering step with more flexibility in how to cover the network using library gates. By applying standard tree-decomposition technique [28], for a given circuit, we create a NAND-decomposed network whose base functions are INVERTERs and 2-input NANDs. The results of Unger [35] and Kung [16] ensure that this process preserves hazard-freedom.

An important observation is that there may be many possible NAND-decomposed networks obtainable through tree decomposition but *they are all hazard-free*. The choice of which

NAND-decomposed network to use, however, can have a significant impact on the performance of the final mapped circuit. Thus, in order to obtain a better final mapped circuit, we propose an efficient heuristic to find a NAND-decomposed network with optimal performance.

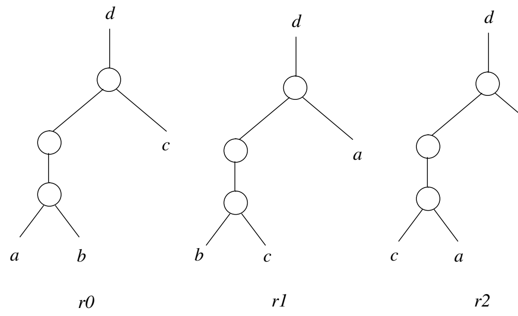


Fig. 3. The three isomorphic 3-input NANDs.

Our heuristic is motivated by Rudell's well-known optimization for worst-case delay [28]. Rudell repeatedly identified each 3-input NAND subgraph in a decomposed network and rearranged their fanin cones to optimize the worst-case latency of the network. We call this rearrangement of fanin cones "NAND3 rotation." Each NAND3 rotation creates one of three isomorphic NAND3 subgraphs, as illustrated in Fig. 3. Consider the isomorphic NAND3 labeled $r_0$. By reconnecting fanin $a$ to input $c$, fanin $c$ to input $b$, and fanin $b$ to input $a$, we complete a clockwise "rotation" of fanins and produce the isomorphic NAND3 $r_1$. Rudell repeatedly rotates each NAND3 subgraph in a decomposed network such that the latest arriving input of the NAND3 is pushed closer to the output. Similarly, our heuristic repeatedly rotates each NAND3 subgraph in a decomposed network and chooses the rotation which minimizes the user-specified cost function. Since NAND3 rotation is equivalent to simply choosing a different tree decomposition of the network, it is hazard-preserving. Currently, the user can specify as a cost function either average latency or average cycle time, however, extensions to any combination of the two are trivial.

Although this heuristic is somewhat simple, our experimental results (see Section VI) demonstrate that it is both computationally manageable and very effective.

**Algorithm 1** (Rotate decomposed network)
rotate_decomposed_network(network) {
  **do**
    **foreach** nand3 $m$ in network
      rotate_nand3($m$)
  **while** (no improvement in performance objective function)
}

Fig. 4. Heuristic rotation algorithm for decomposed networks.

Psuedocode of our implementation is given in Algorithm 1. Note that it calls Algorithm 2 to do rotation on each NAND3 of a network and uses a while loop to repeatedly rotate all NAND3s in the network until no further improvement is possible. More specifically, the objective function we measure is based on two parameters: the average latency or cycle time of the network (depending on user specification), and the average pattern arrival time of the node $n$ where NAND3 $m$ is rooted. The average latency or cycle time of the network is the principle objective function and the average pattern arrival time of the node is used to break ties in the overall average latency/cycle time of the network. This tie breaking feature often allows us to move out of local minimums.

**Algorithm 2** (Rotate NAND3)
rotate_nand3($m$) {
  **for** $r = r_1$ to $r_2$   /* rotate $m$ clockwise twice */
    compute new average performance and average pattern
      arrival time for $m$ rotated to $r$
    **if** (new average performance is better than previous best)
      OR
    (
      (the average performance is the same as previous best)
        AND
      (the average pattern arrival time at $m$ is better than
        previous best average pattern arrival time at $m$)
    )
    rotate $m$ to $r$
    update best average performance and pattern arrival times
}

Fig. 5. NAND3 rotation algorithm.

Note also that to recompute the average latency after each NAND3 rotation, the extended logic and event-driven simulation must be re-performed. Moreover, the average cycle time the fundamental-mode constraints must also be re-calculated. To save run-time, our algorithm re-analyzes only those nodes affected by the rotation, i.e., the nodes in the rotated NAND3 and in their transitive fanout. Since the network is unmapped, we assume that the circuit is trivially mapped to NAND2s and inverters and use estimated delays and load capacitances for these gates when simulating.

## V. Phase 2: Covering for average performance

Our covering procedure is divided into two stages. First, a postorder traversal (from DI's to DO's) uses dynamic programming to determine a set of covering solutions rooted at each node. Then a preorder traversal (from DO's to DI's) selects gates which maximizes the user-specified desired performance (latency or cycle time) subject to a given area constraint.

### A. *Postorder traversal for bottom-up matching*

#### A.1 Algorithm overview

We first introduce our terminology, some of which is borrowed from [7]. The NAND-decomposed graph is referred to as a *subject graph* and the set of available library gates is denoted by $L$. A *match* $h$ for a node $n$ in the subject graph is a *pattern graph* of $L$ that is isomorphic to a subgraph rooted at node $n$ and satisfies the conditions found by Siegel and De Micheli to ensure hazard-freedom [30]. Each match $h$ is associated with a set of gates which have the same pattern graph. The set of nodes in a match $h$ for a node $n$ is referred to as $merged(n, h)$. The set of fanin nodes of $merged(n, h)$ is denoted by $inputs(n, h)$.

A *point* $p$ is a tuple $\langle h.g, pat, fbd, st, early, late, area \rangle$ that contains the key parameters of a cover of a subgraph rooted at a node $n$. First, the point identifies the *matching gate* $h.g$ which is used to derive the point. In addition, the point includes the function $pat : P_i \rightarrow \mathcal{R}$ which returns the pattern arrival time, $pat[i]$, for a set of input patterns $P_i$ for which $i$ is sensitized, which as mentioned earlier, can include the following. For a two-phase state transition, there is one *cld-pattern* modeling the combinational logic delay of the input burst, one *fbd-pattern* for determining the feedback delay, and one *fmc-pattern* for each possible next state transition for determining the corresponding fundamental-mode constraints. For a three-phase state transition, there is an additional *cld-pattern* associated with the state variable burst and an additional *fbd-pattern* to determine the feedback delay needed to avoid essential hazards caused by the state variable burst. From these pattern arrival times, our algorithm estimates the minimum feedback delay *fbd* needed to avoid essential hazards for this mapping of the subgraph rooted at node $n$. In addition, our algorithm computes the settling times $st$ ($st : \delta \times \delta \rightarrow \mathcal{R}$) that are required for pairs of subsequent state transitions needed to ensure that no fundamental-mode constraint is violated. $early$ and $late$ are bookkeeping vectors used to store sets of fanins associated with the early and late transitions involved in any potential hazard at that node. Finally, $area$ is the area at node $n$. Since many pattern arrival times are stored in a point, a point is actually multi-dimensional and the set of points forms a multi-dimensional "surface." This is an extension of the two-dimensional curve in [7].

Algorithm 3 shows the pseudo code for building an area-performance tradeoff surface at a node $n$. First, the set of matching gates $h.g$ at $n$ are found using the modified matching algorithm developed by Siegel and De Micheli [30]. For each matching gate, the algorithm creates many possible area-performance points depending on how the cones of logic rooted at the input of the match $h$, $inputs(n,h)$, are mapped. Because the nodes are visited in postorder, the possibly optimal ways to implement the cones of logic rooted at $inputs(n,h)$ have already been analyzed and stored in different points on surfaces at $inputs(n,h)$.

For each combination of surface points at $inputs(n,h)$, an area-performance point is computed as follows. To compute the average cycle time for a point $p$, the first step is to compute the pattern arrival time for every cld-pattern, fmc-pattern, and fbd-pattern. For each fbd-pattern $i \in P_{fbd}$, the algorithm then selects the maximum feedback delay among the feedback delays stored in $in\_pts$ as the default feedback delay for $p$ for resolving essential hazards which may occur in the fanin cones of $h$. If $g$ is an essential-hazard problem gate rooted at $n$ for pattern $i$, the algorithm also estimates a new feedback delay to avoid an essential hazard occurred at gate $g$ when $i$ is applied (as is explained in Section V-A.2). If the default feedback delay is smaller than the new computed feedback delay, the feedback delay for $p$ is updated using the new feedback delay. For each fmc-pattern $i \in P_{fmc}$, the $fmc$ for $p$ is estimated similarly. But instead of only one $fbd$ being stored in $p$, an array of $fmc$'s for all fmc-patterns is stored. The area of the point is the area of $h.g$ plus the total area of $in\_pts$. Then, the point is inserted into the surface and the function *remove_inferior_pts()* removes all surface points which are deemed unlikely to lead to an optimal covering (as is described in Section V-A.5).

**Algorithm 3** (Compute area-performance surface)
compute_area-performance_surface($n$) {
  **foreach** match $h$ rooted at node $n$
    find all combinations of surface points from the surfaces of $inputs(n,h)$
    **foreach** matching gate $h.g$
      **foreach** point combination $in\_pts$
        $p$ = allocate_point()
        **foreach** $i \in P_{cld} + P_{fbd} + P_{fmc}$
          $p.pat[i]$ = comp_pat_arv_time($n, h, i, in\_pts$)
          **if** $i \in P_{cld}$ **then** $apat$ += $w[i]$ * $p.pat[i]$
        $p.fbd$ = select_max_fbd_at_in_pts($in\_pts$)
        **foreach** $i \in P_{fbd}$
          **if** problem_cover($n, h, i$) = YES
            $fbd$ = estimate_max_diff($n, h, i, in\_pts$)
            $p.fbd = \max(fbd, p.fbd)$
            $p.early[i]$ = fanins with early transitions for $i$ in $p$
            $p.late[i]$ = fanins with late transitions for $i$ in $p$
        **foreach** $i \in P_{fmc}$
          $p.st[i]$ = select_max_diff_at_in_pts($i, in\_pts$)
          **if** problem_cover($n, h, i$) = YES
            $st$ = estimate_max_diff($n, h, i, in\_pts$)
            $p.st[i] = \max(st, p.st[i])$
            $p.early[i]$ = fanins with early transitions for $i$ in $p$
            $p.late[i]$ = fanins with late transitions for $i$ in $p$
        $p.area = \sum_{k=0}^{l}(in\_pts[k].area + m.area)$
        insert $p$ into $n.surface$ in area-ascending order
        remove_inferior_pts($p, n.surface$)
}

Fig. 6. Computing the area-performance surface at a node.

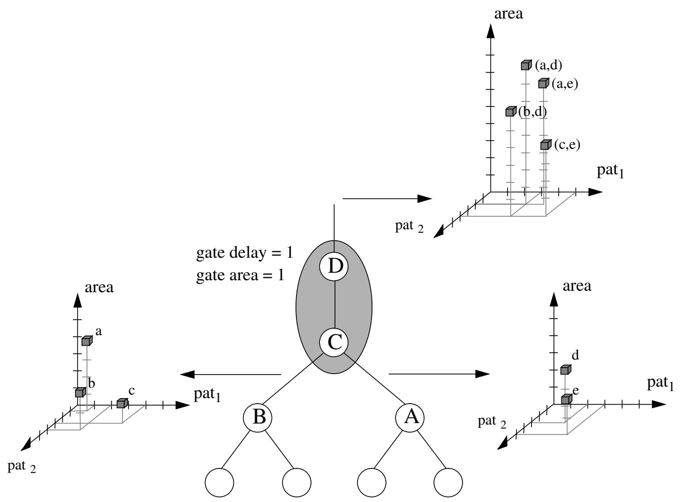


Fig. 7. Computing the surface points for a match.

Consider the example depicted in Fig. 7 where a matched 2-input AND gate $h.g$ rooted at node $D$ covers nodes $C$ and $D$. The set $inputs(D, h)$ is $\{A,\ B\}$. For simplicity, we assume that we are optimizing for latency and that there are two equally weighted cld-patterns, pattern 1 and 2, and no fbd-patterns. The transitional values for the two patterns at $A, B, C$, and $D$ are (R,F), (R,F), (F,R), and (R,F), respectively. $A$ has 2 points $d$, $e$, and $B$ has 3 points $a$, $b$, $c$. There are 6 possible input point combinations, each of which represents a different input gate combination rooted at $A$ and $B$ and determines one point at D. For example, consider the pair $(a, e)$ and assume that all pin-to-pin delays are 1. Then, $pat[1]$ is $max(1+1, 3+1)$ which is 4 and $pat[2]$ is $min(1+1, 5+1)$ which is 2. The area is the sum of the areas of $a$ and $e$ plus the area of the AND gate which is assumed to be 1, yielding 7. As is explained in Section V-A.5, only four of the 6 possible points generated are deemed likely to lead to a good covering; the other two are removed by the function *remove_inferior_pts()*.

### A.2 Estimating the required feedback delay and settling-times

Recall that the required feedback delay and settling-time of a particular problem gate is obtained from the difference between the propagation delay through the feedback signals and the propagation delay from the primary inputs to the problem gate [6]. Once the combinational logic portion of the circuit is mapped, this value can be readily obtained (as described in Section III-B.) During the covering process, however, the delay through the feedback signals is not known since the nodes are processed in postorder. In particular, the logic between the problem gate and the corresponding state variables/primary outputs has not yet been chosen.

To address this problem, our algorithm estimates the delay of each state variable/primary output $z$ to be the pattern arrival time of $z$ in the un-mapped network. Although this estimate is somewhat coarse, our experimental results suggest it is effective.

### A.3 Extending the computation of pattern arrival times to complex gates

To calculate the pattern arrival time during covering, function *comp_pat_arv_time()* must analyze matching gates that may be complex. For such gates, we algorithmically extend our definition of a dominant fanin. Specifically, we recursively traverse the NAND-decomposed subgraph representing the complex gate to simultaneously find the dominant fanin and pattern arrival times for a given point $p$. For each complex-gate input $m$, we calculate $p.pat[i] + \textit{p2p-delay}(m, n, h.g, t)$, where $t$ is the transition that node $n$ makes in pattern $i$. We then recursively propagate these delays up through the subgraph selecting the delay and fanin node which is dominant for the current subgraph node. For example, consider a matching AOI gate $h.g$ rooted at node $n$ where all fanins are non-controlling. The algorithm first chooses the maximum sum in each AND subgraph and then chooses the minimum sum for the OR subgraph among those maximum sums of AND subgraphs. The corresponding $m$ for the minimum of the maximums is the dominant fanin $m_d$ of $n$ and the minimum of maximums is the pattern arrival time $p.pat[i]$.

### A.4 Accounting for unknown loads

The function *comp_pat_arv_time()* uses the pin-dependent delay model in MIS [7] to model the delay of $h.g$ from input pin $m$ to output pin $n$, as follows,

$$\begin{aligned}\textit{p2p-delay}(n, m, h.g, t) &= \tau(n, m, h.g, t) + \\ &\quad R(n, m, h.g, t) \cdot C_n, \end{aligned} \tag{8}$$

where $\tau(n, m, h.g, t)$ is an intrinsic pin-to-pin delay of the gate, $R(n, m, h.g, t)$ is a pin-specific output drive-resistance, and $C_n$ is the capacitive load at the output of $h.g$.

It is important to note that when the postorder procedure first visits each $m \in inputs(n, h)$, the exact output load $C_m$ is unknown because some of its fanouts may not have been mapped. In particular, this node may have fanouts in cones of logic that have been traversed as well as cones of logic that have yet to be traversed. For this reason, our algorithm must estimate its load. For the mapped fanouts it accounts for the input capacitance of the mapped gate. For the unmapped fanouts it uses a default input capacitance $C_d$. This is known as the *unknown load approximation.*

A useful observation made in [7] is that when the main procedure visits node $n$ and computes a point for $h.g$, the exact output load for each node $m$ is more precisely known because the input capacitance of its fanout $n$ is determined by the gate $h.g$. More specifically, the previously unknown load contribution of $h.g$ is now known.

To obtain a better estimate of the pattern arrival time at $n$, the routine *comp_pat_arv_time()* compensates for the difference between the default load and the exact load by adding a *load shift* term to the pattern arrival times at $m$. Specifically, this *load shift* term is $R(m, m_d, h.g', t') \cdot (C_m - C_d)$, where $m_d$ is the dominant fanin of $m$ and $t'$ is the transition for pattern $i$ at node $m$ [7].

### A.5 Removing points with inferior average performance

The criteria of determining the inferiority of a point affects the efficiency and effectiveness of the algorithm. A good criteria reduces a large number of points but still keeps the final solution as close to optimal as possible.

We first propose an exact technique to remove inferior points that depends on what the tool is optimizing. When optimizing for latency, a point is *inferior* if there exists another point on the surface which has less or equal latency for *all* state transitions. When optimizing for cycle time, the other point must have less or equal to cycle times for *all* state transition pairs. These definitions yields the optimal covers subject to load shift errors. However, under this definition, it is difficult for a point to be inferior since any individual latency/cycle time may violate the above condition. Our experimental results in Section VI show that the number of non-inferior points becomes unmanageable for large circuits.

Thus, in order to reduce the number of points, we propose a new heuristic to quantify the potential merits of one point over another based on the notions of a point's average latency and cycle time. Specifically, the *point average latency* is an estimate of the average latency of the circuit up to that node for the cover of the subgraph represented by the point. It is defined as the sum of the point's state transition latencies weighted by the probability of the state transition. For a two-phase state transition the point's state transition latency is simply the point's pattern arrival time for the cld-pattern associated with this state transition. For a three-phase state transition, the point's state transition latency also consists of the pattern arrival time for the cld-pattern associated with the state burst and the current estimated value of the required feedback delay.

Similarly, the *point average cycle time* is an estimate of the average cycle time of the circuit up to that node for the cover of the subgraph represented by the point. It is defined as the point average latency plus the current estimate of the average required fundamental-mode constraint.

A point has *inferior average performance* if there exists any other point on the surface which has less or equal area and less or equal average performance, i.e., latency or cycle time depending on which the user directs the mapper to optimize. Our experimental results indicate that judging inferiority based on point average latencies and cycle times is very effective. In other words, these quantities adequately reflect the relative overall performance impact of various mapping options for a given subgraph.

In our implementation, we maintain the area-performance surface as an area-ascending ordered list to make removing average performance inferior points efficient. Specifically, a point $p$ divides the list into two sets $A$ and $B$. Each point in set $A$ has no larger area than the area of $p$ and each point in set $B$ has larger area than $p$. If there exists any point in set $A$ with better (or equal) average performance than $p$, $p$ has inferior average performance and is removed from the list. If not, $p$ is not removed and any point $p'$ in set $B$ that has worse (or equal) average performance than $p$ is removed. Thus, the list remains not only area-ascending order but also average-performance-descending order by construction. Consequently, removing inferior average performance points takes linear time with respect to the number of points in the list.

Recall that in Fig. 7 only 4 points are stored at $D$ instead of 6 since both $(c, d)$ and $(b, e)$ generate likely inferior points. This is because $(b, e)$, $(c, d)$, and $(c, e)$ all have the same average latency of 4.5 but point $(c, e)$ has the lowest area of 4.

It is important to note why dropping points with inferior average performance may lead to a sub-optimal solution. Consider the case where the critical path for Pattern 2 is not through this AND gate and is through a different part of the graph. Then, the only pattern arrival time of importance is Pattern 1. For the point from $(b, e)$, the pattern arrival time for pattern 1 is 4. Because this is the smallest pattern arrival time for Pattern 1 between the points with an area of 4 or higher, it could be part of the optimum solution and should not be dropped. Fortunately, our experimental results indicate that this heuristic effectively removes most surface points yet still maintains close to optimal solutions. This is because if the heuristic picks a point that yields a non-optimal solution, the difference from the optimal solution is probably small since the point would most likely have significantly sub-optimal pattern arrival times for only very infrequent patterns.

We also note that for multi-fanout nodes, we adopt Chaudhary and Pedram's heuristic of dividing the area of each point by the number of its fanouts [7]. This means that the relative area cost of solutions in which the multi-fanout node is internal to a gate is very high. Consequently, this heuristic favors solutions in which the multi-fanout node is a gate output rather than internal to a gate.

In addition, to further reduce the number of points, we adopt the well-known binning technique [28] in which if the difference between two performance metrics is less than a user-defined *bin size*, the performance metrics are considered equal, yielding many more inferior points.

### *B. Decomposing the mapping of multi-output circuits*

Ideally, once area-performance tradeoff surfaces at all DO's are computed, we would select a combination of DO points (one for each DO) which yield the best average performance over all possible point combinations that satisfy a given total area constraint. This, however, can be computationally very expensive since the number of DO point combinations that would need to be analyzed can be huge. Thus, we propose a fast heuristic to find a good, but potentially non-optimal, solution which is to iteratively cover each output cone individually and sequentially, using a decomposed area constraint. Another important advantage of this approach is that once an output cone is covered, all surface points on nodes in that cone can be freed, dramatically reducing the total memory needed for the algorithm.

It may be useful to explain why this heuristic may not yield a circuit with optimum performance. Consider the case where we are optimizing the average latency of a circuit that has two output cones, referred to as $A$ and $B$, and two state transitions, $t_1$ with probability $0.9$ and $t_2$ with probability $0.1$. Assume for simplicity that these state transitions have only two-phases and thus feedback delays need not be considered. In the frequent state transition $t_1$ assume both outputs transition, whereas in the less frequent state transition $t_2$ assume only the output of cone $A$ transitions. Because we process cones individually and sequentially, when we optimize output cone $A$ we favor the pattern arrival times corresponding to $t_1$. Unfortunately, it might be the case that the latency of $t_1$ is dictated by output cone $B$. Thus, in this case, the ideal thing to do would be to optimize $A$ for the less frequent state transition $t_2$ (rather than $t_1$) because this would yield lower average latency.

This problem occurs because cone $A$ is only critical for relatively infrequent patterns. Fortunately, this means that the difference from the optimal solution is usually small since only the pattern arrival times of relatively infrequent patterns remain non-optimal. In addition, this problem can be mitigated by relaxing the required times for state transitions at a DO to be equal to the maximum corresponding pattern arrival times obtained from previously mapped cones. Also note that our estimate of the required feedback delay and settling times are automatically updated during the process of mapping each cone. Thus, cones mapped later in the algorithms have the benefit of more accurate estimates.

### *C. Preorder traversal for top-down selection*

Now assume we have selected a point in a DO's surface whose performance is the best of all points that satisfy the given local area constraint. Since the area and performance of this point is based on the unknown load approximation, the set of gates that lead to this point may actually yield different area and performance characteristics from that estimated by the point [7]. Thus, our algorithm performs a separate preorder traversal to actually select the remaining gates in the cover of this cone. In this way, we can keep the overall difference between estimated and actual area and performance metrics to be typically less than 10%.

The first step of Algorithm 4 is to call *find_best_point()* to find the best point at node $n$. To facilitate this, the node data structure contains an array of *pattern required times*, a corresponding array of *directions of importance*, an array of *settling required times*, and a *feedback required delay*. The direction of importance for a given pattern indicates whether the pattern arrival time of the point chosen should have a larger or smaller value. The direction of all cld-patterns is always smaller because the chosen point should have a delay equal to or smaller than the required times. However, for fmc-patterns and fbd-patterns, both directions are needed. Recall that our algorithm computes the settling time and feedback delay by taking the difference of two pattern arrival times, one corresponding to an early transition and one corresponding to a late transition (these are computed from $req\_max\_fbd.early[i]$ and $req\_max\_fbd.late[i]$ respectively). To reflect the desire for the point to have a smaller difference, the chosen point's pattern arrival time for the early transition needs to be equal to or *smaller* than the required pattern arrival time while the point's pattern arrival time for the late transition needs to be equal to or *larger* than the required pattern arrival time.

Due to the unknown load problem, it may not be possible to find a point that satisfies all these constraints. For this reason, as illustrated in Algorithm 5, we compute a *load shift error* to represent by how much a point violates these constraints. Specifically, the load shift error is the weighted sum of the amounts that the pattern arrival time violates the corresponding pattern required time. For a point to satisfy all pattern required times, the load shift error must equal zero. When no point satisfies all the pattern required times, the function selects the point with the smallest load shift error.

After finding the best point, Algorithm 4, sets the pattern required times for all fanins of the chosen gate. Note that for a DO $n$, whose point is selected by the user, $prt(n, i) = n.best\_p.pat[i]$ for all patterns $i$ for which $n$ is sensitized and $prt(n, i) = \infty$ for all other patterns. For all other nodes, the algorithm initializes the pattern required times as follows.

$$prt(m, i) = prt(n, i) - \textit{p2p-delay}(n, m, best\_p.h.g, t), \quad (9)$$

where $best\_p.h.g$ is the best matched gate rooted at $n$ stored in point $best\_p$.

Next, Algorithm 4 checks whether the chosen gate is an essential-hazard problem gate. If so, it computes the slack between $best\_p$'s stored feedback delay and the maximum feedback delay allowed. This slack, which can be either positive or negative, is used to compute a secondary pattern required times for the fanins. If this secondary pattern required time is more constraining than the initially computed pattern required time, the pattern required time is updated accordingly. The algorithm performs a similar process if the chosen gate is a fundamental-mode problem gate. Note that we initialize the maximum required feedback delay and settling times at the DO's to the values in the point $best\_p$ selected at the DO.

**Algorithm 4** (Recursively select best gates)
recur_select_best_gates($n$) {
  $n.best\_p$ = find_best_point($n$)
  **foreach** $m \in inputs(n, n.best\_p.h.g)$
    **foreach** $i \in P_{cld}$
      $\textit{p2p-delay}(m, n, n.best\_p.h.g, t(i)) =$
        $\mathrm{load}(n) * n.best\_p.h.g.drive[t(i, n)] +$
        $n.best\_p.h.g.intrinsic[t(i, n)]$
      $m.prt[i] = n.prt[i] - \textit{p2p-delay}(m, n, n.best\_p.h.g, t(i))$
    /* propagate $req\_max\_fbd$ top-down */
    $m.req\_max\_fbd = n.req\_max\_fbd$
    **foreach** $i \in P_{fbd}$
      $\textit{p2p-delay}(m, n, n.best\_p.h.g, t(i)) =$
        $\mathrm{load}(n) * n.best\_p.h.g.drive[t(i, n)] +$
        $n.best\_p.h.g.intrinsic[t(i, n)]$
      $m.prt[i] = n.prt[i] - \textit{p2p-delay}(m, n, n.best\_p.h.g, t(i))$
      /* flags for comparing $prt[i]$ and $pat[i]$
        set to be SMALLER if $m$ is a DO */
      $m.g\_or\_s[i] = n.g\_or\_s[i]$
      **if** problem_cover($n, n.best\_p.h.g, i$) = YES
        $early = n.best\_p.fbd.early$
        $late = n.best\_p.fbd.late$
        $slack = m.req\_max\_fbd - n.best\_p.fbd$
        **if** $m \in n.best\_p.early[i]$
          $m.g\_or\_s[i]$ = SMALLER
          $2nd\_prt = early + slack/2 -$
            $\textit{p2p-delay}(m, n, n.best\_p.h.g, t(i))$
          **if** $m.prt[i] > 2nd\_prt$ **then** $m.prt[i] = 2nd\_prt$
        **else if** $m \in n.best\_p.late[i]$
          $m.g\_or\_s[i]$ = GREATER
          $2nd\_prt = late - slack/2 -$
            $\textit{p2p-delay}(m, n, n.best\_p.h.g, t(i))$
          **if** $m.prt[i] < 2nd\_prt$ **then** $m.prt[i] = 2nd\_prt$
    **foreach** $i \in P_{fmc}$
      /* propagate $req\_st[i]$ top-down */
      $m.req\_st[i] = n.req\_st[i]$
      /* the remainder of algorithm is similar to $P_{fbd}$ section,
        but with every $max\_fbd$ and $req\_max\_fbd$ substituted
        with $st[i]$ and $req\_st[i]$ */
    recur_select_best_gates($m$)
}

Fig. 8. Algorithm to recursively select best gates for one DO.

### D. Complexity analysis

We focus our complexity analysis on our covering algorithm because it dominates the overall run time. We first consider a match $h$ at node $n$ where the matching gate $h.g$ has $l$ inputs and each input $k$ has $N(k)$ surface points. The number of points at $n$ is bounded by $\prod_{k=0}^{l} N(k)$. Assuming a fixed library size, the total number of matches at a node is a constant. We can thus conclude that the number of points from one level of the graph to the next grows at most polynomially with a degree that is bounded by the maximum number of gate inputs.

The time complexity to derive each point consists of the sum

**Algorithm 5** (Find best points)
find_best_point($n$) {
  $min\_load\_shift\_error = \infty$
  **foreach** $p$ in $n.surface$ in area-ascending order
    $load\_shift\_error = 0.0$
    **foreach** $i \in P_{cld}$
      **if** $p.pat[i] > n.prt[i]$
        $load\_shift\_error$ += $w[i] * (p.pat[i] - n.prt[i])$
    **foreach** $i \in P_{fbd}$
      **if** $(p.pat[i] > n.prt[i]) \oplus (m.g\_or\_s[i] = \text{GREATER})$
        $prt\_violation =$
          $\max(|p.pat[i] - n.prt[i]|, prt\_violation)$
    $fbd\_violation = req\_max\_fbd - p.fbd$
    $fbd\_load\_shift\_error = \max(prt\_violation, fbd\_violation)$
    $load\_shift\_error$ += prob$(P_{fbd}) * fbd_load_shift_error$
    **foreach** $i \in P_{fmc}$
      **if** $(p.pat[i] > n.prt[i]) \oplus (m.g\_or\_s[i] = \text{GREATER})$
        $prt\_violation = |p.pat[i] - n.prt[i]|$
      $st\_violation = req\_st[i] - p.st[i]$
      $st\_load\_shift\_error = \max(prt\_violation, st\_violation)$
      $load\_shift\_error$ += $w[i] * st\_load\_shift\_error$
    **if** $load\_shift\_error = 0.0$ **then**
      return($p$)
    **else if** $load\_shift\_error < min\_load\_shift\_error$
      $min\_load\_shift\_error = load\_shift\_error$
      $min\_load\_shift\_error\_p = p$
  return($min\_load\_shift\_error\_p$)
}

Fig. 9. Algorithm to find the best point in a surface.

of the time to generate the point and the time to update the node's area-performance surface. The time to generate the point can be assumed to be a constant. The time to insert the point into the surface and remove points with inferior average performance, however, depends on the number of points on the node's surface. If the surface is implemented as a linked list, the time complexity for inserting the $(N(k)+1)^{th}$ point into the surface and removing points with inferior average performance is $O(N(k))$. The preorder traversal also involves a search of the surface of every node $k$ visited, taking $O(N(k))$ time.

Thus, the overall complexity of the algorithm is $O(m \cdot N_{max}^2)$, where $m$ is the total number of nodes in the network and $N_{max}$ is the maximum number of surface points on any surface.

The high computational complexity of the algorithm is mitigated by a variety of factors. First, asynchronous control circuits tend to be relatively small (less than 1K gates). This is because to achieve good performance asynchronous systems are typically designed using a distributed control paradigm consisting of many relatively small controllers running concurrently rather than a single large controller. Second, as mentioned earlier, the number of patterns of interest for each output tends to be small. And, finally, most derived area-performance points have inferior average performance and are therefore dropped. Indeed, our preliminary results, described in Section VI, are much more promising than the above worst-case complexity analysis suggests.

## VI. Experimental Results

We conducted experiments on a suite of benchmark circuits, presented in Table II, using a AMD-K6-233MHz/Linux PC with 128 Megabytes of memory. Each circuit is specified in an unoptimized Verilog netlist along with an XBM specification annotated with conditional probabilities. Using Markov chain analysis on the annotated XBM specification, we automatically generate the set of input patterns and their associated probabilities. The decomposition and covering algorithms are implemented on top of the POSE environment [12], which is the extension of the SIS framework [29] in which Chaudhary and Pedram's mapper is implemented.

| Circuits Tested | | | | |
|---|---|---|---|---|
| Circuit | # DI's | # DO's | # Pats | # Nodes |
| merge | 6 | 3 | 8 | 58 |
| bufctrl1 | 5 | 3 | 3 | 66 |
| p3 | 7 | 3 | 11 | 75 |
| q42 | 7 | 3 | 4 | 101 |
| scsi-init-send | 10 | 4 | 10 | 141 |
| dramc | 14 | 7 | 14 | 287 |
| binary-counter | 16 | 7 | 32 | 369 |
| pe-send-ifc | 15 | 5 | 14 | 385 |
| trcv | 21 | 7 | 30 | 746 |
| ircv | 21 | 7 | 30 | 779 |
| tsend | 34 | 9 | 42 | 1410 |
| p1 | 45 | 17 | 31 | 1606 |
| isend | 45 | 9 | 44 | 2096 |
| scsi | 47 | 11 | 93 | 2470 |
| cache-ctrl | 54 | 20 | 49 | 3032 |

TABLE II
Description of circuits tested.

### A. Decomposition results

We first used the SIS function *tech_decomp* to tree-decompose each benchmark into a hazard-free NAND-decomposed unoptimized network. We then applied our decomposition heuristic to rotate the network optimizing first for latency and secondly for cycle time. Table III reports the average latency/cycle-time for both the un-optimized and optimized circuits. The run-time for all circuits were less than 1 hour, with all but two circuits taking less than 7 minutes. The performance comparison between un-optimized and optimized circuits demonstrates that a rotated NAND-decomposed network typically has significant improvement (from 4% up to 39%) in average performance compared to its un-optimized NAND-decomposed network.

### B. Covering results

We conducted three mapping experiments on each benchmark: average-case mapping without rotation, average-case mapping with rotation, and worst-case mapping using a modified version of Chaudhary and Pedram's mapper that minimizes worst-case delay.

| Experimental Results of Decomposition Heuristic | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Circuit | w/o Rotate | | w/ Rotate | | Imp. | | RunT (s) | |
| | AveLat | AveCyc | AveLat | AveCyc | AveLat | AveCyc | AveLat | AveCyc |
| merge | 6.79 | 6.79 | 4.16 | 4.16 | 39% | 39% | 0.2 | 0.2 |
| bufctrl1 | 8.75 | 8.77 | 7.96 | 7.98 | 9% | 9% | 0.0 | 0.1 |
| p3 | 4.93 | 4.93 | 4.59 | 4.71 | 7% | 4% | 0.2 | 0.2 |
| q42 | 11.38 | 11.40 | 9.80 | 9.82 | 14% | 14% | 0.2 | 0.2 |
| scsi-init-send | 13.62 | 13.62 | 10.68 | 10.68 | 22% | 22% | 0.9 | 0.9 |
| dramc | 10.24 | 10.58 | 8.52 | 8.52 | 17% | 19% | 4.3 | 4.3 |
| binary-counter | 14.47 | 14.54 | 12.31 | 12.44 | 15% | 14% | 11.6 | 12.7 |
| pe-send-ifc | 14.82 | 14.82 | 12.29 | 12.30 | 17% | 17% | 10.0 | 10.3 |
| trcv | 25.64 | 25.64 | 21.56 | 21.56 | 16% | 16% | 62.9 | 63.7 |
| ircv | 26.04 | 26.68 | 20.78 | 21.12 | 20% | 21% | 50.8 | 52.1 |
| tsend | 30.91 | 31.07 | 24.20 | 24.20 | 22% | 22% | 399.8 | 422.9 |
| p1 | 19.66 | 20.16 | 17.22 | 17.49 | 12% | 13% | 219.3 | 229.1 |
| isend | 30.07 | 30.50 | 27.46 | 27.89 | 9% | 9% | 406.8 | 646.5 |
| cache-ctrl | 28.64 | 29.13 | 23.90 | 24.23 | 17% | 17% | 1857.9 | 1609.5 |
| scsi | 26.35 | 27.59 | 20.77 | 20.71 | 21% | 25% | 2561.7 | 2343.5 |

TABLE III
EXPERIMENTAL RESULTS OF OUR DECOMPOSITION HEURISTIC. DELAYS ARE MEASURED IN NANO-SECONDS.

For all experiments, we used the *lib2* gate library included in the SIS package. For simplicity, we assumed that all gates in *lib2* are hazard-free without running De Micheli and Siegel's algorithm to determine hazardous gates in *lib2*. The matching routine we used is borrowed from SIS and always associates the pin of a simple gate that has the smallest intrinsic delay to the input of a pattern graph with the longest path to the output. This choice of pin association favors covering longer paths of a network with shorter delay, leading to a more balanced covering that effectively minimizes worst-case delay. This unfortunately counteracts our efforts to create an unbalanced covering to shorten true critical paths. In order to truly reflect the potential advantage obtained from the rotation, we reversed the pin delay assignment of simple gates (e.g., NANDs, NORs) such that the pin with the shortest path to the output has the smallest intrinsic delay for our average-case mapping.

### B.1 Evaluation of inferior point schemes

We first present Table IV that shows the number of surface points we allocated and the run-time for each benchmark for both the exact and heuristic point removal schemes. It also lists the average performance using the heuristic and exact technique in the experiments of our mapper with rotation. As we can see, the performance of the circuits obtained with the heuristic and the circuits obtained with the exact technique are almost identical in all cases, demonstrating that the heuristic technique does not significantly reduce the circuits' quality. Moreover, the results demonstrate the memory and run-time efficiency of the heuristic. For example, when optimizing PE-SEND-IFC for average latency, the exact technique with rotation allocated 58,401 points, whereas the heuristic only allocated 1469 points. The run-time of the algorithm is reduced dramatically (1640.8s vs. 17.1s). For the five largest benchmark circuits, we used binning technique to keep the complexity manageable. Given equal bin sizes, the exact technique cannot complete after 1 hour on twelve of the experiments, whereas the heuristic technique completes in all cases.

### B.2 Analysis of impact of rotation on mapped circuits

We next performed experiments to determine the impact on our rotation scheme on the mapped circuits. The results, shown in Table V, show that rotation leads to better mapping in almost every circuit, most often yielding improvements of over 5%.

Interestingly, the results suggest that the improvement in the mapped circuits is sometimes limited by the structure of the circuit. For small shallow networks, such as Q42, the improvement of average latency in the mapped circuits is significantly smaller than observed in the un-mapped circuits. This is because a large reduction of critical path delay can be achieved when covering using unbalanced library graphs without doing any rotation. In some of the larger and deeper networks (e.g., BINARY-COUNTER, TSEND and SCSI), however, rotation yields substantial improvements in the delay of the mapped circuits. This suggests that, sometimes, less gates are needed to cover the critical path of the rotated networks compared to the un-rotated networks.

## *C. Comparison to worst-case mapped circuits*

We also compare our best average-case mapper (with rotation) to Chaudhary and Pedram's worst-case mapper and report the percentage improvement in average performance. It is very important to note that this comparison is not intended to yield an apple-to-apple measurement between comparable technology mappers but rather is intended to establish an estimate of how much our mapper can benefit from optimizing for the average case.

Table VI compares the average-case performance of the two technology mappers. The results demonstrate that our average-

| Number of surface points and run-time (sec) | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Circuits | Bin Size | AveLatency Optimization | | | | | | AveCycleTime Optimization | | | | | |
| | | Heuristic | | | Exact | | | Heuristic | | | Exact | | |
| | | AveL | Point | Time | AveL | Point | Time | AveC | Point | Time | AveC | Point | Time |
| merge | 0.0 | 1.82 | 74 | 0.3 | 1.82 | 92 | 0.3 | 1.76 | 75 | 0.4 | 1.76 | 95 | 0.4 |
| bufctrl1 | 0.0 | 2.14 | 114 | 0.1 | 2.13 | 156 | 0.2 | 2.54 | 121 | 0.4 | 2.62 | 363 | 0.5 |
| p3 | 0.0 | 1.01 | 164 | 0.5 | 1.00 | 602 | 0.8 | 1.28 | 191 | 1.7 | 1.23 | 1925 | 8.5 |
| q42 | 0.0 | 2.60 | 217 | 0.5 | 2.60 | 407 | 0.6 | 2.77 | 278 | 1.7 | 2.77 | 1951 | 5.1 |
| scsi-init-send | 0.0 | 2.29 | 293 | 1.3 | 2.28 | 542 | 1.4 | 2.29 | 310 | 3.1 | 2.28 | 1537 | 7.6 |
| dramc | 0.0 | 2.08 | 913 | 10.2 | 2.09 | 6297 | 58.6 | 2.16 | 1001 | 71.0 | 2.12 | 11166 | 339.3 |
| binary-counter | 0.0 | 1.54 | 720 | 14.8 | 1.53 | 2537 | 17.4 | 2.95 | 772 | 38.4 | 3.13 | 20607 | 313.6 |
| pe-send-ifc | 0.0 | 2.74 | 1469 | 17.1 | 2.74 | 58401 | 1640.8 | 3.14 | 1633 | 90.5 | – | – | – |
| trcv | 0.0 | 3.57 | 2820 | 94.1 | – | – | – | 3.55 | 2938 | 474.8 | – | – | – |
| ircv | 0.0 | 3.17 | 2632 | 76.9 | – | – | – | 3.18 | 2623 | 367.7 | – | – | – |
| tsend | 0.01 | 3.92 | 2308 | 420.5 | 3.99 | 10449 | 1039.3 | 3.91 | 2342 | 615.8 | – | – | – |
| p1 | 0.01 | 3.62 | 3132 | 250.7 | – | – | – | 4.20 | 3206 | 668.1 | – | – | – |
| isend | 0.05 | 4.55 | 2820 | 428.0 | 4.55 | 6835 | 757.1 | 4.62 | 2818 | 775.6 | – | – | – |
| cache-ctrl | 0.05 | 5.34 | 3818 | 1900.0 | 5.39 | 8774 | 2283.0 | 5.44 | 3789 | 1851.1 | – | – | – |
| scsi | 0.1 | 4.21 | 2566 | 2605.6 | 4.29 | 5542 | 2722.8 | 4.39 | 2588 | 2540.9 | – | – | – |

TABLE IV
COMPARISON OF COVERING ALGORITHMS USING EXACT VS. LIKELY INFERIORITY HEURISTICS. AVEL CORRESPONDS TO AVERAGE LATENCY AND AVEC CORRESPONDS TO AVERAGE CYCLE TIME. IN ADDITION, "–" DENOTES TIME-OUT.

case mapper can yield significant average-case performance improvements over the traditional worst-case mapper. The source of improvements can be decomposed in two major factors. First, we find and optimize only the true critical path delay for each pattern. Second, we find the probability for each pattern and prioritize the mapping for patterns with higher probabilities. Since the circuits MERGE, BUFCTRL1, Q42 and BINARY-COUNTER have uniform pattern frequencies, the obtained improvements can be attributed to the first factor. The other circuits have a more skewed distribution of pattern probabilities and thus the combination of both factors lead to significant improvement.

Table VII shows that for 12 out of the 15 circuits tested our area is smaller than the area obtained using the worst-case mapper. This area reduction may be attributed to the fact that our algorithm uses large gates only on highly-frequent critical paths.

### *D. Post-layout results*

To further validate our results we integrated our techniques with the Mentor Graphics physical design tools and evaluated the automatically-generated layout using $0.8\mu m$ Government CMOSN standard cells. Specifically, we extracted the wire-capacitances from the layout and used them to analyze the impact on wire-delays. The results demonstrate that the wire capacitance changed the average performance by very little. Consequently, our post-layout improvements are similar in magnitude to our pre-layout values. The delay contribution of wire delay is small because the Mentor Graphic's router does a good job of keeping wires short for these relatively small circuits and because the standard cells in the CMOSN library are relatively large which makes their intrinsic gate delay dominate the associated wire delay.

## VII. CONCLUSIONS

We presented a technology mapping technique for optimizing the average performance of asynchronous burst-mode control circuits. We use stochastic techniques to determine the relative frequency of occurrence of each state transition and extend known analysis, decomposition, and covering techniques to optimize the mapped circuits for average latency/cycle time. Our results demonstrate that our techniques can simultaneously lead to significantly higher average performance and significantly smaller area when compared to traditional techniques. We also believe that these technology mapping algorithms are applicable to other fundamental-mode design styles, such as Nowick's UCLOCK method [23].

Extensions of our approach to fundamental-mode circuits implemented with generalized C-elements [37] is an interesting area of future research. Using generalized C-elements often increases the performance of the circuits, but their optimization would most likely involve handling hazard-free sequential decomposition as well as transistor-level delay analysis. Another possible direction for future work is to extend this type of average-case optimization to other control circuit design styles (e.g., quasi-delay insensitive, speed-independent, timed, etc.) or other levels of the design hierarchy (e.g., architectural and logic synthesis, placement and routing, fanout optimization, etc.).

Lastly, it may be possible to apply our average-case methodology to the technology mapping of synchronous telescopic units [2]. These blocks of combinational logic have variable latency (they either take one or two clock cycles) and mapping them for the average case can increase the probability that the unit takes only one clock cycle, thereby increasing system performance.

| Average-Case Performance of Mapped Circuits | | | | | | |
|---|---|---|---|---|---|---|
| Circuits | AveLat | | | AveCyc | | |
| | ACNR | ACR | Imp. | ACNR | ACR | Imp. |
| merge | 2.01 | 1.82 | 9% | 1.91 | 1.76 | 8% |
| bufctrl1 | 2.33 | 2.14 | 8% | 2.57 | 2.54 | 1% |
| p3 | 1.04 | 1.01 | 3% | 1.30 | 1.28 | 2% |
| q42 | 2.62 | 2.60 | 1% | 2.86 | 2.77 | 3% |
| scsi-init-send | 2.57 | 2.29 | 11% | 2.68 | 2.29 | 15% |
| dramc | 2.15 | 2.08 | 3% | 2.18 | 2.16 | 1% |
| binary-counter | 1.76 | 1.54 | 13% | 3.28 | 2.95 | 10% |
| pe-send-ifc | 3.02 | 2.74 | 9% | 3.43 | 3.14 | 8% |
| trcv | 3.85 | 3.57 | 7% | 3.95 | 3.55 | 10% |
| ircv | 3.38 | 3.17 | 6% | 3.62 | 3.18 | 12% |
| tsend | 4.49 | 3.92 | 13% | 4.42 | 3.91 | 11% |
| p1 | 3.95 | 3.62 | 8% | 4.67 | 4.20 | 10% |
| isend | 4.74 | 4.55 | 4% | 4.76 | 4.62 | 3% |
| cache-ctrl | 5.92 | 5.34 | 10% | 6.13 | 5.44 | 11% |
| scsi | 5.13 | 4.21 | 18% | 5.39 | 4.39 | 18% |

TABLE V

COMPARISON OF AVERAGE PERFORMANCE (MEASURED IN NANO-SECONDS) OF ROTATED AND UNROTATED NETWORKS. ACNR CORRESPONDS TO AVERAGE-CASE MAPPING WITHOUT ROTATION. ACR CORRESPONDS TO AVERAGE-CASE MAPPING WITH ROTATION.

| Area of Average-Case and Worst-Case Optimized Circuits | | | | | | |
|---|---|---|---|---|---|---|
| Circuits | AveLat Opt. | | | AveCyc Opt. | | |
| | ACR | WC | Imp. | ACR | WC | Imp. |
| merge | 11 | 15 | 29% | 12 | 15 | 21% |
| bufctrl1 | 14 | 13 | -3% | 15 | 13 | -10% |
| p3 | 15 | 20 | 23% | 19 | 20 | 5% |
| q42 | 24 | 23 | -2% | 21 | 23 | 8% |
| scsi-init-send | 29 | 32 | 10% | 29 | 32 | 9% |
| dramc | 64 | 68 | 7% | 64 | 68 | 5% |
| binary-counter | 76 | 78 | 3% | 73 | 78 | 6% |
| pe-send-ifc | 79 | 87 | 9% | 80 | 87 | 8% |
| trcv | 153 | 173 | 12% | 152 | 173 | 12% |
| ircv | 158 | 175 | 10% | 162 | 175 | 7% |
| tsend | 283 | 295 | 4% | 292 | 295 | 1% |
| p1 | 333 | 350 | 5% | 320 | 350 | 8% |
| isend | 431 | 459 | 6% | 432 | 459 | 6% |
| cache-ctrl | 705 | 645 | -9% | 691 | 645 | -8% |
| scsi | 547 | 559 | 2% | 554 | 559 | 1% |

TABLE VII

AREA(X$10^3$) COMPARISON BETWEEN AVERAGE-CASE AND WORST-CASE MAPPED CIRCUITS. ACR CORRESPONDS TO CIRCUITS OPTIMIZED FOR THE AVERAGE-CASE WITH ROTATION. WC CORRESPONDS TO CIRCUITS OPTIMIZED FOR THE WORST-CASE.

| Perf. of Average-Case and Worst-Case Optimized Circuits | | | | | | |
|---|---|---|---|---|---|---|
| Circuits | AveLat Opt. | | | AveCyc Opt. | | |
| | ACR | WC | Imp. | ACR | WC | Imp. |
| merge | 1.82 | 1.97 | 7% | 1.76 | 1.97 | 10% |
| bufctrl1 | 2.14 | 2.34 | 8% | 2.54 | 2.60 | 2% |
| p3 | 1.01 | 1.09 | 7% | 1.28 | 1.40 | 9% |
| q42 | 2.60 | 2.61 | 0% | 2.77 | 2.93 | 5% |
| scsi-init-send | 2.29 | 2.72 | 16% | 2.29 | 2.72 | 16% |
| dramc | 2.08 | 2.40 | 13% | 2.16 | 2.40 | 10% |
| binary-counter | 1.54 | 1.81 | 15% | 2.95 | 3.63 | 19% |
| pe-send-ifc | 2.74 | 3.56 | 23% | 3.14 | 3.93 | 20% |
| trcv | 3.57 | 4.33 | 18% | 3.55 | 4.40 | 19% |
| ircv | 3.17 | 3.89 | 18% | 3.18 | 4.36 | 27% |
| tsend | 3.92 | 4.90 | 20% | 3.91 | 5.05 | 22% |
| p1 | 3.62 | 4.11 | 12% | 4.20 | 4.87 | 14% |
| isend | 4.55 | 4.96 | 8% | 4.62 | 5.15 | 10% |
| cache-ctrl | 5.34 | 6.02 | 11% | 5.44 | 6.03 | 10% |
| scsi | 4.21 | 5.61 | 25% | 4.39 | 5.66 | 22% |

TABLE VI

AVERAGE-CASE PERFORMANCE COMPARISON BETWEEN AVERAGE-CASE AND WORST-CASE MAPPERS. ACR CORRESPONDS TO CIRCUITS OPTIMIZED FOR THE AVERAGE-CASE WITH ROTATION. WC CORRESPONDS TO CIRCUITS OPTIMIZED FOR THE WORST-CASE. DELAYS ARE MEASURED IN NANO-SECONDS.

| Experimental Results for Post-Layout Circuits | | | | | | |
|---|---|---|---|---|---|---|
| Average Latency | | | | | | |
| | Ave-Case | | Worst-Case | | Imp. | |
| Circuits | NCP | WCP | NCP | WCP | NCP | WCP |
| merge | 0.71 | 0.73 | 0.76 | 0.78 | 6% | 6% |
| bufctrl1 | 0.92 | 0.95 | 1.20 | 1.24 | 23% | 23% |
| p3 | 0.47 | 0.49 | 0.50 | 0.52 | 5% | 5% |
| q42 | 1.27 | 1.31 | 1.31 | 1.35 | 3% | 3% |
| scsi-init-send | 1.10 | 1.13 | 1.29 | 1.34 | 15% | 15% |
| dramc | 0.86 | 0.89 | 0.92 | 0.95 | 6% | 6% |
| binary-counter | 0.64 | 0.67 | 0.74 | 0.77 | 14% | 14% |
| pe-send-ifc | 1.34 | 1.39 | 1.58 | 1.65 | 16% | 16% |
| trcv | 1.52 | 1.58 | 1.67 | 1.73 | 9% | 9% |
| ircv | 1.48 | 1.54 | 1.98 | 2.06 | 25% | 25% |
| tsend | 1.70 | 1.77 | 2.00 | 2.09 | 15% | 15% |
| p1 | 1.56 | 1.63 | 1.75 | 1.83 | 11% | 11% |
| isend | 1.70 | 1.78 | 1.89 | 1.97 | 10% | 10% |
| cache-ctrl | 1.70 | 1.78 | 1.93 | 2.03 | 12% | 12% |
| scsi | 1.33 | 1.39 | 1.71 | 1.79 | 22% | 22% |

TABLE VIII

EXPERIMENTAL RESULTS COMPARING THE AVERAGE LATENCY OF AVERAGE-CASE AND WORST-CASE MAPPED CIRCUITS INCLUDING BACK-ANNOTATED WIRE-CAPACITANCE FROM POST-LAYOUT CIRCUITS. WCP (NCP) CORRESPONDS TO CIRCUITS WITH WIRING CAPACITANCE (NOT) INCLUDED. DELAYS ARE MEASURED IN NANO-SECONDS.

| Experimental Results for Post-Layout Circuits | | | | | | |
|---|---|---|---|---|---|---|
| Average Cycle Time | | | | | | |
| | Ave-Case | | Worst-Case | | Imp. | |
| Circuits | NCP | WCP | NCP | WCP | NCP | WCP |
| merge | 0.71 | 0.73 | 0.76 | 0.78 | 6% | 6% |
| bufctrl1 | 1.15 | 1.18 | 1.22 | 1.26 | 6% | 6% |
| p3 | 0.59 | 0.61 | 0.67 | 0.69 | 12% | 12% |
| q42 | 1.31 | 1.35 | 1.32 | 1.37 | 1% | 1% |
| scsi-init-send | 1.22 | 1.26 | 1.29 | 1.34 | 5% | 6% |
| dramc | 0.86 | 0.89 | 0.92 | 0.95 | 6% | 6% |
| binary-counter | 1.20 | 1.25 | 1.46 | 1.51 | 18% | 18% |
| pe-send-ifc | 1.49 | 1.55 | 1.76 | 1.83 | 16% | 16% |
| trcv | 1.59 | 1.66 | 1.67 | 1.74 | 5% | 5% |
| ircv | 1.38 | 1.44 | 1.99 | 2.06 | 31% | 30% |
| tsend | 1.76 | 1.84 | 2.07 | 2.16 | 15% | 15% |
| p1 | 1.73 | 1.80 | 1.97 | 2.05 | 12% | 12% |
| isend | 1.81 | 1.90 | 1.93 | 2.01 | 6% | 5% |
| cache-ctrl | 1.72 | 1.80 | 1.94 | 2.04 | 11% | 12% |
| scsi | 1.47 | 1.53 | 2.07 | 2.16 | 29% | 29% |

TABLE IX

EXPERIMENTAL RESULTS COMPARING AVERAGE CYCLE TIME OF AVERAGE-CASE AND WORST-CASE MAPPED CIRCUITS INCLUDING BACK-ANNOTATED WIRE-CAPACITANCE FROM POST-LAYOUT CIRCUITS. WCP (NCP) CORRESPONDS TO CIRCUITS WITH WIRING CAPACITANCE (NOT) INCLUDED. DELAYS ARE MEASURED IN NANO-SECONDS.

## ACKNOWLEDGMENTS

We would like to acknowledge M. Pedram at USC for inspirational discussions on technology mapping and his former student C.-L. Pu for his invaluable guidance through their technology mapping software. Also, we would like to thank J. Martin at UCI for drafting many of the figures in the paper. Lastly, we are indebted to the anonymous reviewers whose critical feedback has made this a better paper.

## REFERENCES

[1] P. A. Beerel, C. J. Myers, and T. H.-Y. Meng, "Covering conditions and algorithms for the synthesis of speed-independent circuits," *IEEE Transactions on Computer-Aided Design*, vol. 17, no. 3, pp. 205–219, Mar. 1998.

[2] L. Benini, E. Macii, M. Poncino, and G. De Micheli, "Telescopic units: a new paradigm for performance optimization of VLSI designs," *IEEE Transactions on Computer-Aided Design*, vol. 17, no. 3, pp. 220–232, Mar. 1998.

[3] J. Benkoski, E. Vanden Meersch, L. J. M. Claesen, and H. De Man, "Timing verification using statically sensitizable paths," *IEEE Transactions on Computer-Aided Design*, vol. 9, no. 9, pp. 1073–1084, Sept. 1990.

[4] K. van Berkel, R. Burgess, J. Kessels, A. Peeters, M. Roncken, and F.Schalij, "Asynchronous circuits for low power: A DCC error corrector," *IEEE Design & Test of Computers*, vol. 11, no. 2, pp. 22–32, Summer 1994.

[5] S. M. Burns, "General conditions for the decomposition of state holding elements," In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Mar. 1996, pp. 48–57.

[6] S.Chakraborty, D. L. Dill, K.-Y. Chang, and K. Y. Yun, "Timing analysis for extended burst-mode circuits," In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Apr. 1997, pp. 101–111.

[7] K. Chaudhary and M. Pedram, "Computing the area versus delay trade-off curves in technology mapping," *IEEE Transactions on Computer-Aided Design*, vol. 14, no. 12, pp. 1480–1489, Dec. 1995.

[8] H. Chen and D. Du, "Path sensitization in critical path problem," *IEEE Transactions on Computer-Aided Design*, vol. 12, no. 2, pp. 196–207, Feb. 1993.

[9] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, E. Pastor, and A. Yakovlev, "Decomposition and technology mapping of speed-independent circuits using boolean relations," In *Proc. International Conf. Computer-Aided Design (ICCAD)*, Nov. 1997, pp. 220–227.

[10] S. Furber, "Computing without clocks: Micropipelining the ARM processor," in G. Birtwistle and A. Davis, eds., *Asynchronous Digital Circuit Design*, Workshops in Computing, pp. 211–262. Springer-Verlag, 1995.

[11] R. Hitchcock, G. Smith, and D. Cheng, "Timing analysis of computer hardware," *IBM journal of Research and Development*, vol. 26, no. 1, pp. 100–105, Jan. 1982.

[12] S. Iman and M. Pedram, "POSE: Power optimization and synthesis environment," in *Proc. ACM/IEEE Design Automation Conference*, Jun. 1996, pp. 21–26.

[13] K. Keutzer, "DAGON: Technology binding and local optimization by DAG matching," in *Proc. ACM/IEEE Design Automation Conference*, Jun. 1987, pp. 341–347.

[14] A. Kondratyev, M. Kishinevsky, J. Cortadella, L. Lavagno, and A. Yakovlev, "Technology mapping for speed-independent circuits: Decomposition and resynthesis," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Apr. 1997, pp. 240–253.

[15] D. Kung, "Path sensitization in hazard-free circuits," in *Proc. International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, 1995.

[16] D. S. Kung, "Hazard-non-increasing gate-level optimization algorithms," in *Proc. International Conf. Computer-Aided Design (ICCAD)*, Nov. 1992, pp. 631–634.

[17] 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.

[18] F. Mailhot, "Algorithms for technology mapping based on binary decision diagrams and on boolean operators," *IEEE Transactions on Computer-Aided Design*, vol. 12, no. 6, pp. 599–620, Dec. 1993.

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

[20] P. McGeer and R. Brayton, "Efficient algorithms for computing the longest viable path in a combinational circuit," in *Proc. ACM/IEEE Design Automation Conference*, Jun. 1989, pp. 561–567.

[21] C. J. Myers, P. A. Beerel, and T. H.-Y. Meng, "Technology-mapping of timed-circuits," in *Proc. Working Conference on Asynchronous Design Methodologies*, May 1995, pp. 138–147.

[22] S. M. Nowick, "Automatic Synthesis of Burst-Mode Asynchronous Controllers," PhD dissertation, Stanford University., Dept. of Computer Science, Stanford University, Stanford, CA, 1993.

[23] S. M. Nowick and B. Coates, "UCLOCK: automated design of high-performance unclocked state machines," in *Proc. of International Conference on Computer Design (ICCD)*, Oct. 1994, pp. 434–441.

[24] S. Perremans, L. Claesen, and H. DeMan, "Static timing analysis of dynamically sensitizable paths," in *Proc. ACM/IEEE Design Automation Conference*, Jun. 1989, pp. 568–573.

[25] S. Ross, *Introduction to Probability Models*, Academic Press, 1985.

[26] S. Rotem, K. Stevens, R. Ginosar, P. Beerel, C. Myers, K. Yun, R. Kol, C. Dike, M. Roncken, and B. Agapiev, "RAPPID: An asynchronous instruction length decoder," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Apr. 1999, pp. 60–70.

[27] J. P. Roth, "Diagnosis of automata failures: A calculus and a new method," *IBM Journal of Research and Development*, pp. 278–281, Oct. 1966.

[28] R. Rudell, "Logic Synthesis for VLSI Design," PhD dissertation, U. C. Berkeley, Apr. 1989. Memorandum UCB/ERL M89/49.

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

[30] P. Siegel, G. De Micheli, and D. Dill, "Automatic technology mapping for generalized fundamental-mode asynchronous designs," in *Proc. ACM/IEEE Design Automation Conference*, Jun. 1993, pp. 61–67.

[31] P. Siegel and G. De Micheli, "Decomposition methods for library binding of speed-independent asynchronous designs," In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pp. 558–565, Nov. 1994.

[32] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, Jun. 1989.

[33] H. J. Touati, C. W. Moon, R. K. Brayton, and A. Wang, "Performance-oriented technology mapping," in W. J. Dalley, ed., *MIT Conference on Advanced VLSI*, 1995, pp. 79–97.

[34] C.-Y. Tsui, M. Pedram, and A. M. Despain, "Power efficient technology decomposition and mapping under an extended power consumption model," *IEEE Transactions on Computer-Aided Design*, vo. 13, no. 9, pp. 1110–1122, Sept. 1995.
[35] S. H. Unger, *Asynchronous Sequential Switching Circuits*, Wiley-Interscience, John Wiley & Sons, Inc., New York, 1969.
[36] K. Y. Yun, "Synthesis of Asynchronous Controllers for Heterogeneous Systems," PhD dissertation, Stanford University, Dept. of Electrical Engineeering, Stanford University, Stanford, CA, Aug. 1994.
[37] K. Y. Yun, "Automatic synthesis of extended burst-mode circuits using generalized C-elements," in *Proc. European Design Automation Conference (EURO-DAC)*, Sept. 1996, pp. 290–295.
[38] 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," *IEEE Transactions on VLSI Systems*, vol. 6, no. 4, pp. 643–655, Dec. 1998.
[39] K. Y. Yun and D. L. Dill, "Automatic synthesis of extended burst-mode circuits: part I (specification and hazard-free implementations)," *IEEE Transactions on Computer-Aided Design*, vol. 18, no. 2, pp. 101–117, Feb. 1999.
[40] K. Y. Yun and D. L. Dill, "Automatic synthesis of extended burst-mode circuits: part II (automatic synthesis)," *IEEE Transactions on Computer-Aided Design*, vol. 18, no. 2, pp. 118–132, Feb. 1999.

**W**ei-Chun Chou (Student Member, IEEE) received the B.S. degree in Nuclear Engineering from National Tsing Hua University, Hsinchu, Taiwan, in 1985, the M.S. degree in Electricl Engineering from National Taiwan University, Taipei, Taiwan, in 1992, and the Ph.D. degree in Electrical Engineering from the University of Southern California, Los Angeles, in 1999. In 1999, he joined Sun Microsystems, Sunnyvale, as a senior CAD engineer. His research interestes include logic synthesis, high-level synthesis, asynchronous design, and noise analysis.

**D**r. Beerel (Member, IEEE) received his B.S.E. degree in Electrical Engineering from Princeton University in 1989 and his M.S. and Ph.D. degrees in Electrical Engineering from Stanford University in 1991 and 1994, respectively. He was a member of the technical program committee of the Second Working Conference on Asynchronous Design Methodologies, the 1995 ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU'95), and the third International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC '97). He was Program Co-chair for ASYNC'98. His research interests include CAD, formal verification, and mixed asynchronous/synchronous VLSI systems.

Dr. Beerel won the Junior Research Award of the School of Engineering in 1998 and was a recipient of the School of Engineering's Outstanding Teaching Award in 1997. He was also co-winner of the Charles E. Molnar award for two papers published in ASYNC'97 that best bridged theory and practice of asynchronous system design. He also recieved an NSF Career Award and a 1995 Zumberge Fellowship. He also was a primary consultant for Intel Corporation on their Asynchronous Instruction Decoder Project.

**K**enneth Y. Yun (Member, IEEE) received the Ph.D. degree in Electrical Engineering from Stanford University, Stanford, CA and the M.S. degree in Electrical Engineering and Computer Science from Massachusetts Institute of Technology, Cambridge. He is currently an assistant professor in the Electrical and Computer Engineering Department at University of California, San Diego. He had held design engineering positions at TRW and Hitachi for 6 years. His current research interests include high-speed networking, asynchronous circuits and systems, and synchronization issues. He has been working with Intel Corp. as a primary consultant on the Asynchronous Instruction Decoder Project.

Dr. Yun has organized the Fourth International Conference on Advanced Research in Asynchronous Circuits and Systems as a program co-chair. He is a recipient of the 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. He has received best paper awards at the 1998 International Conference on Computer Design and at the 1999 International Symposium on Advanced Research on Asynchronous Circuits and Systems.