Optimizing average-case delay in technology mapping of burst-mode circuits

Peter A. Beerel*
EEB 350, MC 2562
EE-Systems Dept.
USC
Los Angeles, CA 90089

Kenneth Y. Yun†
EBU1-4402, MC 0407
Dept. of ECE
UCSD
La Jolla, CA 92093

Wei-chun Chou
EEB 350, MC 2562
EE-Systems Dept.
USC
Los Angeles, CA 90089

Abstract

This paper presents technology mapping techniques that optimize for average case delay of asynchronous burst-mode control circuits. First, the specification of the circuit is analyzed using stochastic techniques to determine the relative frequency of occurrence of each state transition. Then, subject to timing and area constraints, the technology mapper minimizes the sum of the cycle times of the state transitions, weighted by their relative frequencies. Unlike other technology mappers, our mapper is based on the single step transition model for delay which finds the true critical paths, avoiding the false path problem.

1 Introduction

The goal of logic synthesis is to produce a circuit which satisfies a given specification (*e.g.*, a set of logic equations, a finite state machine, a signal transition graph), occupies minimal area, and meets given timing constraints. Most logic synthesis systems split this task into two phases, a technology independent phase and a technology dependent phase. The first phase involves the synthesis of the specification into an optimized network of technology-independent logic equations. These equations represent ideal gates with simple models for area (*e.g.*, literal count) and delay (*e.g.*, unit delay). The second phase is referred to as *technology mapping*, and its purpose is to finish the synthesis by mapping the optimized equations to gates from a target cell library. Unlike technology-independent synthesis, technology mapping considers more detailed models of gate area (*e.g.*, actual gate area) and delay (*e.g.*, pin-to-pin delays, effects of capacitive loading, and distinct rise and fall times).

While numerous tools and techniques for technology mapping of synchronous circuits [5, 15, 19, 21, 8] optimize for a combination of area, delay, and/or power, the few tools and techniques developed for asynchronous circuits focus on ensuring hazard-freedom during technology mapping [10, 17, 16, 7] and do not consider optimizations particular to asynchronous circuits. One reason for this lack of attention is that the technology mapping step in many design methodologies (*e.g.*, Micropipelines [18, 4], Tangram [23], and Martin Synthesis [9]) is greatly simplified by the use of specialized cell libraries. For other control-oriented synthesis methodologies, such as speed-independent design, timed-circuit design, and burst-mode design, however, gate-level implementations using existing industrial standard-cell libraries are attractive because designing, maintaining, and interfacing other tools with specialized cell libraries for asynchronous design is very expensive. In fact, our motivation for this work arises from the Asynchronous Instruction Decoder Project at Intel Corporation. In this project, the lack of tools and techniques forced us to manually map many control circuits onto Intel's cell library. This was both labor-intensive and inexact since we used back-of-the-envelope techniques to decide amongst various mappings.

This paper deals with the technology mapping of asynchronous burst-mode control circuits. These circuits are specified with an extended burst-mode (XBM) diagram and are implemented using a modified Huffman architecture [25, 26, 12]. As an example, the extended burst-mode specification and implementation of the example *scsi-init-send* are given in Figure 1. This architecture has the advantage of very low latency because the input-to-output path is purely combinational, *i.e.*, no explicit storage elements are

*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, a NSF Career Award, and a gift from Intel Corporation.

†This research was supported in part by a gift from Intel Corporation.

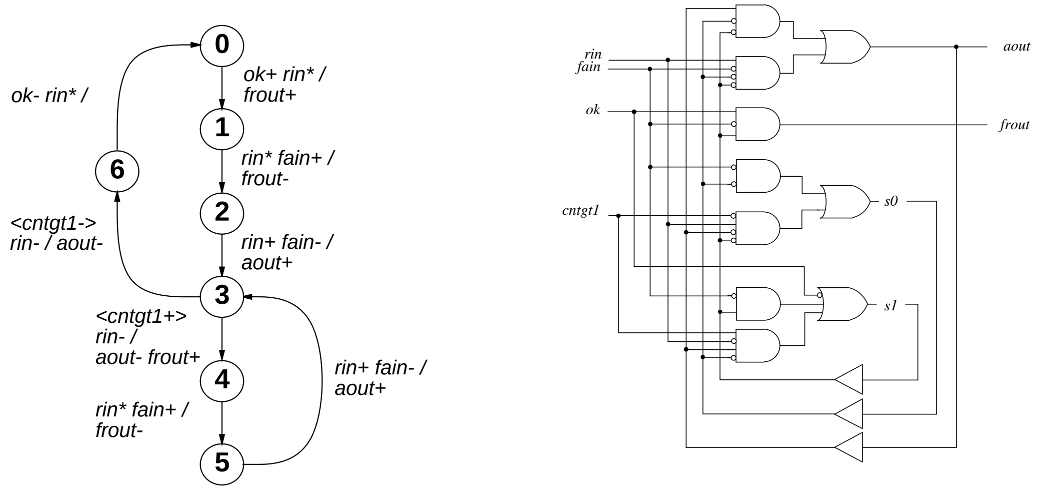

Figure 1: Scsi-init-send example.

used. In each state of the machine, the circuit waits for a set of specified input transitions, referred to as an *input burst*, then changes a variety of output signals, referred to as an *output burst*. In addition, depending on the implementation style, the circuit may internally set any number of state signals either concurrently with the output signals or in a separate *state variable burst* either before or after the output burst. For the machine to operate correctly, the environment must obey the *generalized fundamental-mode restriction* which means it can complete the state transition cycle by introducing a new input burst only after the circuit changes state and has settled [25].

It is important to observe that a variety of different input bursts may occur in a state, driving the circuit into different modes of activity. For instance, in the *scsi-init-send* example there are two possible state transitions from state 3. The first (to state 4) represents the sending of another byte, while the second (to state 6) represents the signaling of the end of transmission. Since an output signal is often specified to change in multiple state transitions, its logic may contain different paths that are executed in different state transitions. To optimize for average cycle-time, the delay along these paths should be prioritized according to the relative frequency of the associated state transition.

We calculate this priority using Markov chain analysis of the burst-mode specification based on the conditional probabilities of different state transitions. These conditional probabilities may be given by the user or estimated through behavioral simulation of the circuit in its environment. In the *scsi-init-send* application, the conditional probabilities depend on the packet size of SCSI transfers. For example, a reasonable packet size is 1K bytes, in which case the conditional probabilities would be as follows: given the circuit is in state 3, the probability of going to state 4 next is 0.999 while the probability of going to state 3 next is 0.001. Using Markov chain analysis we determine the long run proportion of executing each state transition. The results are used to create the set of input *pattern structures* that guide the technology mapper to optimize for average cycle-time.

The first step of our technology mapping is the decomposition of the optimized equations into a set of available base functions, such as 2-input NAND-gates and inverters. This decomposition is necessary to ensure each node in the network can be implemented by at least one library gate. This decomposition also increases the granularity of the network which provides more mapping options, yielding better final circuits [15]. There are many decompositions possible and the choice affects the final quality of the technology mapping solution [15]. We present a simple heuristic that optimizes the decomposition for average-case cycle-time.

After the technology decomposition, the next step of the technology mapping algorithm is to optimally cover the network of base functions with available gates. Our approach is motivated by Chaudhary and Pedram's technology mapper [3] which considers area-delay tradeoffs based on static (pessimistic) timing analysis. Instead of considering a single worst-case delay through the network, we calculate the delays incurred by different input patterns and consider the tradeoffs possible among different pattern delays. This pattern-based approach facilitates the application of the *single step transition model* for delay analysis [6] which finds the true critical paths. Our covering algorithm can thus provide the optimal technology mapping solution for average cycle-time subject to timing and area constraints under the assumptions of fixed feedback delay and fixed load capacitance. It produces near-optimal results when the effects of realistic load capacitances are considered.

The remainder of the paper is organized as follows. Section 2 provides the necessary background on technology mapping for both synchronous and asynchronous circuits and provides an introduction to burst-mode circuits. Section 3 describes how to determine the pattern structures for a burst-mode circuit that guides optimizations within the technology mapper. Section 4 then details the technology mapping algorithm. Section 5 reports our preliminary experimental results. Finally, Section 6 presents our conclusions and plans for future work.

2 Background and previous work

This section first describes previous work on technology mapping for both synchronous and asynchronous circuits and then reviews burst-mode circuits and their average-case performance.

2.1 Technology mapping of synchronous circuits

For synchronous circuits, technology mapping is often reduced to DAG covering that is efficiently approximated by a sequence of optimal tree coverings. A set of base functions is chosen, such as a 2-input NAND gate and an INVERTER. The optimized equations (obtained from the technology independent optimization) are decomposed into a graph where each node is one of the base functions. This graph is called a *subject graph*. Each library gate is also represented by one or more *pattern graphs* consisting of only base functions. The technology mapping problem is then reduced to the problem of finding a minimum cost covering of the subject graph from a collection of pattern graphs for all gates in the library. For area optimization, the cost of a cover is defined as the sum of the gate areas. For worst-case delay optimization in a synchronous circuit, the cost of a cover is defined as the worst-case critical path of the circuit. Both Chaudhary and Pedram [3] and Touati [19] extend these works to solve the minimum area under delay constraints. However, they consider only synchronous circuits and employ pessimistic static timing analysis to determine the worst-case critical paths. As a result, the critical path found may never propagate an event and the true delay can be lower. This problem is called the *false path problem* and to our knowledge has never before been addressed within the scope of technology mapping.

2.2 Technology mapping of asynchronous circuits

For some asynchronous circuits, the key obstacle to technology mapping is the decomposition into a set of base functions. For example, in speed-independent circuits even simple decompositions (*e.g.*, decomposing a 3-input AND into 2-input AND gates) can yield hazards due to an extra unbounded delay element introduced. Techniques to decompose the circuit into a limited-fanin basis have been developed [24], but to our knowledge only apply to a very limited class of circuits which do not allow environmental choice. For more general circuits, techniques to add *acknowledgement wire forks* to remove hazards caused by decomposition have been developed [16, 1]. These techniques, however, do not work on all circuits [16, 1]. For speed-independent circuits, therefore, we currently must assume the existence of an unrealistic library that contains unlimited-fanin gates in order to guarantee that a technology-mapping solution exists. Myers *et al.* studied hazard-free decomposition techniques for timed circuits [10], however, cannot guarantee a solution either. Timed circuit decomposition is also made complicated because the additional delay in the network can sometimes violate the interface timing constraints.

More promising results were obtained by Lavagno *et al.* for bounded-delay circuits specified by signal transition graphs [7]. They applied synchronous technology mapping procedures first and then added delay elements to remove any hazards introduced by the mapping [7]. This technique demonstrated that synchronous technology mapping solutions can sometimes be used, but the added delay elements may limit performance.

For fundamental mode designs, such as burst-mode circuits, Siegel and De Micheli showed that with only small modifications, synchronous technology mapping solutions can be safely used [17]. In particular, Siegel and De Micheli use results from Unger [22] to demonstrate that the decomposition into base functions does not introduce hazards. Moreover, Siegel and De Micheli present an algorithm to identify library gates which might in some cases cause a hazard if used in a fundamental mode circuit. The results demonstrate 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 used to determine if the match causes a hazard in that instance [17]. These results prove the existence of a technology mapping solution for fundamental mode designs and show how synchronous techniques can be appropriately modified. The key shortcoming of this technique, which this paper addresses, is that the underlying synchronous technology mappers are limited to optimizing worst-case performance, not average-case performance.

2.3 Burst-mode control circuits

We now describe the specification, architecture, and the operation of burst-mode circuits.

2.3.1 Extended burst-mode specifications

An extended burst-mode asynchronous circuit [25] is specified by a state diagram which consists of a finite number of states, a set of labeled state transitions connecting pairs of states, and a start state.

Figure 1.a shows the extended burst-mode specification of the *scsi-init-send* circuit. Signals not enclosed in angle brackets and ending with + or − are *terminating edge signals.* The signals enclosed in angle brackets are *conditionals*, which are level signals whose values are sampled when all of the terminating edges associated with them have occurred. A conditional $\langle a^+ \rangle$ denotes "if a is high" and $\langle a^- \rangle$ denotes "if a is low." A state transition occurs only if all of the conditions are met and all the terminating edges have appeared. A signal ending with an asterisk is a *directed don't care.* If a is a directed don't care, there must be a sequence of state transitions in the machine labeled with a^*. If a state transition is labeled with a^*, the following state transitions in the machine must be labeled with a^* or with a^+ or a^- (the terminating edge for the directed don't care). The *scsi-init-send* specification has a conditional input *cntgt1*, 3 edge inputs (*ok*, *rin*, *fain*), and 2 outputs (*aout*, *frout*). Consider the state transitions out of state 3. The behavior of the machine at this point is: "if *cntgt1* is low when *rin* falls, change the current state from 3 to 6 and lower the output *aout*; if *cntgt1* is high when *rin* falls, change the current state from 3 to 4 and lower *aout* and raise *frout*."

A directed don't care may change at most once during a sequence of state transitions it labels, *i.e.*, directed don't cares are *monotonic* signals, and, if doesn't change during this sequence, it must change during the state transition its terminating edge labels. A terminating edge which is not immediately preceded by a directed don't care is called *compulsory*, since it *must* appear during the state transition it labels. In Figure 1, *rin* is low when the specification is in state 4. It can rise at any point as the machine moves through state 5 but it must have risen by the time the machine moves to state 3 because the terminating edge rin^+ appears between states 5 and 3.

2.3.2 3D implementation style

In this paper, we assume that XBM specifications are implemented in *3-D* design style [26]. A 3-D machine is formally represented as a 4-tuple (X, Y, Z, δ), where X is a set of primary input symbols, Y is a set of primary output symbols, Z is a possibly empty set of state variable symbols, and $\delta : X \times Y \times Z \to Y \times Z$ is a *next-state function.* The hardware implementation of a 3-D machine is a combinational network, which implements the next-state function, with the outputs of the network fed back as inputs to the network. There are no explicit storage elements such as latches, flip-flops, or C-elements in a 3-D machine.

2.3.3 State transitions in a 3D machine

The state transitions in 3D machines can be designed in two ways. First, state transitions can have *two-phases* in which the input burst is followed by an output/state variable burst where state variables change concurrently with outputs. Second, state transitions can have *three-phases* in which a state variable burst that either precedes or follows the output burst [25].

2.3.4 Performance modeling and optimizations for state transitions

For burst-mode state machines, two performance measurements for a state transition have been defined [25]. *Latency* is the delay from the last terminating input edge of an input burst to the last edge of the resulting output burst [25]. *Cycle time* is the delay required to avoid circuit malfunction from the last terminating input edge of an input burst to the first compulsory input edge of the next input burst [25].

For the purposes of this paper, a *pattern* is an assignment of input signals and fed-back state and output signals to binary values. The *pattern arrival time* at a node in the circuit for a particular pattern is the time between the application of the pattern and the time the node changes. It is not defined for nodes that do not change in response to the particular pattern. Every burst defines a pattern whose pattern values correspond to the values of all input and feedback signals immediately after the last terminating transition of the burst arrives at the circuit. The *pattern delay* of a burst is defined as the maximum pattern arrival time among those corresponding to output/state variables that change in response to the burst.

The cycle time of a two-phase transition t equals the pattern delay of its input burst plus the delay required for the circuit to settle which includes the feedback delay and the settling time delay. The cycle time of a three-phase transition consists of the pattern delay of the input burst, the pattern delay of the state variable burst, the feedback delay for the state variables and associated settling-time, and the feedback delay for the outputs and associated settling-time.

It has been our experience that most often no feedback delays are necessary and the settling time required is shorter than the fastest environmental response time. For the purpose of technology mapping it seems less critical to minimize these factors and thus we model them as constants *fb-delay* and *st-delay*. The cycle time, $cyc(t)$, of a two-phase state transition t is then

$$cyc(t) = delay(o\text{-}burst(t)) + fb\text{-}delay + st\text{-}delay \quad (1)$$

and the cycle time of a three-phase state transition t is

$$cyc(t) = delay(s\text{-}burst(t)) + delay(o\text{-}burst(t)) + 2(fb\text{-}delay + st\text{-}delay) \quad (2)$$

where $delay(o\text{-}burst(t))$ and $delay(s\text{-}burst(t))$ are the pattern delays of the input and state variable bursts in state transition t, respectively.

Using these definitions, we can quantify the average-case cycle-time of a burst-mode circuit. 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 delay of a trace, denoted $delay(T)$, is the sum of the cycle-times of the trace's state transitions. The average cycle-time, $\overline{\delta}$, is the limit of the trace delay divided by the length of the trace as the length of the trace goes to infinity:

$$\overline{\delta} = \lim_{k \to \infty} \frac{\sum_{T \in T^k} delay(T) \cdot Pr_{XBM}(T)}{k}. \quad (3)$$

3 Obtaining the pattern structure set for a burst-mode circuit

The inputs to our technology mapping algorithm are a set of pattern structures, an area budget, and a NAND-decomposed combinational network. Each pattern structure describes the information concerning a single input pattern of interest and consists of five parts: an input pattern, the values of the primary inputs that will be applied to the circuit; a set of initial values, the values of the circuit nodes before the input pattern is applied; input timing information, the arrival time of the changing primary inputs; any output timing constraints, minimum and maximum delays allowed on all changing output signals; and a weight, a measure of the relative frequency of this input pattern. The technology mapper attempts to map the combinational circuit such that all timing constraints are met and the weighted sum of the pattern delays is minimal.

This section describes the set of pattern structures for a burst-mode circuit that can guide the technology mapper to minimize its average cycle time. For state transitions involving two-phases (input burst followed by output burst) we create one pattern structure describing details of the input burst. For three-phase state transitions, we create two pattern structures: one to model the input burst and the other to model the state variable burst. The derivation of the contents of each pattern structure is described in detail below.

3.1 Input patterns

The input pattern of each compulsory primary input, primary output, and state variable, are uniquely determined by the specification because the XBM diagram satisfies the *unique entry point* property [25]. The values are determined during synthesis and can be found by simulating the XBM diagram that has been annotated with the state variable assignment.

The input pattern values for directed don't cares, however, are not predetermined by the specification because they can change in one of possibly many consecutive bursts. We assume that the user determines the state transition(s) in which the DDC can change and weights the associated pattern(s) accordingly.

When not sampled, conditionals can change arbitrarily. When sampled, they must stabilize according to set up and hold constraints. Because of hazard-freedom, however, the primary output cannot be sensitized to the conditional signal unless the conditional signal is stable. Otherwise, toggling of the conditional signal can cause a hazard. Thus, for a state transition in which a conditional signal is not sampled, its input value can be arbitrarily set to 0. For state transitions in which the conditional signal is sampled, however, the input value is set to its expected sampled value. The exact time at which it changes is modeled by its arrival time, discussed next.

3.2 Arrival times

In aggressive asynchronous designs, timing analysis can sometimes identify the order in which transitions within an input burst always (or usually) arrive.

While this information is not currently used during synthesis, it can improve the technology mapping solution. In particular, signals that often arrive late can be pushed closer to the output to improve average-case latency. If available, this information is stored for each state variable burst and output burst in their associated pattern structure.

In addition, arrival times are used to distinguish the signals that change from the signals that do not change. For all signals that do not change in an input burst or a state variable burst, we set their arrival time to $-\infty$. All other signals must have arrival times set greater. The default value is 0.

For conditional signals that are sampled, two options are possible. For the circuit to be hazard-free, the signal must be sampled after it becomes stable. Its arrival time thus should not affect the delay of the primary output and it is sufficient to set it to $-\infty$. It is also possible, however, to use realistic arrival times to ensure that the setup and hold constraints at various nodes in the network are met in the technology mapped solution. Due to lack of space, this paper does not discuss this option further.

3.3 Initial values

In the single step transition delay model it is assumed that the circuit has settled and retains the values from the previous input pattern before the application of the current input pattern. To find the previous input pattern from the current input pattern and its arrival times, we simply invert the logical values of all signals that are changing (*i.e.*, have arrival times greater than $-\infty$). Using this input pattern, we simulate the combinational network until all signals are settled. The settled values are stored as the initial values in the pattern structure.

3.4 Detailed timing constraints

Although we target optimal average cycle-time, it is important to recognize that this may not yield optimum average-case performance of the system in which the circuits is embedded. This is because not all output signal transitions are in the critical path of the system. Consider an output burst $u+v+$, where $u+$ initiates a slow environmental action and $v+$ initiates a fast environmental action. Also assume that both environmental actions must complete before the environment can generate the next input burst. Thus, the pattern arrival time of $v+$ for this output burst is not critical and v should, instead, be optimized for other patterns. We can achieve this by specifying a lower bound on non-critical pattern arrival times, which results in unnecessarily fast pattern arrival times to be excluded, yielding possibly better overall system performance.

We also note that for aggressive high-performance designs lower bounds can be used to guarantee that communicating burst-mode machines do not violate each other's fundamental mode constraints. Moreover, we allow the user to specify upper bounds on all patterns. While this does not affect the optimizations for system performance, it is useful in ensuring that some output signals are fast regardless of the effect on overall system performance. This timing may be necessary for correctness of some circuits.

It should be noted that it may be difficult to determine appropriate bounds of various transitions if they depend on as of yet unbuilt components. In addition, it may not be possible to build a circuit to meet a given set of requirements. As a result, design (including technology mapping) is must be an iterative process of estimating appropriate bounds and subsequently modifying them as necessary.

3.5 Statistical analysis

This section describes the Markov chain analysis used to derive the proportion of time the circuit is in each state transition. This information is used to weight the corresponding input and state bursts. Our analysis is similar to that used for power estimation of burst mode circuits [2] as well as for finding transition probabilities in finite state machines [20].

First, we model the execution of the circuit with a stochastic process $\{B_n, n = 0, 1, 2, \ldots\}$. We say 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. For example, if from the initial state (state 0) of the circuit, the machine executes state transitions $0 \rightarrow 1$ followed by state transitions $1 \rightarrow 3$ and $3 \rightarrow 5$, then $B_1 = 0 \rightarrow 1$, $B_2 = 1 \rightarrow 3$ and $B_3 = 3 \rightarrow 5$.

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

$$\Pi_t = \lim_{k \rightarrow \infty} \frac{\sum_{T \in \mathcal{T}^k} (\# \text{ of } t \text{ in } T) \cdot Pr_{XBM}(T)}{k} \quad (4)$$

Then, by taking into consideration the delays of the various transitions, the average cycle time can be calculated as follows:

$$\overline{\delta} = \sum_{t \in \delta} \Pi_t \cdot cyc(t) \quad (5)$$

Markov chain theory can be used to obtain the long-term proportion of state transitions. A Markov chain is a stochastic process X_n in which the conditional

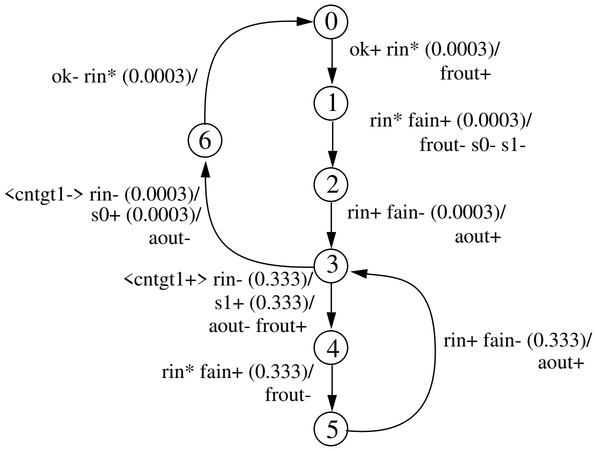

Figure 2: Specification after state encoding annotated with results of our statistical analysis.

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 diagram is strongly-connected this condition holds. Fortunately, these assumptions are usually satisfied in practice. In addition, for XBM diagrams that are not strongly-connected, simple extensions to this analysis apply.

For an irreducible Markov chain it can be shown that the Π_t are the unique non-negative solutions to the following equations [14]:

$$\Pi_t = \sum_{t' \,:\, sink(t')=source(t)} \Pi_{t'} \cdot Pr_{XBM}(t) \quad (6)$$

$$\sum_{t \in \delta} \Pi_t = 1 \quad (7)$$

Consider the *scsi-init-send* example described earlier. Given the conditional probabilities discussed in the introduction, these equations show that Π_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.

For a two-phase state transition t, Π_t is used to weight the input pattern corresponding to the input burst. For a three-phase state transition t, Π_t is used to weight the input patterns corresponding to the input burst and the state variable burst. Figure 2 illustrates the specification for *scsi-init-send* annotated with state variable bursts and weights.

4 Technology mapping for average-case performance

This section presents the timing model we adopt for delay analysis, and the two phases of our technology mapping algorithm, a simple heuristic decomposition phase, and a more complex covering phase.

The objective of the technology mapping is to optimize for average-case performance. Specifically, the technology mapping algorithm minimizes *average pattern delay*,

$$\sum_{i \in I} w_i \cdot delay(i), \quad (8)$$

subject to given timing and area constraints, where I is a set of input pattern structures, w_i is the weight of a pattern $i \in I$, and $delay(i)$ is the delay of pattern $i \in I$.

For the pattern structures of burst-mode circuits (as defined in Section 3), the average pattern delay equals the average cycle time (Equation 5) of the circuit minus a constant. Because the constant does not affect the minimization, the technology mapping algorithm can thus minimize the average cycle time of a burst-mode circuit subject to both timing and area constraints.

4.1 Single step transition model of delay

We calculate the pattern arrival time at a node in the circuit using the single step transition model. This model assumes that before an input pattern is applied, the circuit is stable, *i.e.*, all nodes are either at a 0 or 1 logical value. The model assumes that the nodes retain their logic values set by the previous input pattern. This model of delay is more accurate than the static timing model, but typically requires higher runtimes.

For general circuits, delay analysis using the single step transition model is made complicated by the presence of hazards in the network. For burst-mode circuits, however, the delay analysis is simplified by the fact that the primary outputs are guaranteed to be hazard-free under the unbounded wire and delay model. Thus, no hazard generated at an internal node of a burst-mode circuit can propagate to a primary output. Hence, during the calculation of the arrival time at a primary output for pattern i, any internal node that is hazardous can be ignored. In addition, Kung showed how to ignore nodes that do not change in response to the pattern. Kung describes an efficient implementation of these techniques using 8-valued extended logic simulation in [6]. We abstract the application of this technique in this paper using the function $sens(i)$. This function returns the subset of nodes which can affect the arrival time of any primary output for pattern i, referred to as *sensitized* nodes. In our algorithm, we maintain the pattern arrival times of only sensitized nodes, thus reducing run-time and memory requirements.

To model the delay of library gates, we use the pin-dependent delay model used in MIS. For the k^{th} pin of a gate g, the library specifies both an intrinsic pin-to-pin delay $\tau_{k,g}$ and a pin-specific output drive-resistance $R_{k,g}$. Given a capacitive load C_n at the output of g, the pin-to-pin delays are defined as follows:

$$\textit{p2p-delay}(k, C_n) = \tau_{k,g} + R_{k,g} \cdot C_n. \qquad (9)$$

In our implementation, we use a slightly modified equation which distinguishes between rising and falling pin-to-pin delays. To simplify the exposition of this paper, however, we will refer to the above generic delay equation.

The pattern arrival time at a node bound to a gate is a function of the delays through the input nodes of the gate that are sensitized during that pattern. For example, for the rising (falling) transition of a NAND gate, the pattern arrival time is the *min* (*max*) of the arrival times through all sensitized input nodes, each of which equals the pattern arrival time of the sensitized input node plus the corresponding pin-to-pin delay of the NAND gate. Consider the case in which all pin-to-pin delays are constant. Then, the rising (falling) transition at the output of the NAND gate is caused by the first (last) falling (rising) transition at any of its inputs. For more complicated AOI gates, computing the pattern arrival time may be more complex than computing a simple minimum or maximum. In general, it can involve taking the minimum of maximums or the maximum of minimums.

Notice that unlike static timing analysis that is commonly used in synchronous technology mappers, our delay analysis inherently avoids the false path problem. This is made possible by the derivation and analysis of the patterns of interest that are applied to the circuit. This would be computationally and memory expensive if the number of patterns of interest is large, as is the case for most combinational circuits. However, for burst-mode controllers, the number of patterns of interest is relatively small, as described in Section 3.

4.2 Decomposition for average-case delay

The goal of technology-independent logic optimization and technology decomposition is to obtain a good starting point for the subsequent technology covering step. Both current logic optimization tools and known decomposition techniques minimize worst-case delays. This section describes a heuristic for minimizing average-case delay.

Technology decomposition begins with a netlist of optimized equations and decomposes them into a netlist of 2-input NANDs and inverters. The key issue is that there are many such possible transformations and the choice of transformation affects the quality of the subsequent covering step. This is because the covering paradigm does not allow a node to be partially covered by two gates. In technology mapping for synchronous circuits, one common approach is to decompose the functions into balanced trees of 2-input NAND gates and inverters which optimizes worst-case delay.

One representation of the transformed network is a tree in which each node has 1 or 2 fanins. The nodes with 1 fanin represent inverters and the nodes with two fanins represent NAND-gates. As an example, the three (not counting symmetric representations) possible representation of a 3-input NAND gate are shown in Figure 3.

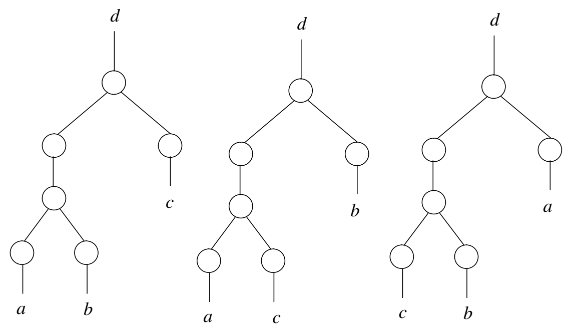

Figure 3: The three decompositions of a 3-input NAND gate.

Our heuristic is motivated by a timing optimization proposed by Rudell. After generating the balanced tree, his heuristic repeatedly restructures the representation of 3-input NAND gates to minimize worst-case delay. In particular, the best representation is chosen is the one that pushes late arriving inputs closer to the output [15]. By repeatedly applying this transformation, a local minimum of worst-case delay can be reached [15].

We modified Rudell's algorithm by guiding the choice of transformation of the subnetwork to minimize average pattern delay. To do this, we repeatedly consider the three representation of all 3-input NAND gates in the network. For each representation we use Kung's algorithm and the unit delay-model to recalculate the circuit's average pattern delay. We then choose the representation that minimizes average pattern delay. If two representations yield the same average pattern delay, we break the tie by selecting the one that minimizes the weighted sum of pattern arrival times at that output (weighted by the pattern weight), referred to as *the average pattern arrival time.* We repeat the application of the transformation until it reaches a locally optimum solution.

Figure 4 illustrates this heuristic with the NAND-decomposed tree for signal *aout* in the *scsi-init-send* example. As can be derived from the annotated specification shown in Figure 2, this output has 4 associated pattern structures. One corresponds to the state variable burst s_0+ between states 3 and 6 and has the weight of 0.0003. Another corresponds to state variable burst s_1+ between states 3 and 4 and has the weight of 0.333. Considering only these two patterns, the average pattern arrival time of this output signal can be significantly improved if we push s_1 closer to the output in the highlighted 3-input NAND gate. It is likely that this will improve the average cycle time of the circuit.

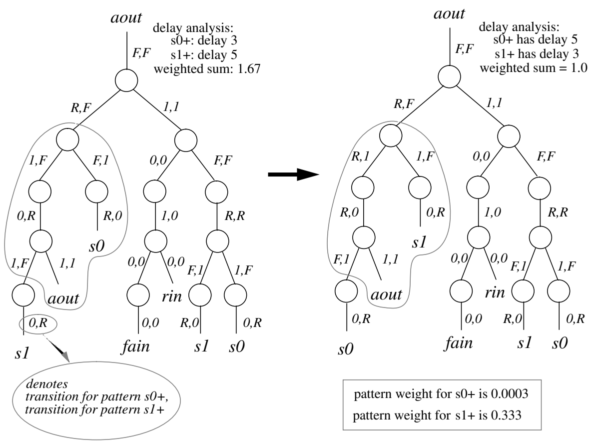

Figure 4: One step in our decomposition heuristic.

4.3 Covering for average-case delay

After the network is decomposed, for each node in the network, all gates that match that node is derived and a covering of gates that implements the network is determined. A brute-force algorithm must find every possible feasible mapping for the circuit, calculate its cost, and pick the lowest cost that meets the given area and timing constraints. To reduce the number of feasible mappings considered, our algorithm uses dynamic programming.

The approach is similar to Chaudhary and Pedram's approach for area and delay tradeoff analysis for synchronous circuits [3]. They find, analyze, and choose from many possible coverings representing many area-delay tradeoffs. They capture these possible tradeoffs in *points* on area-delay curves that are maintained at each node in the network.

Rather than maintaining a single worst-case delay in each point, we maintain the pattern arrival times of each pattern for which that node is sensitized. Thus each point at a node n is of the form $\langle d_1, \ldots, d_{|I_n|}, \mathit{area}\rangle$. (Recall that I_n is the set of patterns for which n is sensitized.) An element, d_i, is the pattern arrival time for pattern $i \in I_n$. The element *area* is the cumulative area for the cover of this sub-cone of logic. Notice that instead of forming an area-delay curve, our set of points represent a multi-dimensional *area-delay surface.* While this moderately increases the computational costs, it facilitates the application of Kung's exact pattern-based timing analysis described earlier.

Our algorithm contains two traversals of the circuit. First, a postorder traversal (from leaves to the roots) is used to determine the set of surface points at each node. Like Chaudhary and Pedram, we do not store one point for every feasible solution. Instead, we store only the *non-inferior* or *pareto* points which represent all possibly-optimal coverings for a sub-tree rooted at this node under the constant output load assumption.

After the first traversal of the circuit, all points that violate the user-specified timing constraints (described in Section 3) are removed. Then, one point from the surfaces of each output signal is chosen. This combination of points identifies the particular technology mapping area-delay tradeoff for the entire circuit. The goal is to choose the combination of points that meets the user-specified area constraint and yields the minimum average pattern delay. We present both heuristic and exact algorithms to achieve this with varying time complexity. Once the combination of points is chosen, a second pre-order traversal (from roots to leaves) is performed to determine the specific gate selections that map the circuit. We now go into more details.

4.3.1 Terminology

We adopt terminology similar to that used by Chaudhary and Pedram [3]. A match m specifies a matching gate $m.g$ whose pattern graph is isomorphic to a subtree rooted at node n of a NAND-decomposed circuit. The nodes which are covered by gate $m.g$ are denoted $\mathit{merged}(n,m)$. The nodes which are not in $\mathit{merged}(n,m)$ but have fanins in $\mathit{merged}(n,m)$ are denoted $\mathit{inputs}(n,m)$. The match also defines a binding $m.\mathit{bind}$ of nodes in $\mathit{inputs}(n,m)$ to distinct input pins of $m.g$. In principle, different matches corresponding two different pin bindings are possible and this is beneficial if the gate has different pin-to-pin delays (however, this is usually not done in practice).

At each node n, we store only *non-inferior* points, where a point $p = \langle d_1, \ldots, d_{|I_n|}, area\rangle$ is non-inferior if there does not exist some other point $p' = \langle d'_1, \ldots, d'_{|I_n|}, area'\rangle$ such that $d_i' \leq d_i$ for all i and $area' \leq area$. Intuitively, inferior points correspond to coverings of the sub-cone rooted at n that can never lead to an optimal solution; there exists a different covering which has equal or smaller delays for all input patterns and equal or smaller area.

4.3.2 Postorder traversal

The first stage of the covering algorithm is a post order (from inputs to outputs) traversal of the NAND-decomposed network to identify the area-delay covering tradeoffs possible at each node of the network. The network is assumed to be a forest of trees, one tree per output signal, and the trees are processed individually.

For each tree, the traversal begins at its primary inputs where the area-delay points are first initialized. For each primary input, the delay vector is set to the arrival times specified in the pattern structures as described in Section 3. The area is set to 0. Once the inputs are initialized, the internal nodes are traversed in post order.

For each internal node n, the set of matches m at that node are found using Siegel and De Micheli's algorithm [17]. For each match, many area-delay points are possible depending on how the cones of logic rooted at $inputs(n, m)$ are mapped. Because the nodes are visited in post order, the possibly optimal ways to implement the cones of logic rooted at $inputs(n, m)$ have already been analyzed and stored in different points on their area-delay surfaces. For each combination of input surface points, an area-delay point is computed for node n as follows. The pattern arrival time d_i for each pattern $i \in I_n$ is computed using the function *comp_p_time()* (described below). The area of that point is simply the area of the matching gate plus the area of the input points.

If this point is inferior it is not stored on the surface at n. On the hand, if it is non-inferior it is added to the surface. The introduction of this new point, may make existing points inferior which are then removed from the surface. After exhausting all matches at a particular node, the next node in the post-order traversal is analyzed. The traversal ends by generating the surface at the root of the tree.

Let us consider a particular matching gate $m.g$ at a node n. If $m.g$ has l inputs and at each input k has N_k points in its surface, the number of combinations of input points is

$$N_1 \cdots N_l. \qquad (10)$$

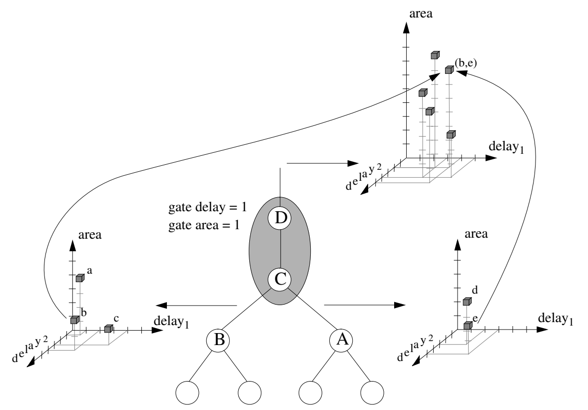

Figure 5: Computing the surface points for a match.

Since each input combination can yield one output point, Equation (10) is also the maximum number of distinct output points. This is a conservative estimate, however, since may points may be inferior and thus will be dropped.

Consider the example depicted in Figure 5 where a simple 2-input AND gate covers nodes D and C. There are two input patterns, one in which the AND gate rises and one in which the AND gate falls. Input B has three points and input A has 2 points. As the upper bound suggests, there are 6 possible combinations of inputs, each corresponding to a different covering of the sub-trees rooted at A and B. Each combination determines one possible output point. For example, consider the combination of point b and point e. The delay of the first pattern is the maximum of the delays of the first pattern in points b and e plus 1 for the delay of the AND gate. This yields a delay of 4. The delay of the second pattern is the minimum of the delays of the second pattern in points b and e plus one for the delay of the AND gate, yielding a delay of 5. This example also demonstrates that Equation 10 is only an upper bound. The reason for that is the combination (c, e) corresponding to a point which is inferior.

Let us now consider the details of the function *comp_p_time()*. The inputs to the function include a match m, a binding of input signals to surface points *in_pts*, a pattern i, and a node n. The function recursively traverses the portion of the decomposed tree that is covered by the matching gate $m.g$ from n until reaching the nodes in $inputs(n, m)$. When the function reaches an input node, it returns one of two values. If the input node is sensitized during this pattern, it returns the pattern arrival time for the given input point. If the input node, however, is not sensitized during this pattern, the point does not have a corresponding pattern arrival time because the node

Algorithm 4.1 (Compute area-delay surface)

compute_area-delay_surface(n) {
  foreach *candidate match* m *at node* n
    foreach *combination of inputs points pt.in_pts*
      foreach $i \in I$ *such that* $n \in sens(i)$
        pt.d[i] = *comp_p_delay(pt.in_pts,i,m,n,n)*
      pt.area = $\sum_{k=0}^{l}$ *pt.in_pts[k].area* + *m.area*
      insert pt into n.surf
      remove_inferior_pts(pt,n.surf)
}

Figure 6: Computing the area-delay surface at a node.

Algorithm 4.2 (Compute pattern arrival time)

comp_p_time(in_pts,i,m,n,curr-n) {
  if *curr-n* $\in$ *inputs*(n, m)
    if *curr-n* $\in$ *sens*(i)
      return *m.g.p2p-delay(m.bind[curr-n],load(n))*
          + *in_pts[n].d[i]* + *shift_delay(in_pts[n],m)*
    else
      return *NON-CONTROLLING*
  else if *pattern is rising at curr-n*
    return $\min_{n_i:\mathrm{FI(curr\text{-}n)}}$ *comp_p_time(in_pts,m,n,*n_i*)*
  else if *pattern is falling at curr-n*
    return $\max_{n_i:\mathrm{FI(curr\text{-}n)}}$ *comp_p_time(in_pts,m,n,*n_i*)*
  else
    return *NON-CONTROLLING*
}

Figure 7: Computing a pattern arrival time at a node.

is stable during the application of this pattern. In this case, the function returns a special arrival time called *NON-CONTROLLING*. In both cases the arrival time propagates up through the decomposed tree to n via the recursion. At each internal node, different input arrival times are analyzed. If the internal node is RISING (FALLING) during that pattern a minimum (maximum) of the input arrival times of its fanin is taken and propagated up the tree. When taking the minimum and maximum, arrival times equal to *NON-CONTROLLING* are ignored. Moreover, if the internal node is STABLE during that pattern, the value *NON-CONTROLLING* is propagated up the tree. At the root of the tree, the resulting delay is the corresponding pattern arrival time.

One additional note is that the pattern arrival times, as defined by Equation 9, depend on the load the gate is driving. When initially setting the delay curve at *inputs*(n, m), the exact load is not known, and a default load is used. When considering a match m, the exact load on *inputs*(n, m) is known. Therefore, when computing the pattern arrival time at n the exact load at *inputs*(n, m) is used to adjust the pattern arrival time at *inputs*(n, m). This is captured in the function *shift_delay()*.

The overall function to compute pattern arrival times is presented in Figure 4.2. Notice that the function seamlessly handles both simple gates, such as NANDs and INVs, and complex AOI blocks. The specific delay function of each type of gate need not be stored; it is essentially extracted from the tree structure that the match covers.

Let us now consider the details of removing inferior points from the surface. After each time Algorithm 4.1 adds a new point to the surface, it removes all inferior points from the surface. Consider, once again, the example presented in Figure 5. In this example, the combination corresponding to points c and d, (c, d), which corresponds to point $\langle 5, 4, 5\rangle$, is made inferior by the output point $(c, e) = \langle 5, 4, 3\rangle$. The rationale is that the point (c, e) corresponds to a binding which is always better than point (c, d) regardless of which input pattern has higher weight. Thus, point (c, d) is not included in the resulting area-delay surface. In our implementation, the surface points are maintained in a simple linked list that is sorted by area. After adding (c, e), the function searches all points with less area to determine if (c, e) is inferior. If (c, e) is not inferior, it then searches all points with larger area than (c, e) to determine if (c, e) makes any of the other points inferior. Thus, if (c, d) is added before (c, e), the algorithm compares (c, e) and (c, d) in the second phase of the search, determines that (c, d) is inferior, and removes it from the list. On the other hand, if (c, d) is added after (c, e), the comparison and removal of (c, d) will occur in the first phase of the search immediately after adding (c, d). More efficient algorithms which use more complicated binary tree data structures are also possible.

4.3.3 Point selection for the primary outputs

After the post order traversal finds an area-delay surface for each output signal, we select one point on the surface of each output signal such that the combination of points minimizes average cycle time but still satisfies the area and timing constraints. Recall that the average cycle time is the average pattern delay plus a constant and that a pattern delay is the *maximum* pattern arrival time at any of the primary outputs that change during that pattern.

A brute force method to calculate the global optimum considers every combination of points on the output signals. It excludes combinations that violate the overall area budget or any timing constraint, calculates the average pattern delay for each remaining combination (using Equation 5), and selects the one which yields the lowest average pattern delay. The brute force method, however, is computationally expensive since the number of combinations is the product of the number of points for each output signal.

A fast heuristic to find a good, but potentially non-optimal, solution is to individually minimize each output signal's average pattern arrival time. This has much lower complexity than minimizing the global average pattern delay since combinations of surface points in different output area-delay surfaces need not be considered.

It may be useful to explain why this heuristic may not find the optimum average pattern delay; the principle reason is that an output signal may only be critical for relatively infrequent patterns. To see this, consider a circuit with two patterns, Pattern 1 with weight 0.9 and Pattern 2 with weight 0.1. In Pattern 1, two outputs, A and B, transition. In Pattern 2, two outputs, B and C, transition. The average pattern delay is 0.9*max(A.d[1],B.d[1]) + 0.1*max(B.d[2],C.d[2]). The heuristic that we suggest is to optimize A, B, and C separately. As an example, to optimize B the average pattern arrival time is 0.9*B.d[1] + 0.1*B.d[2]. When optimizing the average pattern arrival time, paths that transition in Pattern 1 will be prioritized over paths that transition only in Pattern 2. If output A requires complex logic such that its Pattern 1 arrival time is always much large than that of B, then any optimization of B for Pattern 1 will not improve the pattern delay. In fact, if B was optimized for Pattern 2 instead of Pattern 1, then Pattern 2 delay might be reduced which improves the average pattern delay. Thus, optimizing average pattern arrival time may not yield the optimal solution, but the difference from the optimal solution is probably small since only the pattern delay of relatively infrequent patterns remain non-optimal.

Alternatively, if the user divides the area budget among the different output signals, an alternative to the brute force algorithm can be applied that still is guaranteed to find the optimum average pattern delay. The gain in computational efficiency is based on the fact that that the minimum of a function $f = f_1 + f_2$, where f_1 and f_2 depend on disjoint sets of variables, equals the sum of the minimums of f_1 and f_2. This leads to more efficient solutions since combinations of f_1 variables and f_2 variables need not be analyzed.

To take advantage of this observation we use a recursive algorithm in which at each level of the recursion we analyze one an output signal. For each point on the surface of this output signal, a reduced objective function is created by substituting the actual pattern delays of the point into the objective function. The reduced objective function is then recursively analyzed and the point which leads to the smallest objective function is chosen. The final calculation of the objective function is computed only at the leaves of the tree of recursive calls, where a point for each output signal has been selected. The key step in the algorithm is the selection of the output signal to analyze in each level of the recursion. The goal is to choose an output signal that will lead to a reduced objective function that can be decomposed into sub-objective functions that can be minimized independently (as described in the above paragraph). We choose the output signal which is *coupled* with the most other output signals among those for which a point has not been selected. Two signals are considered coupled if any burst contains transitions of both of them. The details of the algorithm are depicted in Figure 8.

Lets consider the operation of the algorithm on the example *scsi-init-send*. We assume that all points which either violate the decomposed area constraint or the timing constraints have already been removed from all output signal surfaces. The first output signal analyzed is *frout*. This is because it is coupled with the most (three) other signals. Namely, *frout* is coupled with *aout* because they both change in state transition $3 \rightarrow 4$, and with s_0 and s_1 because they all change in state transition $1 \rightarrow 2$. For each area-delay point on *frout*'s surface, a reduced objective function is created by substituting the actual pattern delays of *frout* into the objective function. The reduced function is then minimized recursively. In the recursive call, the reduced f is decomposed into two functions, one that depends on s_0 and s_1 and one that only depends on *aout*. This can be done because no pattern causes transitions of both *aout* and either of s_0 or s_1. At the leaves of the tree of recursive calls, one point for each output signal has been selected and the reduced objective function is trivially computed. For this example, the number of combinations of points searched in $O(N^3_{max})$ instead of $O(N^4_{max})$ where N_{max} is the maximum number of points in any output signal's surface.

4.3.4 Preorder traversal

After the point selection for a primary output, a preorder traversal (from root to leaves) is performed to identify the specific covering that is implemented.

Algorithm 4.3 (Point selection)

Let S hold the assignment of output signals to surface points.

```
select_points_for_po's(f) {
  if support of f is empty
    return compute(f)
  else if f can be decomposed
    curr-f-value = 0
    foreach decomposed function f_i
      curr-f-value += select_points_for_po's(f_i)
    return curr-f-value
  else select an output signal u in the support of f
    initialize best-f-value
    foreach for each surface point p of u
      let f_r be the reduced objective function by
        substituting in pattern delays of p
      let curr-f-value = select_points_for_po's(f_r)
      if curr-f-value < best-f-value
        update S to select p for u
        best-f-value = curr-f-value
    return best-f-value
}
```

Figure 8: Efficient algorithm to select points for primary outputs that yield the minimum average pattern delay.

The match m at the root that corresponds to this point and $inputs(root, m)$ are identified. For each input $n_k \in inputs(root, m)$, new timing constraints, $t'_k.i$, for each pattern $i \in I_n$ are calculated. This calculation is dependent on the gate and transition type. Consider the falling transition of NAND gate in which the delay function is the maximum of the rise times of the inputs. Then, all sensitized inputs are considered critical and the upper bound of all sensitized inputs is set to the node delay minus the corresponding pin-to-pin delay. For the rising transition of a NAND gate, the delay is the minimum of the fall times of the inputs. Then, the earliest input (a distinction we save in the point structure during the post order traversal) is the only input considered critical. The other inputs have no constraint and their upper bounds can be set to ∞. Once the required times have been updated, the traversal can continue at $inputs(root, m)$.

4.3.5 Unknown load value approximation

It is important to explain why point selection is performed in a separate traversal of the tree and not simply saved in the surface point structures generated during the post order traversal. The reason is that by recomputing the arrival times at $inputs(n, m)$ using the load-shift function, the area-delay surfaces at $inputs(n, m)$ are essentially shifted. As a result, different input points may be selected during the preorder than would otherwise be expected. This can cause a slight difference between the actual area and/or pattern arrival times than specified by the selected output point. Our experimental results suggest this difference is usually less than 10% for typical burst-mode circuits.

Moreover, different points on the surface may shift different amounts depending on the matching gate. In the worst case, this can imply that a previously inferior point which was dropped by the algorithm becomes the non-inferior. This may cause an optimal mapping to be rejected.

One possible fix is to use *binning* at each node to store multiple area-delay surfaces corresponding to different possible loads [15]. Using this technique, any given tolerance from the optimal can be met with an increase in computational complexity [3].

It is important to note, however, that other factors affecting the quality of the technology mapping result are only estimated. These include variances in wire delays and the effects of finite-slope transitions and varying thresholds. Thus, since our delay model is only an estimate, it is less important to achieve the exact optimum. Near-optimal solutions that can be derived efficiently are more attractive.

4.3.6 Complexity analysis

We first consider a match m at node n where the matching gate $m.g$ has l inputs and each input k has N_k surface points. The number of points corresponding 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 tree to the next grows at most polynomially with a degree that is bounded by the maximum number of gate inputs.

The number of total area-delay points stored at the surface of each output signal, however, still grows exponentially with the number of levels in the tree representation of the output signal. Assuming that the tree remains fairly balanced, the number of levels is logarithmic with the size of the circuit (number of gates). Thus, with respect to the size of the logic, the number of total points is polynomial. It is also possible to reduce the number of points by partitioning the area-delay solution space into bins and only maintaining at most one surface point for each bin [15].

The time complexity to derive each point is dominated by the derivation of the pattern arrival times. This time can be assumed to be a constant. The removal of inferior points at each surface is currently accomplished by searching through all points in a linked list after each point is added (even though more efficient search techniques using a multi-dimensional binary tree are possible). The time contribution is dominated by the removal of inferior points for the surfaces with the most points, which will be at the primary outputs. Assuming there are N_i surface points at the i^{th} output, the time complexity for removing its inferior points is thus $O(N_i^2)$.

The heuristic point selection for the outputs and preorder traversal are very efficient, requiring essentially constant time for each surface point. Assuming m outputs, this heuristic has overall time complexity of $O(m \cdot N_{max}^2)$, where N_{max} is the largest number of surface points at any primary output. The optimal point selection algorithm, on the other hand, has the significantly higher time complexity of $\prod_{i=0}^{m} N_i$. (Here, we are assuming that this complexity overshadows the time complexity due to removing inferior points.) As a result, for some large circuits (*e.g.*, those having greater than 10 outputs), the optimal point selection algorithm may not be computationally feasible. In these cases, we must resort to the heuristic point selection algorithm.

The high computational complexity of both algorithms, however, is somewhat mitigated by a variety of factors. First, asynchronous control circuits tend to be relatively small (less than 1K gates) because asynchronous systems are typically designed using a distributed control paradigm. Second, as mentioned earlier, the number of patterns of interest for each output tends to be small. Third, for each pattern only the gates on the sensitized paths (which tend to be a small subset of all paths) will have corresponding pattern arrival times. And, finally, many derived area-delay points are inferior and therefore dropped.

4.3.7 Extension to DAG mapping

Most general combinational circuits are DAGs that do not fall into the special class of trees—they have reconvergent fanout. Since the problem of mapping a DAG is NP-hard, most researchers resort to heuristics. DAGON uses a heuristic to decompose the DAG into a number of trees such that the inputs for each tree come from other tree outputs or primary inputs. This heuristic, however, does not allow mapping across tree boundaries.

Both the program MIS and Chaudhary and Pedram [3] adopt a heuristic that allows mapping across tree boundaries by modifying the postorder and preorder phases. The major problem they addressed occurs when a multiple fanout node is merged into a mapped gate. Since the multiple fanout node may fan out to multiple cones of logic, its logic may have to be duplicated in multiple cones. To avoid excessive logic duplication, the area contribution of the multiple fanout node can be divided by the cardinality of its fanouts, thus favoring a solution in which multiple fanout nodes are preserved. Another complication arises during the preorder gate assignment stage since it is now possible to arrive at a node which has already been mapped. If the current arrival time does not meet the timing constraints, the mapped gate must be replaced with a faster gate, possibly increasing circuit area [3].

Both heuristics can be applied to our algorithm with minor modifications. Although for general combinational circuits the latter tends to give smaller area [3], their relative performance on asynchronous circuits is yet to be determined.

5 Preliminary Results

We have implemented preliminary versions of both the decomposition heuristic and the covering algorithm in CAD tools written in C. The current implementation does not contain the extensions to DAG mapping or the implementation of detailed timing constraints and input arrival times. We conducted the experiments using the tools on the benchmark of circuits, presented in Table 1 and report the results in Tables 2 and 3. The experiments were performed on a SPARC 20 model 502 with 128 Megabytes of memory.

Circuits Tested					
Circuit	# PIs	# POs	# States	# Patts.	# Gates
merge	8	3	7	8	31
bufctrl	5	3	3	3	36
stetson-p3	7	3	8	11	40
q42	5	3	4	4	45
binary-cntr	11	7	32	32	72
dramc	13	7	12	14	135
scsi-init-send	10	4	7	10	142
pe-send-ifc	12	5	11	14	172
trcv	14	7	16	30	401
ircv	14	7	16	30	451
stetson-p1	28	17	26	31	932

Table 1: Description of circuits tested.

Our decomposition tool accepts unoptimized Verilog netlists files and extended burst-mode specifications satisfying the unique next-state assignment condition. The tool produces both a pattern file and the optimized decomposed network in BLIF format. We compared our optimally decomposed circuits to those decomposed in the form of a balanced tree. The results demonstrate that the optimized circuits showed up to a 30% improvement in average pattern delay under the unit-delay model. The run-times for all circuits tested were under 10 seconds.

Preliminary Results of Decomposition Heuristic			
	Ave. Pattern Delay (ns)		CPU
Circuit	Worst-case	Ave-case	(sec)
merge	4	3	0.1
bufctrl	6	5.33	0.1
stetson-p3	3.16	3.13	0.1
q42	6	6	0.1
binary-cntr	6	5	0.4
dramc	7.1	5.61	0.2
scsi-init-send	6.33	4.67	0.1
pe-send-ifc	9.29	7.03	0.4
trcv	6	5.33	1.9
ircv	6.15	4.15	2.2
stetson-p1	12.73	11.2	6.3

Table 2: Preliminary results of decomposition heuristic for minimizing average cycle-time.

We also implemented a preliminary version of our covering algorithm in an extension to POSE, which is an extended version of SIS that contains the code for Pedram and Chaudhary's technology mapping algorithm [3] along with many other optimizations targeting low-power design. We applied both our and their algorithms on both the balanced NAND-decomposed network and the optimized NAND-decomposed network. Moreover, we performed experiments using both the brute-force and heuristic point selection algorithms. Unfortunately, the implementation of the more efficient exact point selection algorithm is yet to be completed. For these experiments, we used the "lib2" gate library included in the SIS package. Due to lack of time, De Micheli and Siegel's algorithm was not used to determine which library gates are hazardous. Instead, all gates were assumed to be hazard-free.

The results demonstrate that the application of our covering algorithms can yield up to 5% to 20% improvement over the traditional worst-case covering. As expected, the circuits with widely varying state transition frequencies, *i.e., scsi-init-send, dramc, trcv* and *ircv*, show significantly higher improvements than the circuits with uniform state transition frequencies, *i.e., merge, bufctrl, stetson-p3, binary-cntr* and *q42*.

Interestingly, the different decomposition techniques did not have a significant impact on the quality of the final circuits. This has two implications. First, it suggests that finding the optimal decomposition may only have a minor impact on the technology mapping solution for average-case performance. Second, it implies that applying standard technology mapping algorithms to technology-independent optimized networks for average-case can eliminate any improvement obtained at the technology-independent level. Further investigation is necessary to verify this result on a larger benchmark that includes some larger burst-mode circuits.

Moreover, for those circuits for which both the exact and heuristic covering algorithms complete, the results show no significant differences. Although still inconclusive, this suggests that the heuristic algorithm is sufficient for average-case optimization. This is fortunate since the heuristic algorithm can be orders of magnitude faster than the exact algorithm.

Nevertheless, both algorithms have computational limits. For small controllers (less than 150 decomposed gates), both the heuristic and brute-force exact algorithms complete within a few minutes. For medium-size controllers (150-500 gates), only the heuristic algorithm completes. This is primarily because the number of output combinations that need to be considered in the exact algorithm becomes too large (greater than 1.0×10^7). For larger controllers (greater than 500 gates), both algorithms fail to complete. This is because the number of surface points at various node grows well beyond 1000 for which the generation of the area-delay surface becomes computationally infeasible. Our future work is to explore more computationally efficient heuristics.

6 Conclusions

This paper describes technology mapping techniques for optimizing average-case performance of burst-mode circuits. It presents a heuristic technology decomposition algorithm using the unit-delay model and a covering algorithm that provides optimal results under the constant load assumption. The results demonstrate that covering has a higher impact than decomposition and that an improvement of up to 5% to 20% is achievable.

We also believe that these technology mapping algorithms are applicable to other fundamental mode design styles, such as Nowick's locally clocked technique [11] and UCLOCK method [13].

Preliminary Results of Covering Technique				
Circuit	Ave. Pattern Delay (ns)			CPU-Times
	Worst-case	Ave-case		
		Optimal	Heuristic	(ns)
merge-WC	1.95	1.71	1.71	0.2/0.1/0.1
merge-AC	2.0	1.68	1.68	0.2/0.1/0.1
bufctrl-WC	2.84	2.80	2.80	0.3/0.1/0.1
bufctrl-AC	2.84	2.80	2.80	0.2/0.1/0.1
stetson-p3-WC	1.69	1.66	1.66	0.3/0.2/0.1
stetson-p3-AC	1.66	1.66	1.66	0.2/0.1/0.1
q42-WC	3.54	3.39	3.49	0.3/0.2/0.2
q42-AC	3.54	3.39	3.49	0.3/0.2/0.2
binary-cntr-WC	5.53	5.32	5.36	1.8/495/1.1
binary-cntr-AC	5.49	5.11	5.11	1.6/260/0.98
dramc-WC	2.97	2.69	2.70	1.2/6.9/3.3
dramc-AC	2.96	2.76	2.75	1.2/22.4/3.0
scsi-init-send-WC	4.63	3.72	3.72	1.6/0.3/0.3
scsi-init-send-AC	4.43	3.86	3.88	0.6/0.3/0.3
pe-send-ifc-WC	4.31	time-out	4.17	1.6/time-out/7.4
pe-send-ifc-AC	4.31	time-out	4.17	1.4/time-out/7.6
trcv-WC	7.88	time-out	6.93	4.2/time-out/205.4
trcv-AC	7.88	time-out	6.93	4.1/time-out/203.6
ircv-WC	8.58	time-out	7.33	4.9/time-out/134.9
ircv-AC	8.58	time-out	7.33	4.7/time-out/132.9
stetson-p1-WC	7.36	time-out	time-out	10.9/time-out/time-out
stetson-p1-AC	7.36	time-out	time-out	10.7/time-out/time-out

Table 3: Preliminary results of our covering algorithms for minimizing average cycle-time. The "-WC" extension refers to the worst-case (balanced) decomposition of the circuit and the "-AC" extension refers to the optimized (average-case) decomposition.

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 C.-K. Lee at USC for coding of a preliminary version of the burst-mode decomposition routines and J. Martin at UCI for drafting many of the figures in the paper. We would also like to recognize the anonymous reviewers of this paper for many valuable comments and suggestions.

References

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

[2] P. A. Beerel, K. Y. Yun, and S. Nowick. Estimation and bounding of energy consumption in burst-mode control circuits. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, 1995.

[3] K. Chaudhary and M. Pedram. Computing the area versus delay trade-off curves in technology mapping. *IEEE Transactions on Computer-Aided Design*, pages 1480–1489, December 1995.

[4] S. Furber. Computing without clocks: Micropipelining the ARM processor. In Graham

Birtwistle and Al Davis, editors, *Asynchronous Digital Circuit Design*, Workshops in Computing, pages 211–262. Springer-Verlag, 1995.

[5] K. Keutzer. DAGON: Technology binding and local optimization by DAG matching. In *24th Design Automation Conference*, pages 341–347. IEEE/ACM, 1987.

[6] D. Kung. Path sensitization in hazard-free circuits, 1995. In collection of papers of the *ACM International Workshop on Timing Issues in the Specification of and Synthesis of Digital Systems.*

[7] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli. Synthesis of hazard-free asynchronous circuits with bounded wire delays. *IEEE Transactions on Computer-Aided Design*, 14(1):61–86, January 1995.

[8] F. Mailhot. Algorithms for technology mapping based on binary decision diagrams and on Boolean operators. *IEEE Transactions on Computer-Aided Design*, 12(6):599–620, 1993.

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

[10] C. J. Myers, P. A. Beerel, and T. H.-Y. Meng. Technology-mapping of timed-circuits. In *2nd Working Conference on Asynchronous Design Methodologies*, May 1995.

[11] S. M. Nowick and D. Dill. Automatic synthesis of locally-clocked asynchronous state machines. In *IEEE ICCAD Digest of Technical Papers*, 1991.

[12] S. M. Nowick and D. L. Dill. Exact two-level minimization of hazard-free logic with multiple-input changes. *IEEE Transactions on Computer-Aided Design*, 14(8):986–997, August 1995.

[13] S.M. Nowick and B. Coates. UCLOCK: automated design of high-performance unclocked state machines. In *Proceedings of the 1994 IEEE International Conferen ce on Computer Design: VLSI in Computers and Processors*, October 1994.

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

[15] R. Rudell. *Logic Synthesis for VLSI Design.* PhD thesis, U. C. Berkeley, April 1989. Memorandum UCB/ERL M89/49.

[16] P. Siegel and G. DeMicheli. Decomposition methods for library binding of speed-independent asynchronous designs. In *IEEE/ACM 1994 ICCAD Digest of Technical Paper*, 1994.

[17] P. Siegel, G. De Micheli, and D. Dill. Automatic technology mapping for generalized fundamental-mode asynchronous designs. In *Proc. ACM/IEEE Design Automation Conference*, pages 61–67, June 1993.

[18] I. E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, 1989.

[19] H. J. Touati, C. W. Moon, R. K. Brayton, and A. Wang. Performance-oriented technology mapping. In W. J. Dalley, editor, *6th MIT Conference on Advanced VLSI Conference*, pages 79–97, 1995.

[20] C.-Y. Tsui, M. Pedram, and A. Despain. Exact and approximate methods for calculating signal and transition probabilities in FSMs. In *Proc. ACM/IEEE Design Automation Conference*, pages 18–24, 1994.

[21] 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*, 13(9):1110–1122, 1995.

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

[23] 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*, 11(2):22–32, Summer 1994.

[24] Victor I. Varshavsky, editor. *Self-Timed Control of Concurrent Processes: The Design of Aperiodic Logical Circuits in Computers and Discrete Systems.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1990.

[25] K. Y. Yun. *Synthesis of Asynchronous Controllers for Heterogeneous Systems.* PhD thesis, Stanford University, August 1994.

[26] K. Y. Yun, D. L. Dill, and S. M. Nowick. Synthesis of 3D asynchronous state machines. In *International Conference on Computer Design, ICCD-1992.* IEEE Computer Society Press, 1992.