Optimizing Non-Monotonic Interconnect using Functional Simulation and Logic Restructuring

Stephen M. Plaza
University of Michigan
EECS Department
Ann Arbor, MI 48109
splaza@umich.edu

Igor L. Markov
University of Michigan
EECS Department
Ann Arbor, MI 48109
imarkov@umich.edu

Valeria Bertacco
University of Michigan
EECS Department
Ann Arbor, MI 48109
valeria@umich.edu

National Taiwan University
EE Department
Taipei, Taiwan 106

ABSTRACT

The relatively poor scaling of interconnect in modern digital circuits necessitates a number of design optimizations, which must typically be iterated several times to meet the specified performance objectives. Such iterations are often due to the difficulty of early delay estimation, especially before placement. Therefore, effective logic restructuring to reduce interconnect delay has been a major challenge in physical synthesis, a phase during which more accurate delay estimates can be finally gathered. In this work, we develop a new approach to this problem that enhances modern high-performance logic synthesis techniques with flexibility and accuracy in the physical domain. This approach is based on (1) a novel criterion based on path monotonicity, that identifies those interconnects amenable to optimization through logic restructuring and (2) a synthesis algorithm relying on logic simulation and placement information to identify placed subcircuits that hold promise for interconnect reduction. Experiments indicate that our techniques find optimization opportunities and improve interconnect delay by 11.7% on average at less than 2% wirelength and area overhead.

Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]: Computer-Aided Design

General Terms Algorithms, Design, Performance

1. INTRODUCTION

As interconnect plays a more significant factor in overall circuit delay, the focus of design methodology is shifting from logic optimization to interconnect optimization. While this transition has been occurring for over a decade, meeting performance objectives is becoming more and more difficult. In recent years, a few successful methodologies achieved timing closure by combining netlist level minimization in logic synthesis with post-placement physical optimizations. This family of solutions is known as physical synthesis. Related strategies, including interconnect buffering [19], gate sizing [17], and relocation [1], successfully improved delay. In [8, 11, 28], post-placement resynthesis achieved delay improvement with limited placement perturbation, but these techniques are limited to simple signal substitution transformations. As the major portion of the critical delay is shifting into interconnect [32], poor design choices during synthesis cannot be easily corrected by limited scale post-placement optimizations. Therefore, more accurate delay models have been developed to guide logic synthesis. *Wire-load* models that estimate delay by considering the capacitive load of each net were effective until wire capacitance and resistance became predominant. Further knowledge of the impact of placement on wirelength was consequently needed by synthesis algorithms.

To meet the challenge of performance optimization at the 130nm technology node and beyond, the traditional design flow transformed from several discrete optimization phases (such as logic synthesis followed by place-and-route) into a more holistic strategy. In [14] wirelength estimation was incorporated in logic synthesis by constructing a highly placeable netlist with the goal of reducing wire detours. In addition, topographical information has been used to guide current synthesis tools [34]. Due to the importance and inherent difficulty of estimating the impact of placement and routing on interconnect, researchers suggested the idea of maintaining a companion placement estimate throughout the logic synthesis process [10, 15, 24]. However, interconnect-aware logic transformations are still limited by the accuracy of the estimates available. Furthermore, guiding logic synthesis by conservative delay estimates, as in [14], can lead to transformations that do not improve critical path delay but increase area and power consumption.

While performing aggressive logic restructuring using global information is desired to exploit better estimates later in the design flow, such accounts have eluded published literature. One particular complication is that the limited amount of flexibility found in combinational circuits must be combined with physical aspects of performance optimization. In this paper, we introduce a post-placement solution that enables aggressive optimization while minimizing changes to the physical netlist. We consider a wide range of changes to the circuit structure while also tracking their impact on physical parameters of the circuit.

Our contributions are as follows:

1. A novel metric for efficiently identifying *non-monotonic* paths in the circuit, that locates regions where restructuring provides the greatest gains. This metric generalizes the monotonicity metric in [4] and considers longer paths.
2. A generic and powerful technique for discovering logic transformations using functional simulation, which also facilitates fast re-evaluation of physical parameters. Our technique does

ISPD'08, April 13–16, 2008, Portland, Oregon, USA.

not require local equivalence between the optimized subcircuit and the original, but uses simulation and satisfiability to ensure that the circuit's functionality is unmodified.

3. A suite of powerful algorithms that efficiently exploit a circuit's don't-cares and avoid heavy-weight techniques traditionally used in logic synthesis, while allowing tighter integration with placement and a more realistic delay calculation.

In our methodology, we first identify long wires that lie on critical paths. We then generate different candidate topologies that improve circuit delay. The next step enables efficient search for logic transformations. Through logic simulation, we partially characterize the behavior of each node in the subcircuit with a *signature* [8, 7, 25]. We then use these signatures to efficiently determine whether a logic transformation generating the desired topology is possible. In the example of Figure 1, we show that by applying our technique, a subcircuit with a long critical path can be transformed to a functionally-equivalent subcircuit with smaller critical path delay. Unlike most techniques from logic synthesis, our circuit restructuring can work directly on mapped circuits with complex standard cells. Another novel feature is our extensive use of circuit flexibility due to signal masking by downstream logic, also known as observability don't-cares (ODC). Additionally, our approach uses controllability don't-cares (CDC), *i.e.*, circuit flexibility due to upstream logic. Compared to work in [30], our approach exploits global don't cares to enhance logic restructuring.

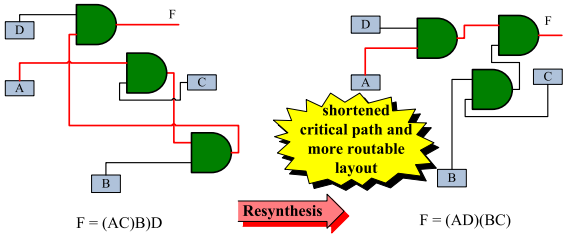

Figure 1: The resynthesis of a non-monotone path can produce much shorter critical paths and improve routability.

Our experiments indicate that large circuits often contain many long critical paths that can be effectively targeted with our restructuring. Improving these paths results in consistent delay improvements of 11.7% on average with minimal degradation to other performance parameters. Our techniques are fast and scale to large designs, whereas completely characterizing node functionality with BDDs would require a prohibitive memory footprint.

In Section 2, we cover background on the use of simulation to guide logic optimization and review state-of-the-art synthesis strategies. In Section 3, we introduce our interconnect optimization strategy. In Section 4, we propose a metric for finding circuit paths that require restructuring. Section 5 introduces a novel physically-aware synthesis approach that uses simulation. Empirical validation is presented in Section 6, and we summarize our work in Section 7.

2. BACKGROUND

This section describes how functional simulation can be used to characterize the behavior of internal nodes in the circuit and guide logic optimization. We then discuss a state-of-the-art approach for logic synthesis, currently limited to the logic domain that provides essential components for our physical synthesis algorithms.

2.1 Simulation and Satisfiability

A node F in a Boolean network can be viewed as a function of primary inputs. Such a node can be characterized by its *signature*, S_F, for K input vectors $X_1 \cdots X_K$.

DEFINITION 1. *$S_F = \{F(X_1), \ldots, F(X_K)\}$ where $F(X_i) = \{0, 1\}$ indicates the output of F for a given input vector.*

A carefully-designed testbench or constrained-random simulator can be used to generate vectors X_i and derive signatures for each node in a circuit. For a network with N nodes, the time complexity of generating signatures for the whole network is $O(NK)$. The functional non-equivalence of two nodes can be determined by the following: $S_F \neq S_G \Rightarrow F \neq G$.

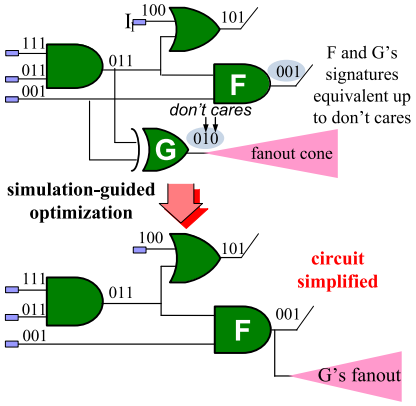

Figure 2: Optimization by merging equivalent nodes in the presence of don't-cares. 3-bit signatures are shown at the output of each gate.

Signatures can be efficiently created and manipulated by taking advantage of bit parallel-operations. Therefore, equal signatures can be used to efficiently identify potential node equivalences in a circuit by deriving a hash index for each signature [18]. Since $S_F = S_G$ does not imply that $F = G$, this potential equivalence must be verified, *e.g.*, using a SAT solver, as explained below.

The signature is a *partial* characterization of a node's functionality. Furthermore, the signature encodes all of the node's CDCs under the input vectors applied. The signature's partial characterization enables fast and aggressive optimizations without requiring a fully specified truth table. However, unlike traditional, correct-by-construction optimizations, these speculative transformations must be validated by a formal proof mechanism. Hence, the efficiency of [18, 21] depends on the underlying engines which formally verify the equivalence of nodes with identical signatures.

Recent advances in SAT solvers, *e.g.*, learning, non-chronological backtracking, and watched literals [22, 27] have made SAT a more scalable alternative to BDDs for equivalence checking. The equivalence of two nodes, F and G, in a network can be determined by constructing an XOR-based miter [5] between them and asserting the output to 1 as shown in the following formula:

$$(F = G) \Leftrightarrow (\forall i \; F(X_i) \oplus G(X_i) \neq 1) \qquad (1)$$

where $\bigcup_i X_i$ is the set of all possible input vectors.

In [18], input vectors are generated dynamically from counter-examples returned by SAT checks proving $F \neq G$. The dynamic input vectors improve the quality of the signatures by limiting situations where $S_F = S_G$ despite $F \neq G$.

2.2 Logic Optimizations with Signatures

Simulation is an effective means for quickly identifying candidates for optimization. In [25, 31], signatures were used to additionally encode ODCs to enable circuit simplification and optimization by merging equivalent nodes. Consider the example in Figure 2 which shows a circuit where logic simulation produces the signatures shown. Notice that through efficient don't-care computation using a fast linear-time simulation [25] of downstream nodes,

don't-care values can be determined for some of the signature's positions. In the example, these don't-cares suggest a potential circuit simplification by merging two nodes. The optimization will need to be verified by a formal proof engine.

Despite these advantages, signature-based optimizations are limited, and general synthesis algorithms have not been developed. A key contribution of this work is the application of signatures to enable logic restructuring while relying on available don't-care computation algorithms.

2.3 Logic Rewriting

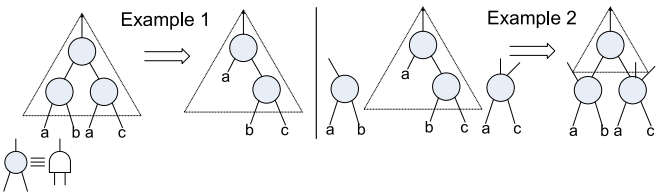

Figure 3: Two examples of AIG rewriting. With structural hashing, it is possible in the second example to reuse external nodes and minimize the subgraph.

Performing scalable logic optimization requires efficient netlist manipulation, typically involving only a small set of gate primitives. Given a set of Boolean expressions that describe a circuit, the goal of synthesis optimization is to minimize the number of literals in the expressions along with the number of logic levels. Several drawbacks of these techniques are discussed in [20], including limited scalability. To this end, the authors of [20] introduced an efficient synthesis strategy called *rewriting*. Logic rewriting is performed over a netlist representation called an And-Inverter Graph (AIG) [18], where each node represents an AND gate and complemented (dotted) edges represent inverters. In logic rewriting, the quality of different functionally-equivalent implementations for a small logic block in a circuit is assessed. In Figure 3, the left transformation leads to a reduction in area. By using a technique called *structural hashing* [18], nodes in other parts of the circuit can be reused. In the right example, there is a global reduction in area by reusing available gates.

The increasing significance of wire delay is addressed by providing more accurate delay models to logic synthesis, from using wire-load models to maintaining companion placements [10]. The delay model is used to modify the literal reduction objective so that transformations or rewrites that improve the delay according to the model are favored. However, delay estimation is becoming more inaccurate before detailed placement and routing as the actual interconnect routes become more important. This trend suggests that new synthesis algorithms should be applied after placement and routing because speculative optimizations can actually increase delay while negatively impacting other performance metrics like area.

3. OUR APPROACH

In this paper, we introduce a new synthesis approach that accounts for physical aspects of performance optimization. We illustrate our approach in Figure 4. Starting from a fully placed circuit, we identify critical paths using static timing analysis. We then apply a novel metric introduced in Section 4 that finds subcircuits for which restructuring could provide the greatest improvements. Next, we perform logic simulation using an even distribution of input vectors and generate signatures that encode don't-cares to obtain a partial characterization of the functional behavior of the circuit. Using this functional information encoded in signatures along with the physical constraints, we efficiently derive a topology that is logically equivalent to the original subcircuit but exhibits better performance. Finally, we legalize the altered placement and update the timing information in the circuit. As a result, we tailor our path-monotonicity metric to find portions of the critical path resulting in the greatest delay improvements. In addition, our techniques can target other objectives as well.

Using signatures for restructuring is advantageous because logic simulation provides a more scalable functional representation than BDDs. Furthermore, signatures can characterize internal nodes for netlists mapped to standard cells as well as for technology-independent netlists. In contrast, the logic rewriting strategy in [20] does not operate on technology-mapped circuits and does not take physical information into account. Our strategy also improves solution quality by considering more don't-cares while being directly guided by physical constraints.

4. IDENTIFYING NON-MONOTONE PATHS

To maximize the effectiveness of our post-placement optimizations, we target timing critical parts of the design that are amenable to restructuring. In this section, we introduce our fast Dynamic Programming (DP) algorithm for finding paths in logic that are *non-monotone*, or paths that are not optimally short. Unlike the work in [4], we consider paths of arbitrary lengths and scale to many more segments in practice. We propose two models for computing path monotonicity: (1) wirelength-based and (2) delay-based. Non-monotonic paths indicate regions where interconnect and/or delay may be reduced by post-placement optimization.

4.1 Path Monotonicity

First, static timing analysis is performed to enable our delay-based monotonicity calculation and identify critical and near-critical paths. We use a timing analyzer whose interconnect delay calculation is based on Steiner-tree topologies produced by FLUTE [12][1] and the D2M delay metric [2] that is known to be more accurate than Elmore delay. Before focusing on critical paths, we will describe a general approach that examines the monotonicity of every path. We define the **non-monotone factor (NMF)** for the path $\{x_1, ..., x_k\}$ with respect to a given cost metric (such as wirelength or delay) as follows:

$$NMF = \frac{1}{c_{ideal}(x_1, x_k)} \sum_{n=1}^{k-1} c(x_n, x_{n+1}) \quad (2)$$

where $c(a, b)$ defines the actual *cost* between a and b and c_{ideal} defines an optimal cost. When $NMF = 1$, the path is monotone under the cost metric. We explore two definitions for cost, one based on rectilinear distance and another on delay.

For the rectilinear case, $c(a, b)$ is the rectilinear distance between cell a and b while $c_{ideal}(a, b)$ is the optimal rectilinear distance assuming a monotonic path. For the delay-based definition, $c(a, b)$ is the $AT(b) - AT(a)$, where AT is arrival time. We define c_{ideal} as the delay of an optimally buffered path between a and b as described by [23] and given by the following formula:

$$c_{ideal}(a, b) = dist(a, b)(R_{buf}C + RC_{buf} + \sqrt{2R_{buf}C_{buf}RC}) \quad (3)$$

where R and C are the wire resistance and capacitance respectively and R_{buf} and C_{buf} are the intrinsic resistance and input capacitance of the buffers. $dist(a, b)$ is the rectilinear distance between a and b. Unlike the distance calculation where the ideal path length between a and b can be equal to the actual path length, the optimal buffered wire between a and b has delay $\leq AT(b) - AT(a)$. We only attempt to optimize paths with large NMFs.

[1]Timing-driven Steiner trees can be easily used instead [3].

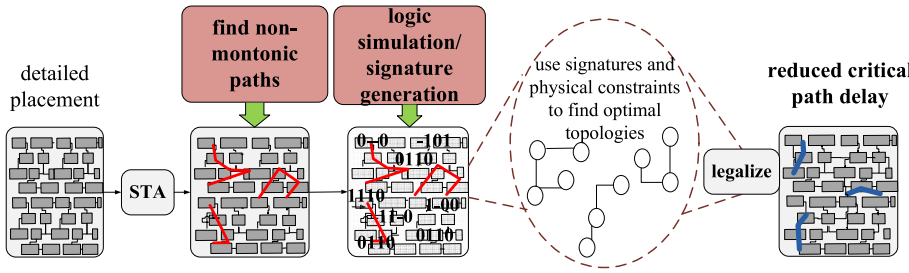

Figure 4: Our approach to optimizing interconnect. First, we identify non-monotonic critical path interconnect, and then we restructure these paths to improve delay. Such netlist transformations include gate cloning, but are substantially more general. They do not require that the subcircuits in question be equivalent. Instead, they use simulation and satisfiability to ensure that the entire circuit remains equivalent to the original.

4.2 Calculating Non-monotone Factors

We now present our algorithm for calculating the NMF of all k-hop paths in a circuit, for a given $k \geq 2$. Our experiments indicate that the greatest NMFs are often observed on relatively short paths, and optimizing such paths brings greatest benefits.

```
inputs
  Nodes: netlist
  Dist: length of paths considered
output
  NMF: NMF between each node
void gen_NMF(Nodes nodes, Dist K) {
    levelize(nodes);
    for_each node1 in nodes
      for_each node2 in range(node1+1, node1+K)
        c_ideal_array[node1,node2] = c_ideal(node1, node2);
    for_each node1 in nodes
      subtot[] = 0;
      for_each node2 in range(node1_succ, node1_succ+K)
        subtot[node1,node2] = max(subtot[node1,node2_pred]
            + c(node2_pred, node2));
        factor = subtot / c_ideal_array[node1,node2];
        NMF[node1,node2] = factor;
}
```

Figure 5: Generating non-monotone factors for a k-hop paths.

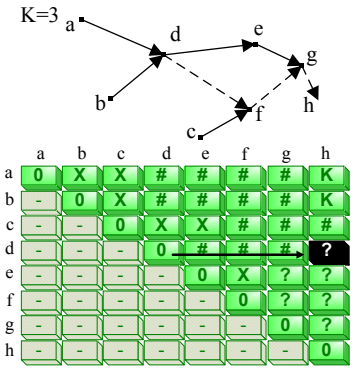

Figure 6: Calculating the non-monotone factor for path $\{d, h\}$. The matrix shows sub-computations that are performed while executing the algorithm in Figure 5.

The non-monotone factor can be efficiently computed for every path using a $O(nk)$-time algorithm for n nodes in the circuit, as shown in Figure 5. First, the circuit is levelized. Then, c_{ideal} is computed for node pairings with a connecting path of $\leq k$ hops, and the values are stored in `c_ideal_array`. All pairs are traversed again, and the `subtot` is generated by computing the maximum cost from *node1* to *node2* through a recurrence relation. The NMF is computed for the subpath, $\{node1,\ node2\}$, by dividing the total cost, `subtot`, by `c_ideal`$[node1, node2]$.

In Figure 6, we illustrate a sub-circuit being traversed using the `gen_NMF` function where $k = 3$ and the current $node1$ is d. The matrix indicates the NMFs already computed with #, and nodes lying on the same path with X. Because we traverse the graph in levelized order, a, b, c have already been examined. Notice, that nodes that are farther than k hops away are not examined (indicated by K in the matrix). For node d, the non-monotone factor is computed for path $d - h$ by determining all the incoming sub-paths to h first. In this example, $d - h$ has the highest NMF if rectilinear distance is the cost function.

5. PHYSICALLY-AWARE LOGIC RESTRUCTURING

We optimize the subcircuits that are identified by the path monotonicity metric as illustrated in Figure 7. After extracting a subcircuit, we clone the logic for the subcircuit's outputs that are not on the critical path. We then optimize only this non-monotone critical path, which can result in global area reduction because the duplicated logic required is often small. Our techniques use signatures to find topologies which are logically equivalent to the original implementation but also improve physical parameters. This transformation is then formally verified by performing SAT-based equivalence checking between the original and new netlists.

Previous work on improving path monotonicity used logic replication [16]. However, that technique is restricted to the topology of the extracted subcircuit and does not consider its functionality. Furthermore, as observed in [16], gate relocation sometimes cannot improve path monotonicity. In the following, we introduce the theoretical framework of using signatures to check the *logical feasibility* of a topology. We then introduce an algorithm for constructing subcircuits using signatures and physical constraints.

5.1 Determining Logical Feasibility with Signatures

Given an extracted subcircuit with x inputs and output F to resynthesize, we express a candidate restructuring as a directed graph T_F with x incoming edges, one outgoing edge F, and n internal vertices. We would like to determine whether there is a mapping G^* of gates $g \in G$ to the n vertices such that F is logically equivalent to the subcircuit T_{F^c} that implements T_F with respect to the outputs of the circuit. We define the **logical feasibility** of the graph T_F as:

DEFINITION 2. *T_F is logically feasible iff* $\exists_{G^*} onset(T_{F^c}) = onset(F)$

where $onset$ represents where the subcircuit is 1 for an input combination. This definition can be relaxed by considering this relation

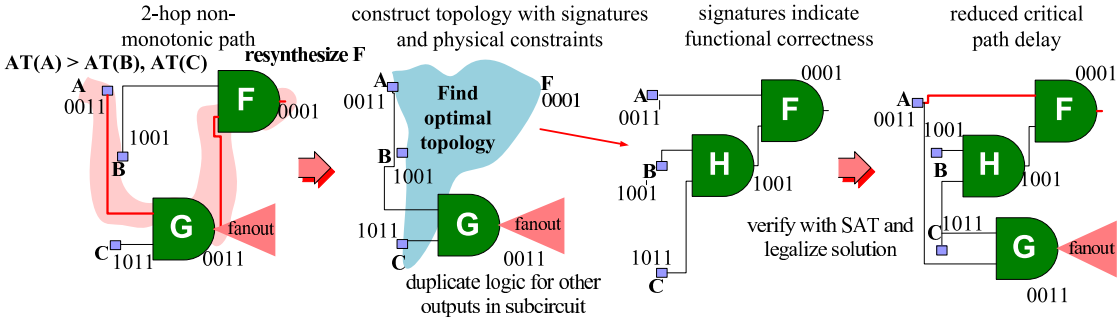

Figure 7: Our flow for restructuring non-monotonic interconnect. We extract a subcircuit determined by our non-monotonic metric and find topologies that are logically equivalent using simulation. This new implementation is then verified by equivalence checking with an incremental SAT solver.

within the care-set which could be considerably smaller than 2^x due to controllability and observability don't-cares.

In the following, we introduce an algorithm for determining logical feasibility on fanout-free circuits, which can be performed with a $O(K * n^2)$-time algorithm using signatures. A candidate restructuring is fanout-free if each node n in T_F has exactly two incoming edges and one outgoing edge. Although our techniques can be applied efficiently on an arbitrary T_F, the advantage of our approach in the fanout-free case is particularly clear. We also consider the set of available gates G as implementing all the 2-input logic functions (16 distinct gates).

A naive algorithm for determining the logical feasibility of T_F requires that every possible mapping G^* is tried. For n vertices, this requires checking $n^{|G|}$ (in this case n^{16}) mappings. Furthermore, performing equivalence checking between T_{F^c} and F is an NP-complete problem. Below, we discuss how signatures can be used to determine a minimal set of inputs that implements a given function and how this can be extended to quickly determine logical feasibility up to the signature approximation.

Pairs of bits to be distinguished: In [7], it was observed that a set of input signatures $S_1, ...S_x$ can implement a target signature S_f, if and only if, every pair of different bits in S_f is *distinguished* by at least one S_x.

DEFINITION 3. *A pair of bits to be distinguished (PBD) is a pair $\{i, j\}$ such that $S_f(i) \neq S_f(j)$.*

DEFINITION 4. *A candidate signature, S_x distinguishes a PBD in S_f if $S_x(i) \neq S_x(j)$ where $\{i, j\} \in S_F^{PBD}$ where S_f^{PBD} is f's set of PBDs.*

Example 1. Assume a target signal $S_f = \{0, 0, 1, 1\}$ and candidates $S_1 = \{0, 0, 0, 1\}$, $S_2 = \{0, 1, 0, 1\}$, and $S_3 = \{0, 1, 1, 1\}$. The PBDs of S_f are $\{0, 2\}, \{0, 3\}, \{1, 2\}, \{1, 3\}$ that need to be distinguished. Note that S_1 and S_2 together cannot implement S_F because they do not distinguish $\{0, 2\}$. However, if all S_x are used, there exists a function that gives S_f. In this example $S_f = S_3 \cdot (S_1 \bar{\oplus} S_2)$. □

Determining logical feasibility with PBDs: We define a PBD that is distinguished by only one S_i as an *essential* PBD for S_i. We associate a signature to each input x of T_F. These signatures implicitly handle controllability don't-cares as impossible input combinations which will never occur in the signatures. By simulating downstream nodes as in [25], observability don't-cares are derived and S_f is reduced to include only care values. If we assume that each S_i under our logic simulation distinguishes at least one essential PBD, we prove the following:

THEOREM 1. *The logical feasibility of an x-input T_F can be determined in $O(K * x^2)$ time for K simulation vectors.*

Proof. Any cut through T_F gives a set of inputs that implements F. Therefore, the S_F^{PBD} must be distinguished by each cut in T_F^c for a feasible topology. Proceeding in topological order, we apply each 2-input transformation (where we ignore the negated case since $OP(a, b)$ and $!OP(a, b)$ distinguish the same number of bits). For a feasible implementation, the operation between any two intermediate nodes S_1 and S_2 must produce a signature that contains all of S_1 and S_2's essential PBDs to uphold the previously stated invariant. Through proof by perfect induction, omitted here, only two Boolean functions (ignoring the negated case) can satisfy this condition. In the worst case, every combination of the *two* different 2-input gates must be applied to each vertex in the graph. □

Our approach is easily extended to arbitrary topologies. By traversing a subcircuit in topological order, we determine that a topology is infeasible for the 2-input transformations already considered. This occurs when an operation cannot preserve the essential PBDs determined by the current cut through the topology. After we determine logical feasibility of a topology, we call a formal proof engine to check for equivalence. By using simulation, the formal proof engine is used in situations where equivalence is most likely.

5.2 Physically-guided Topology Construction

In addition to efficiently *determining* the logical feasibility of various topologies, we propose an algorithm that uses PBDs and physical constraints to efficiently *construct* logically feasible topologies. In this paper, we guide our approach using delay and physical proximity. In the example shown in Figure 8, we try to find an optimal restructuring to implement the target function F with the inputs a, b, and c, using signatures. The functionality of the original circuit is represented by signatures. The figure also shows a table associated with each signal showing the PBDs that are distinguished. The non-essential PBDs for each input signature have light-gray background.

The example shows that the arrival time for c is the greatest, followed by a, then b. Therefore, we consider a topology where c's value is required later. We also consider the proximity of the signals and therefore examine a topology where an operation between a and b is performed. Notice that if all possible 2-input operations are tried, the essential PBDs are not preserved and hence this is not a feasible topology. We then consider another topology where a can be consumed later. For this topology, we see that an XOR-gate will preserve the essential PBDs. We then easily determine that an OR gate is needed to implement F.

Algorithm: We introduce the pseudo-code of our algorithm for restructuring non-monotonic interconnect in Figure 9. After identifying the non-monotonic paths, `Optimize_Interconnect` restructures a portion of the critical path. We first simplify the signa-

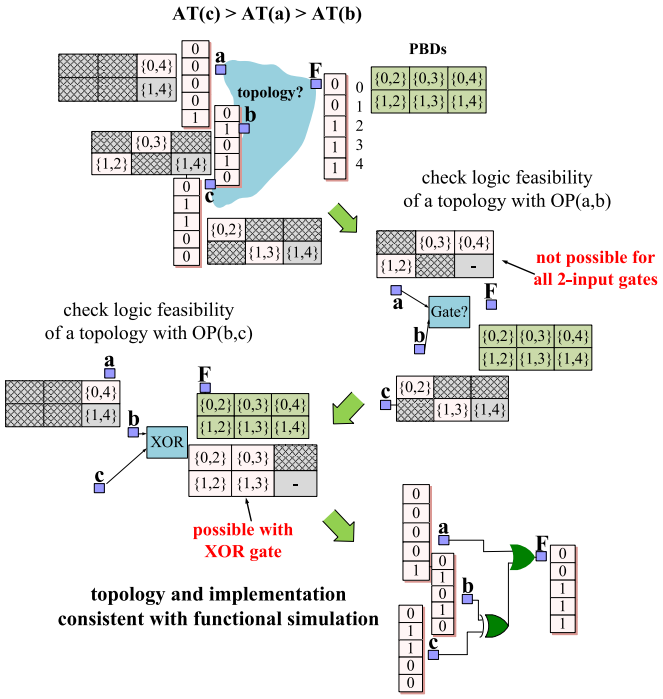

Figure 8: Signatures and topology constraints guide logic restructuring to improve critical path delay. The figure shows the signatures for the inputs of the topology to be derived along with the output. Each table represents the PBDs of the output F that are distinguished. The topology that applies a and b is infeasible because it does not preserve essential PBDs of a and b. A feasible topology uses b and c, followed by a.

tures by `simplify_signatures` by noting that the size of the signature $|S_F|$ can be reduced to the number of different input combinations that occur across $\{S_1, ...S_i\}$. Thus, only a subset of the signature is needed for restructuring because the small subcircuits considered have a maximum of 2^i possible different input combinations, smaller than the number of simulation vectors applied.[2]

We then add any timing or physical constraints, such as locations of the inputs and outputs of the subcircuit being restructured. In `find_opt_topology`, we find a topology that satisfies all the physical constraints and optimizes delay. This topology is constructed by a greedy algorithm where we apply operations on wires that will result in the earliest arrival time at the output. This approximation gives an upper-bound on the best implementation possible that contains the examined combination. If a topology can't be found that satisfies the constraints, the function returns.

We then check the logical feasibility using PBDs and signatures in `check_logical_feasibility`. If the topology is feasible, we associate the derived gates to each vertex and place the subcircuit. Our placement routine considers only the legality of the subcircuit (we call a placement legalizer later). In our approach, we try to place each gate in optimal locations by iterating through each of them and trying nearby locations, until legality and optimality are achieved. For the typically small subcircuits considered, this requires little computational effort. If the topology is not logically feasible, we add a *functional* constraint that will prevent the construction of similar topologies.

If `Optimize_Interconnect` returns a subcircuit, we check

[2]In our experiments, we apply 2048 input vectors and restructure subcircuits with < 10 inputs.

```
void Optimize_circuit(){
    gen_NMF();
    num_tries = X;
    while(worst_nmf > 1)
      if(nckt = Optimize_Interconnect(worst_nmf))
        if(!check_equiv(nckt))
          refine_signatures();
          continue;
        update_netlist();
        legalize_placement();
        update_NMF();
}
Subckt* Optimize_Interconnect(Subckt F){
    simplify_signatures(F);
    Constraints constrs;
    while(find_opt_topology(constrs))
      if(nckt = check_logical_feasibility())
        (*nckt).opt_place();
        return nckt;
      constrs.add(nckt);
}
```

Figure 9: Restructuring non-monotonic interconnect.

the equivalence of the entire circuit using a SAT engine. In the case where our candidate produces a functionally different circuit (which is rare as shown in Section 6), we use the counter-example generated by SAT to refine our simulation hence improving the signatures' quality. If the resulting subcircuit passes verification, we update the netlist and legalize the placement. We update the timing information and the NMFs if a new critical path is found, in which case we select with the next highest NMF and restructure it.

5.3 Efficient Subcircuit Verification

Because we use signatures to limit verification of optimization candidates that are most likely correct, equivalence checking typically confirms the transformation. As in [9], we refine simulation using counterexamples found by failed equivalence checks, so as to reduces additional failed checks. We also minimize the verification time due to equivalence checking by considering only the portions of logic that contributes to the don't-cares used in the transformation. As explained in [25], several don't-cares can exist within a few levels of logic. We invoke a SAT engine so that it considers only these necessary levels of downstream logic. Additionally, we could restrict the equivalence checking to a window around the optimization location to further reduce verification time while still utilizing CDCs and ODCs in the circuit. However, in practice, we observe that the SAT-based equivalence checking requires a small percentage of runtime compared to constructing optimal topologies even for our larger circuit examples. Anecdotal evidence indicates that the MiniSAT engine [13] we use outperforms several SAT solvers developed by EDA companies internally. This may partially explain low runtimes in our experiments.

6. EXPERIMENTS

We implemented and tested our algorithms with circuits from the IWLS 2005 benchmark suite [33], with design utilization set to 70% to match recent practices in the industry. Our wire and gate characterizations are based on a 0.18μm technology library. We perform static timing analysis using the D2M delay metric [2] on Rectilinear Steiner Minimal Trees (RSMTs) produced by FLUTE [12]; here FLUTE can be easily replaced by a timing-driven subroutine, but we do not expect the overall trends in our experiments to change significantly. Our netlist transformations are verified using a modified version of MiniSAT [13] and placed using Capo 10 [6]. We have considered several different initial placements for each circuit by varying a random seed in Capo and report results

as averages over these placements. Our netlist transformations are legalized using the legalizer provided by GSRC Bookshelf [35].

Our delay improvements are achieved by executing the algorithm in Figure 9. We applied 2048 random simulation patterns initially to generate the signatures. We considered paths of less than or equal to 4 hops (5 nodes) using our delay-based metric which allowed us to find many non-monotonic paths while minimizing the size of the transformations considered. We conducted several optimization passes until no more gains were achieved.

6.1 Prevalence of Non-monotonic Interconnect

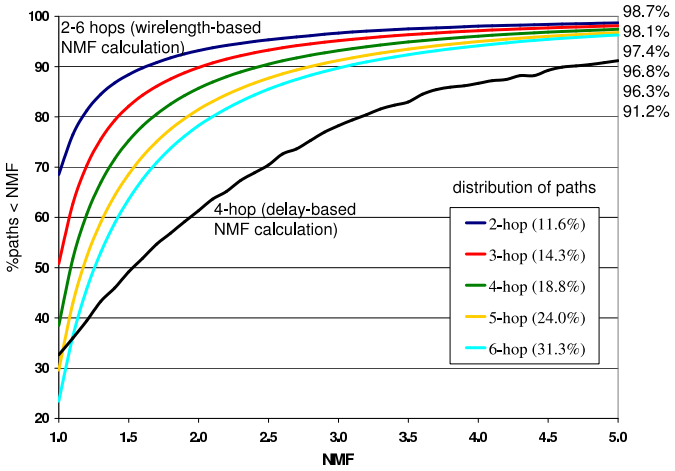

Figure 10: The above graph shows the % of paths whose NMF is below a given value on the x-axis. Notice that longer paths tend to be non-monotonic and at least 1% of paths are > 5 times the ideal minimal length.

Our experiments indicate that circuits often contain many non-monotonic paths. In Figure 10, we illustrate a cumulative distribution of the percentage of paths whose NMFs is below the corresponding value on the x-axis. We generated these averages over all the circuits in Table 1. Each line represents a different path-length examined, where we considered paths from 2 hops to 6 using the wirelength-based NMF metric. We also show the cumulative distribution for the 4-hop delay-based NMF calculation used to guide our delay-based restructuring. Of particular interest is the percentage of monotonic paths, *i.e.*, paths with NMF = 1.

Notice that smaller paths of 2-hops are mostly monotone, whereas the percentage of monotone paths decreases to 23% when considering 6-hop paths. This indicates that focusing optimizations on small paths only, as in [4], can miss several optimization opportunities. It is also interesting to note that there are paths with considerably worse monotonicity having NMFs > 5, indicating regions where interconnect optimizations are needed. We observe similar trends using our delay-based metric. The inclusion of gate delay on these paths results in greater non-monotonicity when compared to the wirelength-metric. Although not shown, each individual circuit exhibits similar trends.

6.2 Physically-aware Restructuring

We show the effectiveness of our delay-based optimization by reporting the delay improvements achieved over several circuits. In Table 1, we provide the number of cells and nets for each benchmark. In the `Performance` columns, we give the %delay improvement, the runtime in seconds, and the percentage of equivalence-checking calls where candidate subcircuits preserved the functionality of the whole circuit. We also report the overhead of our approach with % wirelength increase and the % cell count increase.

Circuit	Cell count	Net count	Performance			Overhead	
			%delay impr	time (s)	%equi	%wire	%cells
sasc	563	568	14.1	41	100	2.35	3.13
spi	3227	3277	10.9	949	82	4.53	0.73
des_area	4881	5122	12.3	503	93	1.09	0.31
tv80	7161	7179	9.1	1075	71	2.50	0.17
s35932	7273	7599	27.5	476	100	2.14	0.19
systemcaes	7959	8220	13.9	748	95	0.89	-0.07
s38417	8278	8309	11.7	481	84	0.68	-0.21
mem_ctrl	11440	11560	9.2	678	37	0.05	-0.02
ac97	11855	11948	6.3	245	100	0.44	0.02
usb	12808	12968	12.2	605	80	0.30	0.06
DMA	19118	19809	14.5	845	65	0.16	0.08
aes	20795	21055	6.4	603	100	0.13	0.01
ethernet	46771	46891	3.7	142	100	0.08	0.06
average			11.7%		85.1%	1.20%	0.34%

Table 1: Significant delay improvement is typically accompanied by a small wirelength increase.

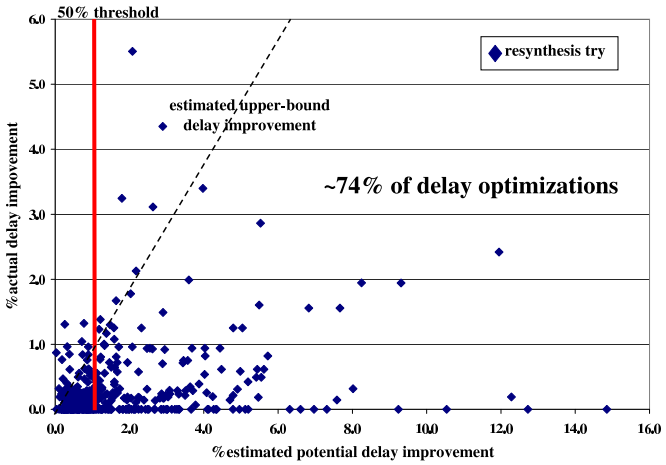

Figure 11: The graph above illustrates that the largest *actual* delay improvements occur at portions of the critical path with the largest *estimated* gain using our metric. The data points are accumulated gains achieved by 400 different resynthesis attempts when optimizing the circuits in Table 1.

We observe improvements on every circuit and a high average delay improvement of 11.7%. For some circuits, such as $s35932$, several don't-care enhanced optimizations enabled even greater delay improvements. We make the following observations:

1. By optimizing only one output of a given subcircuit, we greatly reduce the arrival time of the critical output, while only slightly degrading performance of less critical outputs.
2. Through our efficient use of don't-cares, several x-input subcircuits could be restructured to require fewer than x inputs.
3. As a special case of the previous point, sometimes an input to the subcircuit was functionally equivalent to the output of the subcircuit when don't-cares were considered, enabling delay reduction along with removal of unnecessary logic. Signatures are efficient in exploiting these opportunities.
4. The decomposition of large gates into smaller gate primitives through our restructuring algorithm often produce better topologies because we more precisely construct a topology to meet the physical constraints.
5. We also expect gains due to the duplication and relocation of some cells.

We believe that further gains would be enabled by combining buffering, relocation, and gate sizing strategies between our restructuring optimizations. The runtime of our algorithm scales well for large

circuits due to the use of logic simulation as the main optimization engine. Furthermore, the high % of equivalence checking calls that verified the equivalence of our transformations indicates that signatures are effective at finding functionally equivalent candidates. The wirelength and cell-count overhead are minimal because only a few restructurings are needed and the optimizations can simplify portions of logic. In some cases the number of cells is reduced.

In Figure 11, we demonstrate that our delay-based NMF metric is effective at guiding optimization. Each data point represents a different resynthesis try considering all of the circuits in Table 1. The x-axis shows the predicted % delay gain possible (determined by the optimal-buffered delay). The y-axis indicates the actual gain. Data points that lie on the x-axis indicate resynthesis tries that did not improve delay (a better topology could not be found). The 50% threshold line divides the graph so that the number of resynthesis attempts are equal on both sides. The diagonal line indicates an upper-bound prediction for delay gain. Because some of our optimizations could reduce support, it is possible for some data points to be above the line. Although the NMF and gain calculations do not directly incorporate circuit functionality, 74% of all delay gains are found on the right half of the graph. The correlation to our metric could be further improved by incorporating the % of gain possible with respect to near-critical paths.

7. CONCLUSIONS

Interconnect delay is becoming a major obstacle for achieving timing closure, typically requiring numerous expensive design iterations. Current logic synthesis strategies often sacrifice other performance metrics to improve delay, requiring computationally expensive algorithms and companion placements. Despite these efforts, extensive post-placement optimizations are still needed, especially since buffers will represent a large fraction of standard cells in future technologies [26].

We propose a solution that improves the quality of delay optimization without sacrificing other performance metrics. To this end, we introduce a novel simulation-guided synthesis strategy that is more comprehensive than current restructuring techniques. We develop a path-monotonicity metric to focus our efforts on the most important parts of a design. Our optimizations lead to 11.7% delay improvement on average over several different initial placements, while our delay-based monotonicity metric indicated that 65% of the paths analyzed were non-monotonic. We further observe delay improvements on placements initially optimized for delay, which are consistent with our reported average improvement. We believe that our approach offers an effective bridge between current topological-based synthesis and lower-level physical synthesis approaches. It enables less conservative estimates early in the design flow to improve other performance metrics and reduce the amount of buffering required by shortening critical paths. Future work will explore the benefits of using our technique in conjunction with other physical synthesis strategies such as buffering.

8. REFERENCES

[1] A. Ajami and M. Pedram, "Post-layout timing-driven cell placement using an accurate net length model with movable steiner points", *DAC '01*, pp. 595-600.

[2] C. Alpert, A. Devgan, and C. Kashyap, "RC delay metric for performance optimization", *TCAD '01*, pp. 571-582.

[3] C. Alpert, A. Kahng, C. Sze, and Q. Wang, "Timing-driven steiner trees are (practically) free", *DAC '06*, pp. 389-392.

[4] G. Beraudo and J. Lillis, "Timing optimization of FPGA placements by logic replication", *DAC '03*, pp. 541-548.

[5] D. Brand, "Verification of large synthesized designs", *ICCAD '93*, pp. 534-537.

[6] A. Caldwell, A. Kahng, and I. Markov, "Can recursive bisection alone produce routable placements?", *DAC'00*, pp. 693-698.

[7] K.-H. Chang, I. Markov, and V. Bertacco, "Fixing design errors with counterexamples and resynthesis", *ASP-DAC '07*. pp. 944-949.

[8] K.-H. Chang, I. Markov, and V. Bertacco, "Safe delay optimization for physical synthesis", *ASP-DAC '07*. pp. 628-633.

[9] K.-H Chang, D. Papa, I. Markov, and V. Bertacco, "InVerS: an incremental verification system with circuit similarity metrics and error visualization", *ISQED '07*. pp. 487-494.

[10] S. Chatterjee and R. Brayton, "A new incremental placement algorithm and its application to congestion-aware divisor extraction", *ICCAD '04*, pp. 541-548.

[11] C.-W Chang, C.-K Cheng, P. Suaris, and M. Marek-Sadowska, "Fast post-placement rewiring using easily detectable functional symmetries", *DAC '00*, pp. 286-289.

[12] C. Chu and Y.-C. Wong, "Fast and accurate rectilinear steiner minimal tree algorithm for VLSI design", *ISPD'05*, pp. 28-35. (http://class.ee.iastate.edu/cnchu/flute.html)

[13] N. Een and N. Sorensson, "An extensible SAT-solver", *SAT '03*, (http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/).

[14] W. Gosti, A. Narayan, R. Brayton, and A. Sangiovanni-Vincentelli, "Wireplanning in logic synthesis", *ICCAD '98*, pp. 26-33.

[15] W. Gosti, S. Khatri, and A. Sangiovanni-Vincentelli, "Addressing the timing closure problem by integrating logic optimization and placement", *ICCAD '01*, pp. 224-231.

[16] M. Hrkic, J. Lillis, and G. Beraudo, "An approach to placement-coupled logic replication", *DAC '04*.

[17] L. Kannan, P. Suaris, and H. Fang, "A methodology and algorithms for post-placement delay optimization", *DAC'94*, pp. 327-332.

[18] A. Kuehlmann, V. Paruthi, F. Krohm, and M. Ganai, "Robust Boolean reasoning for equivalence checking and functional property verification", *TCAD '02*, pp. 1377-1394.

[19] C. Li, C-K. Koh, and P. Madden, "Floorplan management: incremental placement for gate sizing and buffer insertion", *ASP-DAC'05*, pp. 349-354.

[20] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: a fresh look at combinational logic synthesis", *DAC '06*, pp. 532-536.

[21] A. Mishchenko, S. Chatterjee, R. Jiang, and R. Brayton, "FRAIGs: A unifying representation for logic synthesis and verification", *ERL Technical Report '05*, Berkeley. (http://www.eecs.berkeley.edu/∼alanmi/publications/).

[22] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: engineering an efficient SAT solver", *DAC '01*, pp. 530-535.

[23] R. Otten and R. Brayton. "Planning for performance", *DAC '98*, pp. 122-127.

[24] M. Pedram and N. Bhat. "Layout driven logic restructuring/decomposition", *ICCAD '91*, pp. 134-137.

[25] S. Plaza, K.-H Chang, I. Markov, and V. Bertacco, "Node mergers in the presence of don't cares", *ASP-DAC '06*, pp. 414-419.

[26] P. Saxena, N. Menezes, P. Cocchini, and D. Kirkpatrick, "Repeater scaling and its impact on CAD", *TCAD '04*, pp. 451-463.

[27] J. Marques-Silva and K. Sakallah, "GRASP: A search algorithm for propositional satisfiability", *IEEE Trans. Comp '99*, pp. 506-521.

[28] G. Stenz, B. Riess, B. Rohfleisch, and F. Johannes, "Timing driven placement in interaction with netlist transformations", *ISPD '97*, pp. 36-41.

[29] L.P.P.P van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay", *ISCAS '90*, pp. 865-868.

[30] J. Werber, D. Rautenbach, and C. Szegedy, "Timing optimization by restructuring long combinatorial paths", *ICCAD '07*, pp. 536-543.

[31] Q. Zhu, N. Kitchen, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "SAT sweeping with local observability don't cares", *DAC '06*, pp. 229-234.

[32] The International Technology Roadmap for Semiconductors, 2005 Edition, ITRS.

[33] `http://iwls.org/iwls2005/benchmarks.html`.

[34] Synopsys DesignCompiler. `http://www.synopsys.com`.

[35] UMICH Physical Design Tools, `http://vlsicad.eecs.umich.edu/BK/PDtools/`