Keeping Physical Synthesis Safe and Sound

Kai-hui Chang, Igor L. Markov, Valeria Bertacco
EECS Department, University of Michigan, Ann Arbor, MI 48109-2121
{changkh, imarkov, valeria}@umich.edu

ABSTRACT

Physical synthesis is a relatively young field in Electronic Design Automation. Many published optimizations for physical synthesis and timing-driven placement end up hurting the final result, often by neglecting important physical aspects of the layout, such as long wires or routing congestion. Our work defines and explores the concepts of physical safeness and logical soundness, and empirically evaluates the effects of physical safeness on routed wirelength, via count and timing. In addition, we propose a new physically safe and logically sound optimization, which provides immediately-measurable improvement and immediately-verifiable correctness that can enhance circuit timing without detrimental effects on wirelength and congestion. We achieve these improvements by performing a series of netlist transformations and re-placements that are individually evaluated for logical soundness (using on-line verification) and physical safeness. When used alone, our techniques improve clock periods of IWLS'05 benchmarks by 5% on average after routing, while increasing routed wirelength by less than 0.2%. Since physically safe techniques provide predictable improvements and do not worsen timing, they can also be used instead of or in addition to more traditional, unsafe physical synthesis algorithms popular in commercial tools. To this end, a combination of safe and unsafe transformations achieves 11% average improvement for seven large benchmarks from the OpenCores suite, and up to 56% for individual circuits.

1. INTRODUCTION

Timing optimization of digital logic is gaining importance with each technology step, as interconnect contributes a larger fraction of critical-path delay due to its poor scaling. Since accurate timing information can only be obtained after the circuit is placed, post-placement timing optimization has been studied extensively. Most techniques either modify the logic or change the physical aspects of the circuit [7]. Physical solutions include net buffering, gate sizing [14] and gate relocation [1]. Logical solutions include gate replication [12], rewiring [5, 6] and restructuring [4, 15, 17, 21]. Techniques based on a placement or routing with the goal to improve timing are often called physical synthesis.

A number of previous works on physical synthesis do not provide an overall improvement because when optimizing one aspect of the design, they may damage other aspects. For example, uncontrollable logic cloning may increase area and wirelength, making critical nets longer than expected during placement and routing [12]. Indiscriminate buffering may also create many gate overlaps, leading to potentially detrimental effects on circuit timing when overlaps are resolved [16]. A number of related publications that try to solve this problem are now available. For example, Li et al. [16] proposed an incremental placement algorithm which maintains the stability of a placement for gate sizing and buffer insertion. Another approach deals with this problem by designing legalizers that seek to preserve circuit metrics, such as the works by Luo et al. [19] and Brenner et al. [2].

Timing-driven placement also suffers similar stability problems. In particular, optimization in placement often increases routing congestion around timing-critical nets. While published papers typically report timing estimates before routing, critical nets often detour during routing, which results in worse performance compared with traditional placement. Timing-driven placement also tends to be unpredictable for reasons specific to placement algorithms. To this end, it is not uncommon for commercial place-and-route tools to produce *better* results when the timing-driven mode is *disabled*. In practice, the best bet to improve timing of a particular design is to try as many timing-driven and non-timing-driven tools as possible. For example, the empirical results shown by Wang and Kahng [13] suggest that there are more routing violations when placers are used in their timing-driven modes, and no single placer, commercial or academic, dominates over the entire benchmark set.

In our work we define and explore *physically safe* and *logically sound* netlist transformations — those that provide immediately-measurable improvement and immediately-verifiable correctness on every step. As seen from our empirical results, these transformations produce more predictable improvements without detrimental effects on other circuit parameters. In addition, they do not conflict with any existing design flows and can be used after unsafe transformations. In the past, safe transformations have been largely neglected because they offered very little improvement [5]. However, the amount of improvement depends entirely on the set of available transformations [7], and our empirical results suggest that safe transformations may improve upon unsafe transformations. Another contribution of our work is a new approach to finding such transformations, based on iterative equivalence checking. By broadening the set of transformations and applying them in a safe way, we show that circuit delay can be improved considerably with very little risk of destabilizing an existing design flow or hampering timing closure, a common problem with new ideas in physical synthesis. Figure 1 shows two examples of our optimizations. In Figure 1(a), the signal that drives $g8$ is resynthesized using gates located closer to it, and a new gate is added to replace the old one. In Figure 1(b), one gate ($g8$) originally driven by $g6$ uses gate *new* as its new source, while the other gate ($g1$) is still driven by $g6$. None of the new gates overlap with old gates, and they are placed in previously-unused locations. Empirical results (in Table 2) show that our technique can improve cycle time by 5% while wirelength and via count increase by less than 0.2%.

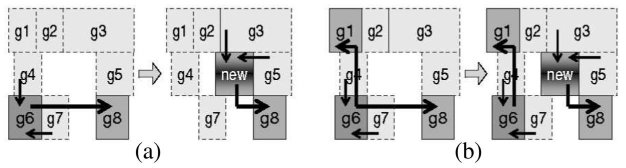


Figure 1: Example transformations for row-based standard-cell layout: (a) resynthesized gate *new* replaces $g6$ to drive $g8$, (b) gate cloning uses resynthesized gate *new* to drive $g8$, while the original driver $g6$ continues to drive $g1$.

The rest of this paper is organized as follows. In Section 2 we describe the concepts of physical safeness and logical soundness, and then review previous work on physical synthesis. We then propose a new powerful, safe and sound timing optimization approach in Section 3. Several aspects of our technique are analyzed in Section 4. Experimental results are reported in Section 5, and Section 6 concludes this paper.

2. SAFENESS AND SOUNDNESS OF PHYSICAL SYNTHESIS TECHNIQUES

Existing techniques for post-placement timing optimization vary in strength and differ in how they affect logic and gate locations [7]. We use the term "physical safeness" to describe their impact on placement and "logical soundness" to describe their impact on logic. In this section, we first describe safeness and soundness in detail. After that, we introduce several physical synthesis techniques and analyze their optimization capabilities and safeness.

2.1 Physical Safeness and Logical Soundness

The concept of *physical safeness* is used to describe the impact of an optimization technique on the placement of the circuit. Physically safe techniques only allow legal changes to the placement, therefore accurate analysis such as timing and congestion can be performed. Such changes are safe because they can be rejected immediately if the layout is not improved. On the other hand, unsafe techniques allow changes that produce a temporarily illegal placement. As a result, its evaluation is delayed, and we cannot reliably accept or reject the change. Therefore, the average quality of unsafe changes may be worse than that of accepted safe changes, as can be seen from Table 4. In addition, other physical parameters, such as via count, may be impacted by unsafe transformations.

Similar to physical safeness, *logical soundness* is used to describe the perturbation to the logic made by the optimization techniques. Techniques that do not change the logic usually do not require verification. Examples for this type of optimization include gate sizing and buffer insertion. Techniques that change the logic of the circuit may require verification to ensure their correctness. For example, optimizations based on reconnecting wires require verification because any bug in the optimization process may change the circuit's behavior. Since local changes to combinational logic can be verified easily using SAT, they are considered logically sound. However, small changes to sequential logic often have global implications and are much more difficult to verify, therefore we do not classify them as logically sound techniques. These techniques include the insertion of clocked repeaters and the use of retiming.

2.2 Physically Safe Techniques

Symmetry-based rewiring is the only timing optimization technique that is physically safe in nature. It exploits symmetries in logic functions, looking for pin reconnections that improve timing [5]. For example, the inputs to an AND gate can be swapped without changing its logic function. Since only wiring is changed in this technique, the placement is always preserved. An example of symmetry-based rewiring is given in Figure 2(a).

The advantage of physically safe techniques is that the effects of any change are immediately measurable, therefore the change can be accepted or rejected reliably. As a result, delay will not deteriorate after optimization and no timing convergence problem will occur. However, the improvement gained from these techniques is often limited because they cannot aggressively modify the logic or use larger-scale optimizations. For example, in [5] timing improvement measured before routing is typically less than 10%. To this end, our experimental results in Section 5 show that post-routing timing improvements typically do not match pre-routing results and must be evaluated directly.

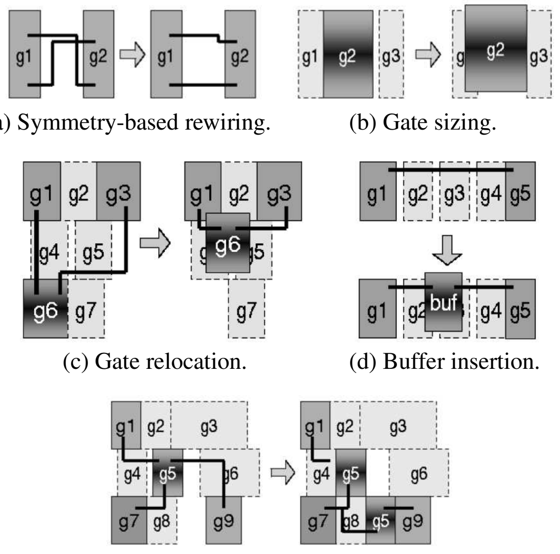
2.3 Physically Unsafe Techniques

Traditional physical synthesis techniques are physically unsafe because they create cell overlaps and thus prevent immediate evaluation of changes. Although some of these techniques can be applied in a safe way, they may lose their strength. Therefore existing physical synthesis tools usually rely on unsafe techniques and fix the newly-created problems later on. These techniques and their impact on logic are discussed below.

Gate sizing and buffer insertion are two important techniques that do not change the logic, as shown in Figure 2(b) and Figure 2(c). Gate sizing chooses the sizes of the gates carefully so that signal delay in wires can be balanced with gate delay, and the gates have enough capability to drive the wires. Buffer insertion adds buffers to drive long wires. The work by Kannan et al. [14] is based on these techniques.

Gate relocation moves gates on critical paths to better locations and also does not change the logic. An example of gate relocation is given in Figure 2(d). Ajami et al. [1] utilize this technique by performing timing-driven placement with global routing information using the notion of movable Steiner points, and they formulate the simultaneous placement and routing problem as a mathematical program. The program is then solved by Han-Powell method.

Gate replication is another technique that can optimize timing without changing the logic. Take Figure 2(e) for example, by duplicating $g5$, the delay to $g1$ and $g9$ can be reduced. Hrkic et al. [12] proposed a placement coupled approach based on such technique. Given a placed circuit, they first extract replication trees from the critical paths after timing analysis, and then they do embedding and post-unification to determine the gates that should be duplicated and their locations. Since duplicated gates may overlap some existing gates, timing-driven legalization is applied at the end. Although their approach improves timing by 1-36%, it also increases wirelength by 2-28%.



(a) Symmetry-based rewiring.
(b) Gate sizing.
(c) Gate relocation.
(d) Buffer insertion.
(e) Gate duplication.

Figure 2: Physical synthesis techniques. Newly-introduced overlaps are removed by legalizers.

Traditional rewiring techniques based on addition and removal of redundant wires are not physically safe. The basic idea is to add one or more redundant wires to make a target wire redundant so that it becomes removable. Since gates must be changed to reflect the changes in wires, overlap of cells may occur. The work by S. C.

Chang utilizes this technique based on an ATPG (Automatic Test Pattern Generation) reasoning mechanism [6].

Optimization techniques discussed so far can be made physically safe by rejecting all changes that create new overlaps, for example, this would allow inserting buffers only in overlap-free sites. However, the prevailing practice for these and many other optimizations is to first allow overlaps and then call a legalizer to fix overlaps. According to our definition, this is physically unsafe. In other words, depending on how many overlaps are introduced, how powerful and how accurate the legalizer is, the physical parameters of the circuit may improve or deteriorate.

Traditional restructuring focuses on directing the synthesis process using timing information obtained from a placed or routed circuit. It is more aggressive in that it will change the logic structure as well as placement. This technique reflects the fact that timing-driven synthesis requires accurate timing, which can only be obtained from a placed circuit. However, a circuit cannot be placed unless it is synthesized. Restructuring tries to bridge the gap between these two different stages in circuit design.

A typical restructuring flow includes: (1) obtaining accurate timing analysis results from placed or routed design, (2) identifying critical paths, (3) selecting gates from critical paths to form critical regions, (4) performing timing-driven resynthesis on the critical regions, and (5) calling legalizers to remove gate overlaps that may be created during the process. This process is repeated until timing closure is achieved. The works by Lu et al. [17], Vaishnav et al. [21] and Changfan et al. [4] are all based on this flow with emphasis on different aspects. For example, the work by Vaishnav focuses on eliminating late-arriving events identified by symbolic simulation, while Changfan analyzes effects of routing on timing and utilizes them in his resynthesis and incremental placement engines.

Traditional restructuring is usually physically unsafe. For example, evaluation of new cell locations cannot be done reliably for technology-independent restructuring unless technology mapping is also performed. Moreover, restructuring techniques based on AIGs are likely to be unsafe because node mergers performed in an AIG may distort a given placed circuit [22]. As a result, the effects of the changes are not immediately measurable. In other words, the delay after optimization may be worse than before. Although carefully designed techniques can be used to alleviate this problem [15, 16, 19], it is difficult to eliminate altogether. The strength and safeness of these techniques are summarized in Table 1.

Techniques	Physical safeness	Optimization strength
Symmetry-based rewiring	Safe	Low
Our work	**Safe**	**Medium**
ATPG-based rewiring, buffer insertion, gate sizing, gate relocation	Unsafe*	Low
Gate replication	Unsafe*	Medium
Restructuring	Unsafe	High

Table 1: Comparison of physical safeness and optimization strength of different techniques. Low strength means only local optimization is possible, high strength means large scale optimization is possible, and medium strength is in between. *Note: some techniques can be made safe but popular implementations allow gate overlap and are unsafe.

3. A NEW POWERFUL, SAFE AND SOUND PHYSICAL SYNTHESIS APPROACH

Our safe and sound physical synthesis approach is discussed in detail in this section. It is "safe" in the sense that the produced placement is always legal and the improvement can be evaluated immediately, and it is "sound" because the logic transformation can be verified easily. Our algorithm is different from previous techniques in that it does not restrict resynthesis to small geometric regions or small groups of adjacent wires. Furthermore, there is much less perturbation to the placement.

3.1 Terminology

A *signature* is a bit-vector of simulated values of a wire. Given the signature s_t of a wire w_t to be resynthesized, and a certain gate g_1, a wire w_1 with signature s_1 is said to be *compatible* with w_t if it is possible to generate s_t using g_1 with signature s_1 as one of its inputs. In other words, it is possible to generate w_t from w_1 using g_1. For example, if $s_1 = 1$, $s_t = 1$ and $g_1 = AND$, then w_1 is compatible with w_t using g_1 because it is possible to generate 1 on an AND's output if one of its inputs is 1. However, if $s_1 = 0$, then w_1 is not compatible with w_t using g_1 because it is impossible to obtain 1 on an AND's output if one of its inputs is 0 (see Figure 5).

A *controlling value* of a gate is the value that fully specifies the gate's output when applied to one input of the gate. For example, 0 is the controlling value for AND because when applied to the AND gate, its output is always 0 regardless of the value of other inputs. When two signatures are *incompatible*, that can often be traced to a *controlling value* in some bits of one of the signatures.

3.2 Our Framework for Safe Restructuring

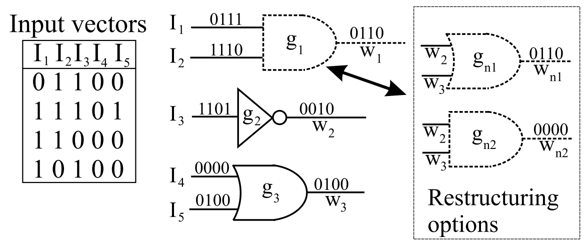


Figure 3: A restructuring example. Input vectors to the circuit are shown on the left. Each wire is annotated with its bit-signature computed by simulation on those test vectors. We seek to compute signal w_1 by a different gate, e.g., in terms of signals w_2 and w_3. Two such restructuring options (new gates) are shown as g_{n1} and g_{n2}. Since g_{n1} produces the required signature, equivalence checking is performed between w_{n1} and w_1 to verify the correctness of this restructuring. Another option, g_{n2}, is abandoned because it fails our compatibility test.

The basic idea behind our physical synthesis approach is to use simulation to identify potential resynthesis opportunities, and a CNF-SAT solver is used to prove their validity [22]. Since our goal is layout optimization, we can prune some of the opportunities based on their promise before formally verifying them since verification is relatively slow. Our framework is given in Figure 4, and an example is given in Figure 3. Initially, *library* contains all the gates to be used for resynthesis. We first generate a signature for each wire by simulating certain input patterns, whose selection will be discussed in detail in Section 3.4. In order to optimize timing, $wire_t$ in line 2 will be selected from wires on the critical paths in the circuit. Line 3 restricts our search of potential resynthesis opportunities according to certain physical constraints, and lines 4-5 further prune our search space based on logical soundness. After a valid resynthesis option is found, we try placing the gate on various overlap-free sites close to a desired location in line 6 and check their timing improvements. In this process, more than one gate may be added if there are multiple sinks for $wire_t$, and the original driver of $wire_t$ may be replaced. We only call equivalence checking when we found certain changes that improve timing because formal verification is time-consuming. In line 10 we remove redundant gates and wires that may appear because $wire_t's$ original driver may no longer drive any wire, which often initiates a chain of further simplifications.

```
1   Simulate patterns and generate a signature for each wire.
2   Determine $wire_t$ to be resynthesized and retrieve $wires_c$
    from the circuit.
3   Prune $wires_c$ according to physical constraints.
4   Foreach $gate \in library$ with inputs selected from
    combinations of compatible wires $\in wires_c$.
5     Check if $wire_t$'s signature can be generated using $gate$
      with its inputs' signatures. If not, try next combination.
6       If so, do restructuring using $gate$ by placing it on
        overlap-free sites close to the desired location.
7         If timing is improved, check equivalency. If not
          equivalent, try next combination of wires.
8           If equivalent, a valid restructuring is found.
9     Use the restructuring with maximum improvement for
    resynthesis.
10  Identify and remove gates and wires made redundant by
    resynthesis.
```

Figure 4: Our resynthesis framework.

3.3 Search-Space Pruning Techniques

In order to resynthesize a target wire ($wire_t$) using a n-input gate in a circuit containing m wires, the brute force technique needs to check $\binom{m}{n}$ combinations of possible inputs, which can be very time-consuming for $n > 2$. Therefore it is important to prune the number of wires to try.

When the objective is to optimize timing, the following physical constraints can be used in line 3 of the framework: (1) wires with arrival time later than that of $wire_t$ are discarded because resynthesis using them will only increase delay, and (2) wires that are too far away from the sinks of $wire_t$ are abandoned because the wire delay will be too large to be beneficial. We set this distance threshold to twice the HPWL (Half-Perimeter Wirelength) of $wire_t$.

In line 4 logical compatibility is used to prune the wires that need to be tried. Wires not compatible with $wire_t$ using $gate$ are excluded from our search. Figure 5 summarizes how compatibilities are determined given a gate type, the signatures of $wire_t$ and the wire to be tested ($wire_1$).

Gate type	$wire_t$	$wire_1$	Result
NAND	0	0	Incompatible
NOR	1	1	Incompatible
AND	1	0	Incompatible
OR	0	1	Incompatible
XOR/XNOR	Any	Any	Compatible

Figure 5: Conditions to determine compatibility: $wire_t$ is the target wire, and $wire_1$ is the potential new input of the resynthesized gate.

To accelerate compatibility testing, we use the "one-count", i.e., the number of 1s in the signature, to filter out unpromising candidates. For example, if $gate$==OR and the one-count of $wire_t$ is smaller than that of $wire_1$, then these two wires are incompatible because OR will only increase one-count in the target wire. This technique can be applied before bit-by-bit compatibility test to detect incompatibility faster.

Our *pruned_search* algorithm that implements lines 4-5 of the framework is outlined in Figure 6. The algorithm is specifically optimized for two-input gates but can be extended to gates with more than two inputs. $Wire_t$ is the target wire to be resynthesized, $wires_c$ are wires selected according to physical constraints, and $library$ contains gates used for resynthesis. All wires in the fanout cone of $wire_t$ are excluded in the algorithm to avoid formation of combinational loops.

```
Function $pruned\_search(wire_t, wires_c, library)$
1  foreach $gate \in library$
2    $wires_g = compatible(wire_t, wires_c, gate)$;
3    foreach $wire_1 \in wires_g$
4      $wires_d = get\_potential\_wires(wire_t, wire_1, wires_g, gate)$;
5      foreach $wire_2 \in wires_d$
6        Restructure using $gate$, $wire_1$ and $wire_2$;
```

Figure 6: Our *pruned_search* algorithm.

In Figure 6, function *compatible* returns wires in $wires_g$ that are compatible with $wire_t$ given $gate$. Function *get_potential_wires* returns wires in $wires_d$ that are capable of generating the signature of $wire_t$ using $gate$ and $wire_1$, and its algorithm is outlined in Figure 7. For XOR/XNOR, the signature of the other input can be calculated directly, and wires with signatures identical to that signature are returned using the signature hash table. For other gate types, signatures are calculated iteratively for each wire (denoted as $wire_2$) using $wire_1$ as the other input, and the wires that produce signatures which match $wire_t's$ are returned.

```
Function $get\_potential\_wires(wire_t, wire_1, wires_g, gate)$
1  if $gate$ == XOR/XNOR
2    $wires_d$= $sig\_hash[wire_t.signature$ XOR/XNOR
        $wire_1.signature]$;
3  else
4    foreach $wire_2 \in wires_g$
5      if $wire_t.signature$ ==
          $gate.evaluate(wire_1.signature, wire_2.signature)$
6        $wires_d \leftarrow wires_d \cup wire_2$;
7  return $wires_d$;
```

Figure 7: Algorithm for function *get_potential_wires*. XOR/XNOR is considered separately because the required signature can be calculated uniquely from $wire_t$ and $wire_1$.

3.4 Implementation Insights

In our implementation, we select desired locations for placing the restructured gates using the following criterion: the first 200 overlap-free slots closest to the center of gravity of the new gate's input and output wires' centers of gravity. Although better initial guesses may exist for desired locations than the center of gravity, they are not necessary because a fairly large number of valid locations around the desired location will be evaluated rigorously. As a result, having an extremely accurate initial guess is not necessary to find the actual best location.

The performance of our algorithm is greatly influenced by the quality of the signatures generated by simulation. Poor signatures cannot distinguish many different wires and require additional calls to equivalence-checking. On the other hand, potentially resynthesizable wires can usually be distinguished from those not resynthesizable if their signatures are different. In light of this, we enhanced the FRAIG package in ABC [22] to dump its patterns and use them for our initial simulation. The purpose of the patterns in ABC is to distinguish different nodes in the AIG (And-Inverter Graphs) netlist built from the circuit, therefore they are also suitable for generating signatures that can distinguish different wires. In particular, if the FRAIG package is run with infinite backtrack limit, at least one simulation vector will exist to distinguish every two nodes. Currently, FRAIGs first simulate 2048 random patterns. Next, they append the counterexamples returned during equivalence checking and their variants as additional simulation patterns.

Despite our efforts to generate high-quality signatures, ill-behaved signatures still exist and may render our simulation-based techniques ineffective. For example, a wire with an all-1 signature can generate a target wire with an all-1 signature using any wire through an OR gate. The same happens to NOR, AND and NAND gates, but not to XOR and XNOR gates. This problem arises because the gate being tried is controlled by one of its input wires. When this happens, only equivalence checking can verify the correctness of resynthesis involving the ill-behaved wire. Needless to say, most such resynthesis opportunities are invalid, making the time spent to verify them worthless. Therefore in our implementation, we abandon resynthesis opportunities with the number of uncontrolled bits (bits in the signature with non-controlling value of the gate) smaller than 4, making sure simulation-based techniques have enough chance to prune impossible combinations of wires.

4. ANALYSIS OF OUR APPROACH

Several aspects of our approach are discussed in this section, including its scalability, optimization power, safeness, advantages and limitations.

Scalability: Suppose that there are m wires in the circuit and g n-input gates are used for resynthesis, then the worst case time complexity of our resynthesis algorithm is on the order of $g \times m^n$ if $n \leq m/2$. However, by using physical constraints and logical pruning techniques, as well as several other heuristics, the time complexity is reduced significantly in practice. From our experimental results, we observe that the runtime is somewhere between linear and quadratic for $n = 2$. For example, a netlist with almost 100K nets can be resynthesized in 14 minutes (Table 2).

Aside from runtime, the use of signatures instead of other logic representations, such as BDDs, makes our approach more scalable in terms of memory usage. For example, comparable methods to find resynthesis opportunities in [8, 9] are evaluated for at most 5K gates at a time, whereas our techniques typically handle 100K-gate circuits in minutes. Commercial tools often use BDDs but achieve scalability by means of (i) netlist partitioning, and (ii) restricting logic optimization to small windows. To this end, our main contribution is a relatively simple framework that is fast and naturally scales to large designs without netlist partitioning or windowing.

Optimization Power and Safeness: Our resynthesis technique tries to reproduce a signal using gates in the library with new inputs selected from the whole circuit, therefore it is essentially a form of technology mapping. Since the selection is not limited by small windows like in previous restructuring techniques [4], it is capable of finding optimizations that are long range. Furthermore, complete controllability don't-cares are automatically utilized in our techniques by construction, while no explicit don't-care computations [18] are required. These don't-cares also give our technique more optimization power to find restructuring opportunities.

When we try to resynthesize a wire, we are either trying to remove a gate and drive all the relevant sinks by a new gate or to speed up the propagation of the signal to the sinks of the wire. The former case subsumes simple gate relocation, gate relocation that simultaneously changes gate type, and also several types of traditional restructuring. The latter case subsumes single-gate logic replication, including the possibility of gate relocation and changing the gate type immediately after cloning.

All our transformations are physically safe in that no gates will be overlapped by our optimization. They also have limited effect on congestion because gates may be removed after each transformation, making white-space almost equal or even better after resynthesis. Furthermore, it is easy to veto transformations that violate designer-specified constraints or worsen designer-specified quality metrics, e.g., involve wires crossing obstacles, increase gate area or aggravate routing congestion. By making sure that every transformation improves major quality metrics without introducing new violations, we ensure that our resynthesis techniques are physically safe. On the other hand, by subsuming and generalizing several existing techniques they achieve considerable strength in practice.

Advantages and Limitations: In summary, the advantages of our resynthesis approach include:

- It is physically safe and logically sound, i.e., it guarantees the correctness of logic, does not increase the number of physical violations and makes sure that major physical parameters are either improved or remain the same.
- It is powerful in that it considers longer range optimizations and utilizes complete controllability don't-cares.
- It subsumes gate relocation and gate replication techniques whenever beneficial.
- It includes a form of physically-safe technology mapping.
- Our experimental results show a 5% cycle-time improvement as measured after routing with less than 0.2% increase in wirelength and via count on average (see Table 2), which confirms that our technique is both powerful and safe.

Sometimes our technique generates a very large number of potential restructuring opportunities that it cannot fully analyze in reasonable time. However, we do not view this as a fundamental limitation and believe that future research may address this drawback.

We observed that our technique does not improve standard arithmetic circuits because they are already heavily optimized. Nonetheless, our technique can be very helpful for large netlists automatically synthesized from HDL descriptions.

5. EXPERIMENTAL RESULTS

We implemented our techniques in C++ including a simple incremental static timing analysis engine for our experiments.[1] In our benchmarks, gate delays range from 0.025ns to 0.15ns, the unit capacitance is $131.53pF/m$ and unit resistance is $337K\Omega/m$. The driver resistance ranges from $2.5K\Omega$ to $10K\Omega$, and the port capacitance is $0.0149pF$. These parameters are based on a 0.18μm technology library, and we expect greater improvements as wire delays become more significant in newer technologies. As in [20], we estimate net delay using Elmore formulas applied to a star topology with the star point placed at the center of gravity of all pins. This model is referred to as the star model in the rest of the paper. After resynthesizing the circuits, we route all the benchmarks and perform more accurate timing analysis based on routed wirelength to calculate the final timing of the circuit [10].

Our hardware platform is an AMD Opteron 250 workstation running Fedora 3 Linux. Our experiments use the min-cut placer Capo

[1]Our techniques should work with any incremental static timing analysis tool. We are currently trying the OAGear-0.94 package, but have not yet been able to use its static timing analyzer because of crashes.

10 from the University of Michigan [3], the QPlace 5.2 placer from Cadence Design Systems, and the NanoRoute 4.1 router also from Cadence. Simulation patterns are generated by the ABC package from UC Berkeley [22], and all transformations are verified by an external equivalence checker based on the MiniSat SAT solver [11]. While our organization licenses a broad range of EDA software, we do not currently have access to relevant physical synthesis tools. However, we do not believe that our techniques are used by EDA vendors, and would be willing to perform empirical comparisons when appropriate tools become available to us. We expect that our tools will achieve comparable improvements, but run considerably faster.

Our testcases are selected from IWLS 2005 benchmarks [23], where the design utilization is 70%. They belong to the following suites: OpenCores (AC97, USB, PCI, AES, WB_CONMAX, Ethernet, DES_PERF, SPI, DES_AREA, TV80, SYSTEMCAES and MEM_CTRL), Faraday (DMA), ITC99 (b14, b15, b17, b18 and b22) and ISCAS89 (s35932 and s38417). The benchmarks in the OpenCores suite are produced by a quick synthesis run of Cadence RTL Compiler and mapped to a $0.18\mu m$ library. Our current implementation can only generate two-input NAND, NOR, AND, OR and XOR gates, as well as their variants where one of the inputs is inverted. In particular, if a three-input gate can be replaced by a two-input gate, our technique will find this restructuring opportunity. Although the netlists used in our experiments have multi-input cells, such as AOI, we do not need to break them down into smaller cells. Multiple gate cloning is not yet supported in the current implementation. As a result, area utilization remains roughly the same after resynthesis is performed.

Our experimental flow is summarized in Figure 8. Three iterations of the resynthesis are carried out for each run, and the maximum number of resynthesis attempts for each wire is limited to 1,000 to further reduce runtime. Characteristics of the benchmarks and our empirical results are summarized in Table 2, where the numbers are averages over three independent runs. Although we performed experiments using both Capo and QPlace, we only report the results produced by Capo due to space limitations. However, results from QPlace show similar trends. In addition to benchmarks in the table, our technique has shown similar performance on other netlists.

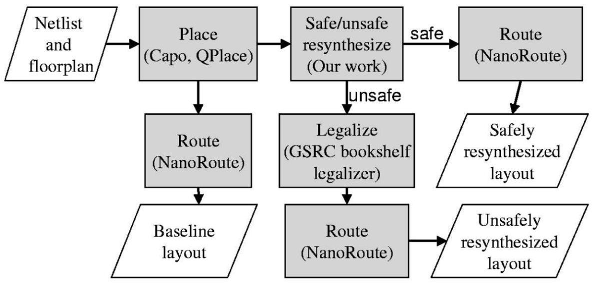
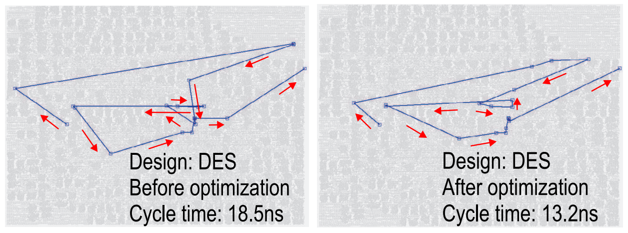
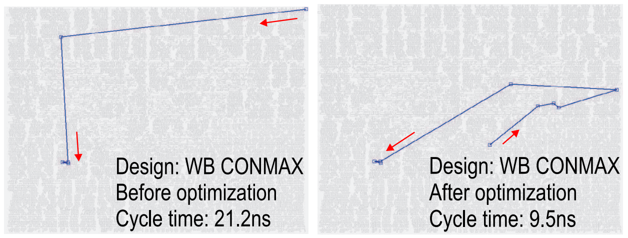

Netlist and floorplan
Place (Capo, QPlace)
Safe/unsafe resynthesize (Our work)
safe
Route (NanoRoute)
unsafe
Route (NanoRoute)
Legalize (GSRC bookshelf legalizer)
Safely resynthesized layout
Baseline layout
Route (NanoRoute)
Unsafely resynthesized layout

Figure 8: Flow chart of our resynthesis experiments.

From the results, we observe that our approach is effective in reducing the maximum delay for most of the benchmarks with minor increase in total wirelength, and sometimes it even results in wirelength reduction. The average cycle time improvement is 23% before routing and 5% after routing, while the routed wirelength and via counts increase by less than 0.2% on average. This is remarkable, compared to the results for logic cloning in [12] where wirelength increases by 2-28%. We can also observe from the results that the cycle time improvements measured using routed wirelength tend to be smaller than those estimated by the star model. The reason is that the star model tends to overestimate wire delay for large or long nets, therefore it may over-estimate improvements. As we will show in the last experiment, this difference tends to decrease when the average net length reduces. Aside from total wirelength, we also measure the average absolute change in routed lengths of individual nets and observe averages between 1% and 2% (the average for AES is 3.2%). This shows that congestion and routability are only slightly affected by our optimization.



(a)



(b)

Figure 9: Two optimization examples, one critical path per plot. Delay calculations are at the $0.18\mu m$ technology node. In (a) the critical path is shortened. In (b) an alternative source to generate the same signal is found. Although the new path is longer, the delay is actually reduced.

Our second experiment counts the number of resynthesizable wires in three of the benchmarks. In this experiment, resynthesis opportunities that reduce to gate relocation are excluded. Our results show that about 50% of the wires are resynthesizable using our techniques on average.

The impact of our techniques is illustrated in Figure 9: (a) the detour of the critical path is reduced, which also reduces the maximum delay; and (b) our resynthesis technique found another source to generate the same signal. Although the new path is longer, the delay is actually reduced.

In order to compare safe and unsafe optimizations, we apply our resynthesis technique in an unsafe way to compare the results with safe resynthesis. In particular, we allow gate overlap during resynthesis and rely on a legalizer to remove the overlaps. In our unsafe resynthesis, the location to place the resynthesized gate is determined by trying 400 sites near the desired coordinate regardless whether these sites are overlap-free or not. We used the legalizer provided by GSRC Bookshelf [24] in our experiments, and noticed that its runtime is typically short. In addition to performing safe and unsafe resynthesis separately, we combined both techniques by performing safe resynthesis after unsafe resynthesis in the hope of leveraging both their advantages. While this experiment does not cover all possible safe and unsafe techniques, we believe that it is representative. The results are summarized in Table 3, where the cycle time improvements estimated by the star timing model and the improvements measured using routed wirelength are both shown. Furthermore, we conducted the same experiments on the same designs with different percentages of whitespace to study the effects of available whitespace on the optimization results, and the

Benchmark	Cell count	Net count	Original				Resynthesized				Runtime (min)
			Cycle time (ps)	Routed cycle time (ps)	Routed wirelength (μm)	Via count	Cycle time improv.	Routed cycle time improv.	Wire-length increase	Additional vias	
SPI	3227	3277	3001	2947	238056	22661	4.45%	3.48%	0.21%	0.31%	1.0
DES_AREA	4881	5122	4551	4506	285701	30269	1.93%	1.56%	0.40%	0.57%	1.4
TV80	7161	7179	5642	5545	506243	50951	10.84%	9.94%	0.25%	-0.15%	2.1
SYSTEMCAES	7959	8220	9380	5007	783394	61878	31.23%	0.19%	0.09%	-0.17%	1.2
MEM_CTRL	11440	11560	5365	5220	1108789	90876	3.38%	3.42%	0.09%	0.16%	69.3
AC97	11855	11948	4291	2857	875680	94969	35.32%	0.00%	0.26%	0.11%	0.5
USB	12808	12968	5478	3570	1005478	87342	33.68%	1.67%	0.16%	-0.08%	1.5
PCI	16816	16990	4206	5252	1383941	131305	5.05%	0.00%	0.69%	0.08%	0.7
AES	20795	21055	4334	4266	1362586	131305	2.11%	1.78%	-0.14%	0.16%	2.6
WB_CONMAX	29034	30165	20302	8766	2751329	257614	61.96%	12.20%	-0.14%	-0.17%	6.5
Ethernet	46771	46891	94515	45363	7753231	427693	83.15%	35.18%	0.33%	-0.12%	13.6
DES_PERF	89341	98576	16065	11403	7656496	594781	18.10%	1.81%	-0.33%	0.04%	4.3
DMA	19118	19809	6743	5472	2055086	153406	13.19%	1.01%	0.08%	-0.38%	22.1
b14	8679	8716	6704	6367	703240	59178	7.38%	4.16%	0.08%	-0.35%	9.3
b15	12562	12605	10815	4821	1029036	94015	54.93%	0.93%	0.02%	0.07%	1.5
b17	37117	37167	9946	6299	3192066	280868	31.64%	0.00%	0.02%	0.00%	5.2
b18	92048	92214	12601	11304	7423151	686753	19.32%	15.33%	-0.08%	-0.10%	13.2
b22	28317	28354	7680	7343	2292342	192405	8.96%	6.69%	0.02%	-0.35%	16.5
s35932	7273	7599	7558	4167	770432	59242	28.75%	0.29%	0.10%	0.01%	0.3
s38417	8278	8309	2879	3203	562242	58937	2.44%	0.00%	0.05%	0.04%	1.6
Average							22.89%	4.98%	0.13%	-0.02%	

Table 2: Improvement achieved by our techniques: cycle times, wirelengths and via counts for unoptimized layouts are shown, followed by relative improvement due to resynthesis. "Cycle times" were estimated by the star timing model, while "routed cycle times" were measured using routed wirelength. The last column shows the program runtime. We only show results using Capo because of space limitations, but results produced with Cadence QPlace exhibit the same trends.

Benchmark	Estimated cycle time improvement				Routed cycle time improvement			Wirelength increase			Via count increase		
	Safe resynth.	Unsafe resynthesis		Unsafe + safe resynth.	Safe resynth.	Unsafe resynth.	Unsafe + safe resynth.	Safe resynth.	Unsafe resynth.	Unsafe + safe resynth.	Safe resynth.	Unsafe resynth.	Unsafe + safe resynth.
		Before legal.	After legal.										
AC97	35.32%	31.07%	31.10%	40.67%	0.00%	0.04%	1.93%	0.11%	0.26%	0.29%	0.11%	3.25%	2.84%
USB	33.68%	33.63%	33.12%	36.57%	1.67%	-0.06%	2.86%	0.10%	0.16%	-0.10%	-0.08%	2.55%	0.10%
PCI	5.05%	6.34%	6.21%	6.48%	0.00%	0.00%	-0.01%	0.11%	0.69%	0.41%	0.08%	8.14%	3.79%
AES	2.11%	3.47%	3.44%	3.36%	1.78%	3.13%	3.22%	0.09%	-0.14%	-0.12%	0.16%	2.10%	1.93%
WB_CONMAX	61.96%	60.81%	60.83%	61.13%	12.20%	11.45%	11.78%	0.87%	-0.14%	-0.02%	-0.17%	-0.23%	-0.55%
Ethernet	83.15%	84.10%	84.09%	84.61%	35.18%	53.46%	55.98%	0.05%	0.33%	0.42%	-0.12%	7.39%	7.31%
DES_PERF	18.10%	22.87%	22.78%	22.88%	1.81%	1.95%	1.86%	0.03%	-0.33%	-0.32%	0.04%	0.39%	0.38%
Average	34.20%	34.61%	34.51%	36.53%	7.52%	9.99%	11.09%	0.19%	0.12%	0.08%	0.00%	3.37%	2.26%

Table 3: A comparison of safe resynthesis, unsafe resynthesis, and unsafe followed by safe resynthesis. Relative improvements in cycle time and increases in wirelength are shown. Unsafe optimizations allow cell overlaps, and legalization is required to remove the overlaps. The GSRC bookshelf provides a legalizer, which is used in this experiment. We only show results for the largest seven benchmarks from the OpenCores suite due to space limitation; however, the remaining benchmarks show the same trends.

results are summarized in Table 4.

The routed cycle time improvement in Table 4 shows that when the percentage of whitespace is high, unsafe resynthesis performs better than safe resynthesis. The reason is that unsafe resynthesis allows placing a cell on the best location regardless whether the location is occupied, and the legalization process only needs to change other placements slightly due to abundant whitespace around. As a result, most circuit parameters are still preserved. However, when the percentage of whitespace is low, safe resynthesis clearly leads to better cycle time improvement after routing. The reason is that the legalization process required by unsafe resynthesis may trigger a chain of placement changes due to insufficient whitespace around, and these changes may alter circuit parameters considerably. For example, the via count increase is much more significant in unsafe resynthesis than in safe resynthesis. As a result, although the estimated cycle time improvement of unsafe resynthesis is only slightly worse than safe resynthesis, the routed cycle time improvement becomes much worse. This supports our claim that the effects of unsafe optimizations are more difficult to evaluate reliably, especially when the percentage of whitespace is low. Furthermore, after unsafe resynthesis is performed, the optimization objective may actually deteriorate during legalization and routing. For example, in the USB benchmark shown in Table 3, the cycle time after unsafe resynthesis is worse than the unoptimized circuit. To obtain the greatest improvement, the advantages of both safe and unsafe techniques should be leveraged. As suggested by the results in Table 4, this goal can be achieved by applying safe resynthesis after unsafe resynthesis.

The comparison between estimated and routed cycle time improvement shows an interesting phenomenon: the largest routed cycle time improvement occurs at 10% whitespace; however, the largest estimated cycle time improvement occurs at 30% whitespace. One possible explanation is that smaller whitespace reduces the overall circuit size, which also reduces average net length. Since the star timing model is more accurate when nets are smaller or shorter, our resynthesis is able to perform more accurate optimizations, which lead to better routed cycle time improvement at 10%

Percentage of whitespace	Estimated cycle time improvement				Routed cycle time improvement			Wirelength increase			Via count increase		
	Safe resynth.	Unsafe resynthesis		Unsafe + safe resynth.	Safe resynth.	Unsafe resynth.	Unsafe + safe resynth.	Safe resynth.	Unsafe resynth.	Unsafe + safe resynth.	Safe resynth.	Unsafe resynth.	Unsafe + safe resynth.
		Before legal.	After legal.										
30%	34.20%	34.61%	34.51%	36.53%	7.52%	9.99%	11.09%	0.19%	0.12%	0.08%	0.00%	3.37%	2.26%
10%	26.27%	25.80%	25.62%	28.11%	14.75%	13.28%	13.84%	0.10%	0.03%	0.07%	0.04%	0.38%	0.84%
3%	21.64%	21.48%	21.57%	24.10%	4.72%	1.14%	6.70%	0.15%	0.05%	0.04%	0.41%	1.12%	0.86%

Table 4: A comparison of cycle time, wirelength and via count for layouts with different percentage of whitespace.

whitespace.[2] The discrepancy between estimated and actual (post-routing) cycle-time improvement suggests that to evaluate physical synthesis techniques reliably, circuit timing after routing should always be reported.

6. CONCLUSIONS

In this paper we proposed and evaluated the concepts of physical safeness and logical soundness to analyze circuit transformations in physical synthesis. Logically sound techniques only make changes to the circuit that can be easily verified, and physically safe techniques only modify the circuit in a way that the effect is immediately and reliably measurable. Safe and sound techniques are usually preferable because the optimizations are more stable, more reliable and more predictable. However, the majority of safe and sound techniques known before our work are limited in their optimization power because of the small number of transformations allowed.

To overcome the limitations of traditional physically safe techniques, we proposed a new resynthesis algorithm that is powerful, safe and sound. It utilizes simulation to generate a signature for each wire, and the wires on the critical path are resynthesized using new gates with their inputs selected from compatible wires. On-line equivalence checking is then carried out to verify the correctness of logic transformations. Since we allow inserting additional gates only when unused space is available, the original gate locations are preserved. At the same time, the global search for candidate wires gives our technique the power to find longer range optimizations. Experimental results show that our technique can improve timing considerably with little risk to other circuit parameters, such as wirelength and via count. Our technique can be applied in practically any design flow with little risk, but the cycle-timing improvement may be different from what we report. In addition, our comparison between safe and unsafe optimizations further highlights the importance of developing more powerful physically safe techniques, or methods to apply unsafe transformations in a safe way with minimal loss in strength.

Our results can be further improved with physically safe net buffering and gate sizing. Less obviously, these steps may need to be applied both before and after our techniques: first, to focus our techniques on the right critical paths, and after resynthesis buffer the remaining critical paths.

Our ongoing work suggests that efficient search pruning mechanisms exist for many gates with three or more inputs. These extensions promise to make our technique even more powerful and efficient.

[2]We are currently investigating the use of Steiner-tree packages, such as FLUTE, to improve the accuracy of timing analysis during resynthesis.

7. REFERENCES

[1] A. H. 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] U. Brenner, A. Pauli and J. Vygen, "Almost Optimum Placement Legalization by Minimum Cost Flow and Dynamic Programming", *ISPD'04*, pp. 2-9.

[3] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Can Recursive Bisection Alone Produce Routable Placements?", *DAC'00*, pp. 693-698.

[4] C. Changfan, Y. C. Hsu and F. S. Tsai, "Timing Optimization on Routed Designs with Incremental Placement and Routing Characterization", *IEEE Trans. on CAD*, Feb. 2000, pp. 188-196.

[5] C. W. Chang et al., "Fast Postplacement Optimization Using Functional Symmetries", *IEEE Trans. on CAD*, Jan. 2004, pp. 102-118.

[6] S. C. Chang, L. P. P. P. van Ginneken and M. Marek-Sadowska, "Circuit Optimization by Rewiring", *IEEE Trans. on Computers*, Sep. 1999, pp. 962-969.

[7] W. Donath et al., 'Transformational Placement and Synthesis", *DATE'00*, pp. 194-201.

[8] S.-Y. Huang, K.-C. Chen and K.-T. Cheng, "AutoFix: A Hybrid Tool for Automatic Logic Rectification", *IEEE TCAD*, pp. 1376-1384, Sep. 1999.

[9] C.-C. Lin, K.-C. Chen and M. Marek-Sadowska, "Logic Synthesis for Engineering Change", *IEEE TCAD*, pp.282-202, Mar. 1999.

[10] T. Lin and C. A. Mead, "Signal Delay in General RC Networks", *IEEE Trans. on TCAD*, Oct. 1984, pp. 331-349.

[11] N. Eén and N. Sörensson, "An Extensible SAT-solver", *Theory and Applications of Satisfiability Testing, SAT*, 2003, pp. 502-518.

[12] M. Hrkic, J. Lillis and G. Beraudo, "An Approach to Placement-Coupled Logic Replication", *DAC'04*, pp. 711-716.

[13] A. B. Kahng and Q. Wang, "Implementation and Extensibility of an Analytic Placer", *IEEE Trans. on CAD*, May 2005, pp. 734-747.

[14] L. N. Kannan, P. R. Suaris and H. G. Fang, "A Methodology and Algorithms for Post-Placement Delay Optimization", *DAC'94*, pp. 327-332.

[15] V. N. Kravets and P. Kudva, "Implicit Enumeration of Structural Changes in Circuit Optimization", *DAC'04*, pp. 438-441.

[16] C. Li, C-K. Koh and P. H. Madden, "Floorplan Management: Incremental Placement for Gate Sizing and Buffer Insertion", *ASPDAC'05*, pp. 349-354.

[17] A. Lu, H. Eisenmann, G. Stenz and F. M. Johannes, "Combining Technology Mapping with Post-Placement Resynthesis for Performance Optimization", *ICCD'98*, pp. 616-621.

[18] A. Mischenko and R. K. Brayton, "SAT-Based Complete Don't-Care Computation for Network Optimization", *DATE'05*, pp. 412-417.

[19] T. Luo, H. Ren, C. J. Alpert and D. Pan, "Computational Geometry Based Placement Migration", *ICCAD'05*, pp. 41-47.

[20] B. M. Riess and G. G. Ettelt, "Speed: Fast and Efficient Timing Driven Placement", *ISCAS'95*, pp. 377-380.

[21] H. Vaishnav, C. K. Lee and M. Pedram, "Post-Layout Circuit Speed-up by Event Elimination", *ICCD'97*, pp. 211-216.

[22] Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification, Release 51205. `http://www-cad.eecs.berkeley.edu/~alanmi/abc/`

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

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