Ultra-Fast Interconnect Driven Cell Cloning For Minimizing Critical Path Delay

Zhuo Li[1], David A. Papa[2,1], Charles J. Alpert[1], Shiyan Hu[3],
Weiping Shi[4], Cliff Sze[1], Ying Zhou[4,1]

[1] IBM Austin Research Laboratory, 11501 Burnet Road, Austin, TX 78758.
[2] Dept. of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109.
[3] Dept. of Electrical and Computer Engineering, Michigan Technological University, Houghton, MI 49931.
[4] Dept. of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77840.
{ lizhuo, dapapa, alpert, csze, nancyz }@us.ibm.com, iamyou@umich.edu, shiyan@mtu.edu, wshi@ece.tamu.edu

ABSTRACT

In a complete physical synthesis flow, optimization transforms, that can improve the timing on critical paths that are already well-optimized by a series of powerful transforms (timing driven placement, buffering and gate sizing) are invaluable. Finding such a transform is quite challenging, to say nothing of efficiency. This work explores innovative cloning (gate duplication) techniques to improve timing-closure in a physical synthesis environment.

With a buffer-aware interconnect timing model, new polynomial-time optimal algorithms are proposed for timing-driven cloning, including both finding optimal sink partitions (identifying the fan-outs) for the original and the duplicated gates, as well as physical locations for both gates. In particular, we present an $O(m)$-time optimal algorithm to minimize the worst slack if the original gate is movable, and an $O(m \log m)$ algorithm if the original gate is fixed, where m is the number of fan-outs. To the best of our knowledge, this work is the first one considering the timing-driven cloning problem under a buffer-aware interconnect delay model.

For a hundred testcases in 45nm technology node, we demonstrate significant timing improvement due to our cloning techniques as compared to other existing timing-optimization transforms. Extensions to other factors, such as wirelength, FOM and placement obstacles are further discussed.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids—*Placement and Routing*

General Terms

Algorithms, Design, Performance, Theory

Keywords

Gate Duplication, Physical Synthesis, Timing-Driven Placement

1. INTRODUCTION

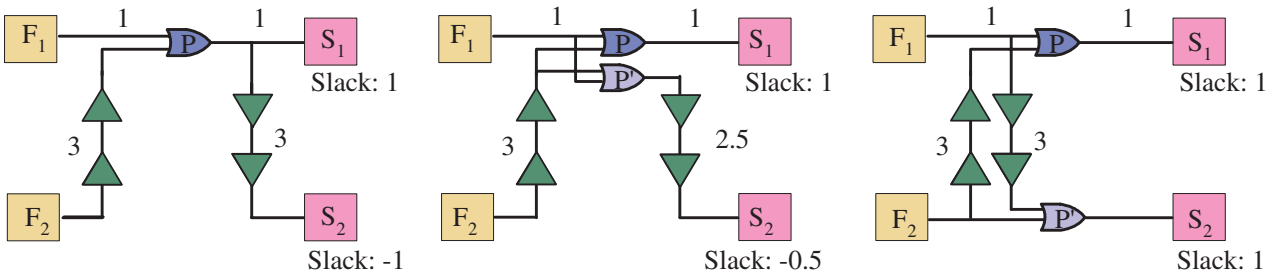
Physical synthesis is a complex process that combines physical design with synthesis to perform design closure. As described in [7], physical synthesis typically consists of several stages including placement, legalization, critical path optimization, etc. Among these stages, the critical path optimization stage is particularly important. It takes a design that is legally placed and initially optimized for timing, and restructures critical paths by applying a multitude of different transforms, such as gate sizing, V_{th} tuning, and buffering. It is often not difficult to optimize timing on some nets early in a physical synthesis flow. However, it is usually more challenging to improve timing if the nets have been optimized by a series of powerful transforms in a physical synthesis flow. This imposes great challenges in critical path optimization.

We believe that fixing all types of timing problems will require some new transforms to be developed, because each transform may fix certain problematic structures. In this paper, we design several highly efficient cloning techniques, also known as cell replication, to improve delays along critical paths. Cloning is not a new synthesis optimization. For example, Brglez [8] and Hwang et al. [10] use cloning as a mechanism to reduce net cut during partitioning, and [11, 13] study cloned gate placement in the FPGA domain. Since cloning helps in reducing the total capacitance loading of a high-fanout net, many works [16, 14, 12] focus on technology independent delay optimization. Under the load-dependent gate delay model with zero-wire delay, the authors of [16] prove that the cloning problem is NP-complete. Under the same delay model, the authors of [17] propose a right-to-left cloning scheme to improve the timing of a technology-mapped circuit. Due to the computational complexity of the problem, heuristics are often proposed to speed up the technique. However, all of these works neglect key features of the problem: the placement of the duplicated gate and interconnect delay. Thus, these models can be used in the logic synthesis stage of design but will be less accurate during the core stages of a physical synthesis flow.

For today's deep sub-micron technologies, previous cloning algorithms, which ignore wire delay, buffering and placement, are largely ineffective in the context of critical path optimization. This is explained in part by interconnect scaling, which has necessitated that buffers be inserted at shorter intervals to overcome wire resistance. Only very short wires will not require buffers [1]. Consequently, when one wants to apply cloning to improve path delay, buffers that have been inserted previously limit the scope of cloning for timing improvement. To make cloning effective, one must account for buffers by considering only non-buffer sinks, and re-buffering the resulting subcircuit.

To the best of our knowledge, the only work which handles both cloning and buffer insertion in the placement stage is BufDup [15]. Unfortunately, they consider cloning and buffer insertion separately. In addition, BufDup uses a timing-oblivious, simple k-means based clustering algorithm to partition the fanout gates. It does contain a timing-driven post-processing step, but it can only be used to balance the capacitance loading of the two partitions and is not designed to improve timing. In contrast to [15], our cloning is based on a linear-delay model [2, 3] with the knowledge that buffered interconnect delay is linearly proportional to its length. This model handles simultaneous buffering and cloning in an abstract and unified way. Adoption of such a delay model also helps to reduce the

ISPD'10, March 14–17, 2010, San Francisco, California, USA.



(a) Original Subcircuit. (b) New subcircuit after cloning leaving buffering intact. (c) New subcircuit after cloning considering buffering.

Figure 1: Example of cloning. The arrival time of F_1 and F_2 are 0. The required arrival time of S_1 and S_2 are 5.

complexity of the gate cloning problem. This work reveals that cloning with buffer-aware linear-delay model can be accomplished very efficiently in polynomial time. Thus, such a problem is not NP-complete any more.

Recently, there are some works on simultaneous timing-driven gate placement and buffering. RUMBLE [9] uses linear-delay buffering model and linear programming techniques to solve timing-driven latch and gate placement problem considering practical constraints. Pyramid [18] uses computational geometry techniques to efficiently solve one gate placement problem with a similar delay model. Note that the timing-driven gate placement problem is subsumed by the timing-driven gate cloning problem, since a fixed sink partitioning reduces the cloning problem to a gate placement problem. Thus, the cloning problem is complicated by the need to find sink partitions and gate placements simultaneously.

An example of simultaneous cloning and buffering is shown in Figure 1. The arrival time of F_1 and F_2 are 0, and the required arrival time of S_1 and S_2 are 5. Consider the instance in Figure 1(a) where we consider cloning gate P. There are two sinks S_1 and S_2 with slacks +1 and -1. The delays from fan-ins F_1 and F_2 to P are 1 and 3 respectively, as are the delays from P to S_1 and S_2, including the delay of buffers and wires along the path. If we clone P to P' while leaving the original buffer trees intact, we may get the result shown in Figure 1(b) in which P' is placed very close to P, and the slack only improves to -0.5. Here the new location of P' is restricted by the buffers that must drive both P and P'. However, if one restructures the buffering solution to eliminate this constraint, one can obtain the far superior solution in Figure 1(c) which increases both slacks to +1 and obtains the physically shortest possible paths from F_1 and F_2 to S_1 and S_2. This example suggests that one must consider buffering and cloning together to effectively reduce delay.

Timing-driven buffering alone can be computationally expensive when used excessively [4]. It is also difficult to use it to derive any theoretical guidance for simultaneous cloning and buffering. To be most accurate, one should explore all possible partitionings of sinks for each net, find gate placements (i.e, with the technique in [9]), re-buffer with dynamic programming, and legalize the design. The whole process is highly expensive when applying to modern designs with hundreds of thousands of nets. It may also waste the majority of its runtime, because in many cases the new solution may be worse than the old solution, and will therefore be retracted.

Unlike the above approach, we use abstract timing models and build a theoretical guide on top of them. In our approach, the effect of buffering is modeled by a linear-delay model, similar to [9, 18]. Our algorithms guarantee optimality under this delay model, and can also be used as a filter to identify a group of critical gates that may benefit from cloning. Even if our solution does not fix all timing problems, one can still apply more accurate gate placement techniques (such as [9]) based on our sink partitioning and re-buffer on a small group of nets. In that way, success rate and the total turn-around-time will be improved.

The main contributions of this work are summarized as follows.

- We propose several polynomial-time optimal algorithms for simultaneous timing-driven cloning and buffering under a linear-delay model. Our algorithms will "see through" buffer trees in the original circuit.
- When the original gate is a movable object, an $O(m)$-time algorithm to compute the optimal cloning that maximizes worst slack is proposed, where m is the number of fanouts.
- When the original gate is a fixed object, an $O(m \log m)$-time algorithm to compute the optimal cloning is proposed.

Note that we assume the load-based cloning techniques have been used in the logic synthesis or early design stage, and we will not focus on the problem of reducing capacitive load. Also, buffering should have processed all high-fanout nets before the cloning we propose. The techniques in this paper are designed primarily for gates driving substantial interconnect delay (medium to long nets).

2. PRELIMINARIES

We outline our problem formulation as follows.

2.1 Linear Buffered-Path Delay Model

Timing driven buffering has become indispensable in timing closure [1, 6, 3]. To capture the effect of buffering in the cloning problem without applying dynamic programming based buffer insertion, it is best to use a buffer-aware interconnect delay model. Like previous work [2, 3, 9], a linear-delay model is used in this paper. In a linear-delay model, the delay along an optimally buffered interconnect of length l is given by $delay(l) = \tau \cdot l$, where τ is a technology dependent constant. In general, τ depends on the buffer library size and the input slew. Therefore, for each technology, we perform optimal buffering on a long 2-pin net with the saturated input slew (which is the slew after several stages of optimal buffering). Subsequently, $\tau = delay(wire)/length(wire)$. For a multi-pin net, we break it into a set of 2-pin driver-to-sink edges and use the linear-delay model for each 2-pin edge. This simplification is generally valid since non-critical paths can be decoupled from the critical paths. With this model, the computationally expensive Steiner tree construction for multi-pin nets can also be avoided.

Empirical results in [3] indicate that linear-delay model is accurate with error within 0.5% when at least one buffer is inserted along the net. Results from [9] also show a 97% correlation between the linear-delay model and an industrial timing analysis tool.

2.2 Problem Formulation

The subcircuit for the cloning problem is a directed graph $G = (V, E)$, where $V = P \cup F \cup S$, and $E = (F \times P) \cup (P \times S)$. Vertex P is the *target* gate to be duplicated, F is the set of *fan-in* gates that drive P with size n, and S is the set of *fan-out* gates that P drives with size m.[1] Every gate $g \in V$ is a logic gate performing certain logic functions, such as AND, OR, XOR but not buffers or inverters, and is associated with physical coordinates $(X(g), Y(g))$. If there are any buffers/inverters in the circuit that

[1]Without loss of generality, we assume n and m are in the same order for simplicity of the complexity analysis.

are fan-ins or fan-outs of P, we will look through them to find the first non-repeater logic gate. Each fan-out gate $S_i \in S$, is associated with required arrival time $RAT(S_i)$ at its input pin, and each fan-in gate $F_i \in F$ is associated with arrival time $AT(F_i)$ at its output pin.

The location of each gate in S and F can not be changed in our problem formulation, and we refer them as "fixed" (not movable) gates. Note that these gates may be allowed to move in other transforms (e.g., legalization after cloning) but their locations are constrained for cloning itself to simplify the analysis. There are also cases that they are "fixed" by designers who want to keep certain gates in specified locations, or in the late stage of design flow, one prefers minimal perturbation to the design for the stability. Gate P may be movable or fixed.

After cloning, we create a duplicated gate for P, denoted by P'. Finding a location for P' is one objective of this work. The graph G becomes $G' = (V', E')$, where $V' = P \cup P' \cup F \cup S_P \cup S_{P'}$, and $E' = (F \times P) \cup (P \times S_P) \cup (F \times P') \cup (P' \times S_{P'})$. S_P is the set of fan-out gates that P drives, and $S_{P'}$ is the set of fan-out gates that P' drives. In the G', each fan-in gate F_i is also connected to the duplicated gate P', but fan-out gates S are divided into two disjoint sets S_P and $S_{P'}$. We refer to the division of S into S_P and $S_{P'}$ as a *sink partitioning*, and S_P and $S_{P'}$ as *sink partitions*. All other notations in G are valid for G'.

For each edge $e = (g_1, g_2)$ in G and G', the Manhattan length of edge e is $dis(e) = |X(g_1) - X(g_2)| + |Y(g_1) - Y(g_2)|$, where $g_1 \in F \cup P \cup P'$, and $g_2 \in P \cup P' \cup S$. Recall that all multi-pin nets will be broken into 2-pin nets with a linear-delay model. For each edge e, edge delay is $D(g_1, g_2) = \tau \cdot dis(e)$. Each edge is also referred as a "net" where g_1 is the driver, and g_2 is the sink.

For gates P and P', we denote their gate delays by $D(P)$ and $D(P')$, respectively. In this paper, we treat these gate delays as constants. This is fairly accurate since we maintain that buffering must be performed with cloning, and after that, the load of P and P' will remain almost the same. Gate sizing can be performed before or after cloning if the original driver is too weak or strong, which will further control the error of this constant gate delay model.

For a gate g in $P \cup P'$, the required arrival time at the output pin of g is $RAT(g) = \min_{S_i \in S}\{RAT(S_i) - D(P, S_i)\}$, where S is the set of its fan-out gates. The arrival time at the output pin of g is $AT(g) = \max_{F_i \in F}\{AT(F_i) + D(F_i, P)\} + D(g)$, where F is the set of its fan-in gates. The slack of a gate g is $Q(g) = RAT(g) - AT(g)$.

Without loss of generality, we set gate delays $D(P)$ and $D(P')$ to zero in the following discussion to simplify the analysis. All algorithms are still valid as long as gate delays are constants.

It is easy to see that the slack of P and P' determines the slack of the subcircuit G and G'. For subcircuit G, we have $Q(G) = Q(P)$, and for G', we have $Q(G') = \min\{Q(P), Q(P')\}$. For each edge (net) $e = (g_1, g_2) \in E \cup E'$, we also define the slack of e as

$$Q(e) = RAT(g_2) - D(g_1, g_2) - AT(g_1). \quad (1)$$

Note that $Q(G') = \min_{e \in E'} Q(e)$ and $Q(G) = \min_{e \in E} Q(e)$ too.

Cloning Problem: Given a graph $\boldsymbol{G} = (V, E)$, where P is the target gate, RAT for all fan-outs S_i, AT for all fan-ins F_i, and a linear-delay constant τ, create a cloned gate P' for P, which obtains a new graph G', find S_P, $S_{P'}$ and locations of P' and P (if P is movable) such that $Q(G')$ is maximized.

In contrast to most previous work ([16, 12]) which only identifies the partitions S_P and $S_{P'}$, our algorithms will not only provide a partitioning of fan-outs, but also the placement of P and P'. In addition, if the solution is worse than the original subcircuit, no cloning will be performed.

3. FAST TIMING-DRIVEN GATE CLONING

In this section, we present our algorithms for the cases where P is movable and P is fixed. We start with several new concepts.

3.1 Best Region and Best Arrival Arc Segment

Recall that the set of fan-in gates F is connected to both of original gate P and the duplicate gate P' after cloning. The set of fan-out gates S is split into two disjoint sets (partitions) S_P and $S_{P'}$ such that $S = S_P \cup S_{P'}$.

Let treat the whole circuit image as a 2D plane H. For each fan-in gate F_i in F, the arrival time at any point v in the H is $AT(F_i) + D(F_i, v)$, and $D(F_i, v) = \tau \cdot dis(F_i, v)$. Therefore, if we place a gate at v with the fan-in set F, according to the static timing analysis rule, we have

$$AT(v) = \max_{F_i \in F}\{AT(F_i) + D(F_i, v)\}.$$

Clearly, $AT(v)$ a 2D function, where the variable is the locations of v. Define the set of points minimizing $AT(v)$ on the plane H as

$$K(F) = \{a \in H | AT(a) \leq \min_{\forall v \in H} AT(v)\}.$$

So $K(F)$ is the set of points which have minimum arrival time for all fan-ins. In the following, we will show that $K(F)$ is either a single point or a line segment with 45° degree slope. Refer to Fig. 2 and 3 for examples of $K(F)$.

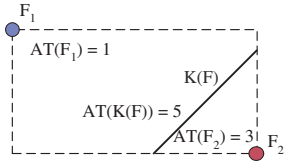


Figure 2: An example of *arrival time arc* $K(F)$**.** $dis(F_1, K(F)) = 4$**,** $dis(F_2, K(F)) = 2$**,** $\tau = 1$**.**

If there is only a single fan-in, it is obvious that $K(F)$ is the point of this fan-in itself, with $AT = AT(F)$. If there are two fan-ins F_1 and F_2, then there are 3 cases,

$$K(F) = \begin{cases} \{a \in H | AT(F_1) + D(F_1, a) = AT(F_2) + D(F_2, a)\}, \\ \text{if } max(AT(F_1), AT(F_2)) - min(AT(F_1), AT(F_2)) \leq D(F_1, F_2); \\ v_{F_1}, \text{if } AT(F_1) > AT(F_2) + D(F_1, F_2); \\ v_{F_2}, \text{if } AT(F_2) > AT(F_1) + D(F_1, F_2); \end{cases}$$

Here v_{F_i} refers to the location of fan-in gate F_i. In the first case, where the difference between the arrival time at F_1 and F_2 is smaller than $D(F_1, F_2)$, $K(F)$ is a *Manhattan Arc*, which is a segment with slope 45° or -45° in the bounding box of F_1 and F_2. This slope will always be 45° or -45° as long as technology dependent coefficient τ is a constant. Note that when F_1 and F_2 are either horizontally or vertically aligned, $K(F)$ is a point, which is a degenerate case of *Manhattan Arc*. For the other two cases, where one of the arrival times dominates the other one, $K(F)$ is at the location of one of the fan-in gates.

Denote $K(F_i)$ as the set of points minimizing $AT(v)$ for fan-ins $F_1, \ldots, F_i$. If we have more than two fan-ins, we will first form $K(F_2)$ for F_1 and F_2, and then merge $K(F_2)$ with F_3 to get $K(F_3)$. $K(F_3)$ will be another *Manhattan Arc* or a single point, depending on the relationship among $AT(K(F_2))$, $AT(F_3)$, and $dis(K(F_2), F_3)$ which is the shortest Manhattan distance between F_3 and $K(F_2)$. Repeating this procedure for all fan-ins, we can find the final $K(F)$. This bottom-up merging process is very similar to the Deferred-Merge Embedding (DME) algorithm in clock tree construction [19] though the formula there is to get a zero skew arc. With a similar procedure to the one shown in [19], it is not hard to prove that $K(F)$ is always a *Manhattan Arc* or a single point, and our merging process guarantees the optimality that $K(F)$ will have minimum arrival time for all fan-ins. The detailed proof is omitted here due to space limitations.

In the rest of paper, we denote $K(F)$ as *arrival time arc* (a point can be considered a degenerate case of an arc), and the arrival time

on this arc as $AT(K(F))$. An example of $K(F)$ for two fan-ins is shown in Fig. 2.

Similarly, we can find $K(S)$, the set of points maximizing $RAT(v)$ on the plane H, for the set of fan-outs S. We denote $K(S)$ as *required arrival time arc* and the required arrival time on this arc as $RAT(K(S))$. Refer to Fig. 3 for a $K(S)$. With similar procedure in [19], it is also easy to prove that computation of $K(F)$ and $K(S)$ takes $O(m)$ time assuming m and n are in the same order. Also, given any order of fan-ins and fan-outs, denote $K(F_i)$ as *arrival time arc* for the set of gates $F_1, \ldots, F_i$, and $K(S_i)$ as *require arrival time arc* for the set of gates $S_1, \ldots, S_i$. We can compute all values of $K(F_i)$ and $K(S_i)$ in $O(m)$ time with dynamic programming by incremental updating and storing all arcs. Therefore, the amortized cost for computing each $K(F_i)$ and $K(S_i)$ is only $O(1)$. We introduce the following Lemma to be used in Section 3.3.

LEMMA 1. *It takes $O(m)$ time to compute $K(F)$, $K(S)$, and all values of $K(F_i)$ and $K(S_i)$. The amortized cost of computing each $K(F_i)$ and $K(S_i)$ is $O(1)$.*

The next Lemma states that for any point in the plane, its arrival time (required arrival time) can also be represented by the arrival time at the $K(F)$ ($K(S)$) and the shortest Manhattan distance between the point and $K(F)$ ($K(S)$). The proof is straightforward from the merging process and the fact that computation of arrival time (or required arrival time) is a max (min) operation.

LEMMA 2. *For any point v in the plane H, $AT(v) = AT(K(F)) + \tau \cdot dis(K(F), v)$, and $RAT(v) = RAT(K(S)) - \tau \cdot dis(K(S), v)$.*

Now we will introduce the concept of *Best Region*. Define $Z(F, S)$ as a region formed by $K(F)$ and $K(S)$,

$$Z(F,S) = \{v \in H | dis(v, K(F)) + dis(v, K(S)) = dis(K(F), K(S))\},$$

where $dis(v, K(F))$, $dis(v, K(S))$ and $dis(K(F), K(S))$ are the shortest distance between a point or *Manhattan Arc* to another point or *Manhattan Arc*. When $K(F)$ and $K(S)$ are both single points, then $Z(F, S)$ is the rectangle bounding box formed by two points. Other examples of region $Z(F, S)$ for different shapes of $K(F)$ and $K(S)$ are shown in Fig. 3.

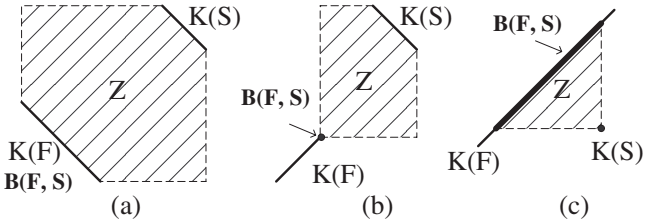


Figure 3: Examples of region Z. (a) Both $K(F)$ and $K(S)$ are -45° line segments; (b) $K(F)$ is a 45° line segment and $K(S)$ is a -45° line segment; (c) $K(F)$ is a 45° line segment and $K(S)$ is a single point.

It is easy to show that for any point v outside region Z, it will have $dis(v, K(F)) + dis(v, K(S)) > dis(K(F), K(S))$, and no point exists in H with $dis(v, K(F)) + dis(v, K(S)) < dis(K(F), K(S))$. Also, all points in region Z will have the same slack

$$Q(Z(F,S)) = RAT(K(F)) - AT(K(S)) - \tau \cdot dis(K(F), K(S)).$$

The following theorem states the property of region $Z(F, S)$.

THEOREM 1. *Given the location of fan-in gates F, fan-out gates S, if the gate P is placed in any point inside region $Z(F, S)$ formed with the above process, it achieves the maximum slack.*

PROOF. If P is located outside of region Z with a bigger slack, then based on Lemma 2 we have

$$\begin{aligned} Q(P) &= RAT(P) - AT(P) \\ &= RAT(K(S)) - AT(K(F)) - \tau \cdot (dis(K(F), v) + dis(K(S), v)) \\ &< RAT(K(S)) - AT(K(F)) - \tau \cdot dis(K(F), K(S)) \\ &< Q(Z(F,S)), \end{aligned}$$

which contradicts the assumption. □

We refer the region Z as *Best Region* since it gives the region with best slack. We also refer the above procedure to find Z as *FindBestRegion*. The complexity of *FindBestRegion* is $O(m)$ since the only cost is to compute $K(F)$ and $K(S)$.

THEOREM 2. *FindBestRegion finds Best Region Z in $O(m)$ time for a net with m fan-outs.*

Now we introduce the concept of *Best Arrival Time Arc*. We define *Best Arrival Time Arc* $B(F, S)$ as the intersection of $K(F)$ and $Z(F, S)$. $B(F, S)$ is part of $K(F)$, while the detailed shape is decided by $K(F)$ and $K(S)$. In examples illustrated in Fig. 3, $B(F, S)$ is $K(F)$ in Fig. 3 (a), a single point in Fig. 3 (b), and a partial segment of $K(F)$ in Fig. 3 (c). From Theorem 1, we know that every point on $B(F, S)$ still achieves the maximum slack. Define the slack on $B(F, S)$ as $Q(B(F, S))$, and we have $Q(B(F, S)) = Q(Z(F, S))$. In next section, the concept of $B(F, S)$ is used to design our algorithm.

3.2 When the Original Gate is Movable

In this section, we are going to present the algorithm when the original gate P is movable. The main idea is to limit the solution search space to $K(F)$, and then find *Best Arrival Time Arc* $B(F, S_P)$ and $B(F, S_{P'})$ efficiently by dividing plane into 6 regions (Fig. 4) and using the unique properties of fan-out slack of each region to find the best locations of P and P'.

When P is movable, we are free to place both P and P'. From Section 3.1, given a partitioning S_P and $S_{P'}$, we can simply place P and P' on the best arrival time arc $B(F, S_P)$ and $B(F, S_{P'})$ to achieve the optimal solution. The goal is to find the best partitioning S_P and $S_{P'}$, which gives best slack among all possible partitionings. However,, without the knowledge of partitionings, $B(F, S_P)$ and $B(F, S_{P'})$ are not known to us.

An important observation is that both arcs need to be on $K(F)$, which is known. Therefore, rather than trying all partitionings, we will limit our solution search space for both P and P' on $K(F)$, which enables the efficient computation as well. This is the key to drive the partitioning and computing *best arrival time arcs*. By limiting P and P' on $K(F)$, we have $AT(P) = AT(P') = AT(K(F))$.

LEMMA 3. *If arrival time arc $K(F)$ is a single point, no cloning is needed.*

PROOF. If $K(F)$ is a single point, then $B(F, S_P) = B(F, S_{P'})$ is a single point. One can place P at $B(F, S_P)$ and achieve the maximum worst slack without cloning. □

As stated in Section 1, the assumption here is that the cap load of the gate is reasonable and no capacitance-based cloning needed.

Now we discuss the case when $K(F)$ is a *Manhattan Arc*. Since both P and P' are movable, we use P as the example in the following discussion. Without loss of generality, we assume $K(F)$ is a 45° line segment (all analysis for the -45° case are similar), as shown in the Fig. 4.

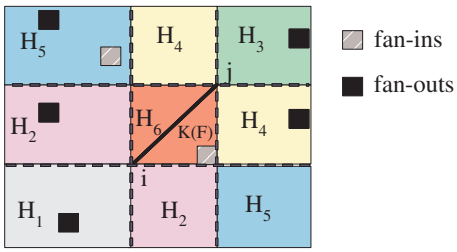


Figure 4: The region division for the *arrival time arc* $K(F)$.

Denote the lower left endpoint and upper right endpoint of $K(F)$ as i and j, respectively. The plane H is divided in 6 regions, H_1, H_2, H_3, H_4, H_5, and H_6, based on i and j as shown in the same figure. Please note that, some fan-ins may be located outside region H_6 as shown in Fig. 4 since the arrival time of all fan-in gates could be different. One can also refer Fig. 2 as an example.

For any fan-out gate S_i in each region, we then analyze the relation between the slack of the edge (net) $e = (P, S_i)$ and the location of P on $K(F)$. Note that $Q(e)$ is purely determined by $RAT(S_i) - D(P, S_i)$ since $AT(P) = AT(K(F))$.

Fig. 5 shows the typical curves of edge slack vs. location of P on $K(F)$ for each region. The horizontal coordinate is the distance along the line segment from point i to point j. For example, if a fan-out is located in region H_1, then when P is located at i, we will get the maximum slack for this net, and when P is located at j, we will get the minimum slack for this net. When a fan-out is located in H_2, then when P is located from i to a certain point on $K(F)$, the slack will be same, and begins to decrease when P is moving towards j.

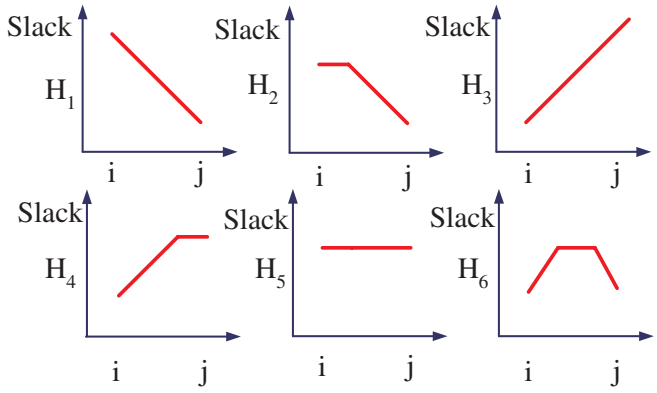


Figure 5: The slack vs $K(F)$ curves for each region.

If we intersect all slack curves in the set S_P ($S_{P'}$), a minimum slack curve can be generated by taking the minimum slack among all slack curves for each point on $K(F)$. The segment with maximum slack on this new curve will be the best slack we can achieve for this set of fan-outs. This segment is either a level segment or a single point from Fig. 5. Let refer this segment as *Best Slack Segment*. Then the corresponding segment for *Best Slack Segment* on $K(F)$ is $B(F, S_P)$ ($B(F, S_{P'})$). Clearly, we seek the partitioning with the greatest *Best Slack Segment*, and if we find such *Best Slack Segment*, $B(F, S_P)$ and $B(F, S_{P'})$ are found too. Two examples of *Best Slack Segment* are shown in Fig. 6.

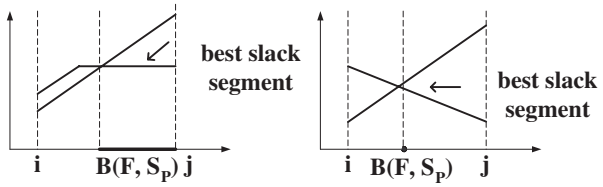


Figure 6: Examples of Best Slack Segment.

We have now two cases, namely, whether there are fan-outs in region H_6.

When there are no fan-outs in region H_6, from Figure 5, by putting all fan-out gates from H_1 and H_2 in one set (say S_P), and all fan-out gates from H_3 and H_4 in another set (say $S_{P'}$), *Best Slack Segment* for each set is the maximized since it avoids potential intersection (i.e. fan-outs from H_1 and H_3). In that case, $B(F, S_P)$ is a line segment on $K(F)$ starting from i, and $B(F, S_{P'})$ is a line segment on $K(F)$ starting from j. Fan-out gates from H_5 can be put in either set and do not affect the results since for every point in region H_5, the distance to all locations on $K(F)$ are same. This partitioning is one of the best partitionings and achieves the best slack.

LEMMA 4. *If both P and P' are movable, and no fan-out gates are located in the region H_6, the cloning problem can be solved optimally in $O(m)$ time.*

PROOF. If we put all fan-out gates from H_1 and H_2 in S_P, all fan-out gates from H_3 and H_4 in $S_{P'}$, and all fan-out gates from H_5 in either set, we have an optimal partitioning. One of the optimal placement solutions places P at i and P' at j. The time complexity is $O(m)$, which is the time of computing $K(F)$. The case when the slope of $K(F)$ is -45° can be proved similarly. □

From Lemma 4, it follows that

LEMMA 5. *If P is movable, and no fan-out gates are located in the region $H_6 \cup H_1 \cup H_2$ (or $H_6 \cup H_3 \cup H_4$), no clone is needed and the optimal slack can be achieved by placing P at j (or i).*

Now we present the general algorithm to *when there are fan-outs in region H_6*.

A slack curve as shown in Figure 5 for any region H_1, H_2, H_3, H_4, H_5 and H_6 can be regarded as a trapezoid-like curve (referred to as trapezoids for notational convenience henceforth) or a degenerate case (e.g., a line segment) of a trapezoid. Consider a graph containing slack curves corresponding to all fan-out gates. In the following, a side of a trapezoid will be called a line segment. The slope of any such line segment is of 0° or τ or $-\tau$. A 0° line segment in a trapezoid is called a level segment. In the degenerate case where the slack curve is a single line segment, the level segment is defined as the end point with maximum slack.

In all trapezoids, we first find the rightmost τ-slope line segment and the leftmost $-\tau$-slope line segment. For example, left (resp. right) side of the dotted t_1 (resp. t_2) in Figure 7(a) shows the rightmost τ-slope (resp. leftmost $-\tau$-slope line segment). The line segment of a trapezoid is rightmost (resp. leftmost) if no line segment of the slope is to the right (resp. left) of the line segment. The leftmost and rightmost line segments can be found in linear time.

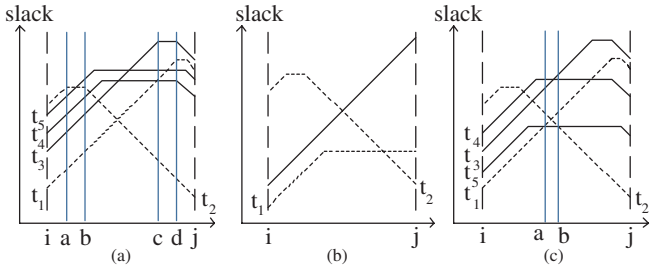


Figure 7: Examples of Slack curves v.s. locations: (a) an example that needs gate duplication; (b) an example in which the rightmost and leftmost segments do not intersect; (c) an example that does not need gate duplication.

First note that any point in a slack curve for fanout S_i refers to the net slack $Q(P, S_i)$ when placing P along i, j as defined in Figure 4. Given a single slack curve t_1, the best slack it can achieve is the slack corresponding to the level segment. To achieve it, one can place the gate anywhere on that level segment.

Case 1: When the rightmost τ-slope line segment and the leftmost $-\tau$-slope line segment do not intersect, as shown in Fig. 7(b), the lower level segment of all trapezoids, which is the *Best Slack Segment*, determine the maximum worst slack and no gate duplication is needed. Note that in this case, pure line segments in regions H_1, H_2, H_3 and H_4 are considered as well, since they are degenerate cases of trapezoids. One can just place P anywhere on that level segment and achieves the best slack.

Case 2: When the rightmost τ-slope line segment and the leftmost τ-slope line segment intersect, first find the trapezoids that these two line segments belong to. Without loss of generality, the identified trapezoids are as t_1 and t_2, respectively, in Fig. 7(a). We compute the intersections of all other trapezoids with t_1 and t_2, and put them into the sets S_P and $S_{P'}$ formed by t_1 and t_2, respectively. All other trapezoids can be divided into three groups.

Group A: For any trapezoid not intersecting any of t_1 and t_2, called *zero-intersecting trapezoid*, we arbitrarily assign it to a set. The zero-intersecting trapezoids will not impact the worst slack. Note that if all trapezoids other than t_1 and t_2 are zero-intersecting

trapezoids, the lowest level segment in t_1 and t_2 are *Best Slack Segment* of each set.

Group B: For any trapezoid intersecting only one of t_1 and t_2, called *one-intersecting trapezoid*, we can always assign it to the opposite set (formed by the line segment not intersecting with it). For example, the trapezoid t_3 in Fig. 7(a) only intersects t_2 and it is assigned to S_P formed by t_1. The one-intersecting trapezoids will not impact the worst slack as long as they are assigned appropriately. Note that if all trapezoids other than t_1 and t_2 are one-intersecting trapezoids, the lowest level segment in t_1 and t_2 are *Best Slack Segment* of each set.

Group C: For any trapezoid intersecting both of t_1 and t_2, called *two-intersecting trapezoid*, we have two intersecting points. A two-intersecting trapezoid will be assigned to the set containing the higher intersecting point. For example, both t_4 and t_5 are assigned to the S_P formed by t_1. One then needs to find the two-intersecting trapezoid with lowest level segment such as t_4 in Fig. 7(a). Subsequently, the lowest level segment in t_1, t_2 and t_4 determines the *Best Slack Segment*. In Fig. 7(a), *Best Slack Segment* for P is with t_2. This means that P can be anywhere between a, b and P' can be anywhere between c, d. For the partitioning of the set of fan-out gates S, P will connect to S_P which contains all the trapezoids assigned to S_P determined by t_1, and P' will connect to $S_{P'}$ which contains all the trapezoids assigned to $S_{P'}$ determined by t_2. Note that the lowest level segment of a two-intersecting trapezoid can be lower than the intersection of t_1 and t_2, see, e.g., t_5 in Fig. 7(c). However, it will not impact our algorithm. This just means that one cannot improve the slack by gate duplication since the worst slack is determined by the level segment of t_5.

The algorithm is optimal since the above two cases cover all possible situations and in each situation, it is easy to see that the optimal solution is computed. In the algorithm, one needs to first compute the rightmost τ-slope and the leftmost $-\tau$-slope line segment. If they do not intersect, the slack is determined by the lower level segment. Otherwise, for each of the remaining $m-2$ trapezoids, compute its intersections with t_1 and t_2. Assign the trapezoids to partitions accordingly based on their groups. For a two-intersecting trapezoid, one also needs to record its higher intersection point. Next, find the trapezoid with lowest higher intersecting point (e.g., t_4 in Fig. 7(a)), which takes linear time. One can then immediately find the maximum possible worst slack the circuit can achieve by comparing it with the level segment of t_1 and t_2. Clearly, the above algorithm runs in linear time.

THEOREM 3. *The optimal cloning can be computed in $O(m)$ time if the original gate is movable.*

The pseudo-code of the algorithm is shown as follows.

Algorithm `Cloning_Movable`
Input Graph G
Output Location of P and P', S_P and $S_{P'}$
Begin
1: Find *arrival time arc* $K(F)$ given the set of fan-in gates F.
2: **If** $K(F)$ is a point
3: Move P to that point and return.
4: Divide the plane by $K(F)$ into 6 regions
5: **If** No S_i in H_6
6: Compute S_P, $S_{P'}$, P and P' according to Lemma. 4, return
7: Put all slack curves into a single graph.
8: Find rightmost τ-slope and leftmost $-\tau$-slope line segment and trapezoids
9: **If** They do not intersect
10: Slack is determined by the lower level segment.
11:**Else**
12: Compute intersections between remaining trapezoids with the above trapezoids and assign them to S_P and $S_{P'}$ accordingly. Compute P and P' as above.
13:Return the location of P and P', S_P and $S_{P'}$;
End.

3.3 When the Original Gate is Fixed

When the original gate P is fixed, the above algorithms do not work since we can not expect P to be placed on the *arrival time arc* $K(F)$. Let us assume all fan-outs in S is sorted in the non-increasing order of $RAT(S_i) - D(P, S_i)$.

LEMMA 6. *There are at most m unique $Q(P)$ values if P is fixed.*

PROOF. Since P is fixed, $AT(P)$ and $D(P, S_i)$ are constant. Then for all possible partitionings, $Q(P)$ can only be one of the value among $RAT(S_1)-D(P,S_1)-AT(P)$, $RAT(S_2)-D(P,S_2)-AT(P)$, ..., $RAT(S_m)-D(P,S_m)-AT(P)$. □

The above Lemma states that if fan-out S_i is in S_P, then we can put all fan-outs S_j, where $j < i$ into S_P, and $Q(P)$ does not change. With Lemma 6, we can start with putting S_1 in $S(P)$, while putting all other gates in $S(P')$, and get the worst slack of $Q(P)$ and $Q(P')$. If $Q(P') \geq Q(P)$, we can stop since this is the best slack we can get. If not, we can put S_1 and S_2 in $S(P)$, which will decrease $Q(P)$, but may increase $Q(P')$. Again, if $Q(P') \geq Q(P)$, this will be the best slack we can get since further process will further decrease $Q(P)$, and give smaller worst slack for the whole subcircuit. The pseudo-code of the algorithm is shown as follows.

The sorting of S takes $O(m \log m)$ time. After S is sorted, we can precompute all $K(S_{P'})$ for all possible m cases in $O(m)$ time based on Lemma 1. Each *FindBestRegion* takes $O(1)$ time now since $AT(K(F))$ is a constant and $K(S_{P'})$ has been precomputed. The total complexity is $O(m \log m)$.

Algorithm `Cloning_Fix`
Input Graph G, Original Slack Q_{ori}
Output Location of P', S_P and $S_{P'}$
Begin
1: Sort S in non-increasing order of $RAT(S_i) - D(P, S_i)$.
2: **For** $i = 1$ to $m - 1$
3: $S_P = \{S_1, \ldots, S_i\}$, $S_{P'} = S - S_P$.
4: Call **FindBestRegion** to get the location of P'.
Compute $Q(P)$ and $Q(P')$
5: **If** $Q(P') \geq Q(P)$
6: break.
7: Compare the solution with Q_{ori} and
return the location of P', S_P and $S_{P'}$;
End.

One note is that one can disconnect all sinks and just let P' drive all fan-outs and move it, which is similar to RUMBLE [9], and we can compare the solution with the above results and find the best one.

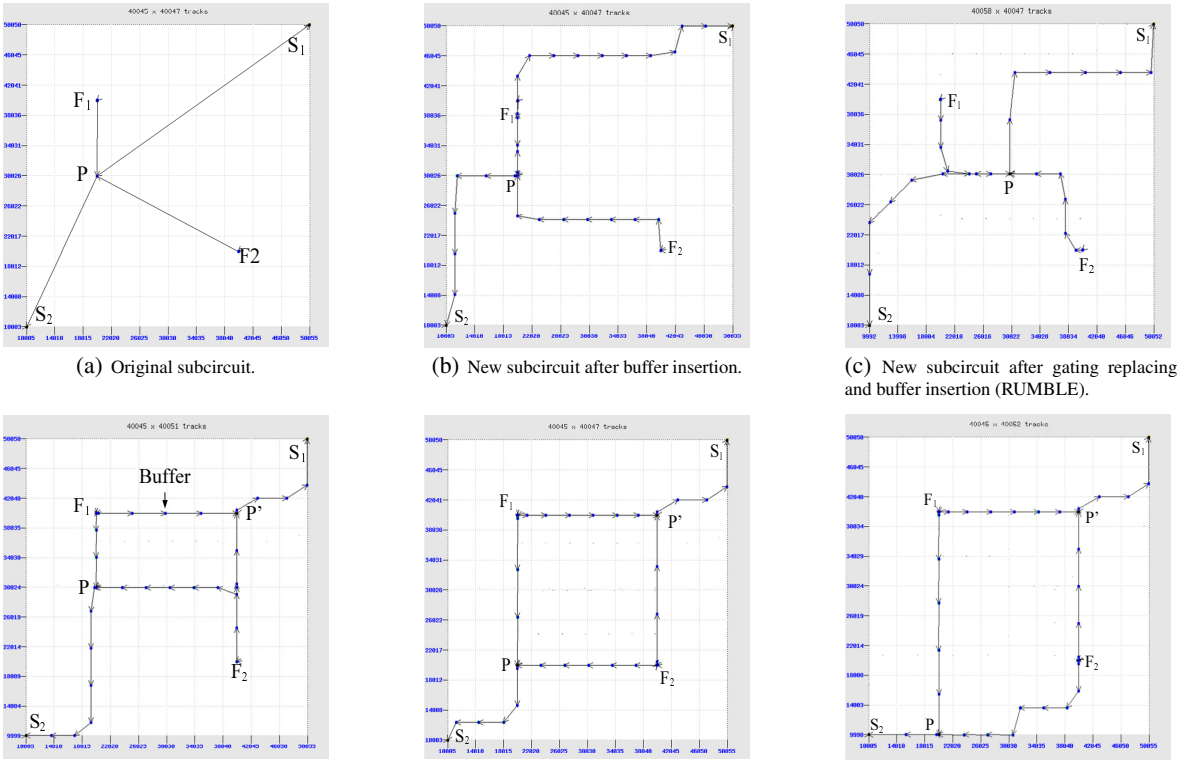
THEOREM 4. *The optimal cloning can be computed in $O(m \log m)$ time if the original gate is fixed.*

4. EXPERIMENTAL RESULTS

To show the effectiveness of cloning, especially compared to other optimizations, we first create 100 random testcases in the 45nm node (which means the logic gates and buffers are taken from a 45nm library). We randomly created subcircuits with different fan-in and fan-outs and placed them in a region with the bounding box size ranging from 1mm to 15 mm. The number of fan-ins range from 2 to 4, and the number of fan-outs range from 2 to 8. We choose 16 buffers and inverters for the buffer insertion.

We implemented 4 different optimizations including cloning as follows, to show the benefit of our techniques. They are

- Buffering: Timing driven buffer insertion [6]. This data is treated as the baseline and the data of all other optimizations is compared to this one.
- RUMBLE: Moving the original gate and rebuffering as described in [9].
- Clone1: Our cloning algorithm when the original gate is fixed.
- Clone2: Our cloning algorithm when both the original and duplicated gates can be moved.



(a) Original subcircuit.

(b) New subcircuit after buffer insertion.

(c) New subcircuit after gating replacing and buffer insertion (RUMBLE).

(d) New subcircuit after cloning and replacing new gate only (Clone1).

(e) New subcircuit after cloning and replacing both gates (Clone2).

(f) New subcircuit after Clone2 with wirelength-inefficient solution.

Figure 8: Example of different optimizations, including buffering, RUMBLE and cloning. F_1 and F_2 are fan-ins with same arrival time and S_1 and S_2 are fan-outs with same required arrival time. P is the original gate, and P' is the new duplicated gate.

To be fair, for RUMBLE, Clone1, and Clone2, we always first run buffer insertion before the optimization. The results are also compared to buffer insertion results (which means "Buffering" is the baseline). This is to guarantee that any improvement we see from our techniques, are due to cloning instead of pure buffering on the original net. In addition, we also use the RUMBLE algorithm inside our cloning algorithms to determine the best gate location after a partitioning is fixed. For each partition, we will run RUMBLE to find the gate location and slack, and then choose the best solution for all partitions derived from our algorithm. Note that it is only for the comparison purpose, and one can still apply our algorithm first to find the best partitioning and only apply the RUMBLE algorithm once.

All algorithms including buffering and RUMBLE are implemented in C++ and tested on an AMD Opteron computer with 2.8GHz CPU and adequate memory. For the cloning, we did the full optimization steps, including ripping up buffer trees for the subcircuits, duplicating and placing the gates, re-buffering and legalization. For RUMBLE, we also rip up buffer trees and place the original gate in the new location. Also, we use an industrial static timing analysis engine for the timing analysis. For the rebuffering, we implement the buffering algorithm in [6] to get best timing-area tradeoff, and buffer tree is constructed to be placement congestion aware.

To clearly illustrate the impact of each optimization, we first choose one subcircuit and show the circuit layout after each optimization from Fig. 8(b) to Fig. 8(d), where Fig. 8(a) shows the original circuit without buffering. The Manhattan distance between S_1 and S_2 is 13 mm. The timing information after each optimization algorithm is shown in Table 1. It clearly shows that the benefit of buffering, RUMBLE, Clone1 and Clone2 approaches. Clone2 gives the best results in terms of worst slack and FOM. Clone1 is still better than RUMBLE and get the same worst slack as Clone2, but can not do better for S_2. RUMBLE achieves better slack than pure buffering by placing the original gate in the middle, however, it sacrifices the slack at S_1 for S_2. Note that the slack of S_1 and S_2 are not exactly same as for RUMBLE and Clone2, it is due to slew impact, the buffering topology chose from placement congestion aware buffer-tree algorithm which considers the placement density, as well as the order of buffer insertion for all the nets which results in asymmetric timing constraints.

Table 1: Experimental results comparing cloning to other optimization techniques for the subcircuit shown in Fig. 8.

Optimization	Slack at S_1 (ns)	Slack at S_2 (ns)
Buffering (Fig. 8(b))	-2.855	-2.206
RUMBLE (Fig. 8(c))	-2.410	-2.403
Clone1 (Fig. 8(d))	-1.606	-2.076
Clone2 (Fig. 8(e))	-1.606	-1.590

For the rest of the circuits, due to space limitations, we list the top 10 subcircuits with the best improvement due to cloning with detailed information. The results are shown in Table 2. For all experiments, we present worst slack (WSLK) improvement over "Buffering", Figure of Merit (FOM, the sum of all negative paths) improvement over "Buffering", final area and wirelength, where Buffering experiment serves as the baseline. The area includes the original fan-in gates, fan-out gates, cloned gate and buffering area. We also list the summary results of all 100 subcircuits in Table 2 by averaging all metrics. The runtime for all testcases is pretty fast, less than 5 seconds, including all static timing analysis, buffer insertion, linear programming inside RUMBLE, I/O processing and model build-up time.

The table clearly shows the same trend as shown in Fig. 8. In terms of worst slack, RUMBLE is better than buffering, and

Clone1 and Clone2 are better than RUMBLE, where Clone2 gives the best timing results in general, although with the cost of area and wirelength increase. We also notice that for all cases, Clone1 and Clone2 both achieve better FOM improvement than buffering. Note that our algorithms may not get the best FOM, especially for Clone1 where the original gate can not be moved to improve the slack of other paths. The summary data shows that Clone2 and Clone1 still outperform RUMBLE and buffering on average.

Table 2: Experimental results comparing cloning to other optimization techniques for 100 random subcircuits. All numbers refer the final results after each optimization.

subckt	Transforms	WSLK Improvement (ns)	FOM Improvement (ns)	Area	WL
1	Buffering	0	0	1158	181960
	RUMBLE	1.548	3.630	609	117637
	Clone1	1.553	3.645	799	117632
	Clone2	1.581	3.747	601	141977
2	Buffering	0	0	1110	166546
	RUMBLE	1.111	2.895	859	162461
	Clone1	1.175	3.091	889	162419
	Clone2	1.542	4.660	1026	164254
3	Buffering	0	0	942	142242
	RUMBLE	0.956	1.859	722	131794
	Clone1	1.030	2.298	850	145908
	Clone2	1.073	2.611	765	135896
4	Buffering	0	0	709	95520
	RUMBLE	1.050	1.113	636	88441
	Clone1	1.022	1.092	636	88441
	Clone2	1.050	1.113	636	88441
5	Buffering	0	0	1758	253393
	RUMBLE	0.839	6.128	1120	194261
	Clone1	0.814	6.262	1109	194260
	Clone2	1.028	5.413	1410	241818
6	Buffering	0	0	1604	225577
	RUMBLE	0.773	3.282	998	177139
	Clone1	1.014	1.041	1529	241626
	Clone2	1.017	2.152	1293	233053
7	Buffering	0	0	1781	268237
	RUMBLE	0.302	0.189	1583	257903
	Clone1	0.830	1.049	1990	315835
	Clone2	0.815	1.121	2047	330826
8	Buffering	0	0	1578	227047
	RUMBLE	0.262	4.270	1153	195097
	Clone1	0.681	2.118	1836	272108
	Clone2	0.732	4.866	1633	251854
9	Buffering	0	0	998	140556
	RUMBLE	0.685	1.512	861	122344
	Clone1	0.687	1.514	848	122411
	Clone2	0.718	1.530	871	122360
10	Buffering	0	0	998	159705
	RUMBLE	0.269	1.312	831	140127
	Clone1	0.672	1.759	916	150529
	Clone2	0.673	1.754	899	150490
Avg. of 100 circuits	Buffering	0	0	1407	205891
	RUMBLE	0.192	0.797	1337	198247
	Clone1	0.279	1.050	1472	216617
	Clone2	0.309	1.267	1471	220089

5. EXTENSIONS

Our algorithms naturally accommodate several additional objectives that we briefly summarize in this section.

5.1 Wirelength Efficiency

Note that in our formulation, we did not directly consider wirelength. However, our approach can be extended to consider wirelength while not sacrificing slack. For example, in the case where both gates are movable and no gates are placed in the region H_6, after we determine the partitioning, and put P at i and P' at j, we can still find the best region Z which is bounded by i and S_P for P (similarly for P' with a region bounded by j and $S_{P'}$). When region Z is not a single point, it may be possible to find a solution with same slack but better wirelength. Due to space limitations, the detailed algorithm is omitted, but we briefly summarize the $O(m^2)$ algorithm as follows. Each rectangle region in the Hanan grid formed by all fan-in and fan-out gates has a unique function of total wirelength and its locations. Scan through the part of Z in each region and find the best location with the minimal wirelength, then choose the best solution among all Hanan regions covered by Z, which is the solution with the best wirelength and optimal slack. Note that the wirelength optimal region may be contained within Z, in which case the wirelength optimal solution is also slack optimal.

A wirelength inefficient Clone2 example is shown in Fig. 8(f). It has the same slack and FOM as Fig. 8(e), however, Fig. 8(e) clearly shows smaller wirelength (and less number of buffers), and it can be proved that the location of P in Fig. 8(e) is wirelength optimal solution too.

5.2 FOM Optimization

Though our algorithms can improve the FOM objective by improving worst-slack, our algorithms do not directly optimize the FOM objective. It can, for example, hurt FOM by reducing slack on two paths, while endeavoring to improve the slack on a third. In the late stages of the flow, this may be unacceptable, and we may wish not to harm FOM, or to directly optimize FOM or the number of negative paths. When both gates are movable and there is no fan-out in H_6, it is easy to prove that our solution gives the best solution in terms of FOM. When there are gates in H_6, one can tune the algorithm `Cloning_Movable` to be FOM aware. When we assign trapezoids, even if it does not change worst slack, we can assign based on its own slack and achieve better FOM. Finally, we can prevent harm to the FOM objective by incorporating it into the acceptance criteria for any cloning solution.

5.3 Placement Blockages

When there are placement blockages in the design, such as IP, macros, or just the region of high gate densities, one may not place the duplicated gate or original gate to the optimal location. Our algorithms can be extended to handle blockages as follows. When the best region Z is not a single point, and not completely blocked by placement blockages, we place P (or P') in the region inside Z with free space and still achieve the optimal slack. If Z is completely blocked, then P is placed at the legal location with the minimum Manhattan distance to the region Z.

6. CONCLUSIONS

In this work, the cloning problem is revisited for timing-driven objectives under a linear delay model in the context of physical synthesis. We present several highly efficient algorithms for timing driven cloning to optimize circuit worst slack.

7. REFERENCES

[1] P. Saxena et al., "Repeater scaling and its impact on CAD," *IEEE Trans. CAD* , vol. 23, no. 4, pp. 451–463, 2004.

[2] R. Otten, "Global wires harmful," in *ISPD* , 1998, pp. 104–109.

[3] C. J. Alpert et al, "Accurate estimation of global buffer delay within a floorplan," *IEEE Trans. CAD* , vol. 25, no. 6, pp. 1140–1146, 2006.

[4] W. Shi, Z. Li, and C. J. Alpert, "Complexity analysis and speedup techniques for optimal buffer insertion with minimum cost," in *ASPDAC*, 2004, pp. 609–614.

[5] S. Dhar and M. A. Franklin, "Optimum buffer circuits for driving long uniform lines," *IEEE J. Solid-State Circuits*, vol. 26, no. 1, 1991, pp. 32-40.

[6] Z. Li et al., "Making fast buffer insertion even faster via approximation techniques," in *ASPDAC* , 2005, pp. 13–18.

[7] C. J. Alpert et al., "Techniques for Fast Physical Synthesis," in *Proc. of IEEE*, 2007, vol. 95, no. 3, pp. 573-599, 2007.

[8] R. Kužnar and F. Brglez, "PROP: A recursive paradigm for area-efficient and performance oriented partitioning of large FPGA netlists," in *ICCAD* , 1995, pp. 644–649.

[9] D. A. Papa et al., "RUMBLE: An incremental, timing driven, physical-synthesis optimization algorithm," in *ISPD*, 2008, pp. 2–9. 2007.

[10] J. Hwang and A. El Gamal,"Optimal replication for min-cut partitioning," in *ICCAD* , 1992, pp. 432–435.

[11] G. Chen and J. Cong,"Simultaneous timing-driven placement and duplication," in *ISFPGA* , 2005, pp. 51–59.

[12] J. Lillis, C. K. Cheng and T. Y. Lin,"Algorithms for optimal introduction of redundant logic for timing and area optimization," in *ISCAS* , 1996, pp. 452–455.

[13] H. Kim, J. Lillis and M. Hrkić,"Techniques for improved placement-coupled logic replication," in *GLSVLSI* , 2006, pp. 211–216.

[14] C. Chen and C. Tsui,"Timing optimization of logic network using gate duplication," in *ASPDAC* , 1999, pp. 233–236.

[15] D. Ba neres, J. Cortadella and M. Kishinevsky,"Layout-aware gate duplication and buffer insertion," in *DATE* , 2007, pp. 1367–1372.

[16] A. Srivastava et al., "On the complexity of gate duplication," *IEEE Trans. CAD* , vol. 20, no. 9, pp. 1170–1176, 2001.

[17] A. Srivastava et al., "Timing driven gate duplication," *IEEE Trans. VLSI* , vol. 12, no. 1, pp. 42–51, 2004.

[18] T. Luo et al., "Pyramids: An Efficient Computational Geometry-based Approach for Timing-driven Placement." in *ICCAD*, 2008, pp. 204–211.

[19] T. H. Chao et al., "Zero skew clock routing with minimum wirelength," *IEEE. Trans. CAS* , vol. 39, no. 11, pp. 799–814, 1992.