# Addressing the Effects of Reconvergence on Placement-Coupled Logic Replication

Miloš Hrkić
mhrkic@cs.uic.edu

John Lillis
jlillis@cs.uic.edu

University of Illinois at Chicago, CS Dept., Chicago, IL 60607

## ABSTRACT

We present a general and robust approach to timing-driven, placement-coupled logic replication. The approach is similar in nature to [7] and it is designed to address issues that arise due to the *reconvergence* in the circuit specification. We build on the *Replication Tree* idea and modify the timing-driven fanin tree embedding algorithm to optimize sub-critical paths. We have built optimization engine for the FPGA domain and report promising preliminary results including clock period reduction of up to 38% compared with a timing-driven placement from VPR [14] and up to 9% compared to RT-Embedding replication algorithm from [7].

## Categories and Subject Descriptors

B.7.2 [**Hardware**]: Integrated Circuits—*Design Aids*

## General Terms

Algorithms, Performance

## Keywords

Timing Optimization, Logic Replication, Placement, Programmable Logic

## 1. INTRODUCTION AND BACKGROUND

The idea of *logic replication* is to duplicate certain cells in a design so as to enable more effective optimization of one or more design objectives. This primitive re-synthesis operation has been applied in several different contexts including min-cut partitioning (e.g., [12] [13]) and fanout tree optimization (e.g., [11] [15]).

Recently a few papers including [1], [7], [2] and [3] have explored the idea of using replication to effectively deal with interconnect-dominated delay at the physical level. In these papers it is observed that, because replication effectively separates multiple signal paths it becomes easier to, at the physical design level, "straighten" input-to-output (flip-flop to flip-flop) paths which might otherwise have been very circuitous (and therefore high delay).

A simple example from [1] reproduced in Figures 1 and 2 illustrates the idea. Suppose that the terminals at $a$, $b$, $d$ and $e$ are fixed. There are 4 distinct input-to-output paths; any movement of the central cell $c$ from the shown location will degrade the delay of at least one of these paths (assume for the moment a linear delay model). Thus in Figure 1 we have no choice but to tolerate non-monotone input-to-output paths. Now suppose that we replicate cell $c$ as shown in Figure 2 to form $c'$ computing the same function, but feeding only output $b$ while $c$ drives only $d$. If we produce such a logically equivalent netlist all input-to-output paths become virtually monotone.

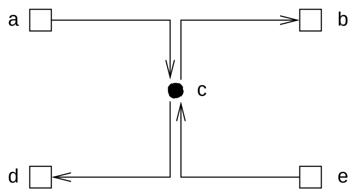

**Figure 1: Example with forced non-monotone paths.**

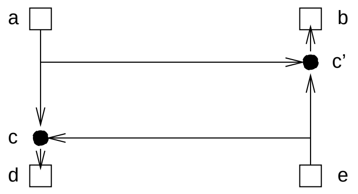

**Figure 2: Example illustrating path straightening by replication of cell $c$.**

The recent work of [1] made a compelling case for the potential of replication by observing that not only do typical placements contain critical paths which are highly non-monotone, but also that the number of cells which have near-critical paths flowing through them is relatively small (thus, one may conjecture that a small amount of replication may be sufficient). Then an incremental replication procedure was proposed and evaluated experimentally with promising results. Roughly speaking the algorithm examined the current critical path and looked for cells to replicate; for such cells, it placed the duplicate, performed fanout partitioning and then legalized the placement. The criteria for selecting a cell was based on the goal of inducing *local monotonicity*. Local monotonicity was defined by a sequence of 3 cells on a path $v_1, v_2, v_3$. Let $d(u, v)$ be the rectilinear distance between cells $u$ and $v$; then we say that the path from $v_1$ to $v_3$ is non-monotone if $d(v_1, v_3) < d(v_1, v_2) + d(v_2, d_3)$ (i.e., traveling to $v_2$ creates a detour). In such a case, $v_2$ is a good candidate for replication so as to straighten this path

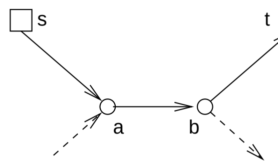

**Figure 3: Limitation of local monotonicity. Cells $a$ and $b$ are locally monotone yet $s$-to-$t$ path is not.**

without disturbing other paths passing through $v_2$.

While this strategy proved effective in reducing clock period, it is observed in [7] that a technique based on local monotonicity has limitations. Figure 3 demonstrates this limitation. In the figure we see a critical path $(s, a, b, t)$ (dashed lines indicate other signal paths which may be near critical). Clearly, this path is non-monotone and yet, all sub-paths (of length 3) are locally monotone. In this case (which is not unusual), the approach is unable to improve the delay.

Thus, [7] proposed a more robust and general replication strategy. First, a fanin tree embedding algorithm is proposed. In this problem, given the root of a fanin tree (e.g., a flip-flop), a tree circuit which produces its inputs and arrival times at the inputs (leaves) of the tree, goal is to embed the tree so as to obtain a tradeoff between the cost of the embedding and the arrival time at the root (sink) of the tree. Unfortunately, most circuits, because of reconvergence, do not contain large sub-circuits which are fanin trees. To cope with this problem, [7] proposed the *Replication Tree* which gives a systematic way of taking a set of edges in a circuit forming a directed tree (e.g., with the root being the input of a flip-flop), and, using replication, induce a genuine fanin tree which can be optimized by the fanin tree embedder.

Part of the philosophy behind the replication tree is that by inducing fanin trees, very strong, physically coupled algorithms can be employed with guarantees of optimality for the subproblem being solved under very general and flexible formulations. As a contrasting example, delay-optimal technology mapping [10] can be solved efficiently if one is not concerned with area/cost overhead. Of course, in most practical situations we cannot ignore such fundamental design objectives.

Although [7] was able to achieve larger improvement than [1] in minimizing critical delay, a significant number of circuits from the test suite have non-monotone critical paths.

It this work we analyze effects caused by reconvergence in netlist specification on the *Replication Tree* construction. These effects prevent the fanin tree embedding algorithm to "straighten" some paths. This particular problem is related to the way how replication tree handles reconvergences in circuit specification, which occurs quite often in practice. While we still use the same replication tree construction algorithm we propose a new fanin tree embedding scheme which has more success in addressing reconvergence issues by optimizing sub-critical paths. Since the optimization framework is iterative in its nature, we optimize sub-critical paths hoping to create better optimization choices for subsequent iterations of the flow.

The paper is organized as follows. Section 2 describes the problem caused by optimal cost/max-arrival-time embedding of the replication tree which is linked to netlist reconvergence. We introduce modified fanin tree embedding approach that optimizes sub-critical paths in order to address reconvergence issues in section 3. Section 4 gives a top-level flow of our approach. Experimental results are presented in Section 5. In section 6 we discuss potential problems that are still present and we conclude in section 7.

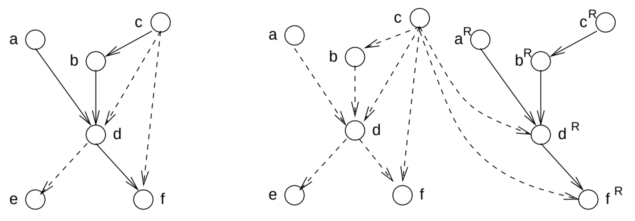

**Figure 4: Replication Tree example.**

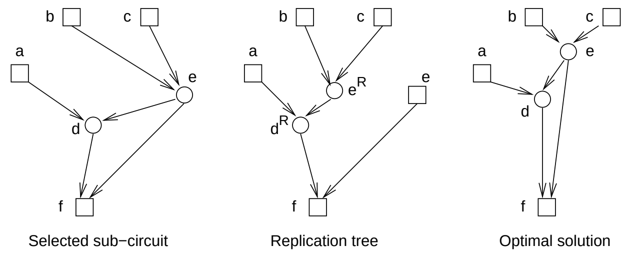

**Figure 5: Example of reconvergence effect on the replication tree.**

## 2. EFFECTS OF RECONVERGENCE

Since most circuits do not have large fanin trees due to reconvergence, [7] proposed the *Replication Tree* which induces large fanin trees in a logically equivalent circuit. To better understand the potential weakness of this approach we will briefly reintroduce the main replication tree concepts through an example.

In the left part of Figure 4 we have a portion of a circuit with a set of edges in bold. These edges form a tree with all edges pointing toward the root $(f)$. Note that in the left figure, this tree does not form a valid fanin tree due to reconvergence. To induce a fanin tree we (temporarily) make a copy of each node in the tree $(f, d, a, b, c)$. If the original cell is $v$ and the copy is $v^R$, we assign connections as follows: let $u_1, ..., u_k$ be the inputs to $v$. If $(u_i, v)$ is a tree edge, then $v^R$ receives its $i$'th input from $u_i^R$; otherwise, it receives its $i$'th input from $u_i$.

This construction is applied to the circuit in the Figure 4 and results in the circuit on the right and yields a fanin tree sub-circuit formed by the replicated cells. Notice that cells $d^R$ and $f^R$ connect to $c$ rather than $c^R$ – otherwise the replicated cells would not form a proper fanin tree (technically speaking it is a Leaf-DAG because, for example "leaf" node $c$ connects to two cells in the tree, however, since the timing properties of $c$ are fixed and known, this does not complicate the embedding process). Two main claims from this construction are that if the circuit is modified in this way, the result is functionally equivalent; and the set of replicated nodes form the internal vertices of a legitimate fanin tree which can be embedded.

It sounds natural to use optimal cost/max-arrival-time fanin tree embedding algorithm on replication tree (as in [7]). However, the optimality of the embedder creates a problem when reconvergence is present. Let us use and example to illustrate this issue. In Figure 5 on the left we have an extracted critical sub-circuit. Inputs are nodes $a$, $b$ and $c$. Internal nodes are $d$ and $e$ and sink is node $f$. Nodes that are fixed are represented by squares and movable/replicable nodes are represented with circles. Let as assume that nodes $a$, $b$ and $c$ have signal arrival time of zero. Also let the arrival time at $e$ be 2 units, and 3 units at sink $f$, while delay of the path from $a$ to $d$ is one unit. The replication tree construction procedure applied to the net on the left will yield the replication tree shown in the middle of the Figure 5. Node $d$ becomes replicable $d^R$, but since there is a reconvergence on node $e$ we will have two nodes: movable/replicable $e^R$ and fixed node $e$, where reconvergence "breaks". When we apply the optimal cost/max-arrival-time tree embedding algorithm to this replication tree the fastest solution with smallest cost will have internal nodes placed at exactly the same location as they originally were (on the left in Figure 5). Path from $a$ to $f$ is not critical. Since path from $e$ to $f$ is monotone, it cannot be improved any further and initial signal arrival time at $e$ is set to 2 units, so arrival time at $f$ will remain 3 units. The embedder is not going to over-optimize sub-critical path that goes through node $e^R$ and this will yield minimum placement cost for node $e^R$ and $d^R$ which is achievable by placing those nodes on top of $d$ and $e$ respectively. [1] However, solution on the right of the Figure 5 has clearly better delay (i.e. all monotone paths) and potentially uses less wiring resources.

To address this problem we propose a modified fanin tree embedding algorithm that allows over-optimizing sub-critical paths.

## 3. OVER-OPTIMIZED TREE EMBEDDING

In the fanin tree embedding problem we are given a fanin tree, placement of leaves (inputs) and root (sink), arrival times at the inputs and a target placement region (in our case this is encoded in an embedding graph). The goal is to place the internal tree nodes (gates) minimizing cost subject to an arrival time constraint at the root (typically there is a tradeoff between cost and arrival time).

To address the optimality issue presented in the previous section we propose a modification to the embedding problem. The optimization criteria is to place internal tree nodes minimizing cost subject to *composite* arrival time constraint at the root. Going back to the example in the Figure 5, if we could optimize delay of the *second critical path* then the reconvergence could be broken and in the next iteration we could potentially optimize the remaining path of the critical net and achieve global delay improvement.

Dynamic programming approach for fanin tree embedding in [7] is based on 2-dimensional variant of the tree embedding problem from [5] and [6]. A candidate solution (embedding) for a subtree rooted at node $i$ in the tree with node $i$ placed at vertex $j$ in the embedding graph is represented by its *signature* $(c, t)$, indicating that this sub-solution incurs cost $c$ and has latest arrival time $t$ at $i$. The variant is called 2-dimensional since solution signature contains two parameters that we are simultaneously optimizing (in this case, cost and max arrival time). In [6] one can find a description for more complex 3-dimensional case, i.e. simultaneously optimizing three parameters. However, top-level algorithms for 2D and 3D case are almost the same and the major difference is how some of the primitives are handled. Pseudo-code of tree embedding algorithm is shown in Figure 6. In our case, where we want to optimize cost, critical delay and chosen path delay, it seems that by simply using the 3D variant of the embedding tree algorithm would solve the problem. Using 3D embedding would certainly (and significantly) affect the run-time of proposed approach. Also it would present significant problem if we would to try optimizing additional 2 or 3 critical paths, since we would have to increase number of dimension in the embedding algorithm which could yield higher complexity.

[1]Note that there are some rare exceptions where savings in wiring resources could outweigh the replication cost for $e^R$ and the path could be straightened, and then in subsequent iterations node $e$ could be moved as well and unified with new location of $e^R$, but a significant amount of luck is required.

```
Subroutine: GenDijkstra($A^b, G$)
        $A^b$: Joined solutions; G: Target routing graph
d1      $A \leftarrow \emptyset$
d2      for each solution in $A^b(v)$
d3        Insert $A^b(v)$ into Queue
d4      endfor
d5      while $Queue \neq \emptyset$
d6        $v_1 \leftarrow$ Top Queue
d7        if Solution($v_1$) is not suboptimal
d8          $A(v_1) \leftarrow A(v_1) \cup$ Solution($v_1$)
d9          for each edge $(v_1, v_2)$ adjacent to $v_1$
d10           $v_2 \leftarrow$ Augment($v_1, v_2$)
d11           Insert $v_2$ into Queue
d12         endfor
d13       endif
d14     endwhile
d15     return $A$
Subroutine: JoinTree($T, G$)
        T: Topology subtree; G: Target routing graph
c1      for each vertex $v \in G$
c2        $A^b[v] \leftarrow$ Join($A.left[v], A.right[v]$)
c3      endfor
c4      $A \leftarrow$ GenDijkstra($A^b, G$)
c5      return $A$
Subroutine: ComputeSubTree(T,G)
        T: Topology subtree; G: Target routing graph
b1      if($leaf(T)$)
b2        $A(T) \leftarrow$ ComputeInitial($G, T$)
b3      else
b4        $A(T.left) \leftarrow$ ComputeSubTree($T.left, G$)
b5        $A(T.right) \leftarrow$ ComputeSubTree($T.right, G$)
b6        $A(T) \leftarrow$ JoinTree($T, G$)
b7      endif
b8      return $A(T)$
Algorithm: TreeEmbedding($T, G, s$)
        T: Topology; G: Target routing graph
        s: source node
a1      $A(T) \leftarrow$ ComputeSubTree($T, G$)
a2      $Final \leftarrow$ AugmentRoot($A(T), s$)
a3      return $Final$
```

**Figure 6: General Tree Embedding Algorithm.**

To better understand the proposed solution, let us describe differences between 2D and 3D algorithm. The main difference is in determining the solution *dominance* property. Given a solution signature, a solution is *non-dominated* if no other solution is superior in *all* dimensions. Any dominated solution may be discarded. In 2D case there is a total order of solutions due to this property, while that is not the case in 3D variant. In Line $d7$ of the pseudo-code, in 2D case, if we keep solutions sorted in the list, dominance test is trivial (check the tail of the list) and takes constant time. However, 3D case is more complex and balanced binary search trees are needed for efficient dominance test ([6]). Another difference is in the implementation of *Join* primitive in Line $c2$ which joins two sets of sub-solutions. In 2D case *Join* could be performed by linear traversal through sorted solution lists, while in 3D case we have to perform a cross product off all sub-solutions in those sets.

In our case, solution signature is a triple $(c, t, t2)$ (cost, critical delay and sub-critical delay). Observe that sub-critical delay cannot be larger that critical delay. This means that we have a total order of non-dominated solutions which means that we can use dominance test from 2D variant of the tree embedding algorithm. Even more, we can extend the solution signature to include more sub-critical path delays, as long as we keep them in sorted order and treat delay as lexicographically ordered values. In other words, if two solutions have the same cost and the same critical delay, we will keep the one which has better sub-critical path delay. If there is more expensive solution with the same critical delay but has better sub-critical path delay we will keep it, instead of discarding it as it happens in [7]. Unfortunately, for *Join* primitive we have to perform a cross product.

With this framework in place, a natural choice for sub-critical path would be a second most critical delay. As mentioned earlier, in the bottom-up DP procedure we must combine candidate solutions from subtrees to form new candidate solutions. At internal node $i$ in the tree and vertex $j$ in the graph we *join* single sub-tree solutions as follows:

$$\begin{aligned} c &= p_{i,j} + c_1 + c_2 + ... + c_k \\ t &= max(t_1, t_2, ..., t_k) \\ t2 &= max(\{t_1, t_2, ..., t_k\} \cup \{t2_1, t2_2, ..., t2_k\} \setminus \{t\}) \end{aligned}$$

where $k$ in the number of inputs for gate at $i$, and $p_{i,j}$ is placement cost. We will refer to this version as `Lex-2` (we have 2 lexicographically ordered delay values).

Having virtually no restrictions on number of sub-critical paths that we can over-optimize, we have implemented `Lex-3`, `Lex-4` and `Lex-5` in the similar fashion. Due to multiple reconvergences in the circuit description it is possible that many paths in the replication tree have exactly the same delay and more sub-critical paths are needed to resolve the deadlock. The single solution *join* for `Lex-3` is shown bellow:

$$\begin{aligned} c &= p_{i,j} + c_1 + c_2 + ... + c_k \\ t &= max(t_1, t_2, ..., t_k) \\ t2 &= max(\{t_1, t_2, ..., t_k\} \cup \{t2_1, t2_2, ..., t2_k\} \setminus \{t\}) \\ t3 &= max(\{t_1, t_2, ..., t_k\} \cup \{t2_1, t2_2, ..., t2_k\} \cup \\ &\quad \{t3_1, t3_2, ..., t3_k\} \setminus \{t\} \setminus \{t2\}). \end{aligned}$$

One other scheme of interest is to additionally optimize path from some specific input of the replication tree. In the replication tree the actual inputs are identified as leafs of the tree that have zero signal arrival time (in this way we can distinguish them from the leaves that are created as reconvergence terminators). Among those inputs we can identify *critical* input, as the one with largest downstream delay (which is easily obtained by performing static timing analysis). Solution signature is $(c, t, tc, w)$ where $tc$ is delay from critical input, and $w$ is a weight factor which is not included in dominance test. The weight factor is set to 1 for the critical branch and 0 otherwise. The join operation for

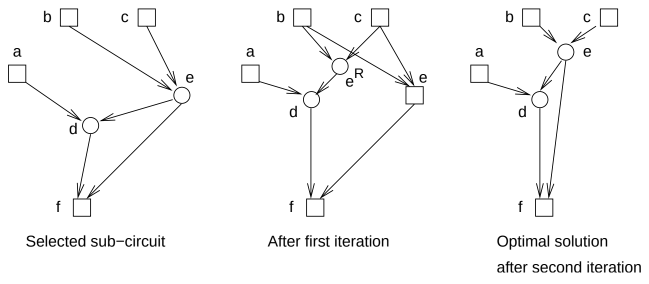

**Figure 7: Example of applying Lex-3 version of the algorithm on the same sub-circuit in two consecutive iterations. Sub-circuit on the left is transformed to the one in the middle by over-optimizing sub-critical path and using replication. In the second iteration replicated cells get unified and an optimal configuration on the right is achieved.**

`Lex-mc` (max and critical) is as follows:

$$\begin{aligned} c &= p_{i,j} + c_1 + c_2 + ... + c_k \\ t &= max(t_1, t_2, ..., t_k) \\ tc &= tc_1 * w_1 + tc_2 * w_2 + ... + tc_k * w_k \\ w &= w_1 + w_2 + ... + w_k. \end{aligned}$$

Going back to the example in the Figure 5, we are able to achieve solution on the far right by applying the `Lex-3` variant of the embedding algorithm. In the first iteration, paths from $b$ and $c$ to $f$ will get over-optimized in respect to dominating arrival time from $e$ to $f$ and we will obtain a solution similar to the figure in the middle. Then, in next iteration, since reconvergence is "broken", we will optimize paths from $b$ and $c$ through $e$ to $f$. Due to cost minimization $e$ would most likely be placed on top of $e^R$ and thus unified with it, achieving configuration on the right. Figure 7 shows how in two consecutive iterations in the optimization flow, sub-circuit on the left can improve delay by applying the `Lex-3` version of the algorithm.

## 4. OPTIMIZATION FLOW

The top-level view of how our optimizer relates to VPR [14] is similar to [1] and [7]: VPR is invoked to give an initial placement; we optimize the placement by replication; we then give the result to the VPR detailed router to accurately assess the results.

As in [7] the core replication procedure is focused on highly timing-critical sub-circuits and thus, while the embedding algorithm is nontrivial, the runtime penalty for using such a sophisticated algorithm is very small in the scope of the entire flow (this has been verified experimentally).

In each iteration of the replication loop we start with static timing analysis to identify the most critical sink. We extract a *Replication Tree* with the critical sink at its root as in [7]. Then we pass the tree to the new embedder which will produce a family of solutions that trade off cost and delays. After the embedding phase we analyze circuit for possible post-process unifications, since it is possible to have equivalent cells that are not exactly on top of each other, but close enough that unifying them would not harm timing. In addition we also use enhancement from [7]: a simultaneous flip-flop relocation (if the flip-flop is the root of the tree and the embedding was not able to improve delay). As a final step we invoke timing-driven placement legalizer (as in [8] and also explained in [7]) to resolve any possible cell overlaps that may have occurred.

## 5. EXPERIMENTS

We have performed some initial experiments to evaluate effectiveness of proposed techniques. The experiments were conducted in a LINUX environment on a PC with an Intel PentiumM 1.3GHz CPU and 256MB of RAM. The main criteria of interest are the maximum delay through the circuit (i.e., clock period), wire length and number of logic blocks. All such statistics are reported by the VPR timing-driven router. We compared the following versions of our approach: `Lex-mc`, `Lex-2`, `Lex-3`, `Lex-4`, `Lex-5` [2] with Timing Driven VPR [14] and the RT-Embedding replication algorithm from [7]. Table 1 shows the experimental results for 20 MCNC benchmark circuits. Due to space constraints we decided to include complete results only for `Lex-3` version. In Table 2 we report average values for the same set of 20 MCNC benchmark circuits for all compared algorithms (as in the bottom rows in Table 1). In addition we have divided circuits info small ($<$ 3K cells) and large ($\geq$ 3K cells) based on the number of LUTs they have. We present the average numbers for small and large group as well. As mentioned earlier, we used timing driven VPR to place the circuits. Then we applied the corresponding algorithm to optimize the placement and re-synthesize the net-list. All placements were routed using VPR in timing driven mode. The circuits were placed on the minimum square FPGA able to contain the circuit. As in [14] we use definition of low-stress routing as routing where FPGA has about 20% more routing resources available then the minimum required to successfully route the circuit. Also from [14], infinite-resource routing is when FPGA has unbounded routing resources. It is argued in [14] that the former represents the situation how FPGA will be routed in practice and the latter is a good placement evaluation metrics. For post-place-and-route experiments we present both low-stress ($W_{ls}$) and infinite-resource ($W_\infty$) critical path delay numbers. Reported numbers are normalized to VPR results.

From Table 2 we can see that `Lex-3` has the best average delay improvement among all proposed algorithms. We also observed that the average improvement for larger circuits is better then for smaller ones, which is encouraging since majority of circuits today belong to the large group. The average improvement over RT-Embedding for large circuits is 4% of the initial (unoptimized) delay. Indeed this may look as a small improvement, but it is averaged over 20 circuits and not all could be equally improved. The best improvement is 9% for the circuit `s38417`, improved from 0.93 to 0.84 (see Table 1). For `elliptic` we have 7% improvement and about 6% for `alu4`, `dsip`, `seq` and `apex2`. It should be pointed out that for circuits `misex3`, `diffeq`, `dsip`, `des`, `bigkey` and `s38584.1` we have reached theoretical lower bound, i.e. all flip-flop to flip-flop paths are monotone (assuming fixed flip-flop locations). Furthermore, most of the circuits in this test suite have very high density, all but 3 of them have density above 95% and 6 of them above 99%. For circuits `ex5p`, `apex4`, `seq`, `spla` and `ex1010` we ran out of free slots for replication and thus had to terminate early (for some of them we were able to replicate under 1% of total number of cells).

[2] In fact, we have implemented "Lex-N" version of the algorithm, but for values of N above 5 we cannot claim modest runtime overhead any longer.

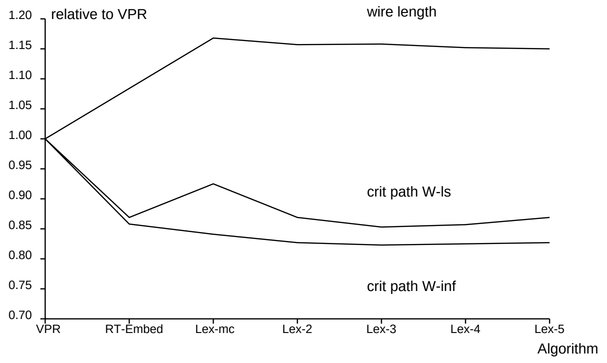

**Figure 8: Average improvement for all 20 MCNC benchmark circuits.**

The best improvement over VPR is almost 38% for circuit `pdc`, and we have 9 circuits with improvement larger than 20%. Amount of cells introduced by replication is on the average only 0.9% of the total number of cells. However, usage of wiring resources increased to 15.8% on the average compared to 8.4% increase of RT-Embedding approach. It looks a bit surprising that such small amount of replication can yield wiring overhead of this magnitude. Since our assumption was that we are optimizing circuits of high cell density, post-process unification was designed to be very aggressive in attempts to unify replicated cells as long as they do not violate current critical delay, hoping to clean-up intermediate replications as much as possible ([7] presents some statistics on unification). The biggest wiring overhead is almost 56% for circuit `dsip`. It turns out that cell density of `dsip` is only 47%, and it is not congested at all. Similarly `bigkey` has second larges wiring overhead of almost 33% and it's density is also low, only 58.5%. This suggests that we have to revisit unification strategy for circuits of smaller cell density.

Runtime overhead of our approach is very modest – under 5% of the time of VPR flow (place and route) and comparable to RT-Embedding.

In Figures 8, 9 and 10 we have plotted values from Table 2. On X-axis we have algorithm variant, and on Y-axis we have scale normalized to VPR results. It can be seen that all over-optimization approaches have similar max delay improvements and similar problems related to wiring resources. The exception is `Lex-mc` which developed local wiring congestion which can be seen from increased delay of low-stress routing.

## 6. DISCUSSION

We have shown that the reconvergence in circuit description presents an obstacle for further timing optimization by approach presented in [7]. Our proposed approach of optimizing sub-critical paths does seem to address the reconvergence issue, but still needs further improvements.

The issue of most concern is the potential overuse of wiring resources. We believe that the aggressive unification strategy creates excessive wire usage problems, which may lead to routing problems and degradation of circuit performance due to routing congestion (i.e. routes need to take detours around congested regions thus increasing interconnect de-

**Table 1: Comparison between Timing-Driven VPR, RT-Embedding and Lex-3**

| Circuit | Timing Driven VPR | | | | RT-Embedding normalized to VPR | | | | Lex-3 normalized to VPR | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | crit path [ns] | | wire | | crit path | | wire | | crit path | | wire | |
| | $W_\infty$ | $W_{ls}$ | length | blk | $W_\infty$ | $W_{ls}$ | length | blk | $W_\infty$ | $W_{ls}$ | length | blk |
| `ex5p` | 80.59 | 81.99 | 20020 | 1135 | 0.764 | 0.774 | 1.090 | 1.011 | 0.764 | 0.783 | 1.110 | 1.019 |
| `tseng` | 50.54 | 53.65 | 10495 | 1221 | 0.987 | 0.978 | 1.060 | 1.002 | 0.970 | 0.933 | 1.068 | 1.010 |
| `apex4` | 72.12 | 75.41 | 22332 | 1290 | 0.888 | 0.913 | 1.107 | 1.011 | 0.854 | 0.871 | 1.193 | 1.024 |
| `misex3` | 64.44 | 65.87 | 21784 | 1425 | 0.852 | 0.891 | 1.148 | 1.010 | 0.835 | 0.872 | 1.273 | 1.021 |
| `alu4` | 77.20 | 81.07 | 20796 | 1544 | 0.922 | 0.925 | 1.053 | 1.002 | **0.860** | 0.945 | 1.197 | 1.013 |
| `diffeq` | 55.29 | 57.49 | 15560 | 1600 | 0.989 | 0.969 | 1.026 | 1.001 | 0.999 | 0.990 | 1.020 | 1.002 |
| `dsip` | 65.38 | 67.21 | 17237 | 1796 | 0.793 | 0.804 | 1.277 | 1.001 | **0.731** | 0.822 | 1.559 | 1.001 |
| `seq` | 76.93 | 77.82 | 28493 | 1826 | 0.870 | 0.885 | 1.048 | 1.003 | **0.818** | 0.859 | 1.100 | 1.008 |
| `apex2` | 94.61 | 95.47 | 30998 | 1919 | 0.811 | 0.838 | 1.120 | 1.010 | **0.755** | 0.799 | 1.262 | 1.016 |
| `s298` | 124.20 | 127.35 | 22762 | 1941 | 0.915 | 0.903 | 1.034 | 1.001 | 0.875 | 0.899 | 1.066 | 1.002 |
| `des` | 90.44 | 91.31 | 27415 | 2092 | 0.876 | 0.876 | 1.039 | 1.001 | 0.876 | 0.886 | 1.043 | 1.002 |
| `bigkey` | 59.69 | 60.65 | 21074 | 2133 | 0.855 | 0.892 | 1.190 | 1.000 | 0.801 | 0.901 | 1.328 | 1.000 |
| `frisc` | 119.02 | 124.61 | 61109 | 3692 | 0.999 | 0.983 | 1.018 | 1.001 | 0.958 | 0.917 | 1.069 | 1.007 |
| `spla` | 111.03 | 113.57 | 68308 | 3752 | 0.812 | 0.824 | 1.108 | 1.008 | 0.793 | 0.829 | 1.164 | 1.008 |
| `elliptic` | 105.96 | 108.50 | 47456 | 3849 | 0.853 | 0.838 | 1.030 | 1.001 | **0.780** | 0.792 | 1.132 | 1.009 |
| `ex1010` | 184.84 | 185.56 | 70300 | 4618 | 0.818 | 0.847 | 1.148 | 1.006 | 0.795 | 0.821 | 1.144 | 1.006 |
| `pdc` | 167.81 | 169.33 | 105073 | 4631 | 0.641 | 0.707 | 1.072 | 1.005 | **0.624** | 0.690 | 1.142 | 1.009 |
| `s38417` | 97.20 | 100.61 | 64490 | 6541 | 0.930 | 0.944 | 1.017 | 1.000 | **0.840** | 0.888 | 1.069 | 1.009 |
| `s38584.1` | 99.74 | 102.10 | 58869 | 6789 | 0.842 | 0.839 | 1.048 | 1.001 | 0.819 | 0.845 | 1.115 | 1.000 |
| `clma` | 211.78 | 217.24 | 145551 | 8527 | 0.746 | 0.745 | 1.053 | 1.005 | 0.708 | 0.707 | 1.100 | 1.006 |
| **average** | | | | | **0.858** | **0.869** | **1.084** | **1.004** | **0.823** | **0.853** | **1.158** | **1.009** |
| **average for small ckts ($<$ 3K cells)** | | | | | **0.877** | **0.887** | **1.099** | **1.004** | **0.845** | **0.880** | **1.185** | **1.010** |
| **average for large ckts ($\geq$ 3K cells)** | | | | | **0.830** | **0.841** | **1.062** | **1.003** | **0.790** | **0.811** | **1.117** | **1.007** |

**Table 2: Comparison of average improvements (RT-Embedding, Lex-mc, Lex-2, Lex-3, Lex-4 and Lex-5)**

| Algorithm | Average normalized to VPR | | | | Average for small ckts normalized to VPR | | | | Average for large ckts normalized to VPR | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | crit path | | wire | | crit path | | wire | | crit path | | wire | |
| | $W_\infty$ | $W_{ls}$ | length | blk | $W_\infty$ | $W_{ls}$ | length | blk | $W_\infty$ | $W_{ls}$ | length | blk |
| `RT-Embedding` | 0.858 | 0.869 | 1.084 | 1.004 | 0.877 | 0.887 | 1.099 | 1.004 | 0.830 | 0.841 | 1.062 | 1.003 |
| `Lex-mc` | 0.841 | 0.925 | 1.168 | 1.013 | 0.852 | 0.951 | 1.197 | 1.014 | 0.824 | 0.886 | 1.124 | 1.010 |
| `Lex-2` | 0.827 | 0.869 | 1.157 | 1.008 | 0.850 | 0.889 | 1.185 | 1.010 | 0.794 | 0.838 | 1.114 | 1.006 |
| `Lex-3` | 0.823 | 0.853 | 1.158 | 1.009 | 0.845 | 0.880 | 1.185 | 1.010 | 0.790 | 0.811 | 1.117 | 1.007 |
| `Lex-4` | 0.825 | 0.857 | 1.152 | 1.008 | 0.848 | 0.889 | 1.175 | 1.009 | 0.790 | 0.809 | 1.117 | 1.006 |
| `Lex-5` | 0.827 | 0.869 | 1.150 | 1.008 | 0.849 | 0.901 | 1.168 | 1.008 | 0.795 | 0.823 | 1.124 | 1.008 |

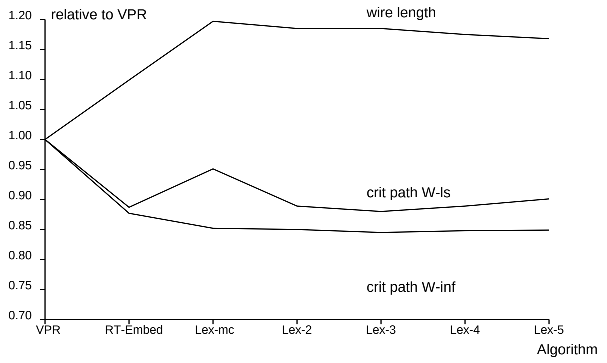

**Figure 9: Average improvement for small circuits.**

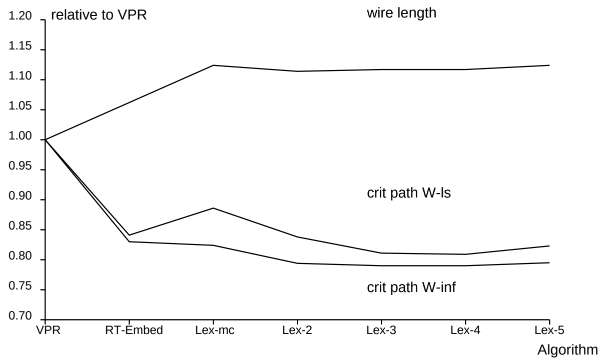

**Figure 10: Average improvement for large circuits.**

lay). At this point we need to experimentally verify this claim and revisit the unification strategy (and possibly make it adaptive to circuit density).

The tree embedding algorithm itself can be used to better address the routing issue, since in a certain extent it does perform global routing. We could run the actual routing algorithm on unoptimized circuit. Then use the actual channel occupancy to assign wire costs in the embedding graph which is then used by the tree embedder. In this way embedder is biased to place cells in regions with smaller wire utilization.

Another anomaly that we have observed is that some replication trees are constructed from sub-circuits that do not reconverge. In those cases, optimizing sub-critical paths seems unnecessary. This may explain why `Lex-5` which perform more powerful optimization does worse than `Lex-3` on the average, since it may over-optimize too many sub-critical paths that turned out to really be non-critical, thus consuming more free cell space and routing resources. A possible solution would be to invoke different versions of the tree embedding algorithm based on the amount of reconvergence that we encounter while constructing the replication tree.

## 7. CONCLUSIONS

We have presented a general and robust approach to logic re-sythesis via timing-driven, placement-coupled replication. The approach is similar in nature to [7] and it is designed to address issues that arise due to the *reconvergence* in the circuit specification. We build on the *Replication Tree* idea and modify the timing-driven fanin tree embedding algorithm to optimize sub-critical paths

Around these core ideas we have built an optimization engine for the FPGA domain and demonstrated promising preliminary experimental results.

We have identified possible obstacles for practical implementation of our approach and presented some of the possible directions for improvement.

Finally, we argue that the general ideas in this paper hold great promise beyond the context studied here. We suggest that the techniques can provide useful bridges between placement, routing and logic (re-)synthesis. Further, the graph-based modeling of the placement target would seem ideally suited to many practical problems (e.g., placement in the context of heterogeneous FPGA routing architectures).

## 8. REFERENCES

[1] G. Beraudo, J. Lillis, "Timing Optimization of FPGA Placements by Logic Replication," DAC, 2003.

[2] W. Gosti, A. Narayan, R.K. Brayton, A.L. Sangiovanni-Vincentelli, "Wireplanning in logic Synthesis," ICCAD, 1998.

[3] W. Gosti, S.P. Khatri, A.L. Sangiovanni-Vincentelli, "Addressing The Timing Closure Problem By Integrating Logic Optimization and Placement," ICCAD, 2001.

[4] S. Devadas, A. Ghosh, K. Keutzer, "Logic Synthesis," McGraw-Hill, 1994.

[5] M. Hrkić, J. Lillis, "S-Tree: A Technique for Buffered Routing Tree Synthesis," DAC, 2002.

[6] M. Hrkić, J. Lillis, "Buffer Tree Synthesis With Consideration of Temporal Locality, Sink Polarity Requirements, Solution Cost, Congestion and Blockages," IEEE Transactions on CAD, 2003.

[7] M. Hrkić, J. Lillis, "An Approach to Placement-Coupled Logic Replication," DAC, 2004.

[8] S.W. Hur, J. Lillis, "Mongrel: Hybrid Techniques for Standard Cell Placement," ICCAD, 2000.

[9] M. Jackson, E. Kuh, "Performance-driven Placement of Cell Based IC's," DAC, 1989.

[10] Y. Kukimoto, R. Brayton, P. Sawkary, "Delay-Optimal Technology Mapping by DAG Covering," DAC, 1998.

[11] J. Lillis, C.K. Cheng, T.T.Y. Lin, "Algorithms for Optimal Introduction of Redundant Logic for Timing and Area Optimization," Proc. IEEE International Symposium on Circuits and Systems, 1996.

[12] L.T. Liu, M.T. Kuo, C.K. Cheng, T.C. Hu, "A Replication Cut for Two-Way Partitioning," IEEE Transactions on CAD, 1995.

[13] W.K. Mak, D.F. Wong, "Minimum Replication Min-Cut Partitioning," IEEE Transactions on CAD, October 1997.

[14] A. Marquardt, V. Betz, J. Rose, "Timing-Driven Placement for FPGAs," International Symposium on FPGAs, 2000.

[15] A. Srivastava, R. Kastner, M. Sarrafzadeh, "Timing Driven Gate Duplication: Complexity Issues and Algorithms," ICCAD, 2000.