# Performance-driven Register Insertion in Placement

Dennis K. Y. Tong
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
kytong@cse.cuhk.edu.hk

Evangeline F. Y. Young
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
fyyoung@cse.cuhk.edu.hk

## ABSTRACT

As the CMOS technology is scaled into the dimension of nanometer, the clock frequencies and die sizes of ICs are shown to be increasing steadily [5]. Today, global wires that require multiple clock cycles to propagate electrical signal are prevalent in many deep sub-micron designs. Efforts have been made to pipeline the long wires by introducing registers along these global paths, trying to reduce the impact of wire delay dominance [2, 8].

The technique of retiming to relocate registers in a circuit without affecting the circuit functionality can be applied in this problem. Though the problem of retiming with gate and wire delay has been studied recently [17, 1], the placement of registers after retiming is a new challenge. In this paper, we study the problem of realizing a retiming solution on a global netlist by inserting registers in the placement to achieve the target clock period. In contrast to many previous works [16, 11] that performed simple calculations to determine the positions of the registers, our proposed algorithm can preserve the given clock period and utilize as few registers as possible in the realization. What is more, the algorithm is shown to be optimal for nets with 4 or fewer pins and this type of nets constitutes over 90% of the nets in a sequential circuit on average.

Using the ISCAS89 benchmark suite, we tested our algorithm with a 0.35$\mu$m CMOS standard cell library, and Silicon Ensemble was used to layout the design with row utilization of 50%. Experimental results showed that our algorithm can find the best sharing of registers for a net in most of the cases, i.e., using the minimum number of registers while preserving the target clock period, within a minute running on an Intel Pentium IV 1.5GHz PC with 512MB RAM.

## Categories and Subject Descriptors

B.7 [**Hardware**]: Integrated Circuits; B.7.2 [**Integrated Circuits**]: Design Aids—*layout, placement and routing*


*ISPD'04,* April 18–21, 2004, Phoenix, Arizona, USA.


## General Terms

Performance

## Keywords

Register insertion, post-retiming, placement

## 1. INTRODUCTION

As the CMOS technology continues to scale down, the clock frequencies and die sizes of integrated circuits are increasing steadily. Data showed that the frequencies of high-performance ICs have doubled every process generation while the die sizes increased by about 25% [5]. With such short cycles and long interconnects, it is not uncommon for a global signal to take multiple clock periods to travel across a chip. In order to alleviate this problem, we need to be able to insert registers to pipeline long global interconnect [2, 8]. Nevertheless, arbitrary insertion of registers along a wire is forbidden because it would change the original functionality of the circuit. As a result, retiming, a sequential circuit optimization technique to relocate registers without affecting circuit functionality, can be applied [9, 10].

Retiming is a sequential circuit optimization technique that relocates registers while maintaining the functionality of the circuit. This powerful technique can be used to optimize the clock period, the usage of registers, or a combination of both in a sequential circuit. Since retiming was firstly formulated a decade ago, much effort has been made to improve the efficiency of the technique [15] or to apply this technique in various areas like physical planning [4], circuit partitioning [12], power reduction [14], testability [6] and so on. Recently, there are some research efforts on addressing the retiming problem with dominant global wire delays [17, 1]. In contrast to the traditional settings of retiming where wire delay is ignored, these new methods can handle both gate and wire delay simultaneously. Their assumption that the delay of a wire will grow linearly with its length can be applied in our case of global interconnects as optimal buffering can be performed.

However, the placement of registers after retiming is another new challenge. Since a net is represented as a branch of edges in a retiming graph model which does not bear any information about the topology of the routes or the positions of the registers. It is unknown whether the clock period obtained from retiming can be realized in the design. Being able to insert registers into a placement solution in order to realize a retiming solution and preserve the corresponding clock period is important, or it will make the retiming

optimization meaningless. Meanwhile, minimizing the number of registers used is also essential as the size of a register is usually several times larger than that of a simple gate, regardless of the process technology being used.

Even though there are several previous works on post-retiming register placement, many of them suffer from the problem of over-simplification when wire delay dominates. For example, in [16], the authors assume that the position of a register is located at the geometric center of the connected gates. This assumption is natural, but the clock period resulted from retiming will be easily violated. A similar problem occurs in [11] in which the authors determine the position of a register such that the sum of the lengths of the nets connected to that register is minimized. Although both of these methods provide a fast estimation of register position, the optimized clock period obtained from retiming cannot be achieved in the layout when interconnect delay is a dominant factor of the circuit delay. A sophisticated method to tackle this problem is of utmost need.

In this paper, we study the problem of realizing a retiming solution on a global netlist by inserting registers in placement to achieve the target clock period. This problem involves two main sub-problems, namely, *topology finding* and *register placement*. As we have mentioned before, a net is modeled as a branch of edges in the retiming graph, the problem of *topology finding* refers to the determination of an optimal sharing of the registers among the fanout edges of a net given the geometric positions of the connected gates such that the optimal clock period obtained from retiming can be preserved. After obtaining the topology tree of a net, we need to find an appropriate position for each register given the constraints in placement (some occupied areas do not allow register insertion) and this problem is known as *register placement*. Given a placement (we used standard cell design in our experiments) and a retiming solution (we used the technique in [1] to generate the retiming solutions in our experiments), our proposed algorithm can insert registers into the placement solution to preserve the clock period as much as possible. Notice that our algorithm has no dependency of the retiming algorithm being used, as long as it considers wire delay of global nets and gives a retiming solution with a target clock period, retiming labels and the maximum arrival time at each gate output.

Our algorithm can find the optimal topology, i.e., using the minimum number of registers while preserving the clock period, for 4 or fewer pin nets. Since nets with 4 or fewer pins constitute, on average, over 90% of nets in a circuit, the proposed algorithm offered an agreeable performance in the experiments. Nearly all the nets had their best topology found and registers inserted successfully while maintaining the clock period.

The remainder of this paper is organized as follows. We present the problem statement in Section 2. The problem of retiming with gate and wire delay is briefly reviewed in Section 3. In Section 4, our proposed algorithm for topology finding and register placement will be presented. Experimental results are presented in Section 5 and a conclusion follows in Section 6.

## 2. PROBLEM FORMULATION

PROBLEM STATEMENT. *Given a placed sequential circuit and a retiming solution, i.e., an optimal clock period clk, a retiming label $r(v)$ at each gate $v$ and the maximum arrival time $a(v)$ at the output of gate $v$, we want to insert registers into the circuit layout to realize the retiming solution, preserving the clock clk as much as possible.*

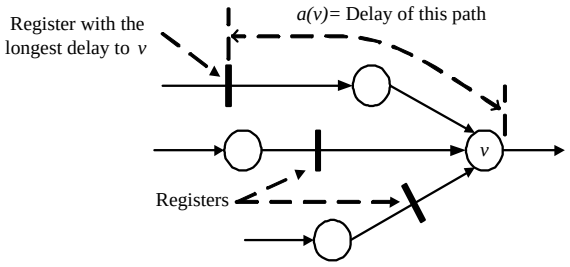


**Figure 1: An illustration of the definition of $a(v)$ for a gate $v$.**

Fig. 1 shows an example that illustrates the definition of $a(v)$.

We can represent the circuit as a graph $G(V, E)$, where each vertex $v \in V$ corresponds to a combinational gate, and each directed edge $e_{uv} \in E$ represents a connection from the output of gate $u$ to the input of gate $v$. Let $w(u, v)$ be the number of registers along the edge $e_{uv}$, $d_{uv}$ be the wire delay of edge $e_{uv}$ if no register lies along the edge. Note that the wire delay $d_{uv}$ is assumed to be proportional to the shortest Manhattan distance between $u$ and $v$.

Now, consider a net $N(s, D, L)$, where $s$ denotes the driving gate, $D$ denotes the set of all driven gates, and $L$ denotes the set of interconnections between $s$ and each of the gates $d_i \in D$. Obviously, $\{s\} \cup D \subseteq V$ and $L \subseteq E$. For each edge $e_{sd_i} \in L$, we have a value $w_r(s, d_i)$ representing the number of registers along the edge $e_{sd_i}$ after retiming. The problem is to insert the minimum number of registers for this net into the circuit according to the retiming solution such that the clock period is preserved as much as possible. This problem comprises two main sub-problems known as *topology finding* and *register placement*. *Topology finding* is the problem of finding a topology, $\Upsilon_N$, of net $N$ given the exact geometric positions of the gates such that the minimum number of registers is used and the target clock period is preserved. *Register placement* is the problem of finding the corresponding position for each register given the topology $\Upsilon_N$ of net $N$.

A topology $\Upsilon_N = (P, K)$ is a tree (an acyclic graph with no designated root yet) that describes the structure of net $N$ on the plane. Each node $p \in P$ corresponds to either a combinational gate or a register, and each edge $k_{uv} \in K$ represents a physical connection between gate $u$ and gate $v$. Each node $p \in P$ that has only one adjacent node in $\Upsilon_N$, i.e., $deg(p) = 1$, represents a combinational gate while an internal node $p \in P$ that has more than one adjacent node, i.e., $deg(p) > 1$, represents a register. In fig. 2, an example of a 4-pin net in which each *source-to-sink* edge has a register after retiming is shown. There are five possible register sharing topologies in this example: (a) all the edges share a single register (maximum sharing) as shown in fig. 3; (b) each edge has its own register (no sharing) as shown in fig. 4; (c) for the rest three equivalent cases, two of the edges share a single register while the other has a separate one, as shown in fig. 5.

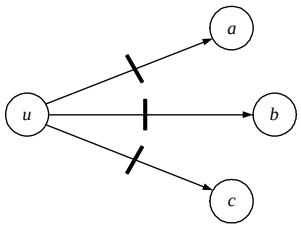


**Figure 2: Graph model of a 4-pin net in which each edge has a single register after retiming.**

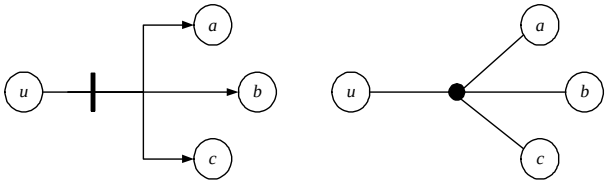


**Figure 3: Case (a) - a single register is shared among the edges (maximum sharing). The topology tree of this configuration is shown on the right.**

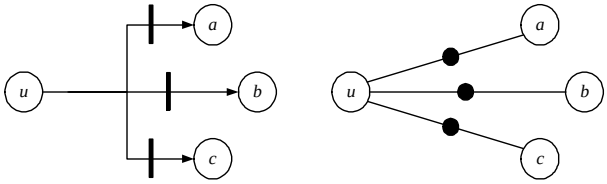


**Figure 4: Case (b) - each edge has a separate register (no sharing). The topology tree of this configuration is shown on the right.**

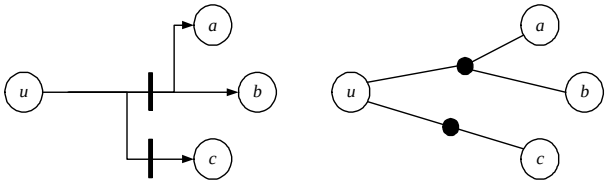


**Figure 5: Case (c) - two of the edges share a register while the other has a separate one. The topology tree of this configuration is shown on the right.**

The above figures show the topology trees of the 4-pin net in the three different cases and the corresponding physical configurations.

Although we can always identify the topology tree which has the maximum sharing of registers for a net, it is not always possible to place the registers on a chip using that topology while preserving the given clock period. Using case (a) in fig. 3 as an example, suppose the clock period resulted from retiming, $clk$, equals 1.5 units and the positions of gate $u$, $a$, $b$ and $c$ are $(0,0)$, $(-3,0)$, $(0,3)$ and $(3,0)$ respectively, as depicted in fig. 6. Obviously, it is impossible to share a single register among the three edges without clock violation. Three separate registers have to be allocated and inserted exactly at $(-1.5,0)$, $(0,1.5)$ and $(1.5,0)$ for edge $e_{ua}$, $e_{ub}$ and $e_{uc}$ respectively in order to satisfy $clk$.

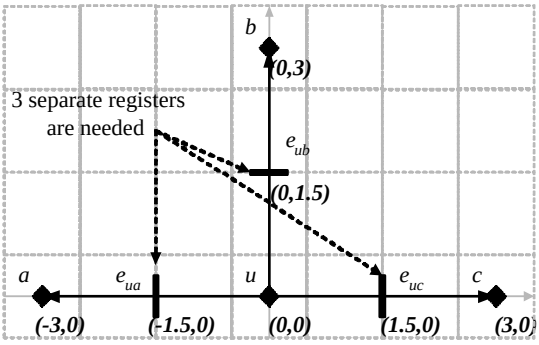


**Figure 6: A situation in which the registers cannot be shared in order to preserve the clock period $clk$ = 1.5 units.**

Even if we have a feasible topology tree, it can happen that the suggested position for a register has been occupied by some other gates, i.e., the target area is blocked, and we have to look for another appropriate position. Section 4 will address how a feasible topology tree can be found and how the positions of the registers can be finalized.

# 3. REVIEW OF RETIMING WITH GATE AND WIRE DELAY

In this section, we briefly mention the technique of retiming with gate and wire delay in [1] which we have employed to generate a retiming solution for our proposed register insertion algorithm to work on. In [1], it is assumed that wire delay is proportional to the length of the wire segment and each gate is associated with a gate delay. Given a sequential circuit, they modeled it as a retiming graph model, $G(V, E)$, as usual, except that each edge has an additional attribute of wire delay. Besides, they have also defined a variable, $a(v)$, as the maximal arrival time at the output of gate $v$ for all $v \in V$.

Two approaches were proposed to solve the problem optimally and near-optimally. Both approaches were based on a transformation of the problem into a system of linear difference inequities such that the Bellman-Ford algorithm could be used to test the feasibility of a trial clock period. Together with the technique of binary search, the optimal or a near-optimal clock $clk$, the retiming labels and the value of $a(v)$ for all $v \in V$ would be obtained.

Since the near-optimal approach runs much faster than the optimal one, we chose the former method to produce the retiming solutions. Our register insertion algorithm will insert the registers with clock preservation accordingly.

# 4. PLACEMENT OF REGISTERS AFTER RETIMING

## 4.1 Topology Finding

In this section, an algorithm is proposed to find the topology of a net given the constraints in placement such that maximum sharing of registers is achieved and the clock period is preserved. This method can find the optimal topology for a net with 4 or fewer pins, and can give near-optimal solution for a net with 5 or more pins according to the experimental results.

### 4.1.1 Algorithm Description

Given a net $N(s, D, L)$, a clock period $clk$, and the maximal arrival time at the output of gate $v$, $a(v)$, we can obtain a feasible topology tree of $N$, $\Upsilon_N$, as described below.

First, we construct the best possible topology $\Upsilon_{N_{opt}}$ for $N$, i.e., a topology having the minimum number of internal nodes (an internal node represents a register). Obviously, the number of internal nodes in $\Upsilon_{N_{opt}}$ equals $Q = max_{d_i \in D}\{w_r(s, d_i)\}$, where $w_r(s, d_i)$ denotes the number of registers on the edge $e_{sd_i}$ after retiming. We label each internal node as $r_i$ representing the $i$-th register on the net from the source $s$ for $1 \leq i \leq Q$. An example of the retiming graph model and the corresponding best possible topology $\Upsilon_{N_{opt}}$ for a 4-pin net is shown in fig. 7.

We call the region of the plane where a register $r$ can be placed the *candidate region* of $r$ and is denoted by $C(r)$. For consistency, the candidate region $C(v)$ of a combinational gate $v$ is the position of $v$ itself, i.e., its coordinates $(x_v, y_v)$, since $v$ is fixed after placement. An *$\delta$-extended region* of a region $\Re$, denoted by $R^{+\delta}(\Re)$, is the region of the plane at a distance $\delta$ or less from some points in $\Re$, assuming that the distance between two points is measured by their shortest Manhattan distance.

Besides, we define an *adjacent-gate region* for each node $p$ in a topology tree, denoted by $A(p)$, as an $\delta$-extended region from its candidate region $C(p)$, i.e., $A(p) = R^{+\delta}(C(p))$ where $\delta$ is defined differently for different types of nodes. The physical meaning of $A(p)$ refers to the region on the plane such that it encompasses *all* the possible positions for an adjacent gate of $p$ in the net. Therefore, the value of $\delta$ for $A(p)$ of a node $p$ is described as follows. If node $p$ is an internal node, $\delta$ equals $clk$. If node $p$ represents a driven gate, $\delta$ equals $a(p) - d_p$, where $a(p)$, given by the retiming solution, is the maximum arrival time at the output of gate $p$ and $d_p$ is the gate delay of $p$. Otherwise, node $p$ represents the driving gate, and we set $\delta$ to $clk - a(p)$. Notice that all these regions are 45°-rotated rectangles on the rectilinear plane because of the Manhattan distance measurement.

Starting from the best possible topology $\Upsilon_{N_{opt}}$, we will modify the topology incrementally until an optimal feasible topology $\Upsilon_N$ is obtained for net $N$. First of all, we choose the node that represents the driving gate $s$ as the root in $\Upsilon_{N_{opt}}$ and direct all the edges away from $s$. Then, we will process each internal node $r_i$ in $\Upsilon_{N_{opt}}$ from $i = Q$ to $i = 1$, i.e., from the furthest register to the closest register to the driving gate $s$, in the following manner.

For each internal node $r_i$ with a set of children $q_1, \ldots, q_m$, find a minimal set of all the overlapping regions between $A(q_j)$ for $1 \leq j \leq m$, denoted by $Y_{min} = (y_1, \ldots, y_k)$, such that the union of the elements in $Y_{min}$ covers at least one point from each region $A(q_j)$. For each $y_l$ in $Y_{min}$, we call the number of regions that $y_l$ has covered at least a point as the *size* of $y_l$, denoted by $s(y_l)$. Sort the elements in $Y_{min}$ in a non-ascending order of their sizes from 1 to $k$, using a greedy procedure ALGSETY as described below.

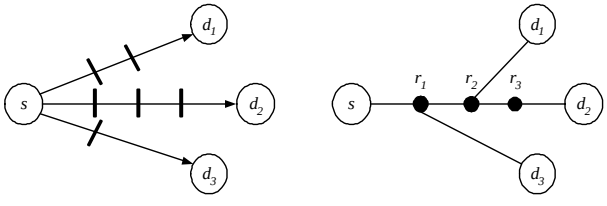


**Figure 7: The retiming graph model (left) and the corresponding best possible topology $\Upsilon_{N_{opt}}$ (right) of a 4-pin net example.**

```
Procedure ALGSETY(r_i, Υ_N);
begin
  overlapped := a boolean flag;
  Y_min ⟵ φ;
  add A(q_1) to Y_min, i.e., y_1 ⟵ A(q_1);
  for j = 2 to m
    overlapped ⟵ false;
    for l = 1 to |Y_min|
      if (y_l ∩ A(q_j) ≠ φ)
        y_l ⟵ y_l ∩ A(q_j);
        sort elements in Y_min in a non-ascending order
        of their sizes from 1 to |Y_min|;
        overlapped ⟵ true;
        break;
      end if;
    end for;
    if (overlapped == false)
      increment |Y_min| by 1;
      add A(q_j) to Y_min at the end, i.e., y_|Y_min| ⟵
      A(q_j);
    end if;
  end for;
  OUTPUT(Y_min);
end.
```

Notice that the union of the elements in $Y_{min}$ covers at least one point from each region, $A(q_j)$, for $1 \leq j \leq m$. Next, we can remove all the edges from $r_i$ to its children $q_1, \ldots, q_m$ in $\Upsilon_{N_{opt}}$, split the node $r_i$ into $k$ new internal nodes, $n_1, \ldots, n_k$, where node $n_l$ corresponds to element $y_l$ in $Y_{min}$ for $1 \leq l \leq k$. In addition, we will assign region $y_l$ as the candidate region of $n_l$, i.e., $y_l = C(n_l)$, for all $l$.

Starting from the $y_l$ whose $s(y_l)$ is the largest in $Y_{min}$, add an edge from $n_l$ to each $q_j$ that has no parent node and whose $A(q_j)$ is covered by $y_l$. Repeat this step until all $y_l$ have been processed. Finally, add an edge from the parent node of $r_i$ to every newly generated internal nodes $n_l$ and $r_i$ can then be removed from the topology tree. The above operations are described in the procedure ALGMODITREE below.

```
Procedure ALGMODITREE(r_i, Y_min, Υ_N);
begin
  remove all the edges from r_i to its children q_1, ..., q_m in
  Υ_N;
  instantiate k new internal nodes, n_1, ..., n_k, where k =
  |Y_min|;
  assign region y_l as the candidate region of n_l, i.e.,
  y_l = C(n_l), for all l;
  for l = 1 to k
    for j = 1 to m
      if (y_l ∩ A(q_j) ≠ φ and q_j has no parent node)
        add an edge from n_l to q_j;
      end if;
    end for;
    add an edge from the parent node of r_i to n_l;
  end for;
  remove r_i;
  OUTPUT(Υ_N);
end.
```

After visiting all the internal nodes $r_i$ in $\Upsilon_{N_{opt}}$ and modifying the topology as described above, we will get a new topology tree $\Upsilon_N$ at the end such that the clock period *clk* is preserved. The whole algorithm for *topology finding* of a net $N$ is described in the procedure ALGTOPOTREE.

*Procedure* ALGTOPOTREE($N$);
begin
  construct the best possible topology $\Upsilon_{N_{opt}}$ for net $N$;
  $\Upsilon_N \longleftarrow \Upsilon_{N_{opt}}$;
  *for* $i = Q$ *to* 1
    $Y_{min} \longleftarrow$ ALGSETY($r_i$, $\Upsilon_N$);
    $\Upsilon_N \longleftarrow$ ALGMODITREE($r_i$, $Y_{min}$, $\Upsilon_N$);
  end for;
  OUTPUT($\Upsilon_N$);
end.

### 4.1.2 Optimality Proof

To prove the correctness of the above algorithm, we have the three following lemmas. However, the proofs of the first two lemmas are omitted due to the limitation in space.

LEMMA 1. *Given a set of n 45°-rotated rectangles $R_1, \ldots, R_n$ on a rectilinear plane, if $R_1 \cap \ldots \cap R_n \neq \phi$, then $R^x(R_1) \cap \ldots \cap R^x(R_n) \neq \phi$, where $x$ is a non-negative real number.*

LEMMA 2. *Given a set of n 45°-rotated rectangles $R_1, \ldots, R_{n-1}$ and $S$ on a rectilinear plane, if $S \cap R_i \neq \phi$ for $1 \leq i \leq n-1$ and $R_1 \cap \ldots \cap R_{n-1} \neq \phi$, then $S \cap (R_1 \cap \ldots \cap R_{n-1}) \neq \phi$.*

LEMMA 3. *Given two 45°-rotated rectangles, $A$ and $B$, on a rectilinear plane, we denote the n times clk-extended regions of $A$ and $B$ as $A_n$ and $B_n$ respectively, i.e., $A_n = R^{+(n \times clk)}(A)$ and $B_n = R^{+(n \times clk)}(B)$. Suppose $A \cap B = R_{AB} \neq \phi$ , we denote the n times clk-extended region of $R_{AB}$ by $(R_{AB})_n$, i.e., $(R_{AB})_n = R^{+(n \times clk)}(R_{AB})$. It is claimed that if there exists a point $x \in A_n \cap B_n$, $x \in R^{+clk}((R_{AB})_{n-1})$ for all $n \geq 1$.*

PROOF. We prove by induction on $n$.

Base case:

Consider the case when $n = 1$. Suppose $x \in A_1 \cap B_1$, the *clk*-extended region from the position of $x$ is given by $R^{+clk}(x)$. Obviously, $R^{+clk}(x) \cap A_0 \neq \phi$ and $R^{+clk}(x) \cap B_0 \neq \phi$ because $x \in A_1 \cap B_1$. Since $A_0 \cap B_0 \neq \phi$ ($\because A_0 = A, B_0 = B$ and $A \cap B \neq \phi$), $R^{+clk}(x) \cap (A_0 \cap B_0) \neq \phi$ by *lemma 2*. Therefore, $x \in R^{+clk}((R_{AB})_0)$ and the claim is true for $n = 1$.

Inductive step:

Assume that the claim is true for $n = j - 1$, where $j$ is a positive integer $\geq 2$, i.e., if there exists a point $x \in A_{j-1} \cap B_{j-1}$, $x \in R^{+clk}((R_{AB})_{j-2})$. Consider the case when $n = j$. Given a point $x \in A_j \cap B_j$ and the *clk*-extended region from its position is denoted by $R^{+clk}(x)$. Obviously, $R^{+clk}(x) \cap A_{j-1} \neq \phi$ and $R^{+clk}(x) \cap B_{j-1} \neq \phi$ because $x \in A_j \cap B_j$. By *lemma 1*, since $A_0 \cap B_0 \neq \phi$, $A_{j-1} \cap B_{j-1} \neq \phi$. Therefore, $R^{+clk}(x) \cap (A_{j-1} \cap B_{j-1}) \neq \phi$ by *lemma 2*. By the induction hypothesis, if $R^{+clk}(x) \cap (A_{j-1} \cap B_{j-1}) \neq \phi$, $R^{+clk}(x) \cap R^{+clk}((R_{AB})_{j-2}) \neq \phi$. Therefore, $x \in R^{+clk}((R_{AB})_{j-1})$. □

THEOREM 1. *The proposed algorithm finds a topology that maximizes the sharing of registers for an i-pin net, where $2 \leq i \leq 4$, and the given clock period clk is preserved.*

PROOF. We prove the three possible cases one-by-one.

Case 1: $i = 2$

This case is trivial because there is only one source $s$, one sink $t_1$ and one edge $e_{st_1}$ in a 2-pin net, there is no other edges to share registers with. The algorithm will start from the furthest internal node $r_Q$ and take the adjacent-gate region of $t_1$, $A(t_1) = R^{+(a(t_1)-d_{t_1})}(C(t_1))$, as the candidate region of $r_Q$, i.e., $C(r_Q) = A(t_1)$. Next, the algorithm will process node $r_{Q-1}$ and take the adjacent-gate region of $r_Q$, $A(r_Q) = R^{+clk}(C(r_Q))$, as the candidate region of $r_{Q-1}$, i.e., $C(r_{Q-1}) = A(r_Q)$.

By substitution, $C(r_{Q-1})$ can be represented as an extended region from the position of the sink $t_1$, as $C(r_{Q-1}) = R^{+((a(t_1)-d_{t_1})+clk)}(C(t_1))$. The algorithm repeats the above steps until it reaches the first internal node $r_1$ where $C(r_1) = R^{+((a(t_1)-d_{t_1})+(Q-1)\times clk)}(C(t_1))$. Since the retiming solution is valid, the distance between $s$ and $t_1$ will not exceed $(clk - a(s)) + ((Q-1) \times clk) + (a(t_1) - d_{t_1})$. Therefore, the algorithm will find the candidate regions for every register and return the best possible topology when it terminates.

Case 2: $i = 3$

Given a 3-pin net, let $s$ be the source, and $t_1$ and $t_2$ be the two sinks. Let $w_r(s, t_1)$ and $w_r(s, t_2)$ be $p$ and $q$ respectively, where $1 \leq p \leq q$. Suppose that there exists a topology tree of maximum register sharing for the 3-pin net such that the first $k$ registers, where $1 \leq k \leq p$, are shared (notice that if the $k$-th register can be shared, the $h$-th register can be shared where $1 \leq h \leq k$), and that the algorithm cannot find such a topology.

Since the algorithm cannot find that optimal topology, it must fail to find an overlapping region for the $k$-th register to be shared. At the point of failure, the algorithm should find that the regions $R^{+((a(t_1)-d_{t_1})+clk\times(p-k-1))}(t_1)$ and $R^{+((a(t_2)-d_{t_2})+clk\times(q-k-1))}(t_2)$ do not overlap. However, these two regions encompass *all* the possible positions for the placement of the $k$-th register from $t_1$ and $t_2$ respectively such that the clock period *clk* would not be violated. Therefore, should the $k$-th register be able to be shared as assumed, it must lie within these two regions and the algorithm must be able to find it. Contradiction occurs.

Case 3: $i = 4$

Given a 4-pin net, let $s$ be the source, and $t_1$, $t_2$ and $t_3$ be the three sinks. Let $w_r(s, t_1)$, $w_r(s, t_2)$ and $w_r(s, t_3)$ be $p$, $q$ and $r$ respectively, where $1 \leq p \leq q \leq r$. Suppose the algorithm is attempting to share the $k$-th register where $1 \leq k \leq p$, i.e., it is trying to find a minimal subset of the overlapping regions such that it covers all the extend regions $R^{+((a(t_1)-d_{t_1})+clk\times(p-k-1))}(t_1)$, $R^{+((a(t_2)-d_{t_2})+clk\times(q-k-1))}(t_2)$ and $R^{+((a(t_3)-d_{t_3})+clk\times(r-k-1))}(t_3)$, denoted by $A$, $B$ and $C$ respectively. Notice that we only consider when $k \leq p$ and assume that the three paths from $s$ to $t_1$, $t_2$ and $t_3$ are not merged yet (i.e., no sharing of registers from $k+1$ to $r$). It is because, otherwise, the situation will fall into case 1 or case 2 discussed above.

There are 4 distinct cases. First, if $A$, $B$ and $C$ are disjoint, it means that the $k$-th register cannot be shared and the algorithm will introduce three new internal nodes to represent the registers and continues with the next internal node $r_{k-1}$. Second, if $A$, $B$ and $C$ overlap with each other, it means that the $k$-th register can be shared among $t_1$, $t_2$ and $t_3$. The algorithm will introduce a single internal node to represent the register and continues. The correctness of the algorithm in these two cases is trivial and will not be elaborated here.

The third case is, without loss of generality, that $A \cap B \neq \phi$ and $B \cap C \neq \phi$ but $A \cap C = \phi$. Denote the region $A \cap B$ as $R_{AB}$ and the region $B \cap C$ as $R_{BC}$. There are three possible options that the algorithm can choose from when evaluating the $k$-th register: (1) it does not share the $k$-th register and introduces three different registers for the sinks; (2) it shares the $k$-th register between $t_1$ and $t_2$ but a separate one for $t_3$; (3) it shares the $k$-th register between $t_2$ and $t_3$ but a separate one for $t_1$. Our algorithm will choose arbitrarily either (2) or (3) as the number of adjacent-gate regions covered by $R_{AB}$ and $R_{BC}$ are the same, but it will never choose (1). We assume that the algorithm chooses (2) in the following analysis.

First, we compare the choices of (1) and (2). Notice that (1) can be better than (2) only when the three separate paths can be merged together at a subsequent step of processing register $h$ where $1 \leq h \leq k$, while the combined path of $t_1$ and $t_2$ and the path of $t_3$ cannot be merged at the $h$-th register. We are going to show that this will not happen.

If we choose (1), suppose that there exists a point $x$ on the plane such that $x \in A_j \cap B_j \cap C_j$, where $A_j$, $B_j$ and $C_j$ represent the $j$ times $clk$-extended regions of $A$, $B$ and $C$ respectively, during a subsequent step of processing register $h$ where $1 \leq h \leq k$. By *lemma 3*, it is shown that $x \in R^{+clk}((R_{AB})_{j-1})$, where $(R_{AB})_{j-1}$ is the $(j-1)$ times $clk$-extended region from $R_{AB}$. This means that if it is possible to share the $h$-th register among the three edges without sharing the $k$-th register at the first place, by choosing (2), i.e., to share the $k$-th register between $t_1$ and $t_2$, the algorithm will also be able to share the $h$-th register among the edges. Therefore, (2) is better than (1) by sharing more registers.

Next, we compare the choices of (3) and (2) similarly. Suppose we choose (3) and there exists a point $x$ on the plane such that $x \in A_j \cap (R_{BC})_j$, where $A_j$ and $(R_{BC})_j$ represent the $j$ times $clk$-extended regions of $A$ and $R_{BC}$ respectively, during a subsequent step of processing register $h$ where $1 \leq h \leq k$. Obviously, there exists a point $y$ covered by $A_j \cap B_j \cap C_j$, i.e., $y \in A_j \cap B_j \cap C_j$. By *lemma 3*, $y \in R^{+clk}((R_{AB})_{j-1})$, i.e., $y \in R^{+clk}((R_{AB})_{j-1}) \cap C_j$, so the $h$-th register will also be shared among the three edges by choosing (2). Therefore, (2) is no worse than (3). As a result, the algorithm will find the optimal solution by choosing arbitrarily either (2) or (3) (using the greedy algorithm).

Finally, if two pairs of the regions overlap while the other is disjoint, i.e., $A \cap B \neq \phi$ but $A \cap C = \phi$ and $B \cap C = \phi$, the analysis is similar to the third case above. □

## 4.2 Register Placement

In this section, we discuss how registers are actually placed using the topology tree yielded from the algorithm discussed in the previous subsection. Using an idea similar to the technique in [13, 7], the positions of the registers are determined.

Since some of the chip areas are occupied by the standard cells, we need to know where on the chip a register can be placed. To tackle this problem, we divide the chip into a mesh of $m \times n$ grids. For each grid $g_{ij}$, we keep track of its center coordinates, $(x_{g_{ij}}, y_{g_{ij}})$, and the size of the free space in the grid, $f(g_{ij})$. We finalize the position for a register in the following manner.

Given a topology tree $\Upsilon_N$, choose arbitrarily an internal node $r$ to be the root of $\Upsilon_N$, and direct the edges of $\Upsilon_N$ away from $r$. Starting from the root $r$, we choose a grid

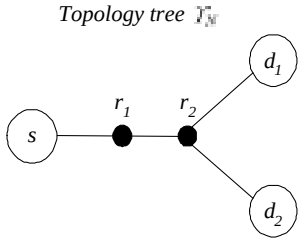


**Figure 8: The topology tree of a 3-pin net $\Upsilon_N$ where $r_1$ and $r_2$ are two shared registers.**

whose center is contained in $C(r)$, i.e., the candidate region for placing the register $r$, and it has the largest free space available. We denote this grid as $g(r)$. If $f(g(r)) \geq z$, where $z$ denotes the size of a register, we take the center of $g(r)$ as the position of the register $r$. Otherwise, we allow a controlled degree of inaccuracy by extending $C(r)$ one grid width further, i.e., $R^{+g_w}(C(r))$, where $g_w$ represents the width of a grid. Repeat the same process using $R^{+g_w}(C(r))$ instead of $C(r)$ in the search of a feasible grid for placing register $r$. If no such grid is found, we report that this register cannot be placed. This could happen because the register counts may increase greatly after retiming.

Let $q_1, \ldots, q_m$ be the set of internal nodes which are the children of $r$ in a topology tree $\Upsilon_N$. After fixing the position of $r$, register $q_j$, for $1 \leq j \leq m$, is placed arbitrarily in its candidate region $C(q_j)$ provided that it is at a distance of $clk$ or less from $r$. After visiting all the internal nodes of $\Upsilon_N$, the position of each register is located.

Suppose we have a 3-pin net $N(s, D, L)$ and its topology tree $\Upsilon_N$ is shown in fig. 8. The topology tree $\Upsilon_N$ shows that the two driven gates $d_1$ and $d_2$ will share two registers represented by the internal node $r_1$ and $r_2$. In this example, we assume that $clk = 3$ units. Consider a $5 \times 5$ mesh as shown in fig. 9, where the positions of the driving gate $s$ and the two driven gates, $d_1$ and $d_2$, are assumed to be the centers of the grids containing them correspondingly, i.e., gate $s$ is located at $(4, 0)$, gate $d_1$ is located at $(0, 4)$ and gate $d_2$ is located at $(2, 4)$. Suppose $\Upsilon_N$ is rooted at node $r_1$ and the algorithm has fixed its position at $(1, 0)$, let us examine how the position of $r_2$ is determined.

The candidate region $C(r_2)$ of $r_2$ covers the centers of grids $g_{03}$, $g_{04}$, $g_{12}$, $g_{13}$, $g_{14}$, $g_{23}$ and $g_{24}$. Starting from the position of register $r_1$, the algorithm expands a rectangle of distance $clk$ from it, denoted by $R^{+clk}(r_1)$ as shown. Next, the algorithm will find that $C(r_2) \cap R^{+clk}(r_1)$ is not empty and covers the center of grid $g_{12}$ and $g_{13}$ - the candidate positions of register $r_2$. If the free space of $g_{12}$ is greater than that of $g_{13}$, i.e., $f(g_{12}) \geq f(g_{13}) \geq z$, the algorithm will assign the center of $g_{12}$ as the position of register $r_2$.

# 5. EXPERIMENTAL RESULTS

We performed retiming and our proposed register placement algorithm on the ISCAS89 benchmark suite. The program was implemented in C language and run on a 1.5GHz Intel Pentium IV processor with 256KB cache and 512MB RAM.

In our experiments, we implemented the circuits using a 0.35$\mu$m CMOS standard cell library from Austria Micro Systems and Silicon Ensemble was used to layout the design

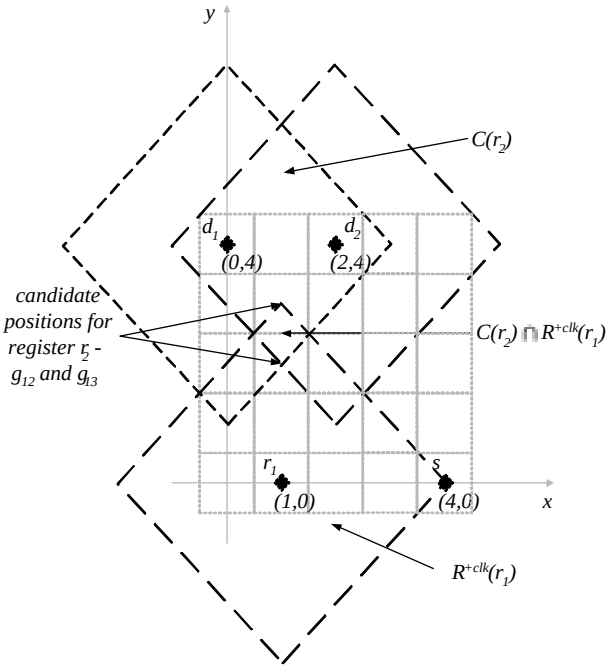


**Figure 9: An illustration of how the final position for the register $r_2$ is determined.**

with a setting of 50% row utilization. Gate delays were referenced from the data book while wire lengths were estimated using the Manhattan distance between the connected cells. We scaled the wire delay according to [3] in which a 1mm wire was assumed to have a delay of 150ps approximately. The size of a grid was set to twice as large as a D-type flip flop. During the placement of a register, we allowed an error of one-grid width, i.e., the width of a D-type flip flop.

The results are shown in Table 1. The first column indicates the name of the circuits. The second column shows the number of logical registers existed in the retiming graph model after retiming. The number of registers had increased after retiming for most of the circuits because the retiming method that we used did not minimize the number of registers as one of its objectives. Although this increase in register counts does hinder our algorithm to place registers, it is not the main concern addressed in this paper.

In the third column, the minimum possible number of register required after sharing is shown, i.e., assuming that every net could be realized using the best topology. The fourth column shows the number of registers that have actually been inserted after using our proposed algorithm. It can be observed that the numbers in the fourth column are the same as those in the third column except for circuit s3271 and s4863. This observation showed that almost all the nets in our test cases could have their registers placed using the best topology, revealing that our proposed algorithm can very often find a near-optimal solution for register insertion.

The fifth column shows the statistics of the number of nets containing 4 or fewer edges with registers whereas the sixth column shows the number of nets having 5 or more edges with registers. The seventh column shows the number of registers that are placed within their candidate regions while the eighth column shows the number of registers that are placed outside their candidate regions but with a controlled error range (one grid size). As we can see, all the registers are placed in their candidate regions successfully in all the test cases. Finally, the CPU runtime is shown in the last column.

# 6. CONCLUSION

In this paper, we have proposed an algorithm that solves the problem of register insertion on global wire given a placed sequential circuit and a retiming solution. The proposed algorithm can preserve a given clock period with a controlled error using as few registers as possible in contrast to many previous works with post-retiming register placement that have the problem of clock period violation. In addition, the algorithm is also proved to be giving the optimal topology for nets with 4 or fewer pins. Since this type of nets makes up for about 90% of the nets in a sequential circuit on average, the algorithm performs very well and effectively under most situations.

Together with any powerful retiming methods which are designed to handle global netlist with block and wire delays, our proposed algorithm can be applied to locate where a register should be inserted to pipeline long global interconnects such that the target clock is preserved. This is particularly useful in today's designs in which multiple clock cycles are required to propagate a signal across a global wire.

Improvements can be made to handle the situation when there is no room for the candidate region of a register. Instead of scanning the neighboring grids for free spaces, incremental shifting and reshuffling of cells can be performed to free continuous rooms for register insertions.

# 7. REFERENCES

[1] C. Chu, E. F. Y. Young, D. K. Y. Tong, and S. Dechu. Retiming with interconnect and gate delay. In *Proceedings of IEEE International Conference on Computer-Aided Design*, 2003.

[2] P. Cochini. Concurrent flip-flop and repeater insertion for high performance integrated circuits. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 268–273, 2002.

[3] J. Cong, L. He, K. Y. Khoo, C. K. Koh, and Z. Pan. Interconnect design for deep submicron ics. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 478–485, 1997.

[4] J. Cong and S. K. Lim. Physical planning with retiming. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 2–7, 2000.

[5] V. De and S. Borkar. Low power and high performance design challenges in future technologies. In *Proceedings of Great Lakes Symposium on VLSI*, pages 1–6, 2000.

[6] S. Dey and S. T. Chakradhar. Retiming sequential circuits to enhance testability. In *Proceedings of 12th IEEE VLSI Test Symposium*, pages 28–33, April 1994.

[7] J. L. Ganley and J. S. Salowe. Optimal and approximate bottleneck steiner trees. In *Operations Reserach Letter*, pages 217–224, 1996.

[8] S. Hassoum, C. J. Alpert, and M. Thiagarajan. Optimal buffered routing path constructions for single and multiple clock domain systems. In *Proceedings of*

**Table 1: Experimental Results of Register Placement with Clock Preservation**

| Circuit | No. of logical regs. after retiming | Minimum possible no. of regs. after sharing | Actual no. of regs. using our method | Nets with 4 or fewer edges with regs. | Nets with 5 or more edges with regs. | No. of regs. placed within candidate region | No. of regs. placed with controlled error | CPU time (s) |
|---|---|---|---|---|---|---|---|---|
| s641 | 87 | 53 | 53 | 53 | 0 | 53 | 0 | 0.01 |
| s713 | 94 | 55 | 55 | 54 | 1 | 55 | 0 | 0.02 |
| s820 | 24 | 23 | 23 | 23 | 0 | 23 | 0 | 0.01 |
| s832 | 23 | 22 | 22 | 22 | 0 | 22 | 0 | 0.01 |
| s1196 | 31 | 18 | 18 | 17 | 1 | 18 | 0 | 0.00 |
| s1238 | 32 | 18 | 18 | 17 | 1 | 18 | 0 | 0.00 |
| s1269 | 259 | 127 | 127 | 113 | 14 | 127 | 0 | 0.02 |
| s1488 | 90 | 73 | 73 | 71 | 2 | 73 | 0 | 0.02 |
| s1494 | 78 | 62 | 62 | 60 | 2 | 62 | 0 | 0.01 |
| s3271 | 826 | 342 | 438 | 276 | 18 | 438 | 0 | 0.18 |
| s4863 | 622 | 408 | 417 | 360 | 22 | 417 | 0 | 0.25 |
| s15850.1 | 1554 | 1264 | 1264 | 1203 | 26 | 1264 | 0 | 2.52 |
| s35932 | 5455 | 2899 | 2899 | 2601 | 280 | 2899 | 0 | 13.41 |


*IEEE International Conference on Computer-Aided Design*, pages 247–253, 2002.

[9] C. Leiserson and B. Saxe. Optimizing synchronous systems. In *Journal of VLSI Computer Systems*, volume 1, pages 41–67, 1983.

[10] C. Leiserson and B. Saxe. Retiming synchronous circuitry. In *Algorithmica, vol. 6, no. 1*, pages 5–35, 1991.

[11] I. Neumann and W. Knuz. Layout driven retiming using the coupled edge timing model. In *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 22, no. 7*, 2003.

[12] P. Pan, A. K. Karandikar, and C. L. Liu. Optimal clock period clustering for sequential circuits with retiming. In *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 17, no. 6*, pages 489–498, 1998.

[13] M. Sarrafzadeh and C. K. Wong. Bottleneck steiner trees in the plane. In *IEEE Trans. on Computers, vol. 41, no. 3*, 1992.

[14] C. V. Schimpfle, S. Simon, and J. A. Nossek. Optimal placement of registers in data paths for low power design. In *Proceedings of 1997 IEEE International Symposium on Circuits and Systems*, pages 2160–2163, 1997.

[15] N. Sheony and R. Rudell. Efficient implementation of retiming. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 226–233, 1994.

[16] T. C. Tien, H. P. Su, and Y. W. Tsay. Integrating logic retiming and register placement. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 136–139, 1998.

[17] H. Zhou. Retiming for wire pipelining in system-on-chip. In *Proceedings of IEEE International Conference on Computer-Aided Design*, 2003.