An ECO Technique for Removing Crosstalk Violations in Clock Networks

Amit Kumar Krishnendu Chakrabarty
Dept. of Electrical and Computer Eng.
Duke University, Durham, NC, USA
{ak, krish}@ee.duke.edu

Chunduri Rama Mohan
Strategic CAD Labs
Intel Corporation, Folsom, CA, USA
chunduri.r.mohan@intel.com

Abstract—Crosstalk noise in the clock network of digital circuits is often detected late in the design cycle, sometimes as late as after first silicon. It is therefore necessary to fix crosstalk violations in the clock tree by making minimal design changes and maintaining skew bounds. We propose a novel approach for making Engineering Change Order (ECO) changes in the clock network to remove crosstalk violations. A two-pronged approach based on wire-translocation and rerouting is presented. Wire translocation, i.e., changing the track for a net to another free track with less crosstalk, eliminates a large percentage of removable violations. A linear time procedure to check if a violation is removable by translocation is developed. Next, an efficient rerouting method based on an incremental loci-finding algorithm is combined with linear time dynamic programming approach to eliminate the remaining crosstalk violations. Simulation results for randomly-generated clock tree of different sizes and benchmark circuits demonstrate the effectiveness of this approach.

I. INTRODUCTION

Crosstalk is becoming a dominant source of noise as chip designs becomes more congested and routing densities increase. In high-speed circuits, crosstalk causes coupling as well as additional timing violations. Since the clock signal has the highest switching rate in a design, any crosstalk on the clock signal has a significant impact on the magnitude of cross-coupling glitches and signal delay [8], [9], [7]. The design of the clock network should therefore directly address crosstalk noise and the associated problems.

Previous work related to clock networks can be divided into two basic areas: tree and the non-tree networks. Tree structures are preferred due to shorter net-length requirements and good skew properties [12]. Mixed strategies combines both the tree and the non-tree approaches [12]. The three most commonly-used tree-based clock routing schemes strive to minimize the clock skew or to achieve a perfect zero-skew tree (ZST). These three schemes are referred to as H-tree-based clock network [3], computational-geometry-based approaches [6], and deferred merged embedding(DME) [2].

Work in the field of synthesis and design of coupling-aware clock networks can be broadly classified into two areas: the construction of a coupling-aware clock tree, and the elimination of crosstalk from a previously constructed clock tree. The problem of constructing a crosstalk-aware clock tree has received much attention. In [8], [9] the authors use a DME-based approach, which can be easily modified to incorporate a non-linear delay model instead of a linear delay model as used in [11]. In an ASIC design flow, crosstalk is handled by providing additional spacing between parallel clock tree routes, shielding the entire clock tree, and making incremental changes to the clock tree to fix specific clock tree nets that cause crosstalk violations.

Shielding and increased spacing in a clock network are preventive steps to avoid crosstalk violation, but they incur additional area cost. The problem of modifying or making incremental changes to an existing clock network to remove crosstalk violations has not received adequate attention.

Incremental design changes that are made late in the design process are refereed to in the design community as Engineering-Change-Order

(ECO) [5], [15]. It is important to note that an ECO cannot be used to eliminate all crosstalk violations as the routing channel is already congested. ECO helps to reduce the amount of rerouting that is necessary to reduce crosstalk violations late in the design cycle. Since ECO relies on local and limited design changes, it is usually not sufficient to remove all violations; nevertheless, it reduces the amount of extensive rerouting that must be carried out, and thereby lessens the likelihood of additional timing problems that may be introduced due to rerouting. The maintenance of skew properties during rerouting in a clock tree is a difficult task. Changes made to the clock tree result in a change of skew values, which in turn may propagate downstream to the children of the particular node where the change takes place.

We propose an ECO approach to remove crosstalk violations in a previously-constructed clock tree. Our approach is divided into two parts. The first part uses the concept of wire translocation to eliminate crosstalk, and present a sufficient condition for a violation to be removable using translocation. In the second part, we introduce a crosstalk-aware rerouting approach. The proposed method of rerouting is based on a linear-time loci finding approach to do alternative placement of a node in a clock tree, followed by a dynamic programming approach to calculate the minimal crosstalk path satisfying a given skew bound δ . The dynamic programming procedure is of complexity $O(|V|)$, which is more efficient than the well-known Dijkstra's algorithm of complexity $O(|V|^2)$, for a given skew bound.

The rest of this paper is organized as follows. The proposed method of wire translocation and crosstalk elimination using translocation is discussed in Section II. In Section III, we explain why DME, H-tree, or computational-geometrybased methods cannot be applied to the problem of incremental modification of a clock tree. Section IV introduces a new incremental loci-finding approach. Section V deals with dynamic programming-based crosstalk-aware rerouting. Experimental results are presented in Section VI. In Section VII, we conclude the paper and present directions for future work.

II. WIRE TRANSLOCATION

Wire translocation is an incremental redesign method where a net is moved to a neighboring track to reduce crosstalk coupling. Before describing wire translocation in detail, we present our assumptions and review some basic definitions. In this work, we assume a rectangular, $m \times n$ gridded routing channel, with m uniformly distributed vertical tracks, denoted by the set $M = \{y_1, \dots, y_m\}$ and n horizontal tracks denoted by $N = \{x_1, \dots, x_n\}$. As shown in Fig 1, the horizontal (vertical) tracks are distance D apart from their adjacent neighboring tracks. The routing channel is modeled by a graph $G(V, E)$, where each vertex corresponds to the intersection of a horizontal and a vertical track. Therefore, for an $m \times n$ channel, the total number of vertices is mn . Each vertex is identified by the index $(i-1) \times m + j$, where i is the horizontal track on which the vertex lies and j is the corresponding vertical track. All nets in the routing channel are placed on the edges

Fig. 1. (a) A 7×7 routing channel (b) Illustration of the associated grid graph of the shaded region.

Fig. 2. An example illustrating crosstalk calculation for nets.

connecting these vertices. The set of edges E is the union of undirected connections defined as follows: there exists an edge between vertices v_i and v_{i+1} if $(i+1) \bmod m \neq 0$ for $0 < i < mn+1$, There also exists an edge and between v_j and v_{j+m} if $j+m < mn$ for $0 < j < m(n-1)+1$. All edges are of equal length as shown in Fig. 1, since all tracks in both directions (horizontal and vertical) are spaced uniformly over the channel.

As in [16], the crosstalk model used in this work considers only the capacitive coupling on a net due to other parallel nets. The capacitive coupling on a net, when another net runs parallel to it, can be measured using the formula $c = kl/t^2$ if $t \leq D$ and $c = 0$ if $t > D$, where k is a coupling parameter, l is the coupling length, t is the distance between the two segments, and D is the length and the width of the grid-cell. Crosstalk elimination approaches described in this paper are generic in nature and other crosstalk models can be used with minor modifications. Each edge in the graph G is therefore of length D . In this work, an edge in the routing graph G is referred to as a grid-cell-edge.

The crosstalk-weight for a grid-cell-edge is defined as the likelihood of a net suffering from crosstalk if it is placed on that grid-cell-edge. For example, if a net is placed on a grid-cell-edge-cell, which also has nets on two exactly adjacent tracks, the crosstalk-weight for that grid-cell-edge is two. If a grid-cell-edge has only one net on an adjacent grid-cell-edge, it is assigned a crosstalk-weight of one. If a grid-cell-edge has no net on adjacent grid-cell-edges, the crosstalk-weight for that grid-cell-edge is zero. Fig. 2 shows three nets a , b and c placed on the 7×7 routing channel of Fig. 1. To calculate the crosstalk for the grid-cell-edges corresponding to net b , which occupies a length of $4D$ on vertical track y_3 , we start from the top end of net b . The segment of net b occupying the first grid-cell-edge D of y_3 has a crosstalk-weight of 0 as it does not have any adjacent neighbors. The crosstalk-weights for the other grid-cell-edges between (x_6, y_3) and (x_3, y_3) , comprising net b are calculated in a similar fashion and these are 2, 2, and 1, respectively (Fig. 2).

Fig. 3. (a) Illustration of wire translocation; (b) An example illustrating the translocation procedure for three parallel nets.

The crosstalk for a net is proportional to the length of the overlap of the net under consideration with nets on adjacent tracks, and inversely proportional to the square of the distance between the net and adjacent nets. In this work, we measure the crosstalk of a net as the sum of the crosstalk-weights of all the grid-cell-edges occupied by the given net, and crosstalk is always defined for a given net. We assume that coupling occurs between two adjacent nets and that, for all practical purposes, the crosstalk effect from all other sources can be assumed to be zero. Referring back to Fig 2, the crosstalk for net b , $Cr(b)$, is calculated as the sum of all crosstalk-weights of grid-cell-edges comprising b , i.e., $Cr(b)=0+2+2+1=5$; similarly, for nets a and c the crosstalk values are calculated to be 2 and 3, respectively. Note that a grid-cell-edge can have a crosstalk-weight even if a net is not placed on it. The maximum-allowed crosstalk value M is defined as the maximum allowable value of crosstalk on a net, and a net with crosstalk value greater than M is said to violate the crosstalk bound. In Fig. 2, the value of M is assumed to be 4; therefore, net b violates the maximum allowable crosstalk bound.

We consider a clock routing problem in which we connect a set of k points lying in a plane [3]. These points are defined as the sinks of the clock tree. Our aim is to connect these k points in a manner such that clock skew and the total interconnect length are minimized [11]. A clock tree is constructed by grouping these sinks together and forming pseudo-sinks to connect sinks efficiently, in a top-to-bottom or a bottom-to-top manner [10].

Translocation is an incremental redesign process that considers two nets a and b placed on two immediate adjacent tracks x_i and x_{i+1} , respectively. If the crosstalk bound M is violated for either net then some part of net a or b can be moved to a neighboring track x_{i-1} or x_{i+2} , such that the crosstalk bound M is no more violated by either net a and b and no new crosstalk violations are created. Fig. 3(a) shows an example of wire translocation.

Example 1: During wire translocation, if two nets are placed on neighboring tracks, these nets induce crosstalk noise on each other due to their *overlap* as shown in Fig 3(a). Suppose the nets a and b are placed on two immediately neighboring tracks A and B. If some part of net b is translocated to the parallel track C, which is empty, the crosstalk effect between nets a and b is reduced. We will revisit this issue in Section II-A.

A. Selection of Tracks for Translocation

Let us define a data structure that is similar to a binary tree. The root of the tree is the clock source and leaf nodes correspond to clock sinks. Fig. 4 shows such a binary tree. The leaf nodes denote the sink of the clock tree. The clock input is fed to the tree at the root. Initially, we assume that the tree is balanced and the skew is zero, with each edge having a skew-weight equal to zero. This binary tree can be modeled as an RC network as shown in Fig. 5. Since, we are only interested

Fig. 4. Justifying the method for net selection.

Fig. 5. RC tree corresponding to the clock tree of Fig. 4.

in delay and skew values at the sink nodes, we ignore the delay and skew at the other nodes of the clock tree.

Example 2: Consider the Elmore delay model for the RC network shown in Fig. 5. Suppose a design modification to the clock tree changes $R2$ and $C2$ to $R2^*$ and $C2^*$, respectively. The new values of delay at Sinks 4, 5, 6 and 7 are obtained as follows:

$$D(4) = R1(C1 + C2^* + C3 + C4 + C5 + C6 + C7) + R2^*(C2^* + C4) + R4(C4) \quad (1)$$

$$D(5) = R1(C1 + C2^* + C3 + C4 + C5 + C6 + C7) + R2^*(C2^* + C5) + R5(C5) \quad (2)$$

$$D(6) = R1(C1 + C2^* + C3 + C4 + C5 + C6 + C7) + R3(C3 + C6) + R6(C6) \quad (3)$$

$$D(7) = R1(C1 + C2^* + C3 + C4 + C5 + C6 + C7) + R3(C3 + C7) + R7(C7) \quad (4)$$

It is clear from above that the design modification does not affect the delay (and thus the skew) at Sinks 6 and 7. It only affects the skew at Sinks 4 and 5, which belong to the subtree for the node where the clock tree was modified.

Any changes that occur at a particular node of a tree is seen by all other children of that sub-tree and a sub-tree closer to leaf nodes has fewer children, hence a net that is closer to leaf nodes is given a higher priority for translocation. The selection of a net for relocating it to a neighboring track is based on the position of the net in a clock tree and the availability of a free adjacent track. The steps involved in translocating a net can be therefore described as follows:

- 1) Step 1: Select the net to be moved, depending on its level from the leaf nodes.
- 2) Step 2: Select a location for net translocation. The location selected should be such that an overlap occurs between two nets involved in crosstalk and the target track (where the net is to be moved to) is empty. If the target track is not empty, the other net is selected for net translocation, assuming it can be moved. The process of translocation is done in an incremental fashion.

- 3) Step 3: Check for crosstalk violations and accept changes if the crosstalk bound is not violated and no other new violations are introduced.

B. Violations removable using translocation

Consider three parallel nets a , b and c as shown in Fig. 3(b). These nets are placed on tracks A, B, and C, respectively. If we want to remove a crosstalk violation on net b , we must translocate length x of the net b . The length $x + 1$ of net b has a crosstalk-weight of one because it has only one neighboring net (Fig. 3(b)). The part of net b of length $l - x - 1$, which remains on its original track, has a crosstalk-weight of 2.

Theorem 1: Translocation is effective only when $l < 3/2M$.

Proof: Translocation, is therefore effective only when $2(l - x - 1) < M$ and $(x + 1) < M$, where M is the maximum crosstalk-weight allowed. This, implies that $l < 3/2M$. ■

Consider n nets that are parallel to each other. The nets labeled $1, \dots, n$ can be divided into groups of three nets: 1 to 3, 2 to 4, and so on. Therefore, for each net except the 1 and n , the condition $l < 3/2M$ should be satisfied. Any violation that satisfies the above properties can be removed using translocation.

C. Computational complexity of the translocation procedure

At the start of translocation procedure, we already know the nets that have crosstalk violations. To check if a violation on a given net is removable by translocation, we need $O(n)$ time assuming an $n \times n$ grid. While checking, we need to know the start and end of a given net under consideration, and we check for the complete set, which can be at most $2n$ in length. The translocation procedure takes $O(n \log(n))$ time as we need to select a point on wire and then see if crosstalk bounds are violated after shifting of the net to another adjacent track. The initial track change for a part of net and checking for crosstalk violations after this change takes $O(n)$ time. However, this procedure should be repeated at the most $\log(n)$ times unless crosstalk bounds are satisfied. A divide and conquer approach can be used to find the exact net length required to be translocated to eliminate crosstalk violation. Therefore, translocation takes $O(n \log(n))$ time in the worst-case for an $n \times n$ grid.

III. OVERALL REROUTING APPROACH

Clock rerouting requires that skew constraints are satisfied for all rerouted nets and their children. To maintain skew within a given bound, we find the loci of points equidistant from two sinks (leaf nodes). Nets of equal length can be drawn from any point on the loci to the two sinks, resulting in an equal value of skew after rerouting.

The overall rerouting scheme can be divided into three basic steps:

- 1) Select a net for rerouting. If there are two nets at different levels that have mutual crosstalk, the net closer to the sink (the leaf nodes) is given priority, due to the smaller number of nodes present in its corresponding subtree.
- 2) Any target net selected for rerouting has one end that is at a higher level than the other. A new set of loci is found for lower points so that net can be re-routed satisfying the skew and crosstalk bounds.
- 3) A routing solution with minimum crosstalk and skew requirements within a given bound is chosen.

Example 3. Consider the situation in Fig. 6, where B4 and B3 are the two points whose loci is to be found. Since nets connecting the new position of A2 (i.e., on the loci) to B3 and B4 should have equal skew, they should be equal in length. Therefore, the possible placement of A2 is searched on the loci of midpoints of B3 and B4. In the above

Fig. 7. (a) An obstacle and two possible paths from A to B. (b) Crosstalk-weights assigned to each grid-cell-edge for two possible paths between A and B. (c) Choice of a path with lower total crosstalk.

Fig. 6. Finding equidistant points from two given points using various methods.

scenario, algorithms such as DME [2] or Edahiro's [6] algorithm will attempt to find the loci of points equidistant from B4 and B3 on a line that is at 45 degrees from horizontal and vertical routing tracks and passes through the midpoint. However, we see that D is a point that is also on loci given by DME. If we go from D to E as shown, points on the path DE and further upstream are equidistant from both B3 and B4. It is evident from this example that DME and Edahiro's algorithm miss points that are equidistant from both B3 and B4 (along line DE and FG) and may satisfy minimal skew condition. These points along DE and FG can possibly be treated as loci (that can be used for possible placement of A2). Points L, K and I, J on the line at forty-five degrees, used by DME and Edahiro's, method are not equi-distant from B3 and B4; therefore, they cannot be used for the possible placement of A2. Therefore, our incremental loci finding approach gives a larger number of points for the possible placement of A2, which results in a higher probability of crosstalk-free rerouting.

IV. INCREMENTAL LOCI FINDING APPROACH

The proposed algorithm for finding the loci can be as described using Fig. 6 as follows:

- 1) Draw X and Y axes from points B3 and B4, as in Fig. 6. Note the points where they intersect. One such point is C, another one is H, diagonally opposite to C.
- 2) Move incrementally to find a point equidistant from both B3 and B4. One such point in this example is D; another point is F, diagonally opposite to D. Note that at each step, a constant number of comparisons (four) are performed.
- 3) All points equidistant from the two given points constitute the loci and is shown by a dark line in Fig. 6.

A. Computational Complexity of the Loci-Finding Algorithm

The complexity of this greedy algorithm is linear with respect to the number of equidistant (loci) points calculated. Since, only four

comparisons are made at every step, we can find an initial point in a loci set in a maximum $\max\{m, n\}$ iterations for a gridded routing $m \times n$ channel. Initially, two lines are drawn such that they intersect at a point, which takes $O(1)$ time. Start moving in a favorable direction in incremental steps. The total number of steps that this incremental procedure takes is at most $m + n$, which is equal to the length of the diagonal for the channel¹ and each time four comparisons are performed. Thus, the overall computational complexity of the incremental loci-finding procedure is $O(m+n)$ for an $m \times n$ routing-channel.

V. CROSSTALK-AWARE REROUTING HEURISTIC

In this section, we present a dynamic programming-based optimization approach for grid-based rerouting. In this method any two given vertices of the grid graph G are connected using a path such that sum of all grid-cell-edge crosstalk-weights along this path is less than any other alternative path connecting these two points; such as path with minimum grid-cell-edge crosstalk-weight is referred to as a *minimum-cost path*. The algorithm starts with a grid similar to that shown in Fig. 7(a). The grid-cell-edges are assigned crosstalk-weights according to the coupling effect of nets on neighboring tracks. As discussed before, nets on adjacent tracks can induce crosstalk on one another. If we have to go from A to B and the grid-cell-edge under consideration has an obstacle as shown in Fig. 7(a), then we have to take an alternative path to reach B, thereby avoiding obstacles.

Example 4. Consider a routing problem in which we have to connect two points A and B as shown in Fig. 7(a). The shaded region in Fig. 7(a) is an obstacle, and no net can be placed on an edge in the obstacle region. Due to the obstacle only two Manhattan paths ACB and ADB are possible in this example.

The various steps involved in a dynamic-programming based minimum-cost path approach is shown in Fig. 8. Let δ be the maximum units of skew allowed. The procedure Mincost-Path calculates the minimum cost path between two given points, "start" and "end", on the routing channel. We consider a pair of source and destination vertices, named "start" (0, 0) and "end" (end-x, end-y). Rename the source vertex as origin. A new set of co-ordinates are defined using procedure *New-coordinates*. The x and y co-ordinates increase as we move towards the end vertex. A bounded box is created in *FormBounded*, with boundaries of the box being $(-\delta, -\delta)$, $(\text{end-x} + \delta, -\delta)$, $(-\delta, \text{end-y} + \delta)$, $(\text{end-x} + \delta, \text{end-y} + \delta)$. In the next step two-dimensional arrays, parent and cost, are defined and initialized as described in Fig. 8. Lines 10-19 calculate the cost for each vertex in this bounded box, and a value is assigned to a corresponding element in the parent array depending on where the current node under consideration

¹The sum of the lengths of two sides of a triangle is always greater than the length of its third side.

Procedure Mincost-Path($G(V,E)$, start, end)

```

1. /*Form a bounded rectangular box with start and end as diagonal.*/
2. FormBounded( $G$ , start, end);
3. /*Give a new co-ordinate system for this box start given (0,0) and end
   (end-x, end-y)*/
4. New-coordinates( $G$ , start, end);
5. /*Define and Initialize two new arrays parent[end-x][end-y] and
   cost[end-x][end-y]. Elements of parent is initialized (0,0) and cost to
   infinity*/
6. InitializeArrays(parent,cost);
7. /* Set cost of start as 0*/
8. cost(0,0)=0;
9. /* Calculate cost for each element in bounded box*/
10. for( i=0; i<=end-x; i=i+1)
11.   for(j=0; j<=end-y; j=j+1)
12.     cost(i,j) = min((cost(i-1,j)+crosstalk-weight(edge((i,j),(i-1,j))),
13.                      (cost(i+1,j)+crosstalk-weight(edge((i,j),(i+1,j))),
14.                      (cost(i,j-1)+crosstalk-weight(edge((i,j),(i,j-1))),
15.                      (cost(i,j+1)+crosstalk-weight(edge((i,j),(i,j+1)))));
16.   /* In parent(i,j) store the value of co-ordinates which result in
   min cost calculated above*/
17.   Allocate-value-parent(parent(i,j));
18. end for;
19. end for;
20. /* Backtrack from end using parent array till we reach start( this is the
   minimum cost path, in reverse direction, from the end to start )*/
21. while(parent(i,j)!=start)
22.   print(i,j);
23.   /*Change values of i to x co-ordinate of parent(i,j) and y to y co-
   ordinate*/
24.   i=x_parent(i,j);
25.   j=y_parent(i,j);
26. end while;
27. print(start);
28. end procedure

```

Fig. 8. Procedure for dynamic-programming-based minimum-cost path solution.

was reached from. After calculation of the complete parent and cost arrays, we can print the path in reverse direction staring at the end node, described in lines from 21-27. This method of printing minimum cost path in reverse direction is referred to as backtracking, and it is different from that used for greedy algorithms [4].

A. Rationale for Dynamic Programming

In this section, we justify the use of the dynamic-programming-based approach for rerouting. In an ECO rerouting problem, we show that we have to make appropriate choices, which result in the solution of one or more sub-problems. The solutions to these sub-problems form a complete solution if the skew bound δ is also satisfied by the given solution.

Suppose we have to re-route a net between nodes $a(1)$ and $a(n)$ in Fig. 7(c), and our solution results in a minimum-cost path, that satisfies skew bounds δ between $a(1)$ and $a(n)$. This is only possible if on any two given points $b(1)$ and $b(n)$, lying on the minimum cost path that satisfying the skew bound δ between $a(1)$ and $a(n)$, the path connecting $b(1)$ and $b(n)$ is also a minimum-cost path satisfying the skew bound δ . This condition is always satisfied. Otherwise, we can prove by contradiction that, if any alternative path exists between $b(1)$ and $b(2)$, it would have been included in minimum-cost solution that satisfies skew bound δ between $a(1)$ and $a(n)$.

Another reason for using dynamic programming is the presence of overlapping sub-problems. A recursive problem solves the sub-problem repeatedly if it has to calculate a minimal-crosstalk path between two given points. However, in a dynamic programming approach, we solve the subproblem once and then store the solution in a table where it can be looked up when needed using a constant-time-per-lookup. As previously discussed, in dynamic programming, we can solve the various sub-problems first and then store the solutions to construct a complete solution. In Fig. 7(c), the minimum-cost path joining $a(1)$ and $a(2)$, can be considered as a sub-problem of a bigger problem of determining the path between A and B. By combining the minimum-

Grid Size	M	Total Violations	Violations Removed	Percentage Removed
400×400	10	98	21	21
400×400	15	92	27	29
400×400	20	81	29	36
400×400	25	74	24	32
400×400	40	62	32	51
400×400	50	53	29	55
400×400	100	34	20	60

TABLE I

TOTAL NUMBER OF CROSSTALK VIOLATIONS AND THE NUMBER OF VIOLATIONS REMOVED AS A FUNCTION OF THE MAXIMUM CROSSTALK BOUND.

cost paths (solutions) for such sub-problems, the minimum cost path along A and B can be calculated. Since we attempt only to re-route a net in a small region, the extra space requirement is linear in terms of $|V|$.

B. Complexity of the Rerouting Algorithm

Consider an $m \times n$ grid. In the worst case, we need to calculate the cost and parent arrays of size mn , once for every vertex. For every vertex, these calculations take constant time. Therefore, the complexity of our dynamic programming approach is $O(mn)$. Since mn equals the number of vertices in the gridded graph, the proposed solution has a worst-case time complexity of $O(V)$, where V is the number of vertices in the gridded graph. Thus, the dynamic-programming based approach is more efficient than the well-known Dijkstra's algorithm, which has complexity $O(|V|^2)$ under a given skew bound.

VI. EXPERIMENTAL RESULTS

A comprehensive set of experiments were performed on a Sun Blade workstation with a 750 MHZ processor and 1 GB memory. The five benchmarks circuits—R1, R2, R3, R4, and R5—are available from [http://vlsi-cad.ucsd.edu/GSRC/bookshelf/Slots/BST/\[1\]](http://vlsi-cad.ucsd.edu/GSRC/bookshelf/Slots/BST/[1]). Experiments were also performed on randomly-generated clock trees using the ZST approach [14]. The proposed approach is different from [8] in that we focus on already-routed clock tree. In this section, we deal with crosstalk violations between two clock nets. The proposed ECO approach is generic and it can be easily applied to a clock tree constructed using any other method. Similarly, proposed crosstalk elimination approaches are generic in nature and other crosstalk models can be used with minor modifications. We assume that our channel is gridded, and all nets are routed in an orthogonal manner. Crosstalk is assumed to be present between two nets placed on adjacent tracks; all other crosstalk values are zero. A linear crosstalk model is used. The crosstalk for a net is calculated on the basis of net overlap and crosstalk-weights. Any net closer to the leaf nodes is a more suitable candidate for wire translocation or rerouting than a net farther from the leaf nodes, as fewer nodes are affected by any such change.

Table I shows the total number of violations, and the number of violations removed, as we vary the maximum crosstalk bound M for a 400×400 grid size. As expected, the percentage of removable violations is low for small values of M . Table II shows the number of crosstalk violations and violations removed for various grid sizes for varying number of sinks. As the number of sinks increases for a given grid size, the routing channel becomes more congested, usually resulting in a reduced number of violations removed. The value of crosstalk-bound (M) used was set to 100. Table II shows that most of the removable violations are eliminated using translocation. Table III shows the number of crosstalk violations removed for benchmark circuits, R1, R2, R3, R4 and R5, in 90nm technology. The area of

Grid Size	Number of Sinks	Total of Violations Removed	Number of Violations Removed by Translocation	Total Number of Violations Removed	Percentage of Violations Removed
100×100	50	8	7	7	87
	150	14	6	9	64
	200	6	6	6	100
200×200	50	13	6	9	69
	100	30	13	17	57
	150	47	17	21	45
300×300	50	19	16	18	95
	100	51	19	24	47
	150	75	25	29	39
400×400	50	19	9	12	63
	100	41	22	23	56
	150	23	10	11	47

TABLE II

TOTAL NUMBER OF VIOLATIONS AND NUMBER OF VIOLATIONS REMOVED AS A FUNCTION OF SINK AND GRID SIZE.

Benchmark Name	Number of Sinks	Chip Area (cm^2) [13]	Crosstalk-inducing wire length (μm)	Total Number of Violations	Number of Violations Removed by Translocation	Number of Violations Removed	Percentage of Violations Removed
R1	267	0.57e-4	0.15	68	16	21	31
			0.20	30	11	12	40
			0.30	7	2	2	29
R2	598	0.78e-4	0.15	151	43	51	34
			0.20	122	42	45	37
			0.30	54	17	22	41
R3	862	1.09e-4	0.15	304	72	85	28
			0.20	202	49	57	28
			0.30	117	34	39	33
R4	1903	1.07e-4	0.15	303	69	84	27
			0.20	153	43	48	31
			0.30	117	36	39	33
R5	3101	1.17e-4	0.15	213	58	68	32
			0.20	197	58	61	30
			0.30	126	42	43	34

TABLE III

TOTAL NUMBER OF VIOLATIONS AND NUMBER OF VIOLATIONS REMOVED FOR BENCHMARKS FROM [14].

the chip is kept the same as in [13]. We define *crosstalk-inducing wire length* to be the extent of overlap of wires on adjacent tracks that tends to cause crosstalk violations. The ECO method described here reduces the extent of major rerouting by removing a significant fraction of crosstalk violations. Thus, it can be used as a precursor to more extensive rerouting techniques for crosstalk removal.

VII. CONCLUSION

In this paper, an ECO technique for removal of crosstalk violations in clock networks has been proposed. This technique is based on the concepts of wire translocation and rerouting. Wire translocation, i.e., the movement of a wire to a neighboring track, is very efficient and has a computational complexity of $O(n)$ for an $n \times n$ grid. rerouting is based on a $O(V)$ loci-finding algorithm and a dynamic-programming-based optimization approach. Experimental results show that a significant fraction ($> 50\%$ in many cases) of crosstalk violations can be removed by the proposed ECO method. As part of ongoing work, we are investigating a computationally-efficient method to identify crosstalk violations that cannot be removed by ECO rerouting.

REFERENCES

- [1] A. Caldwell and A. Markov, Toward CAD-IP reuse: The MARCO GSRC bookshelf of fundamental CAD algorithms. In *IEEE Design and Test*, pp. 72–81, May 2002.
- [2] T. H. Chao et al., Zero skew clock net routing. In *Proc. DAC*, pp. 518–523, 1992.
- [3] J. Cong et al., Matching-based methods for high-performance clock routing. In *IEEE Trans. CAD*, vol. 12(8), pp. 1157–1169, August 1993.
- [4] T. H. Cormen et al., *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd ed., 2004.
- [5] O. Coudert et al., Incremental CAD. In *Proc. ICCAD*, pp. 236–244, 2000.
- [6] M. Edahiro, A clustering-based optimization algorithm in zero-skew routings. In *Proc. DAC*, pp. 612–616, 1993.
- [7] E. Friedman, Clock distribution networks in synchronous digital integrated circuits. In *Proc. of the IEEE*, vol. 89(5), 2001.
- [8] M. F. Hsiao et al., Crosstalk minimization for multiple clock tree routing. In *Proc. MWSCS.*, pp. 152–155, 2002.
- [9] M. F. Hsiao et al., Minimizing inter-clock coupling jitter. In *Proc. ISQED.*, pp. 333–338, 2003.
- [10] M. A. Jackson et al., Clock routing for high performance ICs. In *Proc. DAC*, pp. 573–579, 1990.
- [11] A. B. Kahng and C. A. Tsao, Practical bounded-skew clock routing. In *J. VLSI Signal Process. Syst.*, vol. 16, pp. 199–215, June 1997.
- [12] A. Rajaram et al., Reducing clock skew variability via cross links. In *Proc. DAC*, pp. 18–23, 2004.
- [13] H. Su and S. S. Sapatnekar, Hybrid structured clock network construction. In *Proc. ICCAD*, pp. 333–336, 2001.
- [14] R. Tsay, An exact zero skew clock routing algorithm. In *IEEE Trans. CAD*, vol. 12, pp. 242–249, Feb. 1993.
- [15] H. Xiang et al., ECO algorithms for removing overlaps between power rails and signal wires. In *Proc. ICCAD*, pp. 67–74, 2002.
- [16] H. Xiang et al., An ECO algorithm for eliminating crosstalk violations. In *Proc. ISPD*, pp. 41–46, 2004.