A Search-Based Bump-and-Refit Approach to Incremental Routing for ECO Applications in FPGAs *

Vinay Verma and Shantanu Dutt
Dept. of EECS, University of Illinois-Chicago

Abstract : **Incremental physical CAD is encountered frequently in the so-called engineering change order (ECO) process in which design changes are made typically late in the design process in order to correct logical and/or technological problems in the circuit. As far as routing is concerned, in order to capitalize on the enormous resources and time already spent on routing the circuit, and to meet time-to-market requirements, it is desirable to re-route only the ECO-affected portion of the circuit, while minimizing any routing changes in the much larger unaffected part of the circuit. Incremental re-routing also needs to be fast and to effectively use available routing resources. In this paper, we develop a complete incremental routing methodology for FPGAs using a novel approach called bump and refit (B&R); B&R was initially proposed in [4] in the much simpler context of extending some nets by a segment (for the purpose of fault tolerance) for FPGAs with simple *i*-to-*i* switchboxes. Here we significantly extend this concept to global and detailed incremental routing for FPGAs with complex switchboxes such as those in Lucent's ORCA and Xilinx's Virtex series. We also introduce new concepts such as B&R cost estimation during global routing, and determination of the optimal subnet set to bump for each bumped net, which we obtain using an efficient dynamic programming formulation. The basic B&R idea in our algorithms is to re-arrange some portions of some existing nets on other tracks within their current channels to find valid routings for the incrementally changed circuit without requiring any extra routing resources (i.e., completely unused tracks), and with little effect on the electrical properties of existing nets.**

We have developed optimal and near-optimal algorithms (called `Subsec_B&R` and `Subnet_B&R`, respectively) to find incremental routing solutions using the B&R paradigm in complex FPGAs. We implemented these algorithms for Lucent's ORCA-2C FPGA, and compared our algorithms to two recent incremental routing techniques, `Standard` and `Rip-up&Reroute`, and to Lucent's `A_PAR` routing tool. Experimental results show that our incremental routers perform very well for ECO applications. Firstly, B&R is 10 to 20 times faster than complete re-routing using `A_PAR`. Further, the B&R incremental routers are nearly 27% faster and the new nets have nearly 10% smaller lengths than in previous incremental techniques. Also, the B&R routers do not change either the lengths or topologies of existing nets, a significant advantage in ECO applications, in contrast to `Rip-up&Reroute` which increases the length of ripped up nets by an average of 8.75% to 13.6%.

1 Introduction

An FPGA is a general-purpose, programmable logic device that is customized in the package by the end user. It consists of a large number of programmable logic blocks (PLBs) and programmable routing which allows the logic blocks to be connected to form a larger circuit. The logic is implemented by electronically programming the interconnects, typically by the user instead of the manufacturer.

An SRAM-programmable FPGA is programmed by loading *configuration memory cells* from an external source. The configuration memory cells control the logic and interconnect that perform the application function of the FPGA. Such FPGAs are reprogrammable, and can be used to implement different functions at different times. Such reprogrammability also allows much flexibility for ECO applications late in the design process.

In this paper we present incremental routing algorithms for complex FPGAs that use a novel bump-and-refit (B&R) approach. This approach was initially proposed in [4] in the much simpler context of extending some nets by a segment (for the purpose of fault tolerance) for FPGAs with simple *i*-to-*i* switchboxes (SBox's). Here we significantly develop this concept further to design a complete incremental routing flow for complex FPGAs. We introduce new concepts like bumping costs during global routing, and optimal bumping subsets of nets to realize an efficient incremental routing technique for ECO applications.

The goals of our work are to develop incremental routing algorithms that: (1) Are orders of magnitude faster than complete re-routing; (2) Complete the required incremental routing in the available routing resources if such a solution exists (this will minimize the need for area- and time-expensive fall-back strategies); (3) Complete the routing without significantly changing electrical properties (e.g., power, delay) of existing nets (this will keep the parasitic extraction data and timing/power analysis for the unaffected portion of the circuit valid). Experimental results show that the new B&R incremental routers have significant advantages over existing incremental routing methods with respect to the above metrics. We also prove optimality of a portion of our algorithms.

The rest of the paper is organized as follows. In Sec. 2, we discuss previous research in incremental routing. Section 3 discusses all aspects of the new B&R incremental routing method. Experimental results comparing two versions of the B&R incremental router to the type of incremental routers proposed in [5, 2] that we have implemented, are given in Sec. 4. We conclude in Sec. 5.

2 Prior Work On Incremental Rerouting

This is a new area and few papers have tackled this problem. One of these is the incremental rerouting technique developed by Emmert and Bhatia [5]. In their work the nets connected to faulty/displaced logic blocks (PLB's) are partially ripped up and re-routed. Graph building between the pin of the starting channel (SC) and the pin of the target channel (TC) is attempted. If this is successful, the path with the minimum cost is selected. If the router is not successful, the window size for the router is increased by one unit. Unlike [4], [2] and our current work, they do not perturb or move the already routed nets. We term this incremental routing approach as `Standard` in the rest of the paper.

Cong and Sarrafzadeh present a rip-up and reroute approach to incremental physical design in [2]. The first goal of their incremental routing is to route the new nets without removing any existing nets. When some nets cannot be routed, a rip-up and reroute procedure is used to free routing resources and re-do routing for the newly added nets and ripped up nets. If rip-up and reroute fails to route all the nets, the floorplan and placement of the design is updated to add more routing resources. Their technique of rip-up and reroute is in contrast to our B&R approach wherein we perturb existing nets only within their current channels, rather than rerouting them. Since the nets are

*This work was funded in part by Darpa Contract # F33615-98-C-1318 and in part by a grant from Xilinx Corp.

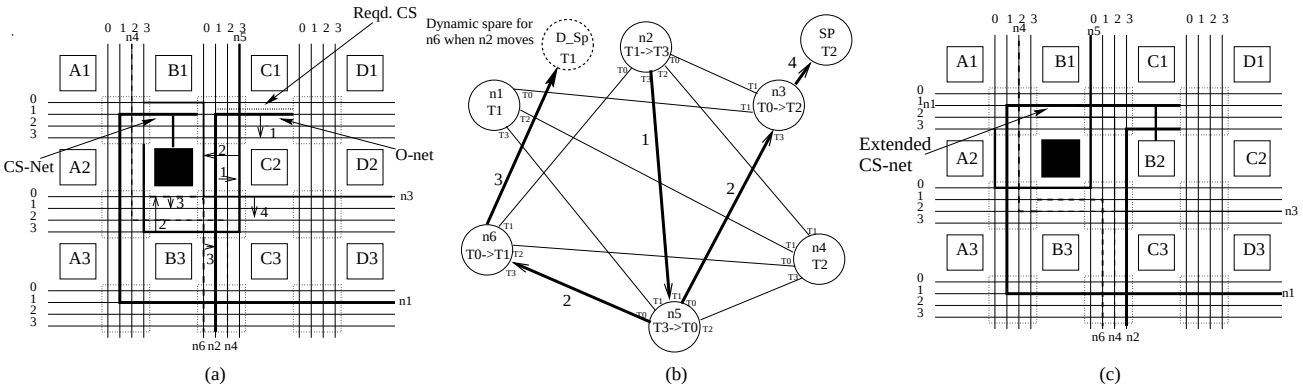

Figure 1: *(a) Presence of occupying net (O-net) n_2 prevents a straightforward insertion of an CS. Thus the O-net must be moved to another track, possibly bumping other nets. (b) Overlap graph representation for the given routing. The track number at an end-point of an edge indicates the track the corresponding net would move to if this edge is traversed. (c) Final routing of the circuit.*

still routed in their original channels, neither their topologies nor their lengths change. Thus most properties of existing nets are preserved which is key to an effective incremental design process.

Cong, Fang and Khoo presented efficient techniques for obtaining a non-uniform routing grid from a given VLSI routing in order to perform incremental routing for ECO [1].

Another recent work is that of Dutt, Shanmugavel and Trimberger [4] in which an incremental rerouting algorithm was developed for fault reconfiguration in FPGAs. In their work they have used segmented FPGAs which uses $i-to-i$ connection, i.e., a net is routed on only one track throughout. It uses the concept of node cover [6] , to cover cell (PLB) faults but is different from [6] in the manner in which net extensions (also called *CS* insertions) are made for the purpose of fault reconfiguration. It makes CS insertions only specific to faults wherever and whenever required in a dynamic manner ([6] uses a static method to provide CS's to cover all possible requirements for the given fault pattern—one fault per row). In Fig. 1a, net n_1 is an CS net—a net the needs to be extended by one segment (the *cover segment (CS)* in order to connect to the PLB that replaces the original PLB (for the purpose of fault reconfiguration) that it was connected to. For each CS, if the required track segment is vacant, the insertion is accomplished by including this segment as part of the corresponding net. However, if the required wire segment is occupied by another net, then the CS insertion will cause a displacement or "bumping" of this net.

As shown in Fig. 1(a), the net occupying the required track segment is termed the *occupying net (O-net)*. The CS-net has to be extended by one segment towards the direction of the cover cell, and this segment is currently occupied by the O-net. Thus the O-net needs to be moved out of its current track. Let a *transition* be defined as the movement of net n_i on a track T_j to another track T_k, and denoted by $n_i^{T_j \rightarrow T_k}$. This transition may result in net n_i bumping into one or more nets on track T_k. These nets will have to move out of their current track T_k, giving rise to a transition for each of them. This transition sequence is shown in Fig. 1b by dark arcs, where net n_2 initiates a set of transitions which finally terminate in "spare" nodes, which are vacant segments of appropriate total lengths in which a bumped net can move in without bumping any other net. The set of transitions take on a directed-acyclic graph (DAG) structure, termed a *transition DAG (T-DAG)*, with the spares forming the leaf nodes. The CS insertion is successful if a T-DAG rooted at the corresponding O-net can be found whose leaves are spare nodes; such a T-DAG is termed a *converging T-DAG*.

The concept of an *overlap graph* (*OG*), which is a graph representation of the circuit routing on the FPGA, is introduced in [4] as an aid to finding a converging T-DAG solution. The OG is an undirected graph with the circuit nets represented by the nodes of the graph. $n_i^{T_j}$ is used to denote a net n_i on track T_j. There exists an edge between n_i and n_j in the OG iff nets n_i and n_j share a channel[1] in the FPGA. Figure 1(b) shows nets n_2 and n_6 having an edge between them in the OG since in Fig. 1(a) they are routed through a common vertical channel to the right of cell B3.

The OG can be used to determine if the required T-DAG exists. Since the OG represents the circuit routing in the FPGA, a T-DAG is a DAG embedded in the OG (the undirected edges of the OG become directed arcs in the direction of the transitions; see Fig. 1b). Thus a converging T-DAG rooted at an O-net can be determined by performing a search for a T-DAG on the OG.

This process is illustrated in Fig. 1 for a small circuit and for a single new net n_1. The corresponding O-net n_2 transits from T_1 to T_3 and bumps into n_5. The movement of n_2 from T_1 creates a "dynamic" spare node (labeled by D_Sp in Fig. 1b) for net n_6, which information is added to the OG. The bumped net n_5 then transits from T_3 to T_0 where it bumps n_6 and n_3. n_6 then transits to the above dynamically created spare on T_1, while n_3 transits to its spare track segments on T_2. Thus a converging T-DAG is determined in the OG. The transition arcs are shown dark in Fig. 1b and numbered chronologically in the order in which they are traversed in the search process. Figure 1c shows the final routing of the FPGA after the bumping sequence converges.

A depth-first search algorithm for finding a converging T-DAG in the OG was developed in [4]. This algorithm is optimal in the sense that it will find a converging T-DAG if one exists. An extended version of this algorithm for more complex FPGAs and for ECO applications is presented later in Fig. 9.

While an existing converging T-DAG will ultimately be found by the depth-first algorithm of [4], it will be time-efficient if some suitable "cost" measure can be used to determine which transitions are more likely to be successful so that fewer T-DAGs are searched and backtracked. A good cost measure will consider both the "magnitude" of bumpings (total length of bumped nets) and the likelihood of convergence of these bumpings.

Two transition cost (TC) (equivalently, bumping cost) measures

[1] A *channel* is the set of all track segments between two adjacent SBox's of the FPGA.

evaluated are as follows:
(1) $sum(n_i^{T_j \rightarrow T_k}) = \sum_{n_j \in adj^{T_k}(n_i)} l(n_j)$, where $adj^{T_k}(n_i)$ are the neighbors of n_i in the OG that are on track T_k, and $l(n_j)$ is the total length of n_j in terms of the track segments (each of length 1) that it occupies. This cost estimate is reasonable, but only considers the bumping magnitude. For example, according to it, it is equally costly to bump a net of length 9 as it is to bump 3 nets each of length 3. However, the latter case has a higher likelihood of convergence since there is greater flexibility in moving 3 bumped nets than a single net of the same total length. This leads to the next cost function.
(2) $sqrt(n_i^{T_j \rightarrow T_k}) = [\sum_{n_j \in adj^{T_k}(n_i)} l(n_j)] / \sqrt{|adj^{T_k}(n_i)|}$.

Using such TC functions to guide the search results in computation time reduction by an order of magnitude compared to a "blind" depth-first search.

The algorithm of [4] is efficient in terms of both track usage and time. It shows considerable improvement over the static method [6] in track overheads for tolerating a single PLB fault in each FPGA row. The static method has a track overhead of 42%, while the depth-first algorithm of [4] has an average overhead of only 12.8%; a 70% improvement. Further, average re-routing times per faulty row over all circuits is about 26.5 secs, which is promising, since complete re-routing for circuits of these sizes can take an hour or more.

In [4] the affected net (nets connected to displaced PLBs) are not rerouted, only net extensions (CS insertions) are made, so [4] cannot be used for ECO routing in its present framework.

We will use a B&R approach of finding new track assignments for existing nets in order to make room for new nets that is similar to that of [4]. However we incorporate the B&R approach within the context of a complete routing system in which we perform full fledged incremental routing (as opposed to extending some nets by one segment at their branch or terminal points as done in [4]). This includes performing global as well as detailed routing taking the potential *bumping cost* into account (besides other cost measures). Further, here we also extend the B&R approach to FPGAs with complex switching and routing capabilities such as those available in commercial FPGAs like Xilinx's Virtex and Lucent's ORCA. The SBox's in these FPGAs allow routing a net on different tracks by interconnecting a segment on track *i* to another segment on track *j*. This adds another dimension to the B&R approach, viz., when a net is bumped in one portion, should only that portion be moved to a different track or should more than that portion be bumped (possibly the entire net)? We have developed a dynamic programming algorithm to optimize metrics like the probability of successful B&R (as measured by the degree of bumping or bumping cost), and SBox and track resource usage to determine how to bump different subnets of a bumped net. All these aspects of our B&R incremental router are discussed in the next section.

3 Our New Incremental Rerouting Algorithm

We have partitioned our incremental rerouting problem into hierarchy of two stages. We first perform global routing taking possible bumping cost and then detailed routing possibly with bumping existing nets if a set of available interconnectable track segments is not found to accommodate the net being routed. If existing nets are bumped by the detailed router, the bump and refit (B&R) algorithm is used to refit the bumped nets on other track segments in a recursive B&R manner. The incremental B&R routing flow that we have developed is shown in the flowchart in Fig. 2 (we have not developed the incremental channel expansion/placement/floorplanning phase shown in the rightmost box of Fig. 2). As shown in the flowchart, both the global and detailed routing phases consider costs that control both net

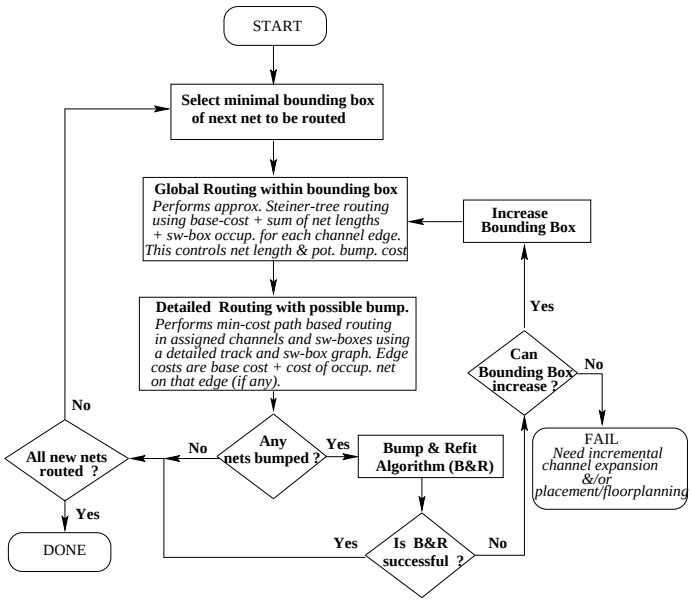

Figure 2: *An incremental routing flow incorporating B&R.*

length and potential bumping cost to obtain a min-cost routing (e.g., the global router will prefer to route along channels where the bumping cost, if any, will be potentially minimal as long as a minimal-length route is obtained).

Before going into the details of our incremental router we define a few terms. We define a *subnet* as a maximal portion of a net which spans at least one horizontal or vertical channel and no part of which can be independently moved to another track without disconnecting it from the rest of the subnet; see Fig. 3. A subnet is on one particular track. We will denote a subnet s_j of net n_i by $n_i.s_j$ We define *subsection* as a set of one or more subnets of a net.

3.1 Global Routing

In global routing we try to optimize the overall wire length of the net to be routed, congestion of the channels and possible bumping cost that may be incurred by the detailed router. The global router assigns channels (or identically SBox's) to the net to be routed. The global routing graph *GR-Graph* is a weighted connection graph of SBox's of the FPGA. Each node in the graph corresponds to a SBox (SBox) and there exists an edge between two nodes in the graph if the corresponding SBox's are adjacent in the layout, i.e., these edges represent channels between SBox's . The weight/cost function of an edge in the graph is discussed in Sec. 3.1. For multipin nets the global route amounts to finding the approximate Steiner tree connecting these pins also called *GTree*. In our technique, we solve this problem by repeated application of Dijkstra's shortest path algorithm [3], in a bounding box of nodes. A formal description of this heuristic is given in Fig. 4

Note that our main goal is to develop an incremental detailed router using a B&R approach. We, however need a global router that can also take possible bumping cost into account when making channel assignments. We thus have used a simple global routing strategy with a primary goal of providing a bumping cost factor in the global routing output, so that the detailed router incurs minimal bumping cost. This in turn means that the B&R algorithm, if invoked by the detailed router will have a high likelihood of finding a solution.

Cost functions for global routing: The cost of the edge between the two nodes in the GR-Graph represents the measure of congestion

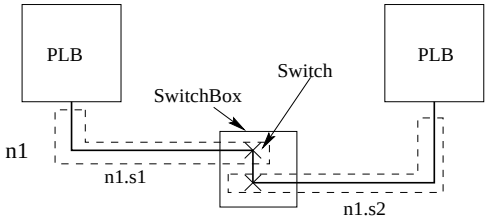

Figure 3: s_1 *and* s_2 *are subnets of net* n_1.

Algorithm GRoute(n_i) */* Steiner-tree approx. global routing using repeated application of Dijstra's shortest path based on a "distance" metric or* cost *that includes: a weighted sum of: (1) net length, (2) channel congestion and (3) a measure of bumping probability of existing nets by the detailed router */*

Begin

Sort pins of n_i by Manhattan distance to center of BBox;
for(p_1 and p_2) */* first and second pins */*
Make bounding box, BBox(p_1,p_2) ;
MinCostPath(p_1, p_2) ; */* Dijstra's shortest path algo. in BBox (p_1,p_2) */*
Mark nodes on path as sink (= s) ;
endfor
q ← center(p_1,p_2) ; */* q is the new center of BBox*/*
for $p \leftarrow 2$ **to** P_0 */* P_0 is total number of pins */*
Make bounding box,BBox(p,q); */* From previous center to the current pin */*
MinCostPath(p,s); */*from* p *to one of the sinks(=s) */*
Mark all nodes on path as sink (= s) ;
$q \leftarrow$ center(p,q) ;
endfor

End

Figure 4: *Steiner tree heuristic for global routing.*

in that channel. Also, each SBox and hence the node has a *basecost*. This takes care of the net length, more nodes on the path implies a longer net and hence higher cost . The congestion in a channel is captured by the resource usage in the SBox of that channel, which is basically the number of switches used in the SBox.

We have used two cost functions. *gcost1* is used for the routing scheme which includes only global and detailed routing.

$$gcost1 = \alpha \cdot base\_cost + \beta \cdot s\_box \qquad (1)$$

where α and β are the weighting factors for net length, channel congestion respectively.

The global routing cost for B&R is different from the one given in Eqn. 1. Here we also estimate possible net bumping cost that may be incurred later by the detailed router:

$$gcost2 = \alpha \cdot base\_cost + \beta \cdot s\_box^i + \gamma \cdot \sum_{s\_box^i} (netlength) \qquad (2)$$

The third term in Eqn. 2 is the total length of the net in that SBox, which captures the global bumping cost—it is easier in general to find converging T-DAGs by bumping shorter nets than longer ones.

3.2 Detailed Routing

After the global router assigns channels (or SBox's) to the net to be routed, detailed routing is done. The objective of the detailed routing is to do a feasible assignment of tracks and switches for the net, with minimum utilization of routing resources (tracks in channel and switches in SBox).

The general SBox model that we use is one in which smaller switches within a SBox are shareable between various $T_i \rightarrow T_j$ interconnections (as opposed to being dedicated to specific interconnections) with only one connection being able to use a switch at any time. Thus a switch resource cost (= the number of switches used) is

Algorithm DRoute(n_i)

Begin

1 **for** each branch Br of GTree */* GTree is the channel Steiner tree returned by GRoute */*
2 **if**(Br is the first branch of GTree)
3 MinCostPath(p_i, p_j); */*p_i, p_j are pin nodes on the first branch of GTree*/*
4 **if** MinCostPath(p_i, p_j) bumps existing nets
5 Get bumped subnets in set $\mathcal{B}$;
6 **endif**
7 **else begin**
8 Let Br = (p_k, S), where p_k is a pin node and S a SBox determined as the Steiner point for Br by GRoute;
9 Mark all switches in SBox S used by the detailed route created so far for n_i as s /* potential sinks for connection to p_k */
10 MinCostPath(p_k,s); */*min. cost path from pin node p_k to any switch node s which is marked.*/*
11 **if** Min_Cost_Path bumps existing nets **then**
12 Get bumped subnets in set $\mathcal{B}$;
13 **endelse**
14 **endfor**
15 **for** each subnet $s_i \in \mathcal{B}$ */* the bumped subnets */*
16 **B&R(s_i)** ; */* use the B&R algorithm described in Fig. 9 */*

End

Figure 5: *Algorithm for detailed routing with possible bumping of existing nets.*

incurred whenever a specific $T_i \rightarrow T_j$ connection is made via a SBox. Just like in track segment assignments, it is also possible to bump into other nets if a particular $T_i \rightarrow T_j$ connection for a net n_k uses a switch currently occupied by another net n_l; in such cases besides the switch resource cost, a bumping cost (see Eqn. 3) below is also incurred. This bumping of n_l also needs to be resolved in a manner similar to the bumping of a net from a track segment; the integrated bump-and-refit for both types of net bumping is solved by Algorithm B&R given later in Fig. 9. The model can also be extended to SBox's with some or all dedicated switches used to make only specific $T_i \rightarrow T_j$ connections by not having any resource cost associated with the use of such switches. This SBox model is general enough to capture SBox's found in commercial FPGAs such as ORCA, Virtex and Virtex-II; parameters such as switch resource cost and bumping cost within an SBox will need to be appropriately instantiated to apply to a specific FPGA.

Figure 6 shows a specific type of SBox that is similar to one used in the ORCA, and which follows the general SBox model. The switches in the ORCA are not dedicated and can be used to connect in either the vertical or horizontal direction. Hence there is a cost associated with their usage. A SBox is modeled as a connection graph called the detailed routing graph *DR-Graph*. The switches of a SBox are nodes in the DR-Graph. Two switches that can be directly connected (electrically), have an edge between them. A track is also modeled as a node (called segment node) in the DR-Graph which has an edge to all switches of the SBox on that track. A segment node is used to enter and leave the the SBox. Also the pins of a PLB have equivalent pin nodes in the DR-Graph. In Fig. 6(a) switch 1 is a segment node of the left SBox on track T_2. It has an edge to switches 4 and 7, on track T_2 in the same SBox. If the two SBox's are connected in the GR-Graph then the corresponding segment nodes (which is the same track) are also connected in the DR-Graph. In Fig. 6(a)&(b) segment nodes 1 and 8 correspond to the same track T_2); hence they are adjacent in the DR-Graph. In Fig. 6(b) node 8 has edges to both switch nodes 11 and 14, since a net on the track segment correspond-

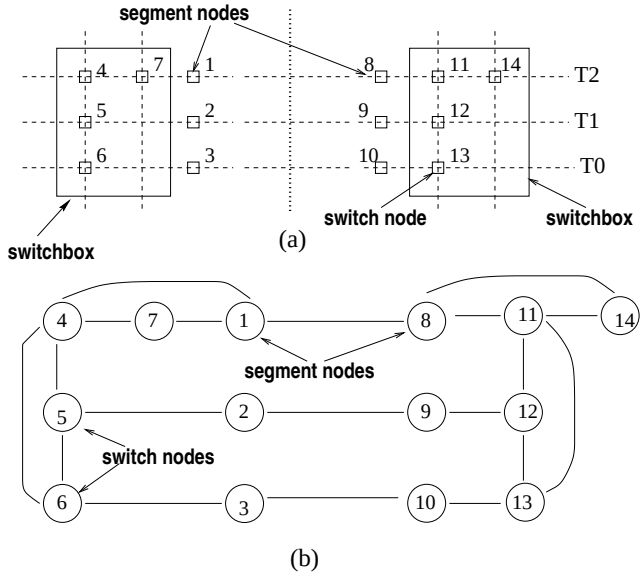

Figure 6: *(a) Internals of a SBox showing switches. The switch box is similar to the one in the ORCA FPGA. (b) Connection graph of switches (DR Graph) used in detailed routing.*

ing to node 8 can connect to the track segment to its right via switch node 14 and bypassing switch node 11; thus another net can use node 11 to make a vertical connection.

In Algorithm DRoute, Detailed routing is performed in the order in which the corresponding branches were created in the global routing tree (*GTree*); see Fig. 5.

Cost function for detailed routing: The cost of a switch node sw_i in DR-Graph for detailed routing is given by

$$dcost(sw_i) = \alpha \cdot sw\_base\_cost + \beta \cdot netlength(sw_i) \qquad (3)$$

where α and β are weighting factors. *sw_base_cost* is used to minimizes the number of switches and hence optimize the resource usage in SBox and $netlength(sw_i)$ is the length of the net, if any, which is currently using switch sw_i; it thus represents the "bumping cost" for any new net that needs to use an occupied switch.

3.3 Detailed Routing with Bump and Refit

Figure 5 describes our detailed router DRoute that incorporates B&R when a non-bumping path does not exist for the net in question. If β is made much larger than α in Eqn. 3, it is clear that DRoute would choose a non-bumping solution if one exists in the channels allocated by the global router[2]. The horizontal channel in the i^{th} routing row is denoted by H_i and a vertical channel in the j^{th} routing column is denoted by V_i. Sometimes the routing resources are not available in those channels to complete a valid route; see Fig. 8(b). In Fig. 8(a) n_1 needs to be to be rerouted and connected to PLB C_1. The global router allocates channel H_1 for the reroute. In Fig. 8(b) net n_4 occupies track T_2 in channel H_1 from V_2 to V_3. Net n_2 occupies track T_1 in channels H_1 from V_1, to V_3 and net n_3 occupies track T_0 in channel H_1 from V_0 to V_1. There are no routing resources available for net n_1 to connect from PLB A_1 to C_1. Hence the detailed router fails to find a valid route. However, if we bump n_4 to track T_0, track T_2 in channel

[2]Though this is not necessarily desirable as longer switch routes within SBox's (this, however, does not contribute significantly to any net length increase, but increases resource usage and net delays) may be chosen to avoid bumping, while a bumping solution may be able to rearrange the nets so that each uses near-minimum switch routes.

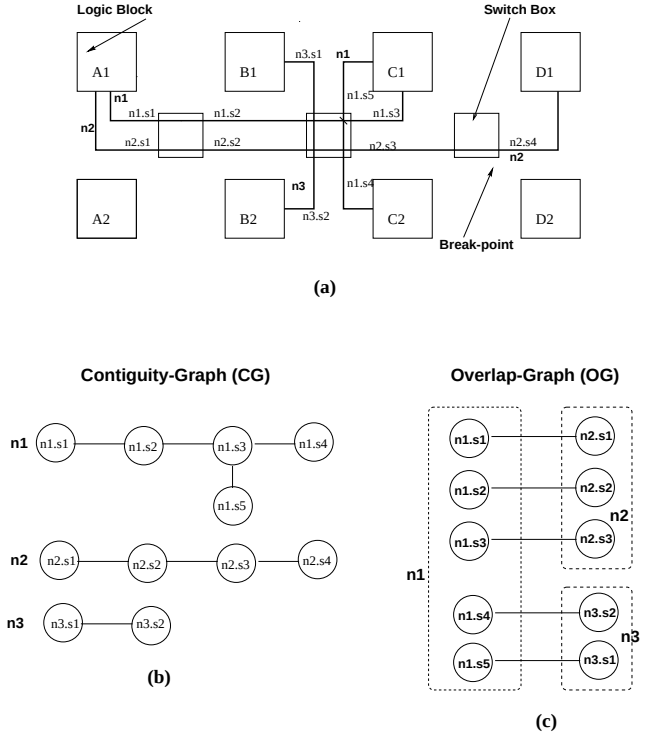

Figure 7: *(a) Routing of nets with their subnets. The corresponding (b) contiguity graph, and (c) overlap graph.*

H_1 is vacated for n_1. Hence a valid route from PLB A_1 to PLB C_1 is created for net n_1; see Fig. 8(c).

A *Contiguity graph (CG)* of a net is the connection graph of subnets. Each subnet is a node in the graph. The subnets which are adjacent in the layout have an edge between them in CG. The adjacent subnets are electrically connected. The concept of *overlap graph* is defined similar to definition in Sec. 2 [4] with a subnet being a node in the graph instead of a net.

Figure 7 shows an example of circuit routing and how the *OG* and *CG* are created. Subnet $n_1.s_1$ is connected to subnet $n_1.s_2$ (see Fig. 7(a)), and hence their corresponding nodes have an edge in the *CG*; see Fig. 7(b). Subnet $n_1.s_1$ and $n_2.s_1$ share a channel in the FPGA (see Fig. 7(a)), and hence their corresponding nodes have an edge in the *OG*; see Fig. 7(c) .

3.4 The B&R Algorithm

The detailed router may not always find a non-bumping path for the net which needs to be rerouted due to ECO. We call this net an *R-net*. The R-net bumps out the the nets occupying routing resources it needs; these nets are called *o-nets*. The subnets of the o-nets which occupy the routing resources of R-net are called *o-subnet*. The o-subnets have to make a transition to a different track. The formal description of the algorithm is give in Fig. 9

We have developed two methodologies for performing B&R: `subnet_B&R` and `subsec_B&R`. In the `subnet_B&R` methodology we bump only the O-subnets of the O-net. The cost of bumping a subnet s from track T_i to T_j is given by equation 4.

$$bumpcost(s^{T_i \to T_j}) = \text{length of subnet on } T_j \text{ bumped by } s. \qquad (4)$$

In equation 4 since s is a subnet, it may bump at most one subnet of another net.

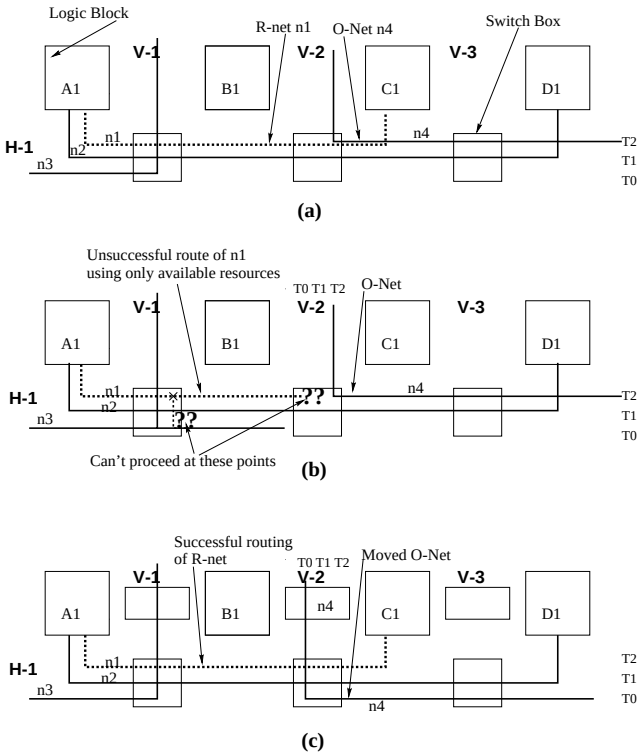

Figure 8: *(a) PLB B_1 is faulty, net n_1 needs to be rerouted. (b) Detailed router is unable to route through the channels allocated by global router (c) Net n_4 is bumped to track T_0 and net n_1 can connect to C_1.*

This bumping cost is similar in concept to the one described in Sec. 2 [4]. The subnets contiguous to o-subnets remain in their own track. and maintain electrical connection to o-subnets via switches of the SBox. This may require some extra switch usage of the SBox. If the switches are not available then the nets occupying them are bumped out.

The `subsec_B&R` methodology was designed to reduce the extra switch usage in the SBox to maintain electrical connection between contiguous subnets of o-net. In this methodology we compute the optimal subsection of the o-net to be perturbed. The optimal subsection includes the o-subnets and maybe some other subnets of o-net. To compute the optimal subsection we calculate the minimum cost of transition of subnets to be perturbed. Cost of transition of subnet s_i to any track T_k is the sum of bumping cost and the cost to connect to all subnets adjacent to it in its CG.

In Fig. 10 subnet x^{T_i} is the *o-subnet* and subnets y^{T_p} and z^{T_l} are subnets adjacent x in the *CG*. The computation of minimum cost of transition ($dynamcost(x^{T_i \to T_j})$) for a generic subnet x from track T_i to track T_j is given by:

$$dynamcost(x^{T_i \to T_j}) = \min_{\forall T_q}[SBox^{xy}(T_j \to T_q) + dynamcost^{-x}(y^{T_p \to T_q})] + \min_{\forall T_k}[SBox^{xz}(T_j \to T_k) + dynamcost^{-x}(z^{T_l \to T_k})] + bumpcost(x^{T_i \to T_j}) \quad (5)$$

In the above dynamic programming formulation the subnet x moves to track T_j so the subnets y and z should make a transition to connect to x on track T_j. If y makes a transition to track T_q, it incurs a SBox connection cost $SBox^{xy}(T_j \to T_q)$ since there needs to be a connection from track T_j to T_q in that SBox and a transition cost from track T_p to T_q ($dynamcost^{-x}(y^{T_p \to T_q})$). In computing

```
Algorithm B&R(s_i)   /*s_i is the bumped subnet */
Begin
1   TranSet = MinCostCalc(s_i);   /* cost-ordered transition set of sub-
    nets (including s_i) of the net n_r containing s_i */
2   for j ← 0 to t − 1 /* t is the total tracks */
3   ChildList = GetChildList(TranSet[j]);   /* set of subnets occupy-
    ing the new tracks of subnets of n_r that are chosen for movement in
    TranSet[j] */
4       if(ChildList == NULL)              /*there is no bumping */
5           DoUpdates(TranSet[j]);         /* update OG and CG */
6           return success;                /*Converging transition set found for s_i */
7       endif
8       else begin
9       if any child is an ancestor /* this leads to a cycle */
10      break;   /* take next best transition */
11      DoUpdates(TranSet[j]); /* update OG and CG */
12      numsuccess = 0; /*keep track of number of successful T-DAGs */
13      for each C ∈ ChildList
14          ReturnFlag = B&R(C)
15          if( ReturnFlag == Fail) break;
16          else numsuccess++;
17      endelse
18      if(numsuccess == |ChildList| ) /* converging T-DAG for all children
        found */
19          return success;
20  endfor
21  ReturnFlag = Subnet_B&R(s_i); /* Subsec_B&R has failed. Use the sim-
    pler version Subnet_B&R(s_i) that bumps only those subnets of bumped
    nets that are bumped by the newly routed net or by other bumped subnets
    */
22  return ReturnFlag;
End
```

Figure 9: *Bump and refit algorithm to find a converging transition set.*

$dynamcost^{-x}(y^{T_p \to T_q}$ the subnet x which is ancestor of y is not considered adjacent to y.The *dynamcost* computation for a node terminates when all nodes adjacent to it in its *CG* are its ancestors in the computation tree. $SBox^{xy}(T_j \to T_q)$ is the minimum cost path within the SBox computed by function MinCostPath to make the $T_j \to T_q$ connection (see Fig. 5), where the cost of each node in the SBox is given by equation 3.

Using the recursive function *dynamcost* of Eqn. 5, Algorithm `MinCostCalc` (Fig. 11) computes an array of the t best transition patterns of all subnets of a net n_i, given that a particular subnet s_i of n_i has been bumped.

We next establish the optimality of the `MinCostCalc` algorithm.

Theorem 1 Algorithm `MinCostCalc` (Fig. 11) for an o-subnet of a net n_j returns an optimal subsection solution with respect to the cost

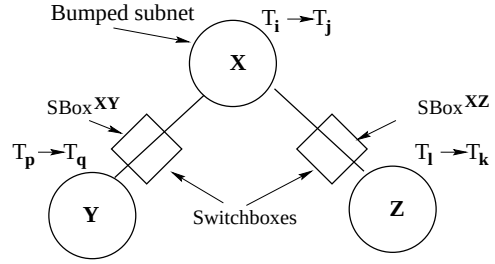

Figure 10: *Subnet X is bumped which is initially on track T_i. Subnets Y and Z are adjacent to X in its CG. $SBox^{XY}$ is the switch box connecting adjacent channels on which subnet X and subnet Y lie.*

Algorithm `MinCostCalc`(s_i)
Begin
for $T_j \leftarrow 0$ **to** $t-1$; $T_j \neq T_{curr}$ /*T_{curr} *is current track of* s_i. *t is total number of tracks */*
dynamcost($s_i^{T_{curr} \rightarrow T_j}$); /* *see Eqn. 5* */
TranSet[j] = *dynamcost* ($s_i^{T_{curr} \rightarrow T_j}$; /**store the track-set for a transition in an array*/
endfor
sort(TranSet) /* *sort the TranSet array in increasing order of cost* */
return (TranSet)
End

Figure 11: *Algorithm to calculate the minimum cost subsection of the bumped net to be perturbed.*

metric used in Eqn. 5. Further, the time complexity of `MinCostCalc` is $O(st^3 \log t)$, where s is the number of subnets of n_j and t the number of tracks in a channel.

Proof: Let x be an *o-subnet* of net n_j and $y \in adj(x)$ in n_j's CG. For each transition T_j of x, where $T_j \neq$ current track of x, *dynamcost* recursively computes the min-cost transition for each y which includes the SBox connection cost to x^{T_j}. Since a net route is a tree (it has no cycles), the transition costs of each y is independent of the transition of its parent x in the dynamic programming computation tree, i.e., in Eqn. 5, $dynamcost^{-x}(y^{T_p \rightarrow T_q})$, for e.g., is independent of the track T_j to which x transits. Thus the transition costs of the y's to different tracks T_q can be computed once and then added to the appropriate $SBox^{xy}(T_j \rightarrow T_q)$ to compute the minimum cost subnet to track assignments of y and all its "children" subnets in the computation tree to remain connected to x, when x transits to T_j.

The minimum transition cost of $x^{\rightarrow T_j}$ is then the sum of minimum transition cost of all y's plus the $bumpcost(x^{\rightarrow T_j})$ (again, the transitions of each y and their children subnets in the computation tree are independent of the other y's and their children subnets). This is exactly the cost computed in $dynamcost(x^{T_i \rightarrow T_j})$ in Eqn. 5. `MinCostCalc` then orders by increasing cost all the $t-1$ $dynamcost(x^{T_i \rightarrow T_j})$ transition costs of the *o-subnet* x; it thus returns an optimal solution as the first cost in the returned cost array.

Since there are $\Theta(t)$ switches and connections in a SBox (and thus in its DGraph), each $SBox^{xy}(T_j \rightarrow T_q)$ cost can be computed in $\Theta(t \log t)$ time, using Dijstra's shortest path algorithm [3] and thus the $\min_{\forall T_q}[SBox^{xy}(T_j \rightarrow T_q) + dynamcost^{-x}(y^{T_p \rightarrow T_q})]$ computation takes $\Theta(t^2 \log t)$ plus the time to compute $dynamcost^{-x}(y^{T_p \rightarrow T_q})$. Since there are a constant number of subnets $y \in adj(x)$, the computation of Eqn. 5 takes $\Theta(t^2 \log t)$ plus the time to compute $dynamcost^{-x}(y^{T_p \rightarrow T_q})$ over all $y \in adj(x)$.

Further, over all possible $t-1$ transitions of x it takes `MinCostCal` a time of $\Theta(t^3 \log t)$ plus the time to compute $dynamcost^{-x}(y^{T_p \rightarrow T_q})$ over all y's; note that the $dynamcost^{-x}(y^{T_p \rightarrow T_q})$ computation needs to be done only once irrespective if the transitions of x. At a leaf subnet u in the computation tree, it takes $\Theta(t)$ time to compute the *bumpcost* of this subnet to $t-1$ possible tracks, and this is the only computation involved in $dynamcost^{-v}(u^{T_p \rightarrow T_q})$ over all T_q, where v is u's parent in the computation tree. If there are k subnets among a y and all its children, then in the worst case (when the subtree rooted at y is a path), it will take $O(t + (k-1)t^3 \log t = O(kt^3 \log t)$ time to compute $dynamcost^{-x}(y^{T_p \rightarrow T_q})$ over all T_q. Since the sum of all such k's over all y's, $y \in adj(x)$, is $s-1$, `MinCostCalc` has a complexity of $O(st^3 \log t)$. □

In Fig. 12 $n_1.s_2$ is the o-subnet which needs to move out from track T_2 in order to route n_2 to connect to pin p_1 of PLB C1 via the switch being currently occupied by $n_1.s_2$. If $n_1.s_2$ alone moves to track T_1

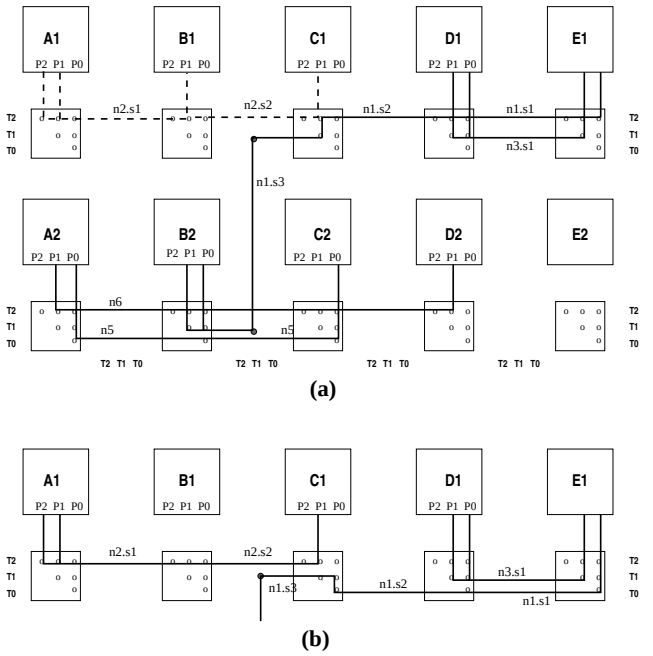

Figure 12: *Net n_2 needs to be reouted and connect to pin p_1 of PLB C_1; subnet $n_1.s_2$ occupies the track and switch required by net n_2. (b) The final routing.*

it will bump $n3.s1$ in order to connect to $n_1.s_1$ ($T_1 \rightarrow T_2$ connection) via the switches used by $n3.s1$; this incurs a high SBox cost. If $n_1.s_2$ moves to track T_0 it will also similarly incur a high SBox to connect track T_0 to T_2. If both $n_1.s_2$ and $n_1.s_1$ are moved to track T_0, then they do not bump into any other net either on T_0 or on any switches needed maintain connection between them; hence a low SBox cost is realized. We also see from Fig. 12(a) that if $n_1.s_3$ makes a transition to track T_2 or T_0 it bumps into net n_6 or n_5, respectively, resulting in a high bumping cost. If $n_1.s_3$ remains at its current track, it neither bumps into any net nor incurs a high switching cost to maintain connection to $n_1.s_2$ if the latter moves to T_0; see Fig. 12(b). Hence $n_1.s_2$ and $n_1.s_1$ is the optimal subsection to bump, and is returned as the min-cost transition by MinCostCal (Fig. 11).

4 Experimental Results

Our B&R incremental routers were tested on Lucent's ORCA-2C FPGA, which has a 10×10 PLB array. In our current implementations we only use track segments of unit length. The input to our software are placed and routed circuits created using Lucent's graphical tools. Nets spanning the range from local to global were created—within the 10×10 PLB array, net lengths ranged from two (local) to 23 (global) spanned PLBs, with the average net length being 7 across all circuits. The experiments were performed on 550MHz Pentium-III Linux and 167 MHz Sun Ultra 1 Solaris machines. Our initial simulations on `B&R`, `A_PAR` and `Rip-up&Reroute` where done on the Ultra 1 machine. We then migrated to Pentium-III machine for later versions of `B&R` ; `A_PAR`, however, does not run on Linux, and so based on the `B&R` and `Rip-up&Reroute` run times on the two machines, we obtained the speed ratio and scaled the `A_PAR` run time to the Pentium-III equivalent time. In Table 1 we report the CPU time for the Pentium-III machine (among other parameters). To simulate ECO, we randomly removed 5%, 10% and 15% of the original nets, and added the same percentage of new nets with the same number of pins but with random pin positions. The metrics of interest for

Circuit	PLBs	Nets	A_PAR	%New Nets	Standard		Rip-up&Reroute			Subnet_B&R		Subsec_B&R	
			time (msec.)		Avg. len	time (msec.)	Avg. len	time (msec)	Ripped-Up %Δlength	Avg. len.	time (msec)	Avg. len	time (msec.)
123.ncl	80	83	1200	5	8.8	74	8.8	70	-	8.8	38	8.8	51
				10	9.5	110	9.18	90	+12.75	9.8	60	9.05	80
				15	9.2	160	8.95	131	+15.75	9.3	110	9.1	120
Eco-Bist2	66	42	550	5	8.1	20	8.1	20	-	8.1	20	8.1	20
				10	9.4	45	8.87	48	-	9.1	37	8.7	39
				15	8.28	59	8.25	67	-3.1	8.22	52	8.15	57
Eco-Big	79	66	1230	5	8.7	50	8.6	48	-	8.6	49	8.6	43
				10	8.6	108	8.2	110	+4	8.5	98	8.4	96
				15	9.4	146	8.71	142	+11	8.8	120	8.69	129
Eco-Bist4	67	47	770	5	7.8	57	7.8	54	-	7.8	51	7.8	51
				10	10.1	90	9.7	83	+7	9.67	30	9.97	27
				15	10.8	101	8.9	98	+30	9.36	42	9.06	46
Eco-Big2	80	84	1380	5	8.6	58	8.4	57	-	8.4	37	8.5	41
				10	9.7	108	9.41	101	+20	9.45	54	8.97	63
				15	10.6	145	8.93	131	+14.5	9.31	88	8.14	108
Average				5	8.4	51.8	8.34	49.8	-	8.34	39	8.36	41.2
				10	9.46	92.2	9.07	86.4	+8.75	9.3	55.8	9.01	61
				15	9.65	122.2	8.75	113	+13.6	9.0	82.4	8.74	92

Table 1: *ECO routing was simulated for 20 runs in each case.* A_PAR *is the time taken by ORCA's auto-place and route tool to do complete routing. The* Avg.length *is the average length of the net to be rerouted.* Ripped-Up %Δlength *is the change in the length of bumped nets in* `Rip-up&Reroute`

incremental rerouting for ECO applications are run time, length and delay of rerouted nets and change in the electrical properties (like power and delay) of the circuit. The aim of any incremental router should be to have minimal effect on the rest of the nets and hence minimally affect the electrical properties of the non-ECO portion of the circuit. We compared the two versions of the B&R algorithm to the two prior techniques `Standard` [5] and `Rip-up & Reroute` [2] implemented by us. All the incremental routing techniques that we experimented with have global and detailed routing phases. In `Standard`, if the detailed router is unsuccessful, the bounding box for the global router is increased. In the `Rip-up&Reroute`, if the detailed router is unsuccessful, the existing nets occupying the needed routing resources are ripped up and rerouted using global and detailed routing. The nets ripped up are exactly those that the detailed router bumps; we use a similar detailed router as in the B&R approach. This provides very specific information on which nets to `Rip-up&Reroute` which is an advantage to it.

From Table 1 we can summarize the following results:

- The `Subnet_B&R` methodology is the fastest. It is nearly 30% faster than `Standard` and nearly 27% faster than `Rip-up&Reroute` for the 15% new net case. The `Subsec_B&R` method is nearly 25% faster than `Standard` and nearly 18% faster than `Rip-up&Reroute` for the above case.
- The `Subsec_B&R` methodology produces the best solution quality. The average length of the new nets is nearly 12% smaller than `Standard` for the 15% new net case.
- The average length of the new nets is almost same for the `Subsec_B&R` and `Rip-up&Reroute`. However, in `Rip-up&Reroute`. the average length of the ripped-up (bumped) nets increased by 13% for the 15% new net case and and by 8% for the 10% new net case. In both the B&R techniques, the length of the existing bumped nets do not change, a very significant advantage.
- As expected, the `Subnet_B&R` is on the average 10% faster than `Subsec_B&R` , but the length of the rerouted nets in `Subsec_B&R` is nearly 3% smaller than that of `Subnet_B&R`.
- Our incremental routers are nearly 10 to 20 times faster than Lucents auto-place and route *A_PAR* when used in complete rerouting framework.

5 Conclusions and Future Work

The B&R approach of [4] proposed in the context of simple net extensions for fault tolerance was significantly extended and new concepts like bumping costs during global routing, and optimal bumping subsets of nets were developed in this paper to realize a novel incremental routing technique for ECO applications in FPGAs. Our incremental routers are nearly 27% faster and nearly 10% shorter in terms of net length than other proposed incremental routing methods. Further, our incremental routers do not change the lengths or topologies of non-ECO nets, and hence has minimal effect on the electrical properties of the non-ECO parts of the circuit. Our new B&R routers thus offer significant advantages in almost all metrics of interest to incremental routing. Our experiments also establish the importance of incremental rerouting over complete rerouting.

In future work we will incorporate all types of track segments and incorporate timing-driven aspects in the B&R process. We will also extend this work to regular VLSI chips.

References

[1] J.Cong, J.Fang and K.Khoo. "An Implicit Connection Graph Maze Routing Algorithm for ECO Routing".*Proc. IEEE Int. Conf. Comput.-Aided Design*, April 1999, pp. 12-18.

[2] J.Cong and Majid Sarrafzadeh. "Incremental Physical Design". *Proc. International Symposium on Physical Design* , April 2000, pp. 84-92.

[3] Cormen, Leiserson and Rivest. "Introduction to Algorithms". *McGraw-Hill Book Company*, 1990.

[4] S.Dutt, V.Shanmugavel and S.Trimberger, "Efficient Incremental Rerouting for Fault Reconfiguration in Field Programmable Gate Arrays",*Proc. IEEE Int. Conf. Comput.-Aided Design*, 1999.

[5] J.M.Emmert and D.Bhatia. "Incremental Routing in FPGA's". *Proc. IEEE Int. ASIC Conference and Exhibit*, 1998.

[6] F. Hanchek and S. Dutt, "Design Methodologies for Tolerating Logic and Interconnect Faults in FPGAs",*IEEE Trans.Computers*, Special Issue on Dependable Computing Jan 1998.