Routing Algorithms: Architecture Driven Rerouting Enhancement for FPGAs

Taraneh Taghavi
CS Dept., UCLA
taghavi@cs.ucla.edu

Soheil Ghiasi
ECE Dept., UCDavis
soheil@ece.ucdavis.edu

Majid Sarrafzadeh
CS Dept. UCLA
majid@cs.ucla.edu

Abstract

The routing channels of today's FPGAs consist of wire segments of various types, which allow the use of new techniques to enhance the routability of net segments in channels.

In this paper we present an optimal greedy algorithm to switch the tracks that net segments are assigned to. This allows us to enhance the rerouting ability by capturing the features of the routing architecture. Suppose the number of tracks in the channels is given. The goal of this algorithm is to increase the number of routed segments of late rerouting requests. This is a good feature for supporting Engineering Change Order(ECO) type of routing. Supporting ECO routing enables the routing algorithms to deal with later changes in routing requests. We used the routing architecture of VirtexII FPGAs from Xilinx as our target architecture and integrated our algorithm into the VPR FPGA routing tool. The experimental results show that our algorithm makes VPR router capable of handling 28.4% more rerouting for segments that are added to the design later.

Keywords

FPGA CAD, Routing, Greedy algorithm, Left Edge Algorithm.

1. Introduction

The problem of routing in FPGAs is a very challenging problem due to the different types of wire segments, and the limited number of connections in channels. Traditionally, routing is done in two stages of global routing and detailed routing sequentially [13, 11, 12, 8]. In the two stage routers, the global router abstracts the details of the routing architecture and performs routing on a coarser architecture. Then, the detailed router refines the routing done by the global router in each channel. Although this approach can be applied to large circuits, the global router may not be able to exploit an accurate abstraction of the routing architecture. This may lead to a degradation of the routing quality. This problem may emerge specifically in FPGAs. Here, the important issue is not only the finding of a routing path, but also the assignment of different wire types to each net segment and the order of this assignment. To alleviate this problem, in [1] a wire type assignment algorithm was presented that is based on iteratively applying the min-cost maximum flow technique. Another approach to solve the problem of routing is to use the one-stage detailed routing algorithms [10, 9, 5, 2]. Although this approach is not able to handle very large routing architectures, it is more accurate, since it can embed all the features of the routing architecture using the detailed routing graphs. In [2], VPR an FPGA placement and routing tool, was introduced which is based on the Pathfinder negotiation-based router for FPGAs [5] that uses Dijkstra's algorithm(i.e. a maze router [15]). This router uses an iterative algorithm that converges to a solution in which all signals are routed while achieving close to the optimal performance allowed by the placement.

In this paper we address the FPGA detailed routing problem. The problem to consider is that given a detailed routed circuit, we want to pack segments in tracks of single wire type as much as possible in order to maximize the number of empty spots in those tracks. These empty spots can accommodate more net segments. In fact, we capture the features of routing architecture of the framework and use them to enhance the rerouting ability. In this way, we are able to support ECO(Engineering Change Order) requirements for routing. For this purpose, we define an optimization problem, present an optimal greedy solution for it, and prove the correctness of it. We applied our technique on the VPR, FPGA placement and routing tool, and present the effect of it on the experimental section of the paper. The main contribution of this paper is that by having the number of the tracks in the channels, we can handle more rerouting requests than pure VPR for net segments which are added to the design later.

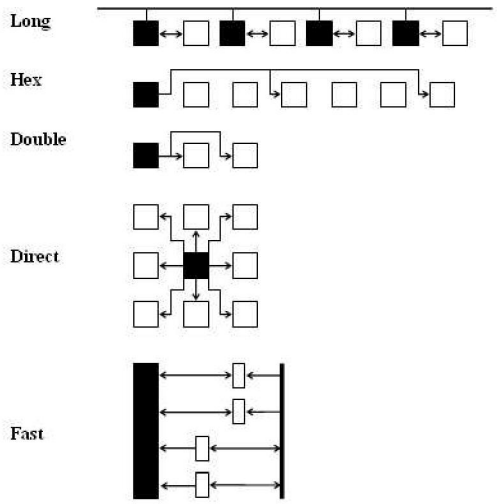

Fig. 1: VirtexII device route

2. Preliminaries

The model we used for FPGA in this paper is an array based FPGA similar to Xilinx VirtexII architecture [7]. Each programmable element in this family of FPGAs is tied to a switch matrix, allowing multiple connections to the general routing matrix. Figure 1 (white squares denote CLBs and black squares denote route switch boxes) shows types of routes that occur in VirtexII family of Xilinx devices. Vertical and horizontal long lines span the full height and width of the chip. Hex lines route signals to every third or sixth CLB in all four directions. Double lines route signals to every first or second CLB in all four directions. Direct lines connect signals to neighboring blocks: vertically, horizontally, and diagonally.

3. Problem Modeling

Basically, the goal is to maximize the number of empty spots of length 2, 3, and 6 that are inevitably made in tracks of single wire type after doing routing. These empty spots can be used for routing other nets which could not be routed previously. The reason that we choose these kinds of segments is the routing architecture of today's FPGAs which has been shown in Figure 1. This routing architecture mostly consists of net segments with length less than 6. Our assumption for connectivity of segments of a wire type is that, each segment can just be connected

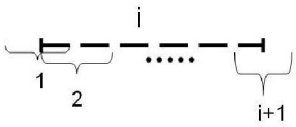

Fig. 2: O_{double} for a segment of length i

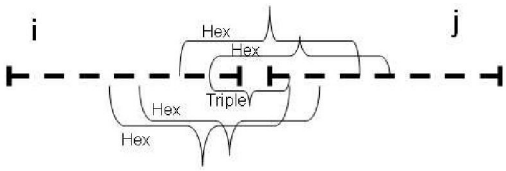

Fig. 3: Gain achievement for two segments of gap 1

from its start and end points to the other segments. We categorize hex segments as two categories, triples and hex. First of all, we define some notations and prove some lemmas based on them. Then we adapt our problem to an optimization problem we are going to solve. For every net of length i we define $O_{double}(i)$, $O_{triple}(i)$, and $O_{hex}(i)$ as the maximum possible number of double, triple(hex line used to connect each CLB to a CLB three spots further), and hex routing segment a net segment with length i can occupy.

Lemma1. The number of spots of length 2, 3, and 6 on the routing segments which a net segment of length i occupies is equal to $i+1$, $i+2$ and $i+5$, respectively.

Proof. Figure 2 shows how the O_{double} for a segment of length i can be calculated as $i+1$. This means that if we put a segment of length i in a track, it may take away the opportunity from each of $i+1$ double wire segments which have overlap with this net segment for later routing. The other parameters can be calculated in the same way. So, we define those parameters as follows:

$O_{double}(i) = i+1, \quad O_{triple}(i) = i+2, \quad O_{hex}(i) = i+5.\square$

Lemma2. On the routing tracks, the total number of occupied spots by two consecutive net segments(the start point of one is connected to the end point of the other), is less than the sum of occupied spots by each of them.

Proof. Let us consider two segments of length i and j that are non-overlapping. If these two segments have a distance gap of zero(i.e. the end point of one of them is the start point of the other), the total number of segments of lengths 2,3, and 6 which they occupy is less than the sum of their O's values. The reason is that some of those occupied spots at the end-points of segments i and j are calculated twice, once for segment i and once for segment j. $\square$

The difference between these two values is the gain that we can achieve by putting these two segments closer to each other as much as possible. Figure 3 shows how we can achieve gain for two segments with gap 1 by calculating the overlapping spots just once. In Table 1, the gain that we can make for different gaps is calculated.

Intuition for greedy approach. A trivial observation based on Lemma1, Lemma2, and Table1 shows that the less the gap between the segments, the more we can pack the segments, the more the gain we can achieve, and thus the greater the number of empty spots with lengths 2,3, and 6 on the routing tracks of segments of length 1 can be created.

segment length	Gap 0	Gap 1	Gap 2	Gap 3	Gap 4
Double	1	0	0	0	0
Triple	2	1	0	0	0
Hex	5	4	3	2	1
Total	8	5	3	2	1

Table 1: The Gain achieved for each segment type based on Gap between two net segments

4. Problem Definition

Segment Track Switch problem (STS): Given a list of routed net segments of length 1 and the number of available tracks of single wire type, we want to switch the place of these routed segments through those tracks in order to maximize the number of empty spots of length 2,3, and 6 created after routing those tracks, meaning maximizing the total amount of gain we get for all channels which can be computed by Table 1.

The number of switches between vertical and horizontal wires, and the connections between CLB inputs/outputs to wires inside tracks are limited. So, one very important point to consider when assigning net segments is dealing with vertical constraints. Net segments cannot be assigned easily because of vertical constraints. To deal with this issue, we make an assumption.

Assumption1. We assume that all switching matrices are the same, and each switching matrix have the same configuration of switches to all segments of the same type.

Also, it should be noticed that the STS problem is not the same as channel routing with vertical constraints which is an NP-hard problem in general. STS problem considers each channel individually and tries to reassign net segments to maximize the number of empty spots. These empty spots can be used later to accommodate more net segments. In the next section, we show how to solve the STS problem with a greedy algorithm and also deal with vertical constraints at the same time.

5. Algorithm

In this section we model the STS problem as a routing problem and propose a greedy algorithm for it by simply assign intervals to the tracks considering the cost function defined in the previous section.

5.1 Greedy Algorithm

First we present a greedy algorithm which is a version of the Left Edge algorithm to solve the STS problem and then we prove the correctness of it. The greedy algorithm for STS is shown in Algorithm 1.

Algorithm 1 Greedy Segment Track Switch Algorithm

Input: Detailed routed nets N of one channel in tracks of single wire type
Output: Track assignment for $\forall n_i \in N$

sort all the net segments in non-decreasing order based on their left endpoints (start points).
assign the first net segment n_1 to the first track with no CVC(Conflict with Vertical Constraints) from either endpoints.
AlreadyAssignedSegments= $[n_1]$.
remove n_1 from N.
mark n_1 as the last segment of that track.
s_i and l_i are left and right endpoints of net segment n_i.
for all net segment $n_i = (s_i, l_i) \in N$ **do**
  find the biggest l_j such that $n_j = (s_j, l_j)$ is marked
  as the last assigned segment of each track,$l_j \leq s_i$,
  $s_i - l_j \leq 4$, and no CVC exists.
  if $\exists\ l_j$ and no CVC to assign n_i to the same track as n_j **then**
    assing n_i to the same track as n_j.
  else
    assign n_i to the the first available track with no CVC.
  end if
  mark n_i as the last segment of that track.
  add n_i to the AlreadyAssignedSegments set.
  remove n_i from N.
end for

In this algorithm, for each channel we just consider tracks that consists of single wire type, meaning wires of length 1. First we sort all net segments on those tracks based on their start points. We start from the first net segment, assign it to the first track which has no conflict with its vertical constraints, and put it into the set of already assigned

segments. For all other net segments, among all available tracks(i.e. are not occupied, so there is no overlapping segment there, or make no conflict with vertical constraints), the greedy choice is to pick the one that the previous net segment assigned to it has smaller gap with this new net segment with the condition that this gap is less than 4. If such a net segment does not exist, we pick the first available track. If the gap between the start and end point of these two segments is more than 4, the gain we obtain becomes 0 based on Table1. So, all tracks with those characteristics are considered as available tracks. They can be used to assign the new net segment, but have no priority over each other. If there is an overlapping segment in that track, we cannot assign new segments to that. So, that track is not available. This technique is similar to Left Edge algorithm for interval packing [14]. The difference occurs when there is more than one available track. In Left Edge algorithm there is no difference among available tracks, but in this algorithm our decision to pick a suitable track is based on the right endpoint of the last assigned net segment of that track.

5.2 Proof of correctness

Theorem1. The greedy algorithm given in Algorithm1 can solve the STS problem defined in the previous section.

Proof by contradiction: Assume there is another optimal solution. We want to show that this solution cannot be better than the greedy solution. Let us consider the first point where the difference between this solution and the greedy solution emerges(i.e. the first segment which is assigned to two different tracks in these solutions). We alter the non-greedy solution to the greedy one in this way. We swap the assignment of the segments of the two tracks from the starting point of the net segment which makes the difference all over to the end of the track. The gain we achieve will increase at the starting point of that segment, and will not change at the other points. So, total gain will increase. For all other points where the segments are assigned to different tracks, the same alteration can be done. It should be noticed that this swapping does not make any conflict with vertical constraints. The reason is that we pick the greedy choice as a choice that does not violate the vertical constraints. Also, the given solution is claimed to be valid so does not have any conflict with vertical constraints. With respect to Assumption1, the configuration of switches is the same for both segments that are considered in swapping, So swapping will not introduce any violation on vertical constraints. This shows that the greedy choice is always a better choice.□

5.3 Demonstration of the proof

Some example cases are shown in Figure 4:

1. Case 1: Point P is the place where the difference emerges in the optimal solution. We can simply swap tracks (a) and (b) from the point P onwards, so we will gain 8 based on the Gain function C defined in Table 1. The number 8 is the sum of the number of doubles, triples and hexes which are not destroyed by greedy choice. So, in every step the greedy choice is the better choice.

2. Case 2: We can simply swap the tracks (a) and (b) from the point P all over to the end, so we will gain $8-5=3$. Again, the greedy choice is the better choice.

3. Case 3: In this case there is no matter where we put the new net segment since none of the tracks makes a better gain. So every choice is the same as greedy choice.

For all other permutations of net segments the cases can be resolved like those mentioned above.

The time complexity of this algorithm is $O(nlogn+nT)$, where n is number of net segments and T is the number of tracks in each channel. Sorting all net segments takes $O(nlogn)$[6]. In each iteration of the for loop we assign one net segment by considering all of the tracks. This contributes to the second term of the time complexity which is $O(nT)$.

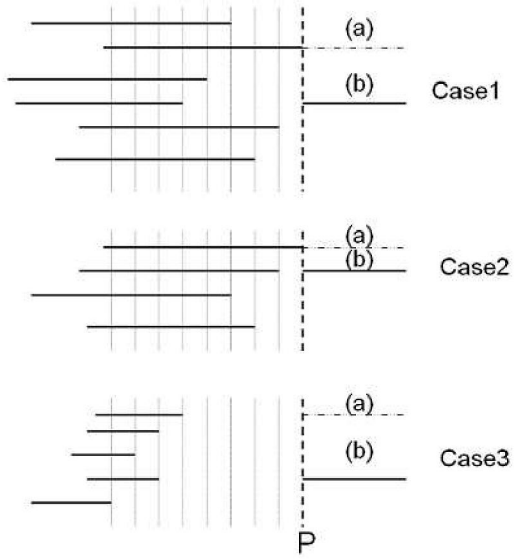

Fig. 4: Examples, demonstrating the greedy choice a and the non-greedy choice b

6. Engineering Change Order(ECO) Enabling

The presented algorithm has a motivating potential to be used for ECO purposes. An ECO is a request to make design changes, typically late in the design process [4]. This makes the framework very useful for the ECO type of routing, since we presented a way to change the detailed routing to be capable of enhancing the result in case the number of routing nets rise up later.

7. Experimental Results

Our experiments are based on the application requirements. Suppose the routing for a complex circuit is done. Later some requests for adding more functionality emerge. In lots of cases, new requests do not require changing the logic elements of the circuit or even their configurations. They just need adding more routing connections to the existing logic elements. Without supporting ECO routing , the designer have to do the routing from the first for every new request. By adding the features to enable ECO routing, the design can tolerate a lot more rerouting ability for new net segments. This may save a lot in designing time and resource consumption. The greedy approach has been implemented using the VPR 4.30 source code and manual [3]. VPR is an academic FPGA routing tool created to do placement and routing for FPGAs. We modified the source code of VPR and embedded our technique in it. VPR uses the Pathfinder algorithm. We implemented Algorithm 1 into the VPR source code after the point that the Pathfinder procedure finds paths for nets. By this point for each channel we know exactly which net segments are going to be routed on which tracks and on what kind of wire type. For all channels we applied Algorithm 1 on net segments located in each channel. It is obvious that the routing after applying our technique is denser and it has more empty spots, while before that the net segments are scattered in the channel. In this way we can reduce the total number of empty tracks throughout the circuit. We tested this idea on 20 MCNC benchmarks. From table 2 we can see that the total number of empty tracks on the circuit is increased by the average of 9%. These empty tracks are used for responding routing requests, which are added later.

To measure how well our technique can enhance the rerouting ability of the circuit, for each channel we created some randomly generated nets with length less than 6 and added them all to that channel. Then we tried to route them along with the existing nets before and after our technique. The results for both cases is given in table 2. It shows that by applying our technique the total number of unroutable nets of those randomly

	Total Used Tracks			Unroutable Nets			Execution Time		
Benchmark	Before	After	Improvement	Before	After	Improvement	Before	After	Increase
alu4	963	868	9.0%	45.7%	15.7%	30.0%	199.18	208.92	4.8%
apex2	1068	1037	3.0%	58.3%	21.3%	37.0%	247.16	261.38	5.7%
apex4	1093	945	13.0%	35.8%	11.6%	24.2%	91.49	99.13	1.1%
bigkey	865	789	8.0%	39.9%	13.5%	24.4%	175.54	190.47	8.5%
clma	2590	2376	8.1%	38.4%	3.7%	34.7%	2321.31	2468.37	6.3%
des	1136	1045	8.0%	22.7%	7.5%	15.2%	142.48	159.37	11.8%
diffeq	710	633	10.8%	49.6%	19.1%	30.5%	74.63	83.35	11.6%
dsip	741	695	6.2%	36.5%	18.8%	17.7%	285.67	298.31	4.4%
elliptic	1352	1232	8.8%	46.2%	13.5%	32.7%	822.85	862.82	4.8%
ex1010	1779	1611	9.4%	40.0%	5.4%	34.6%	483.84	538.18	11.2%
ex5p	1003	909	9.3%	47.0%	14.5%	32.5%	103.21	110.13	6.6%
misex3	1012	883	12.7%	43.5%	12.9%	30.6%	137.40	146.62	6.7%
frisc	2055	1732	15.7%	17.9%	1.0%	16.9%	368.36	411.08	11.6%
pdc	2735	2352	14.0%	25.2%	1.7%	23.5%	1003.60	1068.50	6.4%
s298	712	705	1.0%	75.6%	44.1%	31.5%	394.53	408.43	3.5%
s38417	1304	1282	1.6%	59.1%	21.8%	37.3%	754.55	836.47	10.8%
s38584	1630	1381	15.2%	20.9%	3.1%	17.8%	614.06	688.00	12.0%
seq	1020	969	5.0%	58.6%	22.2%	36.4%	258.59	271.58	5.0%
spla	1968	1728	12.2%	34.0%	4.6%	29.4%	686.00	729.00	6.2%
tseng	535	486	9.1%	57.5%	26.6%	30.9%	49.16	54.68	11.2%
Average	1313	1183	9%	42.6%	14.1%	28.4%	460.68	494.74	7.5%

Table 2: Comparison between two routings before and after greedy technique.

generated nets can be reduced by 28.4% on average. The percentage of unroutable nets before applying this technique on average is 41.8%, while after that is 13.4%. This means that having the same number of tracks in the channels, our technique makes VPR router capable of responding to 28.4% more rerouting requests.

The amount of increase of average running time is around 7.5 percent of the total running time of the program which is totally acceptable in comparison to the amount of enhancement of number of reroutable net segments.

8. Conclusion

In this paper we studied the detailed routing of FPGAs and defined a problem to enhance the rerouting ability of a circuit. We showed that a greedy algorithm can solve that problem optimally, and we proved its correctness.

We modified the VPR routing tool, and integrated our algorithm into it. We found that our technique reduces the total number of used tracks throughout the chip by 9%. So, it allows more rerouting to be done by VPR router. It makes VPR capable of handling 28.4% more rerouting requests. Our technique is very useful for someone who is developing a routing tool to enhance the rerouting ability power of his/her tool. This feature is very useful for supporting ECO, to activate the ability to apply modification to the circuit, later.

9. References

[1] S. Lee,H. Xiang, D. F.Wong and R. Y.Sun. "Wire Type Assignment for FPGA Routing". In *Proc. of International symposium on Field-Programmable Gate Arrays*, pp:61–67, 2003.

[2] V. Betz, and J. Rose. "VPR: A New Packing, Placement and Routing Tool for FPGA Research". In *Proc. of Seventh International Workshop on Field Programmable Logic and Applications*, pp:213–222, 1997.

[3] V. Betz. "VPR and T-VPack User's Manual(Version 3.40)". March 27, 2000.

[4] J. Cong, J. Fang, and K. Y.Khoo. "An Implicit Connection Graph Maze Routing Algorithm for ECO Routing". In *Proc. of International Conference on Computer Aided Design*, pp:163-167, 1999.

[5] L. McMurchie, and C. Ebeling. "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs". In *Proc. of International symposium on Field-Programmable Gate Arrays*, pp:111-117, 1995.

[6] T. H.Cormen, C. E.Leiserson, R. L.Rivest, and C. Stein. "Introduction to Algorithms". *The MIT Press*, 2001.

[7] Xilinx Corporation. "Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet", 2004.

[8] K. Zhu, Y. -W.Chang, and D. F.Wong. "Timing-Driven Routing for Symmetrical-Array-Based FPGAs". In *Proc. of International Conference on Computer Design*, pp. 628–633, 1998.

[9] S. Lee, and D.F. Wong. "Timing-Driven Routing for FPGAs Based on Lagrangian Relaxation". In *Proc. of International symposium on Physical Design*, pp:176–181, 2002.

[10] Y. -S.Lee, and C. -H.Wu. "A Performance And Routability-Driven Router for FPGA's Considering Path Delay". In *Proc. of Design Automation Conference*, pp:557–561, 1995.

[11] Y. -W.Chang, D. F.Wong, and C. K.Wong. "FPGA Global Routing Based on a New congestion Metric". In *Proc. of International Conference on Circuit Design*, pp. 372–378, 1995.

[12] G. G.F.Lemieux, S. D.Brown, and D. Vranesic. "On Two-Step Routing for FPGAs". In *Proc. of International symposium on Physical Design*, pp:60–66, 1997.

[13] S. Brown, M. Khellah, and G. Lemieux. "Segmented Routing for Speed-Performance and Routability in Field-Programmable Gate Arrays". *Journal of VLSI Design*, 1996.

[14] N. Sherwani. "Algorithms for VLSI Physical Design Automation: Second Edition.". *Kluwar Academic Publishers*, Boston, 1995.

[15] C. Y.Lee. "An Algorithm for Path Connections and its Applications". *IRE Transactions of Electron. Compute.*, Vol.EC= 10,pp. 346–365, 1961.