Routing Algorithms: Architecture Driven Routability Enhancement for FPGAs

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 routability by capturing the features of the routing architecture. The goal of this algorithm is to increase the number of routable segments for a given number of tracks in channels. This is a good feature for supporting Engineering Change Order(ECO) type of routing. We used the routing architecture of VirtexII FPGAs from Xilinx as our target routing architecture and integrated our algorithm into the VPR FPGA routing tool. The experimental results show that our algorithm reduces the total number of tracks by 9% on average. It allows 28.4% more rerouting than the existing VPR router, which is based on Dijkstra's maze routing algorithm.

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. 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 to simultaneously route many nets.

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. We introduce a technique to enhance the number of routable nets within each segment type of one channel. The problem to consider is that given a detailed routed circuit, we want to pack intervals in tracks of one wire type as much as possible in order to maximize the number of empty spots in tracks. This is shown in Figures 6 and 7. 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 routability. In this way, we are able to support ECO(Engineering Change Order) requirements for routing.

In order to enhance the routability of a circuit, we define an optimization problem, present a very simple 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 for the given number of tracks in the channels, we can route more segments than VPR. On average, we enhance the routability by 28.4% for short net segments we added to the benchmarks.

2. Preliminaries

The model we used for FPGA in this paper is an array based FPGA similar to Xilinx VirtexII architecture [7].

A new generation of programmable routing resource called Active Interconnect Technology interconnects all of the elements it consists of, which are configurable logic blocks(CLB), basic select RAM memory, multiplier blocks, and digital clock manager blocks. The general routing matrix (GRM) is an array of routing switches. As shown in Figure 4 [7] each programmable element 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. Long lines are bidirectional wires that distribute signals across the device. 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 after doing routing with segments of length 1. 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. We define some notations and prove some lemmas based on them to adjust our goal to the optimization problem we want 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 lines a segment with length i can occupy.

Lemma1. The number of spots of length 2, 3, and 6 which a 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:

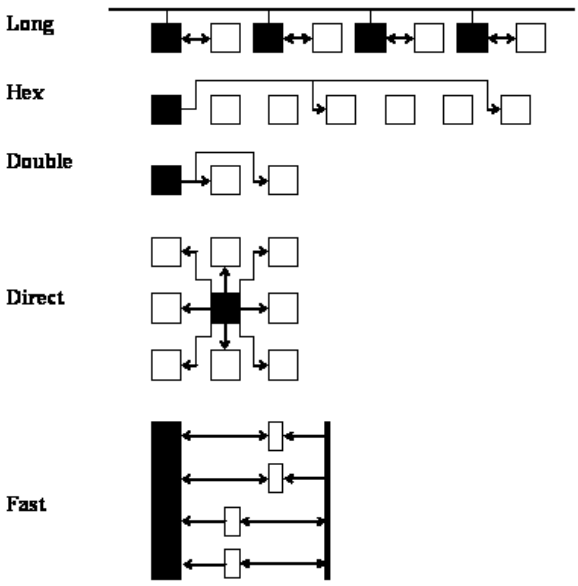

Fig. 1: VirtexII device route

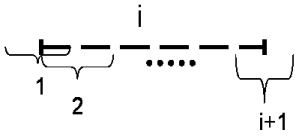

Fig. 2: O_{double} for a segment of length i

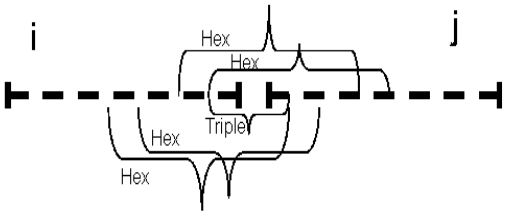

Fig. 3: Gain achievement for two segments of gap 1

$$O_{double}(i) = i+1.$$

$$O_{triple}(i) = i+2.$$

$$O_{hex}(i) = i+5.\square$$

Lemma2. The number of total occupied spots by two consecutive net segments are 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 0(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. □

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.

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

A trivial observation 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 that can be created.

4. Problem Definition

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

One very important point to consider when switching net segments is dealing with vertical constraints. Net segments cannot be switched easily since vertical constraints exist. As shown in the Figure 4, the number of switches between vertical and horizontal wires, and the connections between CLB inputs/outputs to wires inside tracks are limited. In the next section, we show how to solve the STS problem and also deal with vertical constraints.

5. Algorithm

In this section we model the STS problem as a greedy routing problem and 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.

In this algorithm, first we sort all net segments based on their start points. We start from the first net segment, assign it to the first track

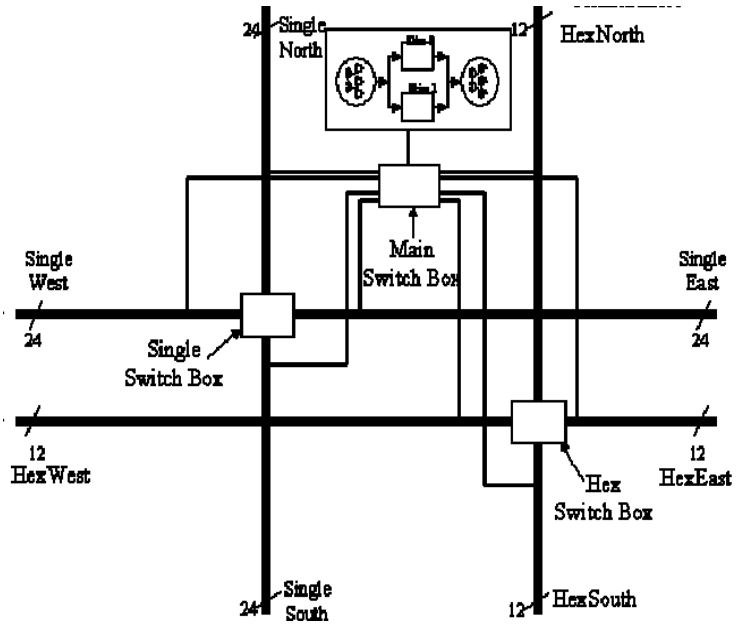

Fig. 4: VirtexII routing switches and connections

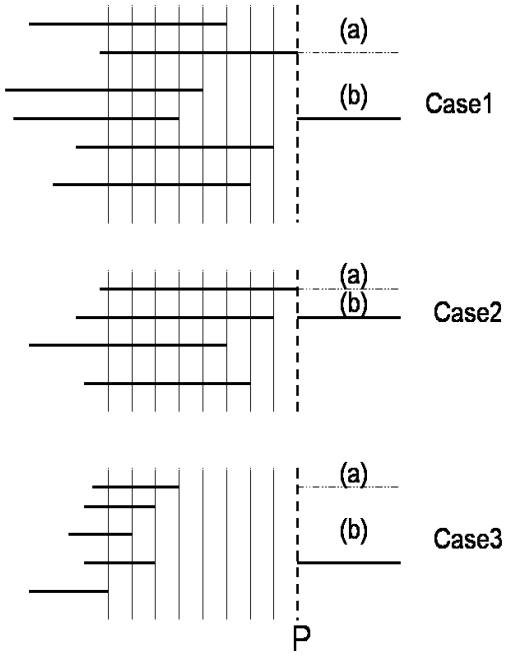

Fig. 5: Examples, demonstrating the greedy choice a and the non-greedy choice b

Algorithm 1 Greedy Segment Track Switch Algorithm

Input: Detailed routed nets N of one channel with 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.
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

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 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. This technique is very similar to Left Edge algorithm for interval packing [14]. The only 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 an optimal solution which is not greedy. 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 easily alter the non-greedy choice to the greedy one in this way. We swap the assignment of the segments of the two tracks from the starting point of the 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. This shows that the greedy choice is always a better choice. So, the given solution was not optimal and the greedy choice gives us the optimal algorithm.□

5.3 Demonstration of the proof

Some example cases are shown in Figure 5:

1. Case 1: Point P is the place where the non greedy choice 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 the solution which is not greedy could not be optimal, since in every step 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 $8-5=3$. So the given solution is not optimal.

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 $O(nT)$.

6. Experimental Results

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 negotiated congestion algorithm. This algorithm is based on ripping up and rerouting methods.

We added the greedy code mentioned in Algorithm 1 to 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. So, for all channels we applied Algorithm 1 on net segments located in each channel that are routed on segments of tracks of single wire type. We switch them in a way that we can pack them as much as possible. Figures 6 and 7 show the routing for one channel of one of the benchmarks without and with our technique. It is obvious that the routing after applying our technique is more dense and it has more empty spots of higher length, while before that the net segments are scattered in the channel. The number of used tracks in this example before the greedy technique is 9 and after it is 5. 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%.

To measure how well our technique can enhance the routability of the circuit by taking advantage of routing architecture, for each channnel we created some randomly generated nets with length less than 6 and add 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 generated nets can be reduced by 28.4% on average. This means that given the same number of tracks in channels, our algorithm can route more segments than VPR.

In some very few benchmarks which are denser ones, this reduction is less than that. The amount of increase of running time is less than 12 percent of the total running time of program which is totally acceptable in comparison to the amount of enhancement of routability.

7. Engineering Change Order(ECO) Enabling

The presented algorithm has a motivative potential to be used for ECO purposes. An ECO is a request to make design changes, typically late in the design process [4]. The ECO problem involves the ability to (i) specify the incremental design changes, (ii) implement the design changes and (iii) update all design databases to reflect the changes. 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.

8. Conclusion

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

We modified the VPR routing tool, and integrated our greedy algorithm into it. We found that our technique reduces the total number of used tracks all over the chip by 9%. Also, it allows more rerouting to be done compared with the original Pathfinder algorithm which VPR uses. The percentage of unroutable nets before applying this technique on average is 13.4%, while with our algorithm on average is 41.8%. Our technique is very useful for someone who is developing a routing tool to enhance the routability power of his/her tool. Also, it 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.

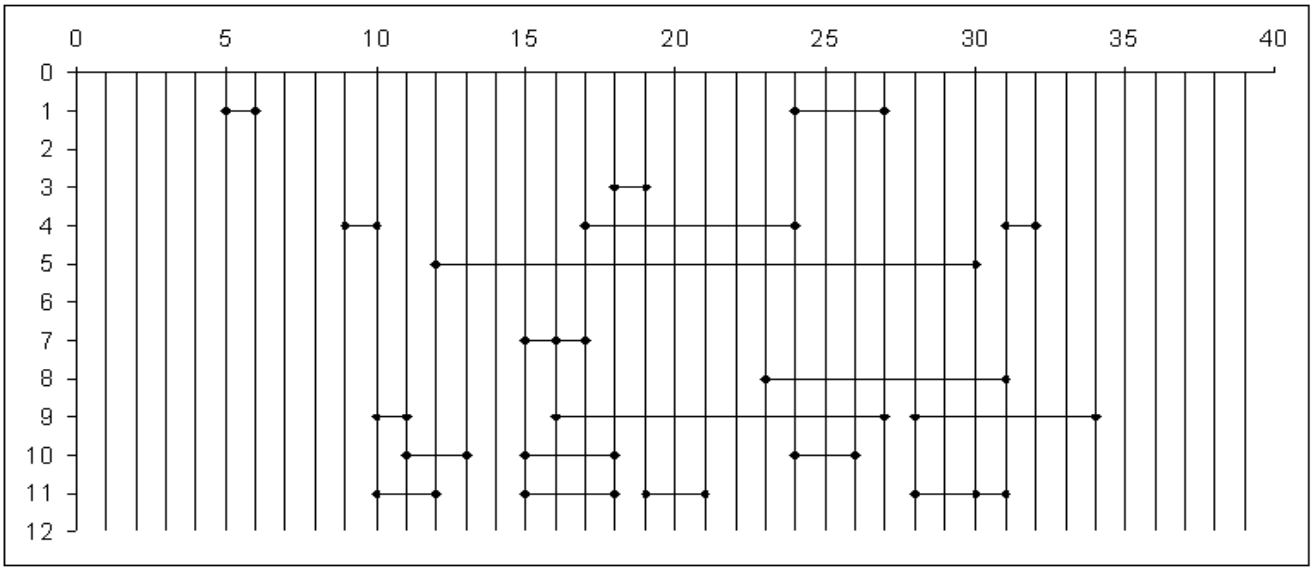

Fig. 6: Net segments order before greedy algorithm for STS

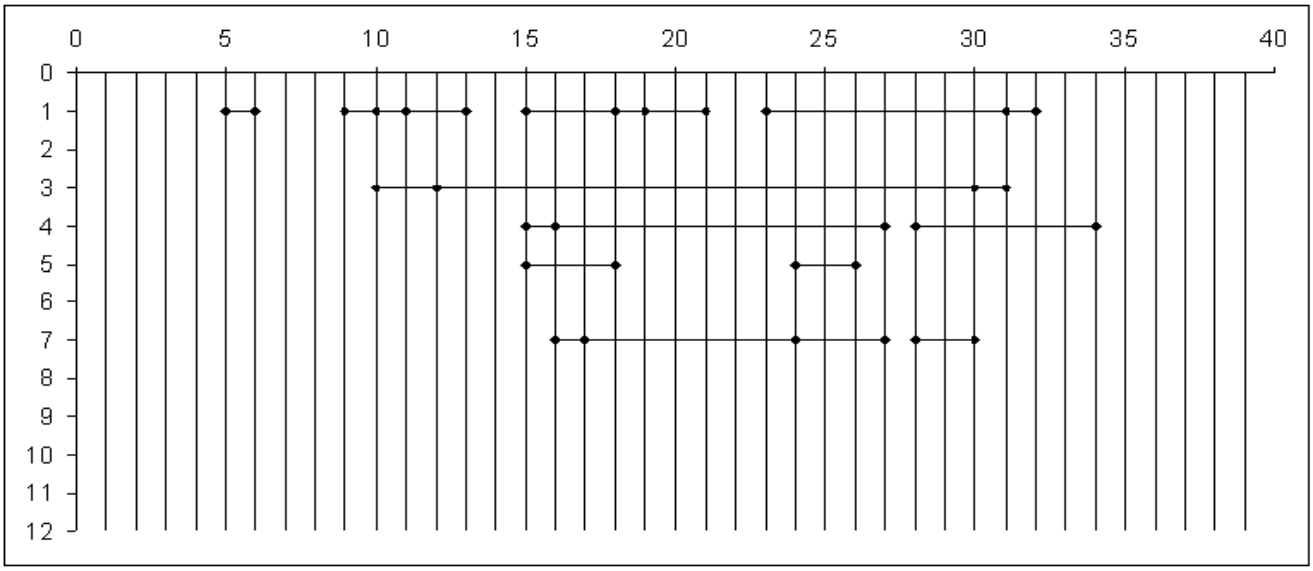

Fig. 7: Net segments order after greedy algorithm for STS

	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%

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