

A Routing Paradigm with Novel Resources Estimation and Routability Models for X-Architecture Based Physical Design^{*}

Yu Hu¹, Tong Jing¹, Xianlong Hong¹, Xiaodong Hu², and Guiying Yan²

¹ Computer Science and Technology Department, Tsinghua University,
Beijing 100084, P.R.China
jingtong@tsinghua.edu.cn

² Institute of Applied Mathematics, Chinese Academy of Sciences,
Beijing 100080, P.R.China

Abstract. The increment of transistors inside one chip has been following Moore's Law. To cope with dense chip design for VLSI systems, a new routing paradigm, called X-Architecture, is introduced. In this paper, we present novel resources estimation and routability models for standard cell global routing in X-Architecture. By using these models, we route the chip with a compensation-based convergent approach, called COCO, in which a random sub-tree growing (RSG) heuristic is used to construct and refine routing trees within several iterations. The router has been implemented and tested on MCNC and ISPD'98 benchmarks and some industrial circuits. The experimental results are compared with two typical existing routers (labyrinth and SSTT). It indicates that our router can reduce the total wire length and overflow more than 10% and 80% on average, respectively.

1 Introduction

As very large scale integrated circuit (VLSI) advances, circuits are getting larger and more complex. Many studies have been focused on high performance integrated circuit (IC) design. However, while designs are implemented based on traditional Manhattan routing architecture, i.e., all interconnects route either horizontally or vertically, the optimization capability is limited due to less routing flexibility. Then, a new routing paradigm allowing interconnects to explore 0° , 45° , 90° , and 135° directions, called X-Architecture, is proposed, which tries to overcome the intrinsic limitation of traditional routing architecture. TX79, a Toshiba micro-processor core, has been replaced and rerouted with the X-Architecture and the liquid routing[1], resulting in an average over 20% wire length reduction and a 20% performance improvement [2].

To cope with the increasing complexity, designers often explore a global router followed by a detailed one. Global routing plays an important role in VLSI physical de-

^{*} This work was supported in part by the NSFC under Grant No.60373012, the SRFDP of China under Grant No.20020003008, and the Hi-Tech Research and Development (863) Program of China under Grant No.2004AA1Z1050.

sign. Wong [3] proved that it is a NP-hard problem to get the optimal routing solution. Useful algorithms have been proposed focusing on routability [4, 5, 6, 7], timing issue [8, 9, 10], coupling noise [11, 12], and routing tree construction [4, 13, 14, 15, 16, 17].

Ryan Kastner [5] developed a global router based on maze routing, called labyrinth. Jing et al. [7] presented a congestion reduction global routing algorithm based on search space traversing technology (SSTT), by which large circuits¹ can be routed in a short running time, while keeping good performance.

Some researches have been done on non-Manhattan routing. Chen et al. [18, 19] reviewed routing architectures. Non-Manhattan routing is now championed by the X Initiative [20]. However, there is no much literature focusing on X-Architecture based routing approaches. Koh et al. [21] proposed a method and implemented a global router to address modern interconnect problems in Manhattan and non-Manhattan architectures. Agnihotri et al. [22] introduced a layer balance approach for congestion reduction in both Manhattan and non-Manhattan architectures. [23] presented a grid-less routing algorithm to reduce both the total wire length and the number of vias.

Motivated by the recent process of X-Architecture, this paper studies X-Architecture based routing to handle large scale circuits efficiently. The goal of our work is to employ X-Architecture in the global router, by which we can achieve highly routing performance that can hardly be obtained in Manhattan routing. In this paper, we route in X-Architecture with a compensation-based convergent (COCO) approach, in which we use a random sub-tree growing (RSG) heuristic to construct and refine routing trees within several iterations. Using our global router, the design flow mentioned in [1] can be accomplished by performing a liquid routing under the guidance of our global router. Our router is tested on MCNC and ISPD'98 benchmarks and some large industrial circuits, which shows that our router makes substantial improvements on both wire length and routability in the reasonable running time while compared with labyrinth [5, 24] and SSTT [7].

The remainder of this paper is organized as follows: in Section 2, resources estimation and routability models in X-Architecture are proposed. Section 3 describes the COCO method for routability analysis. Experimental results, showing routing tree wire length and congestion reduction with MCNC and ISPD'98 benchmarks and industry circuits, are given in Section 4. We conclude this paper in Section 5.

2 Resources Estimation and Routability Models

2.1 Routing Paradigm

[2] mentioned that, a preferred-direction² implementation of the X-Architecture is likely to increase the number of vias and can be worse for delay despite the reduction in wire length. To solve this problem, people adopt the gridless octilinear routing technology, called liquid routing, to finish the detailed routing. In our global router, we allow that a routing can use all possible directions in global routing graph (GRG). To guide the

¹ The largest case used has more than 70K nets.

² The direction of routing in a particular layer is decided in advance.

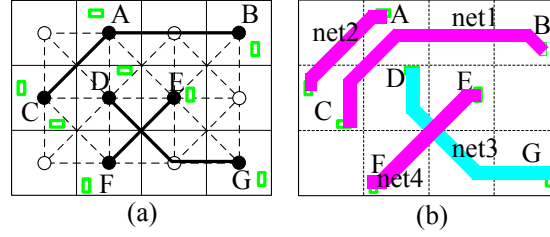


Fig. 1. An example for the global-detailed routing flow

liquid routing, a layer assignment (LA) process should be employed to assign the inter-segment segments into different layers. Then the gridless liquid routing can be performed after LA [23].

To explain how to add our global routing results into the design flow in [1], Fig. 1 shows an example for routing 4 nets, in which net1 connects pin C and pin B, net2 connects pin C and pin A, net3 connects pin D and pin G, net4 connects pin E and pin F. Fig. 1 (a) shows the routing area partition, GRG, and global routing result for these nets and the pins' locations in the chip, in which the green rectangles denote pins. Fig. 1 (b) shows the detailed routing result, in which the purple wires are in one layer and the blue ones are in the other layer.

We obtain the global routing results in Fig. 1 (a) by using the router presented in this paper. Since net3 and net4 have intersectant segment, DG and EF, they should be assigned to different layers. Then all nets can be connected to their pins following the global routing results with some local adjustments.

2.2 GRG Generation

As mentioned above, we design our global router to guide a gridless liquid detailed router. So it's hard to use the existing GRG generation methods [21, 18] in X-Architecture.

A placement result is needed as an input to our global router. Since the placement algorithms for X-Architecture seems not so mature [2], we utilize a traditional standard cell placer to obtain placement results, which can guarantee the same heights of all rows in standard cell design. We should note that our global router can be performed in any other given placement results after generating the corresponding GRG based on the placement architecture.

According to the height of each row, the entire routing region is divided into a set of rectangular tiles. We prescribe that all non-peripheral tiles (see Fig. 2 (a)) should be squares with the same area, which leads the following methods of routing area partition. Assuming the die size of the whole chip is $w \times h$, the vertical distance from top of the chip to the first row is h_t , the one from bottom of the chip to the last row is h_b , and the horizontal distance from the left of the chip to the left most cells is w_l , the number of cells rows is n_{row} . Because all non-peripheral tiles should be squares with the same area, the horizontal width of these tiles is equal to row height, h_{row} . We can see these from Fig. 2 (a). The partition parameters can be computed as follows. The number of columns is $n_{col} = \lfloor \frac{w-w_l}{h_{row}} + 2 \rfloor$. The width of the right most tiles is $w_r = w - w_l - (n_{col} - 2) \times h_{row}$.

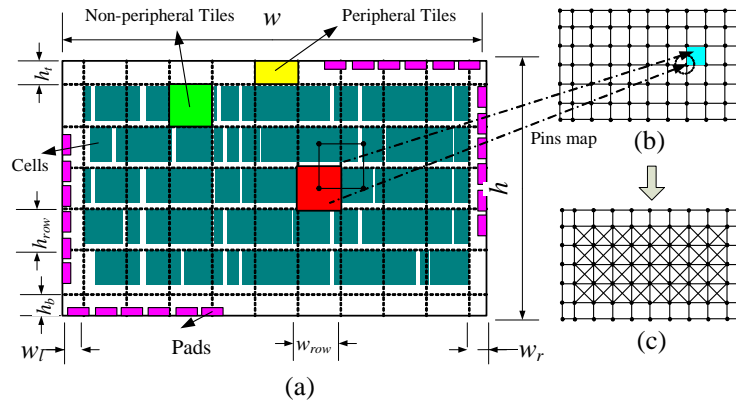


Fig. 2. GRG generation (a) partition of routing region for a standard cell design, (b) the dual graph of (a), (c) the GRG in X-Architecture

After obtaining a partition of the routing area, we generate a dual graph of the partition graph. As shown in Fig.2 (b), each pin is mapped to a vertex corresponding to the tile it is located in. Two adjacent vertices are connected by a horizontal or vertical edge. Then we add some diagonal edges into the dual graph to connect each two diagonal adjacent vertices, except for those in the periphery of the graph, to obtain the GRG in X-Architecture (see Fig.2 (c)). Thus, a net can be specified as a set of vertices in GRG. Then, the problem of routing a net in GRG can be stated as the Steiner tree problem of specified vertices in GRG.

2.3 Routing Resources Assignment (RA) Model

To make sure of the legal via locations in traditional routing algorithms, it's necessary to align the diagonal routing grids to intersect with the Manhattan routing grids below them since each layer employs a preferred routing direction. Then, the result is in an unacceptable waste of wiring resources [2]. In our routing paradigm, routing can explore all possible directions in a layer. So we can compute routing resources according to pitches and routing area, then a liquid detailed router can be used to accomplish the routing process based on our global routing results as we mentioned above. We can use the following method to assign the routing resources.

A positive number c , called edge capacity, is assigned to each edge in GRG. Edge capacity indicates the number of available tracks between the corresponding tiles. Firstly, we compute the number of tracks assigned to each rectilinear edge based on wire widths, pitches, layers and routing area in traditional way. Then we remain half of the capacity of each edge, and reassign the other half to diagonal edges. The Fig.3 shows an example of this strategy.

Fig.3 (a) shows the initial capacity in each edge in rectilinear GRG is 20. We keep the capacities in peripheral edges (not labelled in the Fig.) constant and transfer part of capacities in non-peripheral rectilinear edges to diagonal ones. Then, we half reduce the capacity of each rectilinear edge, which results in capacity 10 in each edge (see

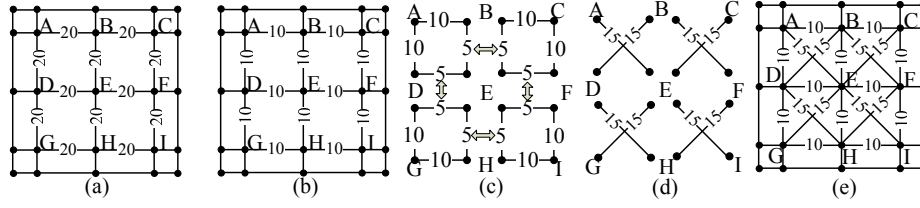


Fig. 3. Resource assignment process

Fig.3 (b)). After that, we assign the other half of capacities into diagonal edges. We consider each non-peripheral square in GRG separately. For example, in square (A, B, E, D), edge (B, E) is shared with square (B, C, F, E), so we half partition the capacity in edge (B, E) for the two separated squares sharing it (see Fig.3 (c)). We assign the total capacities in each square into the two diagonals in it evenly (see Fig.3 (d)). For example, the total capacities in square (A, B, E, D) is 30, so we assign 15 into diagonal edge (A, E) and (B, D), respectively. The final assignment is shown in Fig.3 (e).

2.4 Routability-Aware Global Routing Model

In this paper, both total wire length and total overflow of all edges are considered in our objective. Some basic definitions and formulations are given as follows.

The total wire length is: $L = \sum_{i=1}^{N_n} \sum_{k=1}^{N_e} S_{ik} \cdot l_k$, where N_n is the number of nets, N_e is the number of edges, S_{ik} is equal to 1 if edge e_k is used by the Steiner tree of net n_i , otherwise it's equal to 0, and l_k is the length of edge e_k . The usage of edge e_k is: $u_{e_k} = \sum_{i=1}^{N_n} S_{ik}$, the overflow of edge e_k is:

$$eof_{e_k} = \begin{cases} u_{e_k} - c_{e_k}, & \text{if } c_{e_k} < u_{e_k}; \\ 0, & \text{otherwise,} \end{cases} \quad (1)$$

where c_{e_k} is the capacity of edge e_k . The total overflow of all edges is: $tof = \sum_{k=1}^{N_e} eof_{e_k}$, then, the global routing problem can be described as follows. Find S_{ik} to

$$\text{Minimize } COST = L \times (tof^\beta + \varepsilon). \quad (2)$$

where β is a constant, and ε is a small real constant.

3 COCO Method for Routability Analysis

3.1 RSG for Octilinear Steiner Tree Construction

We find that the existing tree algorithms [25, 26, 13] can't be used directly in our graph-based router because the produced solution could introduce some Steiner points not defined in the routing graph. So we present a random sub-trees growing heuristic (RSG) and integrate it into the global router to construct the initial routing tree for each net and to generate different topologies for each net through the routing process.

In our RSG, we denote the given GRG as $G = (V, E)$ and the terminal set as N . The distance between two vertices is defined as the same as in [13]. We use a random subtrees growing strategy to obtain a Steiner minimal tree in the graph G . The pseudo-code of RSG is shown in Algorithm 1.

Algorithm 1 Graph-based Octilinear Steiner Minimal Tree

Input: Graph G and terminal set T

Output: An octilinear steiner minimal tree in G

```

1: for each terminal  $p \in N$  do
2:    $t_n \leftarrow$  new subtree based on  $p$ 
3:    $t_n.GP \leftarrow p$ 
4: end for
5: while the number of subtrees  $> 1$  do
6:    $t \leftarrow$  select a subtree randomly
7:    $t$  grows from  $t.GP$  to vertex  $v$  based on Eq. (3)
8:   if the vertex  $v \in$  subtree  $t'$  then
9:      $t_{new} \leftarrow$  merging  $t$  and  $t'$ 
10:    Compute the location of  $t_{new}.GP$ 
11:   end if
12: end while
13: Prune all non-terminal leaves in the last subtree ( $tree$ )
14: return  $tree$ 

```

Given a subtree t , whose GP is p , the vertex v it'll grow to is defined as follows:

$$v = \min_k \eta_{p,k}^t, \quad (3)$$

where k is the neighbor vertex with p , and k is not covered by subtree t . The $\eta_{p,k}^t$ is defined as follows:

$$\eta_{p,k}^t = len(p,k) + \gamma \cdot \Psi_k^t, \quad (4)$$

where $len(p,k)$ is the length of edge (p,k) in graph G , γ is a constant, and Ψ_k^t is the shortest distance (in X-Architecture) from vertex p to all the vertices covered by other subtrees, which makes the current subtree t grows towards others as quickly as possible.

We noted that a sub-tree will be selected randomly in each iteration. Hence the solution produced by our RSG could vary in different runs. Tested by running randomly generated cases for tens of times, RSG can produce various topologies while keeping the stable and high performance in wire length. This characteristic of RSG will be useful in our global router. Fig.4 shows different topologies for IBM02-net 106 (22 pins), generated by RSG in three runs.

3.2 COCO Method

In this section, we describe our compensation-based convergent (COCO) method for routability analysis, which tries to achieve a global minimal solution instead of getting trapped in local minima by finding a near optimal trade-off between total wire length and total overflow. We use RSG to construct initial trees for all nets. Then several iterations are needed to refine the solution guided by the objective in Eq.(2).

In each iteration, we compute the overflow of each edge in the global routing graph by Eq. (1) and get a set of congested nets, called N_{cong} . Then we randomly select a subset of N_{cong} , named N_{ran} , according to a probability p_{θ} . The routing resources used by nets in N_{ran} are given back to the edges. The nets in N_{ran} will be rerouted simultaneously. Obviously, there is no net ordering problem among nets in N_{ran} .

To construct a routing tree for each net in N_{ran} , we use the variation of our RSG algorithm. A weight w_e is given in an edge e in the global routing graph, and Eq. (4) is rewritten as follows:

$$\eta_{p,k}^t = \log(\text{len}(p,k) + \gamma \cdot \psi_p^t) + \mu \cdot \log w_e, \quad (5)$$

where μ is a constant, and w_e could be w_{1e} , w_{2e} or w_{3e} , which will be defined later.

To avoid trapping into a local minimization, we change rules for weighting an edge in different iterations. We design three kinds of rules, which have different capabilities to reduce total overflow and total wire length. The definition of these rules is shown as follows:

$$w_{1e} = 1, \quad w_{2e} = \begin{cases} \infty, & \text{if } c_{e_k} \leq u_{e_k} \\ 1, & \text{otherwise} \end{cases}, \quad w_{3e} = \begin{cases} 20 \times u_{e_k} / c_{e_k}, & \text{if } c_{e_k} \leq u_{e_k} \\ (u_{e_k} + \epsilon) / c_{e_k}, & \text{otherwise} \end{cases} \quad (6)$$

where ϵ is a small real number that validates the equation while u_{e_k} is 0.

If the w_1 is used in a certain iteration, nets in N_{ran} will be rerouted based on wire length minimization. In the iterations that w_2 or w_3 is used, the wire length of a net could be increased for detouring the congested edges. Under w_1 rule, a net will be rerouted as a Steiner minimal tree. By doing so, the total wire length can be reduced, and the used routing resources can be expected to reduced too, because less GRG edges are used by a net with a shorter length. For those nets without detours, as mentioned in Section 3.1, the topologies could be expected to change after rerouting, which indicates the possibility of climbing up from a local minimization.

If the w_2 is used, a routing tree will try to avoiding the congested edges, because of their ∞ weights. In the RSG algorithm, an subtree will avoid choosing an edge if the it's congested, otherwise, it'll choose the shortest path to grow to. So this rule is a tradeoff between congestion and wire length.

If the w_3 is used, a routing tree will grew mainly based on the congestion map. As shown in Eq. (6), the congested edges are multiplied by 20, which makes weights increase sharply in these edges. The wire length information should also be added to consideration because the routing tree with too many detours will lead to the more serious congestion problem.

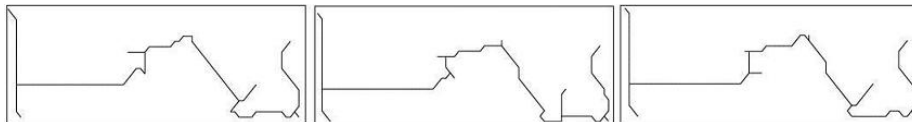


Fig. 4. Different topologies for IBM02-net106 generated by RSG in three runs

The w_3 weight could reduce congestion efficiently while expending longer wire length. On the other hand, the w_1 weight could compensate the wire length increase caused by weight w_3 , while producing more congested edges. And the w_2 rule can be seemed as a tradeoff between w_1 and w_3 , but it has a limitation of reducing congestion and always leads to a local minimization. In the experiments, we found that we can always get a near optimal solution by employing these three rules as a appropriate sequence.

We run several iterations using w_2 after obtaining the initial routing, until the solution (using the Eq. (2) to evaluate our solution) can not be improved any more, then the weight rule alternates between w_1 and w_3 in the following iterations, until no improvement can be produced. In each iteration, the upper bound of total overflow is given based on the previous congestion map. The decrease of overflows leads to the decrease of congested nets and the number of elements in set N_{ran} , which guarantee the convergence of the algorithm.

4 Experimental Results

We implemented our X-Architecture based global router in C Language on a Sun-fire v880 workstation. We tested our router and compare to SSTT³[7] and Labyrinth [11, 24] (the benchmarks are also from the respective literature).

Table 1. Wire length of final routing (COCO vs. SSTT)

Circuit	COCO (μm)	SSTT (μm)	Imp. (%)
C2	5.05e+05	5.45e+05	7.34
C5	1.09e+06	1.26e+06	13.5
C7	1.53e+06	1.83e+06	16.4
sl3207	9.52e+06	9.74e+06	2.26
avq	9.18e+06	1.08e+07	15.0
u100	3.69e+07	4.12e+07	10.4
ut	9.12e+07	1.15e+08	20.7
ucnt500	2.83e+08	3.02e+08	6.29
u05	1.02e+11	1.19e+11	14.3
u11	1.72e+10	1.97e+10	12.7
Average	1.20e+10	1.39e+10	13.7

Table 2. Total overflow and running time (COCO vs. SSTT)

Circuit	COCO		SSTT		Imp (%)
	ovf	cpu	ovf	cpu	
C2	9	0	18	0	50.0
C5	2	5	50	0	96.0
C7	12	9	46	5	73.9
sl3207	1	57	23	7	95.7
avq	1	143	2	9	50.0
u100	7	0	53	1	86.8
ut	1	0	2	1	50.0
ucnt500	2	5	13	3	84.6
u05	6	739	43	574	86.1
u11	5	1890	15	1942	66.7
Average	5	285	26.5	254	82.6

Comparisons for group 1 on total wire length of final routing are shown in Tab.1. Column *Imp.* shows the wire length reduction percentage of our router (denoted by COCO) beyond SSTT.

We can see that, our router optimizes the wire length beyond SSTT in all test cases, and gets an average 13.7% reduction of total wire length of final routing. This result

³ Since both SSTT and our router have strong capabilities of congestion reduction, we reduce the capacity of each edge in the same proportion until our router fails to achieve a completed routing. Then SSTT is tested with these shrunken circuits.

Table 3. Wire length of final routing (COCO vs. labyrinth)

Circuits	COCO	Labyrinth	Imp. (%)
ibm01	69575	76517	9.07
ibm02	188691	204734	7.84
ibm03	158837	185194	14.2
ibm04	187443	196920	4.81
ibm05	42417	689671	38.5
ibm06	314082	346137	9.26
ibm07	391289	449213	12.9
ibm08	440612	469666	6.19
ibm09	467727	481176	2.80
ibm10	685521	679606	-0.87
Average	294619	377883	10.2

Table 4. Total overflow and running time (COCO vs. Labyrinth)

Circuits	COCO		Labyrinth		Imp. (%)
	ovf	cpu	ovf	cpu	
ibm01	60	10	398	72	84.9
ibm02	0	30	492	123	100
ibm03	0	26	209	148	100
ibm04	385	42	882	278	56.4
ibm05	0	57	251	233	100
ibm06	0	53	834	171	100
ibm07	0	66	697	381	100
ibm08	1	73	665	364	99.9
ibm09	1	82	505	553	99.9
ibm10	661	115	588	692	-12.4
Average	110	55	552	302	82.9

is a little conservative because the shrink of the chip increases the detours in the final routing.

Comparisons for group 2 on total wire length of final routing are shown in Tab.3. We can see that our router optimizes the wire length beyond labyrinth in most test cases, and gets an average 10.2% reduction of total wire length.

Tab.2/Tab.4 shows the comparison between our router and SSTT/labyrinth on total overflow (denoted by *ovf*) and running time. Column *Imp.* shows the reduction percentage of total overflow of our router beyond SSTT/labyrinth. We can find that our router optimizes both the total overflow beyond SSTT/labyrinth in almost all test cases, and gets an average 82% reduction of total overflow.

5 Conclusions

This paper presents a novel routing resource estimation and routability analysis models. Based on these models, we design and implement an X-Architecture based global router for standard cell design. Tested on two groups of circuits, our router achieved an average over 10% reduction of wire length and over 80% reduction of total overflow, when compared with a typical Manhattan global router SSTT. Since the RSG is based on graph, our router could be easily transplanted to other architectures.

As future work, we will consider timing and coupling issues in our router.

References

1. Mitsuhashi, T., Someha, K.: Performance-oriented layout design, pervasive use of diagonal interconnects reduces wire-length. *Design Wave Magazine* (2001) 59–64
2. Teig, S.: The x architecture: Not your father's diagonal wiring. In: *Proc. of SLIP*. (2002) 33–37
3. Sarrafzadeh, M., Wong, C.K.: *An Introduction to VLSI Physical Design*. McGraw Hill, USA (1996)

4. Bozorgzadeh, E., Kastner, R., Sarrafzadeh, M.: Creating and exploiting flexibility in rectilinear steiner trees. *IEEE trans. on CAD* **22** (2003) 605–615
5. Kastner, R., Bozorgzadeh, E., Sarrafzadeh, M.: Predictable routing. In: *Proc. of ICCAD*. (2000) 110–114
6. Hadsell, R.T., Madden, P.H.: Improved global routing through congestion estimation. In: *Proc. of the DAC*. (2003)
7. Jing, T., Hong, X., Bao, H., Xu, J., Gu, J.: **Sstt**: Efficient local search for gsi global routing. *J. of Compute Science and Technology* **18** (2003) 632–639
8. Jing, T., Hong, X., Xu, J., Bao, H., Cheng, C.K., Gu, J.: **Utaco**: A unified timing and congestion optimization algorithm for standard cell global routing. *IEEE Trans. on CAD* (2004) 358–365
9. Lin, S.P., Chang, Y.W.: A novel framework for multilevel routing considering routability and performance. In: *Proc. of ICCAD*. (2002) 44–50
10. Hong, X.L., Jing, T., Xu, J.Y., Bao, H.Y., Gu, J.: **Cnb**: A critical-network-based timing optimization method for standard cell global routing. *J. of Computer Science and Technology* (2003) 732–738
11. Kastner, R., Bozorgzadeh, E., Sarrafzadeh, M.: An exact algorithm for coupling-free routing. In: *Proc. of ISPD*. (2001) 10–15
12. Xu, J., Hong, X., Jing, T., Cai, Y., Gu, J.: A novel timing-driven global routing algorithm considering coupling effects for high performance circuit design. *IEICE Trans. on Fundamentals of ECCS* (2003) 3158–3167
13. Zhu, Q., Zhou, H., Jing, T., Hong, X.L., Yang, Y.: Spanning graph based non-rectilinear steiner tree algorithms. *IEEE Trans. on CAD* (2005)
14. Wang, Y., Hong, X., Jing, T., Yang, Y., Hu, X., Yan, G.: Spanning graph based non-rectilinear steiner tree algorithms. *LNCS* **3256** (2004) 442–452
15. Alpert, C.J., Hrkic, M., Hu, J., Kahng, A.B.: Buffered steiner tree for difficult instances. In: *Proc. of ISPD*. (2001) 4–9
16. Xu, J., Hong, X., Jing, T., Cai, Y., Gu, J.: An efficient hierarchical timing-driven steiner tree algorithm for global routing. *INTEGRATION, the VLSI J.* (2003) 69–84
17. Hrkic, M., Lillis, J.: S-tree: A technique for buffered routing tree synthesis. In: *Proc. of DAC*. (2002) 578–583
18. Chen, H., Yao, B., Zhou, F., Cheng, C.K.: Physical planning of on-chip interconnect architecture. In: *Proc. of ICCD*. (2002) 30–35
19. Chen, H., Zhou, F., Cheng, C.K.: The y-architecture: yet another on-chip interconnect solution. In: *Proc. of ASP-DAC*. (2003) 840–846
20. Organization, T.X.I.: <http://www.xinitiative.org> (2005)
21. Koh, C.K., Madden, P.H.: Manhattan or non-manhattan? a study of alternative vlsi routing architectures. In: *Proc. of the GLSVLSI*. (2000) 47–52
22. Agnihotri, A.R., Madden, P.H.: Congestion reduction in traditional and new routing architectures. In: *Proc. of GLSVLSI*. (2003) 28–29
23. Paluszewski, M., Winter, P., Zachariasen, M.: A new paradigm for general architecture routing. In: *Proc. of GLSVLSI*. (2004) 26–28
24. Labyrinth. <http://www.ece.ucsb.edu/kastner/labyrinth/> (2004)
25. Coulston, C.S.: Constructing exact octagonal steiner minimal trees. In: *Proc. of GLSVLSI*. (2003) 1–6
26. Kahng, A., Mandoiu, I., Zelikovsky, A.: Highly scalable algorithms for rectilinear and octilinear steiner trees. In: *Proc. of ASPDAC*. (2003) 827–833