Routing in a New 2-Dimensional FPGA/FPIC Routing Architecture*

Yachyang Sun *C. L. Liu*

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 60801

Abstract - This paper studies the routing problem for a new Field-Programmable Gate Array (FPGA) and Field-Programmable Interconnect Chip (FPIC) routing architecture which improves upon the one proposed in [9] by providing continuing switches along the horizontal and vertical wire segments. The addition of continuing switches leads to higher routability and better timing performance than that for the routing architecture in [9]. A two-phase routing algorithm for the new routing architecture is developed. Both the initial routing phase and the rip-up and reroute phase employ a dynamic programming technique. The rip-up and reroute phase can also be applied to the segmented channel routing problem for row-based FPGA routing structures. Experimental results show that routability is improved dramatically and the number of active programmable switches in connecting paths and the total number of programmable switches are reduced, when compared with the results in [9] and [3]. The running time of the algorithm is less than 7 seconds for each of five industrial circuits.

1 Introduction

Unlike conventional mask-programmable gate arrays, FPGAs use programmable switches in the connecting paths. The ON/OFF status of each programmable switch is programmed by the user without going through the foundry facility. To implement circuits that cannot be fitted on a single FPGA, FPIC was introduced [2, 6]. A large circuit is divided into several parts, and each part is implemented on an FPGA. These FPGAs are then interconnected on a printed circuit board using an FPIC. Fig. 1 shows the conventional two-dimensional FPGA/FPIC routing architecture proposed in [3, 4, 8, 11] which contains only wire segments of unit length. Each square represents a logic block implementing logic functions in the case of FPGA and represents an I/O pin in the case of FPIC. The terminals of a block are connected to wire segments called *terminal segments.* Terminal segments are connected to wire segments in the routing channel through programmable switches at their intersections which are shown as black circles in Fig. 1. A programmable switch is said to be *active*, if its status is ON or is conducting. The switch matrix is used to connect wire segments in the horizontal and vertical channels. In the switch matrix, each wire segment can be connected to a subset of the wire segments on the other sides of the matrix. A routing example of a 2-terminal net is shown in bold lines in Fig. 1. There are eight active switches in

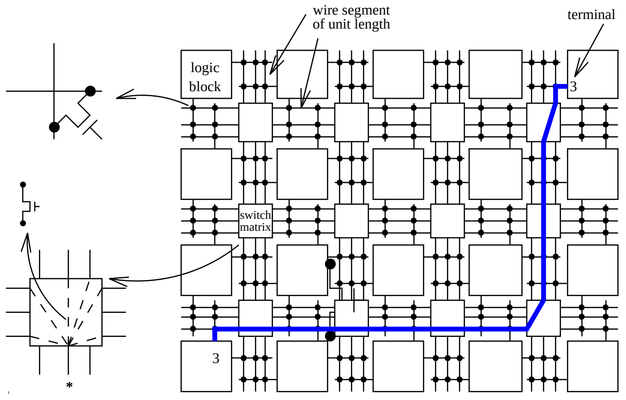

Fig. 1. Conventional two-dimensional FPGA architecture and a routing example

the connecting path, six of them are in switch matrices and each of the remaining two connects a terminal segment to a wire segment in a routing channel. Since the number of active programmable switches in a connecting path is equal to the length of the path (minus one), there will be substantial routing delay in the path. To improve upon the situation, the recently-developed Xilinx XC4000 series provides double-length wire segments which are twice as long as unit-length wire segments. However, the variety of wire segments of different lengths is still very limited.

A new FPGA/FPIC routing architecture that allows a larger variety of wire segments of different lengths was proposed [9]. Programmable switches are available only at some of the intersections between horizontal and vertical wire segments. In this routing architecture, a signal passes through an active programmable switch only when making a turn, whereas in the conventional FPGA routing architecture there is an active switch in each unit length. Consequently, signal delay is reduced dramatically. Programmable switches provided to allow routing paths to make turns are called *turning switches.* An example of the routing architecture is shown in Fig. 2. A routing example for the same 2-terminal net in Fig. 1 is also depicted. Note that only seven turning switches are active in the connecting path, compared with eight switches used in Fig. 1. Nevertheless, low routability is a main drawback of this routing architecture, although asymptotically it has been shown to be as efficient as conventional routing architecture with free tracks. In this paper, we propose a routing

*Work partially supported by NSF under Grant MIP 92-22408 and by Fujitsu Company

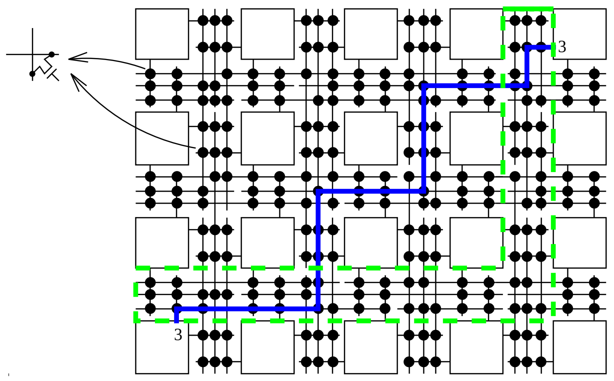

Fig. 2. Two-dimensional FPGA/FPIC routing architecture in [11] and a routing example

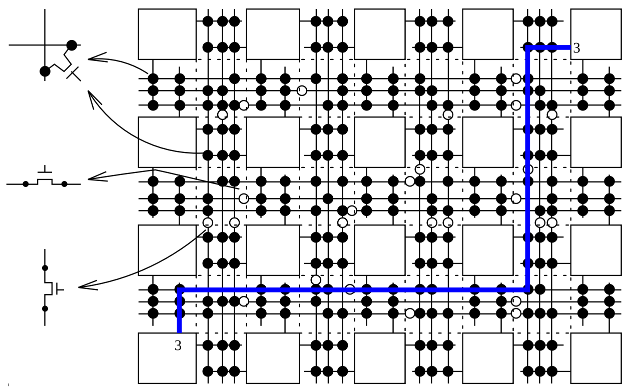

Fig. 3. Proposed two-dimensional FPGA/FPIC routing architecture and a routing example

architecture which improves upon the routability by adding programmable switches between two horizontal (or vertical) wire segments which can be programmed to connect two horizontal (or vertical) wire segments to form a longer horizontal (or vertical) segments. Programmable switches for this purpose are called *continuing switches*. Fig. 3 depicts our new routing architecture corresponding to that in Fig. 2 with the continuing switches shown as white circles. The dotted squares can be viewed as switch matrices in the conventional FPGA/FPIC architecture, except that segments can pass through them without activating any programmable switch. The same 2-terminal net in Fig. 1 and 2 is routed passing through five switches, two of them are continuing switches and the other three are turning switches. It is easy to see that any routing path in Fig. 2 can also be found in Fig. 3, since all routing resource in Fig. 2 is retained in Fig. 3. Moreover, the usage of continuing switches usually reduces the number of turns in the routing path and results in either a shorter path or a path with fewer active switches. Therefore, introduction of continuing switches not only enhances the routability, but also decreases the number of active switches in the interconnecting routes, and thus, the routing delay.

2 A routing algorithm

The first step of our algorithm is to find a conventional two-dimensional global routing solution which satisfies the capacity constraints in the channels. A global routing solution not only ensures an even distribution of the nets in the routing channels to avoid congestion but also reduces the complexity of the overall routing problem. Note that the global routing step is not appropriate for the architecture proposed in [9] where there is no continuing switch in the routing architecture. The very limited amount of routing resource in this architecture makes the capacity constraint a weak necessary condition for achieving a detailed routing solution in most cases. That is, a global routing solution satisfying the capacity constraint often does not lead to a detailed routing solution. Fig. 2 shows an example of such a scenario. The polygon with dashed-line boundary is a given global route for net 3. In the absence of continuing switches, we cannot find a detailed route corresponding to the given global route. However, with continuing switches, a detailed route for net 3 can be found easily as shown in Fig. 3.

Our algorithm for finding a detailed route corresponding to a given global route consists of two phases. In the first phase, we process the nets one at a time and a polynomial time algorithm is used to find an optimal detailed route for each net. When an unroutable net is encountered, a polynomial time rip-up and reroute algorithm is used in the second phase to resolve the conflict. We assume that a route cannot change track in a channel. From a practical point of view, changing track activates more turning transistors which degrades the timing performance. Moreover, the net will occupy more wire segments in the same channel, and thus, increases the possibility of blocking other nets. Note, however, that our algorithm can nonetheless be extended to handle the case in which track changing is allowed.

2.1 An optimal single-net routing algorithm

Suppose we are given a net and its global route as depicted in Fig. 4(a) where the global route is shown as a dashed polygon. We can represent these information using a two-dimensional grid as follows : Each grid point corresponds to a switch matrix and each grid edge (u, v) corresponds to the channel between the two switch matrices corresponding to u and v. Each rectangular region corresponds to a logic block, the terminals of which are represented by points in the region. There is a short edge, corresponding to a terminal segment, which connects a terminal to a grid edge corresponding to a channel. A global route is represented by a tree containing short edges and grid edges. We call such a two-dimensional grid representation of a global route a *channel representation*. Fig. 4(b) shows the channel representation of the global route of net 3 in 4(a). It is not hard to see that the number of detailed routes for a given global route can grow exponentially with respect to the number of terminals. Among these detailed routes, we want to determine one with minimum cost. The cost is defined as a weighted sum of the cost of active programmable switches and the cost of wire segments in the route. The cost of a wire segment is a weighted measure of the delay associated with the wire segment and how much the segment is in demand for routing. The cost of an active programmable switch is the time delay it introduces. If there is no turning switch located at the intersection of a horizontal track and

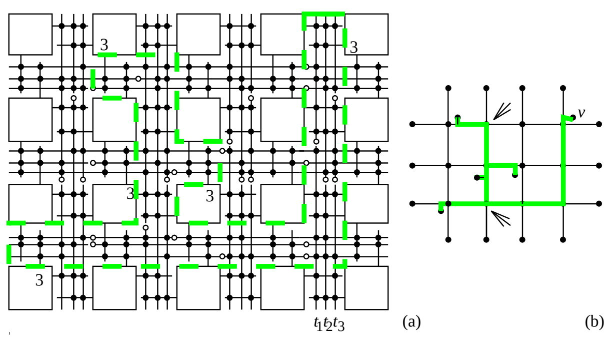

Fig. 4. A global route of net 3 and its channel representation

a vertical track in a switch matrix, we can conceptually assume that there is one with infinite cost at the intersection. Therefore, we shall assume that there is a turning switch at each intersection of a horizontal track and a vertical track in a switch matrix in the rest of this section. Also, we set the cost of any wire segment, turning switch, or continuing switch to be infinity, when it is occupied by a net processed before.

Given the channel representation of a global route, we decompose the channel representation into maximal line segments. A maximal line segment is a line segment that is not properly contained in any other line segment in the channel representation. For example, the line segment between the two arrows in Fig. 4(b) is a maximal line segment. We arbitrarily choose a terminal x and build a rooted tree, $T = (V_T, E_T)$, which represents the intersection relationship between maximal line segments as follows: Each maximal line segment has a corresponding vertex in V_T and $(u, w) \in E_T$ if and only if the maximal line segments corresponding to u and w intersect in the channel representation. The root of T is the maximal line segment incident with x. The rooted tree T for the channel representation in Fig. 4(b) is shown in Fig. 5. Given a node w in T, we use $l(w)$

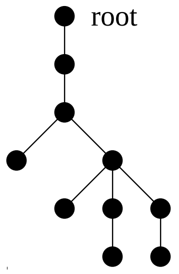

Fig. 5. T for the channel representation in Fig. 4

to denote the maximal line segment corresponding to w. For a maximal line segment h, we use $r(h)$ to denote the channel that contains h. Also, we use $l^{-1}(h)$ to denote the node w of T such that $l(w) = h$. If w is a leaf in T, then $r(l(w))$ is a terminal segment. If w is an internal node of T, $r(l(w))$ is the channel which contains the maximal line segment $l(w)$.

Given T, the routing problem becomes that of choosing a track in $r(l(w))$ for the detailed route of the maximal line segment $l(w)$, for each $w \in T$. Suppose we have chosen a track for the maximal line segment $l(w)$, for each $w \in T$. Then, we can compute the cost of a subtree in T, rooted at a node, say s, by adding the costs of the wire segments used in the route for the maximal line segment $l(w)$, for each descendant w of s, and the costs of all turning switches and continuing switches used in the route. (We assume a node in a rooted tree to be a descendant of itself.)

Suppose that there are t tracks in each channel. We use C_j^s to denote the minimum cost of the route for the entire subtree rooted at s, with track j, $1 \le j \le t$, being chosen for the route of $l(s)$. We call C_j^s, for each j such that $1 \le j \le t$, a *C-cost* of s. In the case that s is a leaf, the cost of the subtree rooted at s is simply the cost of a terminal segment. In this case, we can conceptually assume that there are t terminal segments the terminal is connected to and then C_j^s, for $1 \le j \le t$, is well-defined. In the case that s is an internal node of T, let $v_1, v_2, \ldots, v_n$ denote the children of s in T. Then, we have the following recurrence equation:

$$C_k^s = \sum_{i=1}^{n} \min_{m=1,2,\ldots,t} (C_m^{v_i} + cost\_of\_turning\_switch\_at(k,m,i))$$
$$+\text{costs of continuing switches on track } k \text{ in } r(l(s))$$
$$+\text{costs of wire segments on track } k \text{ in } r(l(s)) \qquad (1)$$
$$\forall k \in \{1,2,\ldots,t\}$$

where $cost\_of\_turning\_switch\_at(k,m,i)$ is the cost of the turning switch located at the intersection of the kth track in $r(l(s))$ and the mth track in $r(l(v_i))$. According to formula (1), to compute the values of all C_k^w's, we need to compute the C-costs of all children of w. Therefore, our algorithm processes the tree T in a bottom-up fashion and computes the C-costs for each node until the root is reached. Then, we choose one with minimum C-cost and trace back to decide the track for each maximal line segment.

Our algorithm computes an optimal detailed route for a given global route as stated in the following theorem:

Theorem 1 *Our algorithm computes a detailed route with minimum cost in $O(mt(mt+p))$ time, where m is the number of nodes in T, t is the number of tracks in each channel, and p is the maximum number of wire segments among all tracks.*

To analyze the time complexity of our algorithm, we observe that there are m nodes in T and for each node there are t C-costs to be computed, one for each track. To compute the C-cost of a node for a track using the formula described above, we need to use the costs of the wire segments along this track and the C-costs of its children for this track. The number of all these variables is $O(p + kt)$, where k is number of children of the node considered and it is at most m. Therefore, the time complexity stated in the theorem follows.

2.2 rip-up and reroute

Since the global router tries to avoid congestion by distributing the global routes evenly in the channels, it is likely that the detailed routing algorithm presented in the previous subsection will succeed in finding a detailed route for each net. In the case that it fails to find a detailed route, we use a rip-up and reroute algorithm to reroute some of the existing routes one track at a time to make room for the route of the current net.

Suppose the global route for net n_7 passes through channel c in which there are existing detailed routes of other nets as shown in Fig. 6(a). Specifically, the detailed route for

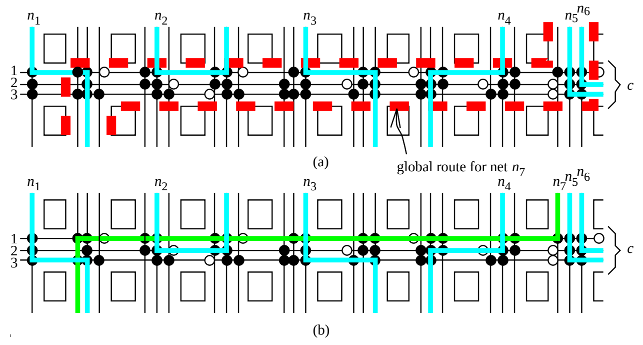

Fig. 6. Dashed lines sketch part of the global route of the currently processed net.

net n_7 is blocked by nets $n1$, n_2, n_3, and n_4 in track 1, by net n_5 in track 3, and by net n_6 in track 2. We examine the tracks in the channel one at a time and shall rip-up and reroute all detailed routes which existed in the track. Let t_1 be the track under consideration. We route the blocked net in track t_1 and then try to reroute all detailed routes which existed in track t_1. If we succeed in rerouting all these nets we then return to the first phase for the next net. Otherwise, we consider the next track in the channel until either rerouting is successful in some track or it fails in all tracks. In the latter case, the global route of the net under consideration is modified and the first phase of our routing algorithm is repeated.

We now consider the problem of rerouting the detailed routes that existed in track t_1. If we want to reroute all the nets in track 2 for the example in Fig. 6, then the answer is no, since it is impossible to reroute the detailed route of net n_6 in track 1 and track 3. So is the case when we try to reroute all the nets in track 3. However, if we try to reroute the nets in track 1, we can reroute the nets n_1, n_2, n_3, and n_4 in tracks 2 and 3 as shown in Fig. 6(b). Therefore, track 1 can be used to route the blocked net n_7.

Formally speaking, let $\Re$ denote the set of detailed routes in track t_1 of channel c, no two of them overlap. Let U denote the set of unused wire segments in channel c. We want to know if there is a way to reroute the nets in $\Re$ using the unused wire segments in U. First of all, our algorithm labels the detailed routes in $\Re$ from left to right as l_1, l_2, …, $l_{|\Re|}$. For each detailed route in $\Re$, say l_m, we use $L_i(m)$ to denote the leftmost unused wire segment in track i whose span overlaps with that of l_m and $R_i(m)$ to denote the rightmost unused wire segment in track i whose span overlaps with that of l_m. For example, in Fig. 6(b), $L_2(2)$ denote the leftmost wire segment in track 2 and $R_2(3)$ denote the third leftmost wire segments in track 2. Suppose we have succeeded in rerouting the detailed routes l_1, l_2, …, l_m, where $m < |\Re|$. When we try to reroute the detailed route l_{m+1}, we list all tracks which are available for l_{m+1}. Let t be the number of tracks in the channel. We can use a binary string $(a_1^{m+1}, a_2^{m+1}, \ldots, a_t^{m+1})$ to represent the availability of the tracks for l_{m+1}. If l_{m+1} cannot be rerouted in track i, then $a_i^{m+1} = 1$. Otherwise, $a_i^{m+1} = 0$. We call such a binary string an *availability vector* for l_{m+1}. According to the definition of an availability vector, if the existing detailed route of another net conflicts with the rerouting of l_{m+1} in track i, then $a_i^{m+1} = 1$, no matter how l_1, l_2, …, l_m are rerouted. Moreover, to reroute l_m in track i, there must be turning switches available for connecting l_m to the remaining route of the net. For the example in Fig. 6(b), we cannot reroute net 2 in track 2, if there is no turning switch between the intersection of the middle track in one of the two vertical channels and track 2 in the horizontal channel. Therefore, a_i^{m+1} will be 1 for all such tracks. We use V_{m+1} to denote the set of tracks which are candidates for the rerouting of l_{m+1}. Given an availability vector for l_m, $(a_1^m, a_2^m, \ldots, a_t^m)$, we determine the following five sets of tracks,

$A = \{i \mid \text{track } i \in V_{m+1},\ a_i^m = 0 \text{ and } R_i(m) \neq L_i(m+1)\}$,
$B = \{i \mid \text{track } i \in V_{m+1},\ a_i^m = 0 \text{ and } R_i(m) = L_i(m+1)\}$,
$C = \{i \mid \text{track } i \in V_{m+1},\ a_i^m = 1 \text{ and } L_i(m) \neq L_i(m+1)\}$,
$D = \{i \mid \text{track } i \in V_{m+1},\ a_i^m = 1 \text{ and } L_i(m) = L_i(m+1)\}$,
$E = \{i \mid \text{track } i \notin V_{m+1}\}$.

We will generate $|B| + 1$ availability vectors for l_{m+1} which satisfy the constraints :

1. $a_i^{m+1} = 0$, for each i in Set A,
2. $a_i^{m+1} = 0$, for each i in Set C,
3. $a_i^{m+1} = 1$, for each i in Set D,
4. $a_i^{m+1} = 1$, for at most one i in Set B,
5. $a_i^{m+1} = 1$, for each i in Set E.

Note that there are exactly $|B| + 1$ availability vectors satisfying the five constraints above. The first constraint states that if track i is available for l_m and rerouting l_m does not prohibit l_{m+1} from being rerouted in track i, then track i is available for l_{m+1}. The second constraint states that if track i is occupied by some existing route and this route does not block the rerouting of l_{m+1} in track i, then track i is available for l_{m+1}. The third constraint states that if some existing route uses the leftmost segment which is needed for l_{m+1}, then track i is not available for l_{m+1}. Note that we can only reroute l_m in track i for i in Sets A and B, since $a_i^m = 0$. If we choose track i in Set A for rerouting l_m, then all the tracks in both Set A and B are available for rerouting l_{m+1}. If we choose track i in Set B for rerouting l_m, then that track is no longer available for rerouting l_{m+1}. The fourth constraint is imposed based on this observation. The last constraint states that if track i is not in V_{m+1}, then it is impossible to reroute l_{m+1} in track i. Note that there is only one availability vector for l_1, the ith coordinate of which is 0, if track $i \in V_1$, and the ith coordinate of which is 1, otherwise. Based on the relationship between the availability vectors for l_m and l_{m+1} and the availability vector for l_1, we can compute all the availability vectors for l_i, for each i, $2 \leq i \leq |\Re|$. Note that if all the availability vectors are $(1, 1, \ldots, 1)$ for some l_i , then it is impossible to reroute $\Re$ using U. Otherwise, if there is a zero entry in any of the availability vectors for $l_{|\Re|}$, then a way to reroute each l_i, $1 \leq i \leq |\Re|$, in $\Re$ using U can be found by tracing back the computation of that availability vector.

Note that if there are two availability vectors for l_m, $(a_1^m, a_2^m, \ldots, a_t^m)$ and $(b_1^m, b_2^m, \ldots, b_t^m)$, such that $a_i^m \leq b_i^m$, for $1 \leq i \leq t$, then the latter is redundant, since the tracks available for l_m specified in the former include those specified in the latter. Therefore, it is unnecessary to keep the redundant availability vector. To store the availability vectors for each route in $\Re$, we use the data structure trie [1]. The number of nonredundant availability vectors is bounded by $\binom{t}{\frac{t}{2}}$, which is the number of binary sequence of length t with $\frac{t}{2}$ ones. If we do not use the data structure tries, the storage size will be $t\binom{t}{\frac{t}{2}}$. However, when a trie is used the storage is at most $2 + 4 + \ldots + 2^t = 2^{t+1} - 2$.

3 Experimental results

Our algorithm was implemented in the C language and executed on a SPARC-10 station. The program was used to route five industrial circuits used in [3, 9]. Each track is segmented such that the number of segments of length l is proportional to $\frac{1}{l}$, where l is a power of 2. Segments in each track are arranged in a random manner so that there is no bias on the distribution of routing resource. The first circuit, BUS_CNTL, is a bus controller. Let k be the number of turning switches allocated to the horizontal segments and vertical segments at the intersection of a horizontal channel and a vertical channel. For each of the cases in which the number of tracks in a channel is 11, 12, 13, and 14, we determined the minimum value of k such that our algorithm produces a 100% completion routing solution. Two different ways to allocate these k turning switches are used in our experiments. One is to randomly generate the positions of these k turning switches such that routing resource is spread evenly without any bias. Fig. 7(c) shows an example of a random arrangement, where the number of tracks in each channel is 6 and $k = 3$. Results corresponding to such an allocation of turning switches are shown in the second column of Table 2. The other way is to arrange consecutive locations for the k turning switches in each row with an offset of one column between adjacent rows. Fig. 8 shows such an arrangement when the number of tracks is 6 and $k = 3$. Results corresponding to such an allocation of turning switches are shown in the third column of Table 2. Note that the experiments show that the arrangement illustrated in Fig. 8 has better routability than the random arrangement. Further justification is needed. BUS_CNTL was routed on an array of 13 × 12 blocks with 151 nets, or 392 equivalent two-terminal nets. The routing result in [3] used a total of 1616 active switches in the connecting paths. In [9], fewer than 1170 are used in all these instances. The results in [9] and [3] for BUS_CNTL are summarized in Table 1. In our case, we further reduce the number of switches used. Note that in [3], the channel density of the global route is used as a lower bound on the number of tracks required in their algorithm. Their algorithm achieves the lower bound in two benchmark circuits and uses at most two more tracks than the lower bound in the other three circuits. However, in our routing architecture, given a global route, the channel density is usually a loose lower bound of the number of tracks required in each channel. Consider the example in Fig. 7.

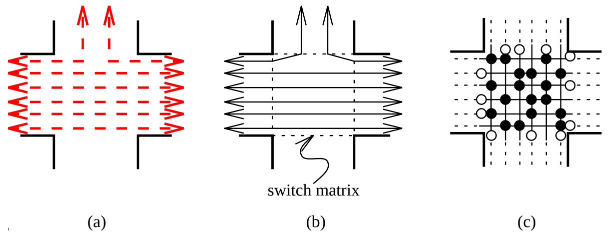

Fig. 7. 6 is a loose lower bound for the routing architecture in (c). 7 tracks are required.

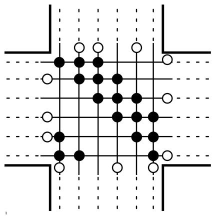

Fig. 8. One way to allocate turning switches

Given the global route in 7(a), if we use the routing architecture in [3], then 6 tracks are enough as shown in 7(b). However, for our routing architecture, 7 tracks are required since each net needs a horizontal wire segment in order to pass through the intersection of the horizontal channel and the vertical channel. Based on this observation, we analyzed the same given global route used in [9] and found that the minimum number of tracks required is at least 11 in our architecture, which was achieved using our algorithm. The number of tracks used in [3] is 10, one more than the channel density. The number of tracks for the architecture in [9] is at least 12. It should be emphasized that the minimum value of k for achieving a 100% completion routing solution decreases dramatically in our architecture. This is due to the increase in routability by the introduction of the continuing switches. The total numbers of ripped up and rerouted nets are only 5, 1, 3, and 0, respectively. Therefore, most of the routing is done by our router in Section 2.1 and only in very rare case the rip-up and reroute algorithm is used to resolve the conflict.

TABLE 1
SUMMARY OF RESULTS IN [9] AND [3] FOR BUS_CNTL

	# of tracks	min. k	# active switches	total # switches
[9]	11	-	-	-
	12	11	1165	17424
	13	9	1151	15444
	14	6	1167	11088
[3]	10	6	1616	15840

For the other four circuits, we used the same number of tracks as in [9] and found the corresponding minimum value of k in order to achieve a 100% completion routing solution. Results for a random allocation of turning switches are shown in the third column of Table 3 and results for the "off-set" allocation of turning switches as illustrated in Fig. 8 are shown in the fourth column which indicates a slight improvement in routability. Compared with the results of [9], the dramatic reduction in the values of k demonstrates

TABLE 2
SUMMARY OF OUR RESULTS FOR BUS_CNTL

# of tracks	min. k	min. $k^{\dagger}$	# active switches	$\frac{\text{active sw. (Ours)}}{\text{active sw. [3]}}$	total # switches	$\frac{\text{total sw. (Ours)}}{\text{total sw. [9]}}$	rerouted nets	time (seconds)
11	4	4	1124	70%	7326	-	5	0.73
12	5	3	1093	68%	9576	55%	1	0.73
13	3	3	1123	69%	6942	45%	3	0.67
14	3	2	1116	69%	7476	67%	0	0.78

TABLE 3
SUMMARY OF OUR RESULTS FOR THE OTHER FOUR CIRCUITS

circuit name	# of tracks	min. k	min. $k^{\dagger}$	# active switches	$\frac{\text{active sw.(Ours)}}{\text{active sw. [3]}}$	total # switches	$\frac{\text{total sw.(Ours)}}{\text{total sw. [9]}}$	rerouted nets	time (sec.)
DMA	13	6	5	2188	65%	23439	54%	4	1.73
EBNR	15	6	6	3383	59%	43950	54%	9	3.12
FSM	16	3	3	3907	64%	29744	57%	1	3.33
Z03	16	8	8	6092	61%	93808	60%	2	6.35

TABLE 4
SUMMARY OF RESULTS IN [9] AND [3] FOR THE OTHER FOUR CIRCUITS

	circuit name	# of tracks	min. k	# active switches	total # switches
[9]	DMA	13	13	2237	43095
	EBNR	15	13	3830	81900
	FSM	16	7	4193	51744
	Z03	16	15	6054	156000
[3]	DMA	10	6	3354	30600
	EBNR	12	6	5700	60480
	FSM	10	6	6132	55440
	Z03	13	6	10039	101400

again that our architecture has much better routability than that proposed in [9]. Also, the number of active switches in the routing solution is substantially smaller than that in [3]. The algorithm is very efficient. It took less than 7 seconds for each circuit. The results are summarized in Table 3 and 4.

4 Conclusion

We have proposed a new two-dimensional FPGA/FPIC routing architecture which has better routability and uses fewer active programmable switches than the architecture proposed in [9]. An effective routing algorithm is developed based on the characteristics of the routing architecture. The rip-up and reroute phase of this algorithm can also be used to solve Actel row-based channel routing problem. Experimental results show that the routing architecture is flexible and the routing algorithm performs well. It is interesting to note that our rip-up and reroute algorithm for two-dimensional FPGA/ FPIC routing architecture can be also used as a heuristic method to solve Actel row-based channel routing problem. First of all, we use the left edge algorithm [7] to divide the 2-terminal nets into d groups, where d is the density of these 2-terminal nets. We then process the 2-terminal nets in one group at a time during which we use our rip-up and reroute algorithm described in Section 2.2 to assign these 2-terminal nets to tracks subject to the constraint that no two nets can share the same wire segment. Recent investigation [10] found that this track assignment problem for non-overlapping nets can be done in linear time.

Acknowledgment The authors are grateful to their colleague Peichen Pan for his discussion in the rip-up and reroute algorithm, Professor Stephen Brown in the University of Toronto for providing experimental examples and anonymous reviewers for their valuable comments .

References

[1] A. Aho, J. Hopcroft, and J. Ullman, *Data Structures and Algorithms*, Addison-Wesley, 1983.

[2] Aptix Inc., *FPIC AX1024D*, Preliminary Data Sheet, August 1992.

[3] S. Brown, J. Rose, and Z. Vranesic, "A detailed router for field-programmable gate arrays," *IEEE Transactions on Computer-Aided Design*, vol. 11, no. 5, pp. 620-628, May 1992.

[4] J. Frankle, "Iterative and adaptive slack allocation for performance-driven layout and FPGA routing," *29th ACM/IEEE Design Automation Conference*, 1992, pp. 536-542.

[5] A. El Gamal, J. Greene, J. Reyneri, E. Rogoyski, K. El-Ayat, and A. Mohsen, "An architecture for electrically configurable gate arrays," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 2, pp. 394-398, April 1989.

[6] R. Guo, H. Nguyen, A. Srinivasan, H. Verheyen, H. Cai, S. Law, and A. Mohsen, "A 1024 pin universal interconnect array with routing architecture," *IEEE Custom Integrated Circuits Conference*, 1992, pp. 4.5.1-4.5.4.

[7] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures, " *Proceedings of 8th IEEE Design Automation Workshop*, 1971.

[8] M. Palczewski, "Plane parallel A* maze router and its application to FPGAs," *29th ACM/IEEE Design Automation Conference*, 1992, pp. 691-697.

[9] Y. Sun, T. C. Wang, C. K. Wong, and C. L. Liu, "Routing on Symmetric FPGA/FPIC," *IEEE/ACM International Conference on Computer-Aided Design*, 1993.

[10] Y. Sun and C. L. Liu, *manuscript*, February 1994.

[11] Xilinx Inc., *XC4000 Logic Cell Array Family*, Technical Data, 1990.