Performance-Driven Board-Level Routing for FPGA-based Logic Emulation

Wai-Kei Mak and D. F. Wong
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712

February 16, 1998

Abstract

In [10] and [11], the board-level routing problem in FPGA-based logic emulators that use crossbars for interconnection was studied, and two different routing algorithms were proposed. However, the performance issue was not considered in the previous algorithms. And they are also unable to handle routing constraints that may arise from certain timing requirement. So, in this paper we propose a performance-driven routing algorithm for the board-level routing problem that can handle additional routing constraints and reduce the delay of the routing solutions. It is an optimization algorithm based on minimum cost flow computation.

1 Introduction

Efficient verification is of high importance in the design of new digital systems. Because of the complexity of the problem, logic simulation by software often cannot completely verify the behaviour of large systems. Recently several logic emulation systems [3-7] that consist of a set of interconnected Field-Programmable Gate Arrays (FPGAs) [1,2] to prototype large digital logic designs were developed. And these systems can emulate complex digital system designs several orders of magnitude faster than software simulators. As a result, FPGA-based logic emulators can verify large designs that otherwise are not verifiable by software simulators.

There are two major steps in doing logic emulation. First, a large design is partitioned into parts each of which can fit inside a single FPGA on the logic emulator [8,9]. Then board-level routing is performed to connect the signals between the FPGA chips.

In logic emulators such as the Realizer system [3] and the Enterprise Emulation system [5] manufactured by Quickturn Design Systems, the set of FPGAs for implementing the logics are

interconnected by a set of small full crossbars. In this paper, we address the problem of board-level routing applicable to the logic emulation systems that use small full crossbars for interconnection.

In the crossbar interconnect architecture, the interconnection crossbars only connect to the FPGAs but not to each other. The I/O-pins of each FPGA are divided into proper subsets of equal size. The pins of each crossbar are connected to the same subset of pins from each FPGA. Thus crossbar i's pins are connected to the pins of subset i from each FPGA (see Fig. 1). The number of crossbars used is equal to the number of subsets in a FPGA, and each crossbar has as many pins as the number of pins in a subset times the number of FPGA chips. An inter-chip net can be connected via crossbar i if its net-pins in different FPGA chips are all assigned to I/O-pins in subset i.

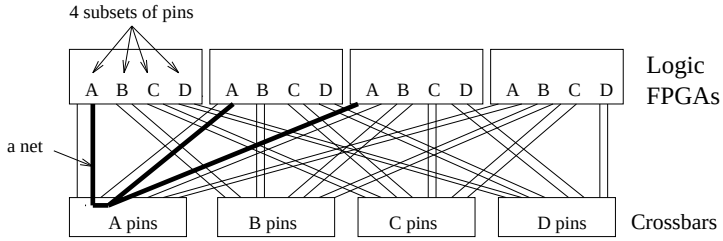

Figure 1: Interconnect Architecture.

Efficient algorithms for board-level routing which always guarantee routing completion when all nets are two-terminal nets and the I/O-pin subset size is even were proposed independently in [10] and [11]. Both algorithms assume that each net can be freely connected via any crossbar. However, since the crossbars are distributed over different locations on a board, the distances a net signal needs to travel when a net is routed via different crossbars can be largely different. So for some highly critical nets, there should be some restrictions as which crossbars each can be routed through to satisfy certain timing constraints. Moreover, for nets on critical paths but do not have any rigid routing restriction, it is still advantageous to route them through shorter routes to reduce the overall delay. So, in this paper we propose a new algorithm that can handle additional routing constraints, and reduce the overall delay by optimizing the routing cost for a given subset of nets.

2 The Routing Problem

We will refer to the FPGAs on a logic emulator simply as chips. We assume that all the chips are identical. Let p be the number of I/O-pins in each pin subset on a chip. And let K be the

number of pin subsets on each chip, which is also the number of crossbars. It follows that the total number of I/O-pins on each chip is Kp.

The board-level routing problem is the problem of assigning the net-pins in each chip to the I/O-pins on the chip so that all pins of the same net are assigned to I/O-pins of the same type. At most one net-pin can be assigned to each I/O-pin. Fig. 2 shows a routing solution to a problem instance with six nets and three chips where $p = 2$ and $K = 2$. The board-level routing problem can also be viewed as the problem of assigning each net to a pin subset type such that in any chip no more than p nets are assigned to the same pin subset, or equivalently, no more than p nets from the same chip are routed via the same crossbar. In this paper, we consider the board-level routing problem with some routing constraints. In particular, we consider the problem when we are given some nets with stringent timing requirement, and thus can only be routed via certain crossbars. And we are given a subset S of nets whose routing cost should be minimized. Our objective is to get a routing solution satisfying the routing constraints that minimizes the total routing cost of all nets in S.

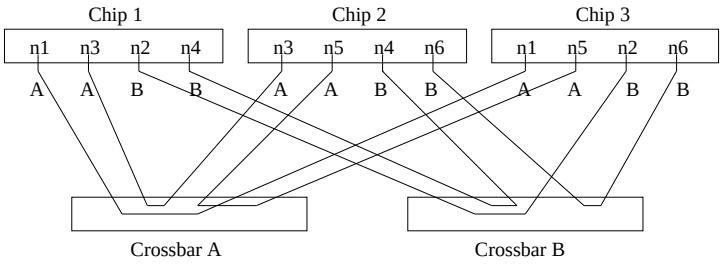

Figure 2: A routing solution.

3 Performance-Driven Routing Algorithm

In this paper, we assume that all nets are two-terminal nets and the I/O-pin subset size p is even. (We have presented a multi-terminal net decomposition algorithm in [12] that can be used to decompose multi-terminal nets in a board-level routing problem instance to two-terminal nets.) Throughout this section, we will use the routing problem instance shown in Fig. 3 as a running example.

The netlist of a routing problem instance can be represented succinctly using a multi-graph where the vertices and the edges correspond to the chips and the nets, respectively. For example, the netlist in Fig. 3 is represented as in Fig. 4.

The very first step of our performance-driven routing algorithm is the same as in [10]. We construct a bipartite graph of the given routing problem instance as described below. (We

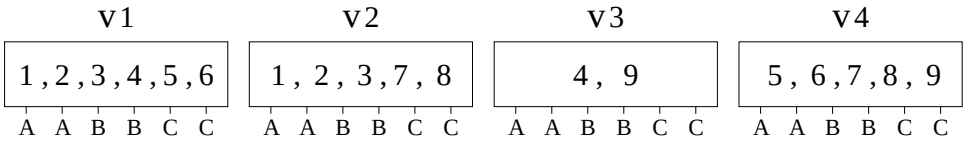

Figure 3: A routing instance.

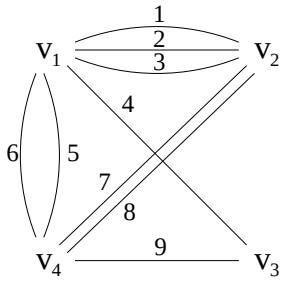

Figure 4: The multigraph representation of a netlist.

assume that the degree of each vertex in the multigraph netlist representation is equal to Kp, the total number of I/O-pins on each chip. Note that we can always ensure that by adding dummy nets to the multigraph as shown in Fig. 5.) First, we give a direction to each net in the multigraph representation so that for each vertex the number of incoming edges is equal to the number of outgoing edges. This can be done by computing an Euler circuit on the mulitgraph, and traversing the Euler circuit once to give directions to the edges along the way (see Fig. 6(a)). Then, we unfold the multigraph into a bipartite graph by splitting each vertex v into two vertices v' and v'', and replacing each directed edge (u, v) by edge (u', v''). Refer to Fig. 6 for an example. This obtains a bipartite graph B_1 and finishes the first step.

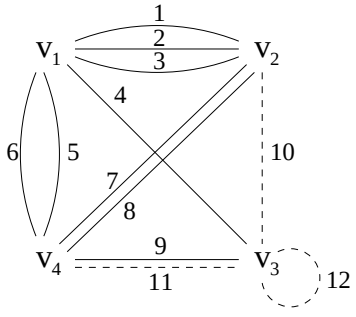

Figure 5: Addition of dummy nets.

What we will do next is different from that in [10]. In [10], disjoint matchings are found

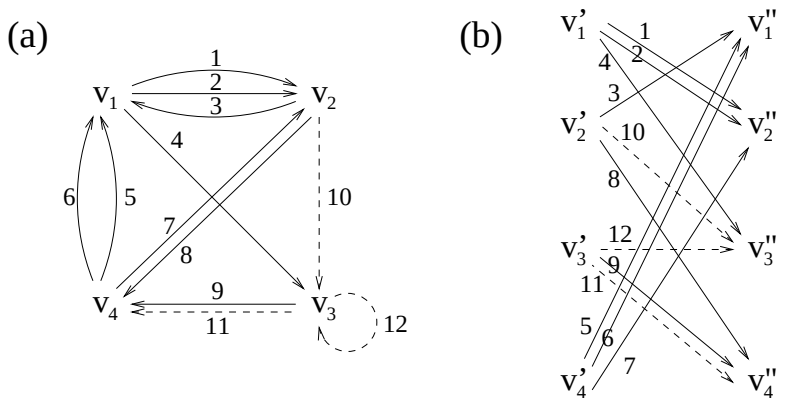

Figure 6: (a) Addition of directions, according to the Euler circuit: 1-10-12-11-6-2-3-4-9-7-8-5. (b) Unfold into a bipartite graph.

in the bipartite graph through a bipartite graph coloring problem formulation. However, additional constraints like "net 1 can be routed through crossbar A or B, but not any other crossbar" cannot be enforced using this approach.

Now we show how we can handle routing constraints like the one above, and at the same time minimize the total routing cost of a given subset of nets. Observe that if we can assign the edges (i.e., nets) in the multigraph to different crossbars such that at any vertex v the number of outgoing edges and the number of incoming edges assigned to crossbar j are both equal to $p/2$ for all $j = 1, 2, \ldots, K$ without violating any routing constraints, then it will give a feasible routing solution. (For example, the assignment in Fig 8(c) corresponds to the routing solution in Fig 9.) Or equivalently, if we can assign the edges in the bipartite graph B_1 to different crossbars such that for any v, the number of edges incident with node v' assigned to crossbar j and the number of edges incident with node v'' assigned to crossbar j are both equal to $p/2$ for all $j = 1, 2 \ldots, K$ without violating any routing constraints, then it will yield a feasible routing solution. Based on this observation, our algorithm is divided into $K-1$ stages. In the first stage, we determine the edges in the bipartite graph to be assigned to crossbar 1 under the above condition, then in the second stage we determine the edges to be assigned to crossbar 2 under the above condition, so on and so forth. After $K-1$ stages, the nets that remain unassigned will all be assigned to crossbar K.

To choose the nets to be routed via crossbar 1, first we take away those nets that cannot be routed via crossbar 1 from B_1. Let B_1^{const} (read as the constrained-B_1) be B_1 without those edges corresponding to the nets that cannot be routed via crossbar 1. See Fig. 7(a)(b) for an example. And we attach a cost $c_i(n)$ to each arc n in B_1^{const}. (We will postpone the discussion

of the suitable value for $c_i(n)$ until the whole algorithm has been described.)

Then we do a minimum cost flow[1] computation [13,14] assuming each node v' on the left hand side of B_1^{const} is a supply node with $p/2$ units of flow supply and each node v'' on the right hand side is a demand node with $p/2$ units of flow demand, and each arc has a unit capacity. See Fig. 7(b) where we have $p = 2$. We note that the minimum cost flow problem instance at hand has a very nice property. Because all arc capacities and external flows are integers (given that p is even), there always exists an integral minimum cost flow (i.e., the flow in each arc is an integer). Moreover, most common minimum cost flow algorithms will produce an integral minimum cost flow solution under this condition. And in our case, the flow in each arc is either 0 or 1. We will assign net n to crossbar 1 if and only if arc n's flow value is 1 (refer to Fig. 7(b) and Fig. 8(a)). After a net has been assigned to crossbar 1, we can remove its arc from the original bipartite graph B_1, this results in a reduced bipartite graph B_2 as in Fig. 7(c).

To determine which nets are to be assigned to crossbar 2, we take away those remaining unassigned nets that cannot be routed via crossbar 2 from B_2. Let B_2^{const} be B_2 without such arcs (Fig. 7(d)). We may do a minimum cost flow computation on B_2^{const} like before to determine which remaining unassigned nets are to be assigned to crossbar 2 as shown in Fig. 7(d) and Fig. 8(b).

In general, after determining the nets assigned to crossbars 1 to i, bipartite graph B_{i+1} is the original bipartite graph B_1 minus all nets assigned to crossbars 1 to i. If we take away those remaining nets that cannot be routed through crossbar $(i+1)$ from B_{i+1}, we can obtain B_{i+1}^{const}. By doing a minimum cost flow computation on B_{i+1}^{const}, we can determine which nets are to be assigned to crossbar $(i+1)$.

Finally, after the nets to be assigned to crossbars 1 to $K-1$ have been determined as above, all remaining nets will be assigned to crossbar K and this completes the routing process.

Now we are going to discuss what is a suitable choice for the arc cost value $c_i(n)$ for each arc n in bipartite network B_i^{const}. Suppose that the cost of routing net n through crossbar i is $d_i(n)$. As our objective is to minimize the total routing cost of the nets in subset S, it may seem natural to minimize the routing cost at each stage, this suggests to set $c_i(n)$ to $d_i(n)$ if net n is in S. But this simple choice has its pitfall. Because the assignment of nets is done from crossbars 1 to K sequentially, a net in S will be assigned to crossbar i if the cost incurred

[1]Minimum cost flow is the problem of shipping commodities at the minimum possible cost from a set of supply nodes to a set of demand nodes in a network where each arc has a capacity that limits the amount of commodities it can carry and an arc cost which is incurred for every unit it carries. It is a generalization of the maximum network flow problem, and can be optimally solved in polynomial time.

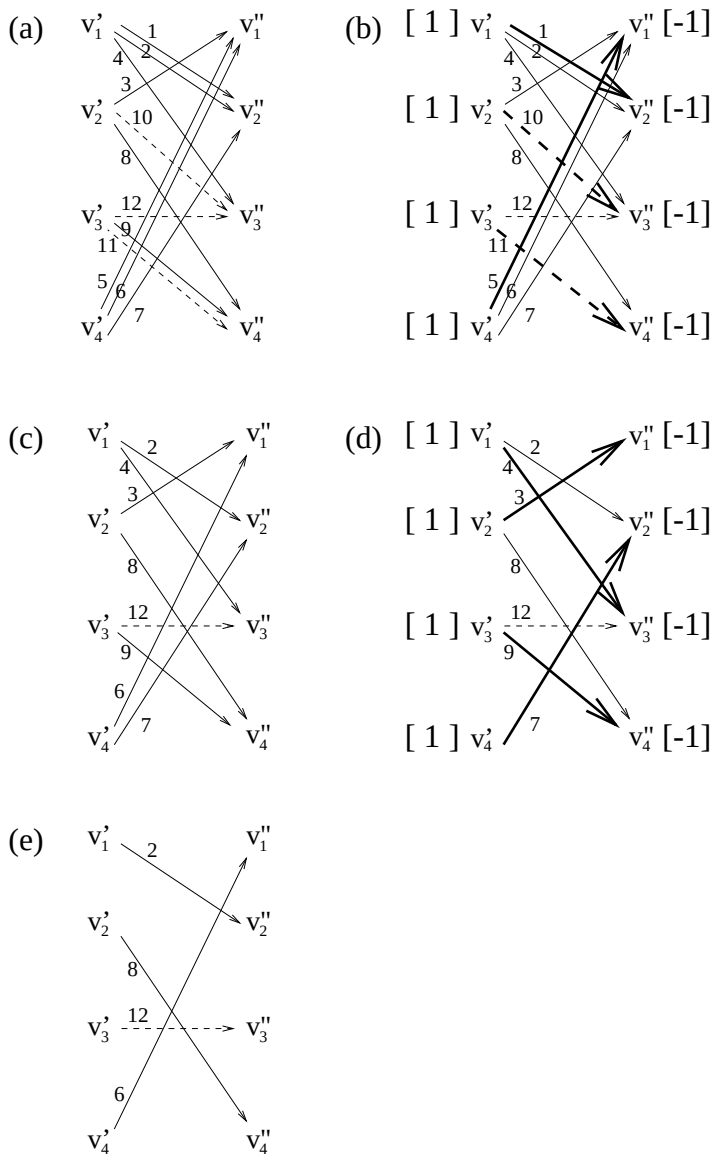

Figure 7: (a) B_1. (b) B_1^{const} (assuming net 9 cannot be routed through crossbar 1). A flow solution is arcs 1, 10, 11, and 5 each carries a unit flow. (c) B_2(= B_1 minus arcs 1, 10, 11, and 5). (d) B_2^{const} (assuming net 6 cannot be routed through crossbar 2). A flow solution is arcs 4, 3, 9 , and 7 each carries a unit flow. (e) B_3(= B_2 minus arcs 4, 3 ,9 , and 7).

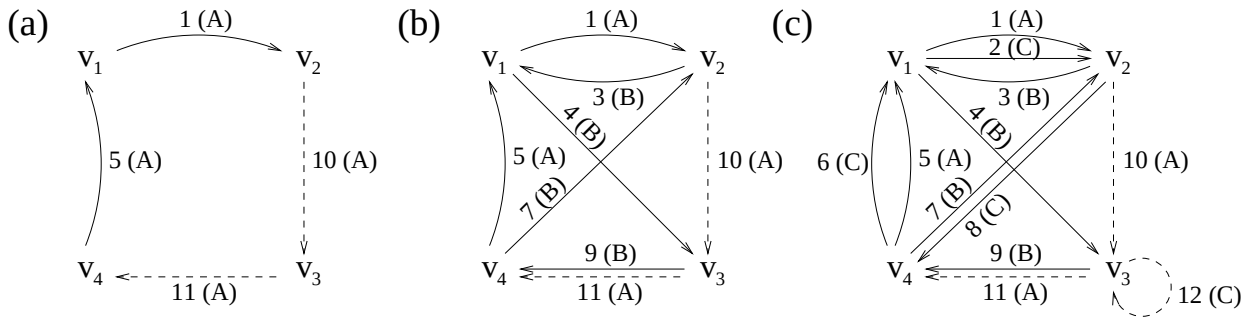

Figure 8: (a) Assignment after stage 1. (b) Assignment after stage 2. (c) Final assignment.

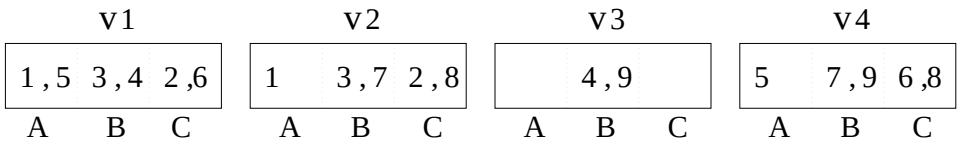

Figure 9: A feasible routing solution.

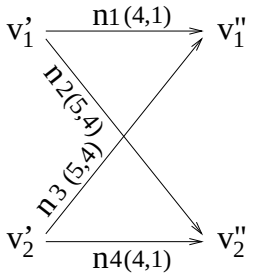

Figure 10: The ordered pair (d_1, d_2) beside an arc n indicates the costs of routing net n through two different crossbars, d_1 is the cost of routing through crossbar 1 and d_2 is the cost of routing through crossbar 2.

immediately is small, however, it is possible that it will incur an even smaller cost if the net is assigned to crossbar j for some $j > i$. To illustrate this, let us consider the example in Fig. 10. If we do the assignment of nets to crossbar 1 before the assignment to crossbar 2, then nets n_1 and n_4 will be selected for routing via crossbar 1 (as $d_1(n_1) + d_1(n_4)(= 4 + 4)$ $< d_1(n_2) + d_1(n_3)(= 5 + 5)$), and nets n_2 and n_3 will have to be routed through crossbar 2. The total routing cost in this case is $d_1(n_1) + d_1(n_4) + d_2(n_2) + d_2(n_3) = 16$. But the optimal total routing cost is 12 which is achieved when nets n_2 and n_3 are routed through crossbar 1, and nets n_1 and n_4 are routed through crossbar 2.

So, we propose to use an adjusted arc cost $c_i(n)$ as follows.

$$c_i(n) = \begin{cases} d_i(n) - \frac{\sum_{j=i+1}^{K} d_j(n)}{K-i} & \text{if net } n \text{ is in } S \\ 0 & \text{otherwise} \end{cases}$$

We let $d_j(n)$, the cost of routing net n through crossbar j, to be equal to M for a very large constant M if net n cannot be routed through crossbar j. The second term in the the arc cost for a net in S represents an opportunity cost. When we assign a net to crossbar i now, we are giving up the opportunity for it to be assigned to some other crossbars later. The opportunity cost is bigger when the average cost of assigning it to latter crossbars is smaller (i.e., we are forgoing a potentially higher saving that can be obtained by assigning it to some latter crossbar). When the arc cost for an arc n in S is negative, it is advantageous to assign net n to crossbar i now, but when the cost is positive, it is better to assign the net to some latter crossbar. And since it does not matter which crossbar a net not in S is assigned to (note that all dummy nets added are not in S), the arc cost for such a net is set to zero.

Using the adjusted cost ensures that a net with more routing restrictions will be routed with a higher priority to achieve routing completion of all nets. Because when a net n has more routing restrictions, there are less choices of crossbars it can be routed through, hence the more the number of j such that $d_j(n) = M$ and the more negative is arc n's adjusted cost at stage i if it can be routed through crossbar i. And an arc with a highly negative cost will always be selected by the minimum cost flow computation.

We also note that if the cost of routing any net n between chips u and v through crossbar j is a constant d_{ujv} independent of n, then except those nets that have some routing restrictions, all arcs between node u' and v'' in B_i^{const} have arc cost equals to either 0 (for nets not in S) or $d_{uiv} - \frac{\sum_{j=i+1}^{K} d_{ujv}}{K-i}$ (for nets in S) at any stage i. Also, multiple arcs between the same pair of nodes and with the same arc cost can be combined into a single arc with capacity equals to the actual number of arcs it represents. So we may reduce the number of arcs tremendously by combining arcs, and improve the efficiency of doing the minimum cost flow computation in

each stage.

The complete performance-driven routing algorithm is summarized below.

Preformance-Driven Routing Algorithm

Input: Netlist of a routing problem instance; K = number of crossbars; p = pin subset size

1. Represent netlist by a multigraph with dummy nets added to ensure all vertices have degree Kp; (Fig. 5)
2. Give a direction to each edge in the multigraph; (Fig. 6(a))
3. Unfold the multigraph into bipartite graph B_1; (Fig. 6(b))
4. **for** $i = 1$ to $K - 1$ **do**
   $B_i^{const} = B_i -$ nets that cannot be routed through crossbar i;
   compute arc cost $c_i(n)$ for each arc n;
   compute a minimum cost flow on B_i^{const} assuming each arc has a unit capacity,
   each node v' has a flow supply of $p/2$ units and each node v'' has a flow demand of $p/2$ units;
   assign all nets with a unit flow to crossbar i and remove them from B_i to obtain B_{i+1};
   rof; (Fig. 7 & 8)
5. Assign all nets in B_K to crossbar K.

4 Experimental Results

We carried out some experiments to test the performance of our performance-driven routing algorithm. In the experiments, we assumed that the chips are arranged linearly in one row while the crossbars are arranged linearly in another row as in Fig 1. And we took $d_i(n)$, the cost of routing net n via crossbar i, as the sum of the horizontal distance between chip u and crossbar i, and the horizontal distance between chip v and crossbar i when net n's two terminals are in chips u and v.

We used ten different board configurations, and for each of them we randomly generated ten routing problem instances. In each problem instance generated, we assumed that 5% of the nets have some routing constraints that restrict which crossbars they can route through, and the routing cost of these nets plus another 25% of nets is to be optimized. Our performance-driven routing algortihm were successful in routing all instances satisfying all routing constraints. When previous algorithms were used to route the same instances, almost all routing contraints were violated. We also compared the optimized routing cost of the given 30% nets to the unoptimized value. The results are shown in Table 1. It can be seen that the optimized routing costs are consistently less than one third of the unoptimized ones.

Table 1: Experimental results.

Chips per board	Crossbars per board	Pins per subset	No. of Nets (avg)	$\frac{\text{optimized routing cost}}{\text{unoptimized routing cost}}$
16	8	16	920	0.31
16	16	8	920	0.30
16	25	6	1076	0.31
16	32	4	920	0.31
20	10	16	1457	0.32
20	20	8	1457	0.32
20	30	6	1641	0.32
25	12	8	1090	0.31
25	20	6	1374	0.31
25	30	4	1374	0.31

5 Conclusions

We have presented a performance-driven routing algorithm for the board-level routing problem in logic emulation. It can handle routing constraints on individual critical nets which cannot be handled by previous algorithms [10,11]. Also, it reduces the delay of the final routing solution by minimizing the routing cost for a set of nets on critical paths. And we have obtained very good experimental results using the performance-driven routing algorithm.

It was shown in [10, 11] that all routing problem instances with two-terminal nets only are feasible when the I/O-pin subset size is even and there is no routing constraint on individual nets. We note that our algorithm can always find a feasible routing solution and with a better delay than the previous algorithms in this case.

References

[1] S.D. Brown, R.J. Francis, J. Rose, and Z.G. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1992.

[2] S. Trimberger (edited), *Field-Programmable Gate Array Technology*, Kluwer Academic Publishers, 1994.

[3] J. Varghese, M. Butts, and J. Batcheller, "An Efficient Logic Emulation System", *IEEE Transactions on VLSI, Vol. 1, No. 2*, p.171-174, June *IEEE Transactions on VLSI, Vol.*

1, No. 2, p.171-174, June 1993.

[4] M. Slimane-Kadi, D. Brasen, and G. Saucier, "A Fast-FPGA Prototyping System that Uses Inexpensive High-Performance FPIC," *ACM/SIGDA International Workshop on Field-Programmable Gate Arrays*, Feb 1994.

[5] L. Maliniak, "Multiplexing Enhances Hardware Emulation," *Electronic Design*, p.76-78, Nov. 1992.

[6] S. Walters, "Computer-Aided Prototyping for ASIC-based Systems," *IEEE Design and Test*, p.4-10, June 1991.

[7] K. Yamada, H. Nakada, A. Tsutsui, and N. Ohta, "High-Speed Emulation of Communication Circuits on a Multiple-FPGA System," *ACM/SIGDA International Workshop on Field-Programmable Gate Arrays*, Feb 1994.

[8] N.C. Chou, L.T. Liu, C. K. Cheng, W.J. Dai, and R. Lindelof, "Circuit Partitioning for Huge Logic Emulation Systems," *Proc. of the 31st ACM/IEEE Design Automation Conference*, p.244-249, 1994.

[9] H.Q. Liu, and D. F. Wong, "Network Flow Based Multi-Way Partitioning with Area and Pin Constraints," *Proc. of the ACM International Symposium on Physical Design*, April 1997.

[10] P.K. Chan, and M.D.F. Schlag, "Architectural Tradeoffs in Field-Programmable-Device-Based Computing Systems," *IEEE Workshop on FPGAs for Custom Computing Machines*, p.138-141, April 1993.

[11] W.K. Mak, and D.F. Wong, "On Optimal Board-Level Routing for FPGA-based Logic Emulation," *Proc. of the 32nd ACM/IEEE Design Automation Conference*, p.552-556, 1995.

[12] W.K. Mak, and D.F. Wong, "Board-Level Multi-Terminal Net Routing for FPGA-based Logic Emulation," *Proc. of International Conference on Computer Aided Design*, p.339-344, 1995.

[13] M.S. Bazaraa, J.J. John, and H.D. Sherali, *Linear Programming and Network Flows*, John Wiley & Sons, 1990.

[14] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, 1993.