Multi-Terminal Net Routing for Partial Crossbar-Based Multi-FPGA Systems

Abdel Ejnioui
Center for Microelectronics Research
University of South Florida
Tampa, FL 33620
(813) 974-2395

ejnioui@cmr.usf.edu

N. Ranganathan
Dept. of Electrical & Computer Engineering
The University of Texas at El Paso
El Paso, Texas 79968
(915) 747-6622

ranganat@ece.utep.edu

ABSTRACT

Multi-FPGA systems are used as custom computing machines to solve compute intensive problems and also in the verification and prototyping of large circuits. In this paper, we address the problem of routing multi-terminal nets in a multi-FPGA system that uses partial crossbars as interconnect structures. First, we model the multi-terminal routing problem as a partitioned bin packing problem and formulate it as an integer linear programming problem where the number of variables is exponential. A fast heuristic is applied to compute an upper bound on the routing solution. Then, a column generation technique is used to solve the linear relaxation of the initial master problem in order to obtain a lower bound on the routing solution. This is followed by an iterative branch-and-price procedure that attempts to find a routing solution somewhere between the two established bounds. In this regard, the proposed algorithm guarantees an exact routing solution by searching a branch-and-price tree. Due to the tightness of the bounds, the branch-and-price tree is small resulting in shorter execution times. Experimental results are provided for different netlists and board configurations in order to demonstrate the algorithm's performance. The obtained results show that the algorithm finds an exact routing solution in a very short time.

Keywords

FPGA routing, FPGA architecture, layout synthesis, interconnect optimization, branch-and-price, integer programming.

1. INTRODUCTION

Multi-FPGA systems play an important role in the verification [5, 28] and the rapid prototyping of large circuits [7, 17, 24]. In addition, they are often used as custom computing machines to solve compute-intensive problems [3, 6, 12, 29]. A multi-FPGA system consists of several FPGA chips placed on a board and connected together through a routing structure. The routing architectures influence the cost and the performance of the emulated and prototyped circuits.

Routing architectures can be either direct or indirect. In the former, the FPGA chips connect directly to each other through a fixed set of wires [16, 29, 30], while in the latter some routing chips are used to interconnect the FPGA chips on the board [7, 17, 28]. In direct routing architectures, the delay is not uniform and routing completion can be difficult to achieve [5, 28]. Consequently, many FPGA chips may be used solely for routing, which diminishes logic utilization [28]. These factors tend to complicate the design of the placement and routing software. On the other hand, indirect architectures rely on special purpose chips used solely for routing. The first structure proposed for these routing chips was the full crossbar [22]. Later, the folded-Clos structure was proposed to implement these routing chips [7]. The full crossbar guarantees complete routing and uniform delay. However, its area cost is very high since it grows squarely with the total pin count. A better alternative is to use partial crossbars as routing architectures [5, 28]. A set of small full crossbars makes a partial crossbar. A partial crossbar size grows linearly with the total pin count and provides uniform delay. In addition, hierarchical expansion is possible by recursively interconnecting a set of partial crossbars to build multiple board systems [28].

The organization of this paper is as follows: Section 2 describes previously proposed approaches to solve this problem. A brief description of the multi-FPGA system architecture used to solve this problem is presented in Section 3. Section 4 introduces several concepts used to model the problem as well as its formulation. A detailed explanation of the proposed algorithm is presented in Section 5. In Section 6, we describe some implementation details and present the experiments. Finally, conclusions are drawn in Section 7.

2. RELATED WORK

In [8], the authors proposed a folded Clos network of interconnections as a routing structure. This network consisted of a set of small crossbars. In order to route all the nets, the Clos networks had to be configured through a control algorithm. In [28], a direct greedy heuristic, which does not reconfigure the interconnect, was proposed to route the nets across a partial crossbar interconnection structure. The heuristic routes most nets but does not guarantee routing completion. Later, a routing algorithm was proposed to route the nets across multiple small crossbars [19, 20]. This algorithm runs in polynomial time for two-terminal nets only. However, the authors proved that the routing problem is NP-complete for multi-terminal nets [21]. Routing multi-terminal nets was achieved by decomposing these nets into two-terminal nets and then applying the two-terminal net routing algorithm [19, 20]. This approach routes most nets, which may result in using more pins that necessary. In high routing traffic, it is possible to exhaust all available pins, which may results in incomplete routing.

In [18], the authors proposed two fast heuristic solutions and an exact solution. The heuristics were based on the one proposed in [28] but with an added feature that tries to balance the nets across all the crossbars. Both heuristics select candidate nets for routing by using a gain function. This function is a linear combination of two parameters where the first and the second parameter represent the number of available pins in a crossbar for a given logic chip and the total number of available pins in the same crossbar across all the logic chips respectively. The first and the second parameters are weighted by two adjustment factors, α and β. However, there was no indication on how to set these two parameters. The exact solution was based on a non-linear programming formulation of the routing problem. Since this solution is time expensive, it was applied only when the heuristic solution failed to complete routing.

In this paper, we propose an exact solution based on an integer linear programming formulation. Contrary to previously proposed approaches, our approach provides theoretically established bounds on the routing solution. The tightness of these bounds is exploited to minimize the search space of the algorithm. In addition, additional bounds are used to improve the performance of the proposed algorithm. The algorithm shows a fairly high degree of stability when tested on different board configurations. The experimental results show that the performance of our algorithm is better than previously published approaches.

3. PARTIAL CROSSBAR MODEL FOR MULTI-FPGA SYSTEMS

In this section, we describe the architecture of partial crossbar-based multi-FPGA systems. In a crossbar-based multi-FPGA system, a set of FPGA chips called the logic chips is placed on a board. These chips are configured to implement logic circuitry. The IO signals between these logic chips are routed through several interconnection chips called the routing chips. Each routing chip is a full crossbar and is connected only to logic chips. No routing chip is connected to another routing chip. The IO pins of each logic chip are partitioned into subsets of pins. Each subset of pins contains the same number of pins. The pins on a routing chip are connected to the pins belonging to the same pin subset on each logic chip. Thus, the number of routing chips on the multi-FPGA system board is equal to the number of pin subsets on each logic chip. The number of pins on a routing chip is the number of pins over all the routing chips belonging to the same pin subset. Figure 1 shows an example of a multi-FPGA system with three logic chips and two routing chips. The pins of each logic chip are divided into two subsets: A and B. Each subset contains two pins. For instance, subset A contains pins A_1 and A_2 in each logic chip. Every pin in the A subset on each logic chip is connected to a pin on the first routing chip. Similarly, every pin in the B subset on each logic chip is connected to a pin on the second routing chip. Since there are two pin subsets, there are two routing chips.

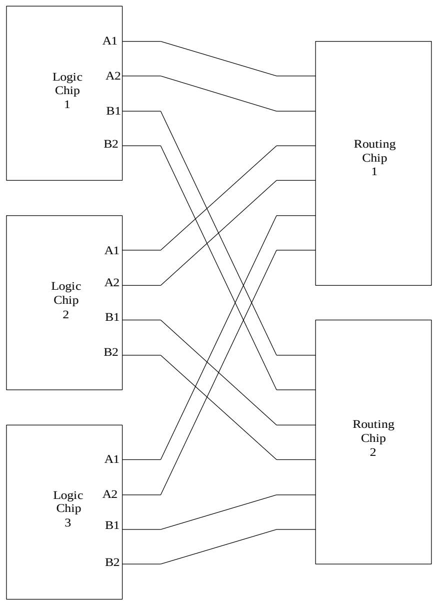

Figure 1 An example of a partial crossbar-based FPGA system.

To map a large circuit on a FPGA based system for emulation or prototyping, the circuit is partitioned such that each partition is packed into a logic chip. After partitioning, the nets between the logic chips must be routed through the routing chips in order to make the circuit functional. As long as there are available pins on a routing chip, the nets can be routed between the logic chips.

4. PROBLEM SPECIFICATION AND FORMULATION

In this section, we introduce some concepts related to the multi-terminal net routing problem on partial crossbars. Then, we describe the problem and model it as a partitioned bin packing problem.

4.1 Problem Specification

We will refer to the chips that contain logic circuitry as the logic chips while those that are used solely for routing as the routing chips. Let C be the number of logic chips on a board. The pins on each logic chip are divided into pin subsets. Let K and m be the number of pin subsets per logic chip and the number of pins per subset respectively. A routing chip will have mC pins. There will be K routing chips per board. A board configuration is represented by $B = (m, K, C)$. In Figure 1, m is 2, K is 2, and C is 3. Let $N = \{n_1, n_2, \dots n_{|N|}\}$ be a netlist. A net n in N is represented by a C-bit binary vector, $(b_1, b_2, \dots b_C)$, where b_j is 1 when n has a terminal on logic chip j and is 0 otherwise. The multi-terminal net routing problem requires that the terminals of a net must be assigned to pins belonging to the same pin subset. In this case, the net will be routed only through one routing chip. For example, a net $n = (1, 1, 1)$ requires one terminal on each logic chip. It can be assigned, for instance, to the pin subset A on the board shown in Figure 1. This means that it can use any pin from pin subset A on each logic chip and subsequently can be routed through only one routing chip. Routing a given net through one routing chip guarantees uniform delay for all signals in the net. Let S_i be a given pin subset on logic chip i and S be the union of such pin subset over all logic chips, $S = \bigcup_{i=1}^{C} S_i$.

For example in Figure 1, set $A = \bigcup_{i=1}^{3} A_i$.

Definition 1: An assignment of a net $n = (b_1, b_2, \dots b_C)$ to a pin set S is called a *legal assignment* iff for each $b_j = 1$ where $1 \le j \le C$, there is an available pin in pin subset S_j.

Given a board configuration $B = (m, K, C)$ and a netlist N, this paper addresses the problem of how to find a legal assignment of each net in N to some subset pin in B. We call this problem *the net routing problem for partial crossbar-based FPGA systems*.

4.2 Problem Formulation

In this section, we present the formulation of the routing problem of multi-terminal nets in crossbar-based multi-FPGA systems. As mentioned in Section 4.1, a net n is represented by a C-bit vector, $(b_1, b_2, \dots b_C)$, where b_j is 1 when n has a terminal on logic chip j and is 0 otherwise.

Definition 2: The weight of a net $n = (b_1, b_2, \dots b_C)$ is $weight(n) = \sum_{i=1}^{C} b_i$ where $b_i \in \{0, 1\}$.

The nets can be either two-terminal or multi-terminal. A t-terminal net has a weight equal to t.

Definition 3: The magnitude of a net $n = (b_1, b_2, \dots b_C)$ is $mag(n) = \sum_{i=1}^{C} 2^i \times b_i$.

The magnitude of a net n is useful in distinguishing n from other nets whose weights are equal to that of n. For example two nets $n_1 = (1, 1, 0)$ and $n_2 = (1, 0, 1)$ have the same weight 2. But their magnitudes are different, $mag(n_1) = 6$ and $mag(n_2) = 10$. It is possible for a netlist to contain nets with identical weights but whose magnitudes are different.

Definition 4: A net type T_i represents a set of nets in N whose magnitudes are equal.

There will be many net types in a given netlist. Each possible magnitude value determines a net type. Let $T = \{T_1, T_2, \dots, T_{|T|}\}$ be the set of net types in N. It is obvious that $|T| \le |N|$. Let t_i be the number of times the net n occurs in N. We associate an "object" O_i with each "net type" T_i, characterized by $(b_1, b_2, \dots, b_C)$, $mag(T_i)$, $weight(T_i)$ and t_i. For clarity, the terms "object" and "net type" will be used interchangeably in the rest of this paper since they represent the same entity.

Let each pin set be represented by a bin. If a set $S = \bigcup_{i=1}^{C} S_i$ is represented by a bin B, then B's total capacity, denoted by $cap(B)$, is $|S|$. Since each set consists of several subsets, the bin will be divided into partitions. Each pin subset S_i is represented by a partition P_i in B. A partition P_i will have as many empty slots as pins in subset S_i. For example, pin set A in Figure 1 can be represented by a bin B. Since A consists of three subsets $\{A_1, A_2, A_3\}$, B will have three partitions $\{P_1, P_2, P_3\}$. Each partition P_i will have two available slots since each subset A_i has two pins for $1 \le i \le 3$.

Definition 5: A bin $B = (|P_i|, C) = (|S_i|, C) = (m, C)$ represents a pin set S where C is the number of partitions in B and $|P_i|$ is the number of available slots in each partition, for $1 \le i \le C$.

Note that (i) C is also the number of logic chips on the board and (ii) every subset S_i in S has the same cardinality m. The capacity of bin B is $cap(B) = |S| = \sum_{i=1}^{C} |S_i| = C \text{ x } |S_i| = m \text{ x } C$. As mentioned in Section 4.1, the problem of routing the nets on a board is equivalent to finding a legal assignment for each net to a pin set. Since each pin set S can be represented by a bin B and each net type T_i in T can be represented by an object O_i, assigning a net type T_i to S is equivalent to packing the object O_i into B. It is possible to pack more than one object into B if $cap(B)$ is large enough.

Definition 6: A *legal packing* of an object $O_i = (b_1, b_2, \dots, b_C)$ into a bin B is the placement of a unit of weight 1 into an empty slot of the partition P_j if $b_j = 1$ for $1 \le j \le C$.

Note that each partition in B has m slots. In order to pack more than one object into a bin, each object must be placed through a legal packing.

Definition 7: A bin B is *feasible* iff each object in B was placed in B through a legal packing.

In a legal packing, the total weight of the objects packed in a bin will always be less than or equal to the bin's capacity. Given a list of objects and a partitioned bin, the number of possible ways in which any set of objects form the list of objects can be legally packed into the bin is quite large. This means that the number of feasible bins is very large. Let Ω be the set of all possible feasible bins. Given a board configuration $B = (m, K, C)$ and a net type set T, the problem of *net routing for partial crossbar-based FPGA systems* is equivalent to packing the objects in T subject to the following constraints: (i) each packing of an object must be a legal packing, (ii) each bin must be a feasible bin, (iii) each object type must occur at least t_i times in the solution. Let a_{ij} be the number of object of type T_i in feasible bin j and the matrix $A = [a_{ij}]$ for $1 \leq i \leq |T|$ and $1 \leq j \leq |\Omega|$. We can formulate the *Net Routing Problem for Partial Crossbar-Based FPGA Systems* as follows:

$$f(x) = \min \sum_{j=1}^{|\Omega|} x_j$$

subject to: $a_{ij}x_j$ ³ t_i , $1 \leq i \leq |T|$, $1 \leq j \leq |\Omega|$.
x_j ³ 0, x_j integer.

The variable x_j represents the number of times the feasible bin j occurs in the solution. The $|T|$ constraints guarantee that there are at least t_i objects of types T_i in the solution. This is the *Integer Programming (IP)* formulation of this problem. It is clear that $|\Omega|$ is an exponential function of $|T|$, which makes the explicit generation of the columns of this IP impractical. Gilmore & Gomory [14, 15] introduced this formulation to solve the cutting stock problem, a well-known problem in the paper industry, which is characterized by an exponentially large number of columns.

5. PROPOSED SOLUTION

In this section, we describe the approach used to solve the IP formulation of the problem. We use a branch-and-price approach to locate the routing solution in a large search space represented by a binary tree. The routing solution is represented by $[x_j]$, $1 \leq j \leq |T|$, which is the number of times the feasible bin j occurs in the solution. A routing solution is routable if $\sum_{j=1}^{|T|} x_j \leq K$, where K is the number of pin subsets per logic chip. Our approach consists of four major parts. The first part represents a procedure to compute an upper bound on the routing solution. The second part represents a column generation technique performed at each node of the branch-and-price tree. The third part represents the branching rules used to add new nodes to the tree. The fourth part represents the strategy used to traverse the tree. We use the following notation to explain the algorithm:

z_{LB} = lower bound
z_{best} = best solution or upper bound
z_{ffd} = First Fit Heuristic solution
z_B = current solution at the current node of the branch-and-price tree

The outline of our branch-and-price approach is as follows:

```
1)  Let $z_{LB} = 0$
2)  Solve the problem instance using the First Fit Decreasing
3)    Heuristic
4)  Update the upper bound: $z_{best} = z_{ffd}$
5)  If $z_{best} \leq K$ then
6)      The problem instance is routable
7)      Stop
8)  Endif
9)  Solve the RMP at the current node using column generation
10) Update the lower bound $z_{LB}$
11) If $z_{LB} > K$ then
12)     The problem instance is not routable
13)     Stop
14) Endif
15) If the solution is integral then
16)     If the current solution $z_B \leq K$ then
17)         The problem instance is routable
18)         Stop
19)     Endif
20)     If $z_B < z_{best}$ then
21)         Update $z_{best}$
22)     Endif
23)     Backtrack to another node
24)     If the node backtracked to is the root node then
25)         The problem instance is not routable
26)         Stop
27)     Endif
28)     Goto step 9
29) Else
30)     If $z_B < z_{best} - 1$ then
31)         Find a maximal column $A_j$ in the solution of the
32)           current LP whose variable is fractional $x_j = \alpha$
33)         Add a left node to the tree in which the RMP to
34)           solve is the problem from the parent node with
35)           the additional constraint $x_j \leq \lfloor \alpha \rfloor$
36)         Add a right node to the tree in which the RMP to
37)           solve is the problem from the parent node with
38)           an additional constraint $x_j \geq \lceil \alpha \rceil$
39)         Branch to the next node
40)         Goto step 9
41)     Else
42)         Backtrack to another node
43)         If the node backtracked to is the root node then
44)             The problem instance is not routable
45)             Stop
46)         Endif
47)         Goto step 9
48)     Endif
49) Endif
```

In the following sections, we explain the important steps of the algorithm.

5.1 Description of the First-Fit Decreasing Heuristic

The second line in the pseudocode of the proposed approach solves the multi-terminal nets routing problem, modeled as partitioned bin packing problem, using the First-Fit Decreasing heuristic [9]. The objects are first sorted in non-increasing order of their weights before being packed in this order. An object is packed into the current bin if room is available. Otherwise an empty bin is added to the list of bins in which the current object will be packed. This process is repeated until every object in the list is packed in a bin. Every packing of an object in a partitioned bin is performed by a legal packing as described in Section 4.2. Initially, the number of bins obtained with this heuristic, z_{ffd}, is an upper bound on the routing solution.

5.2 Column Generation at a Tree Node

In this section, we explain the general version of the column generation performed at each node of the branch-and-price tree, which is represented by line 9 in the pseudocode of the proposed approach. We also formulate the pricing problem and explain how it relates to our column generation technique.

5.2.1 General Version of Column Generation

In [14], the authors proposed a simplex algorithm to solve large-scale IP problems by solving their linear programming relaxations (*LP*) through the generation of a limited number of columns. To circumvent the explicit generation of the columns, the authors proposed a modified version of the simplex method to implicitly generate a finite set of columns in order to improve the objective function [14, 15]. The size of this finite set is at most $|T|$ columns, which is clearly linear in the size of N. Note that a column from this set may occur more than once in the solution. The column generation problem was defined as an optimization problem whose solution is supposed to price out in order to enter the basis at each iteration of the simplex algorithm. The formulation of the column generation problem depends on the application. The formulation of our column generation problem is presented in Section 5.2.2. Column generation stops when no more improvement of the objective function can be obtained. We use the following notation to explain this simplex algorithm:

B^0 = Initial feasible basis (a $|T|$ x $|T|$ matrix)
B^{-1} = Inverse of current feasible basis (a $|T|$ x $|T|$ matrix)
c_B = unit row $|T|$-vector of costs of the basic variables
$t = (t_1\ t_2 \ldots t_{|T|})^T$ = column $|T|$-vector of the occurrences of each object type T_i
$b = B^{-1}$ x t = current basic solution
$\pi = c_B$ x B^{-1} = row $|T|$-vector of the simplex multipliers
$A_s = (a_1\ a_2 \ldots a_{|T|})^T$ = newly generated column $|T|$-vector

The outline of the algorithm is as follows:

1) Make B^0 the initial feasible basis.
2) Compute the current solution $b = B^{-1}$ x t
3) Compute the current simplex multipliers $\pi = c_B$ x B^{-1}.
4) Find a column A_s, using some generating algorithm, by maximizing the function $f(A_s) = \pi$ x A_s.
5) If $1 - f(A_s) \geq 0$ then stop: b is the optimal solution
   Else compute B^{-1} x A_s.
6) If B^{-1} x $A_s \leq 0$ then stop: the optimum is unbounded
   Else determine the pivot row, update the basis and goto step 2.

In our formulation, B^0 is the unit matrix whose diagonal entries are equal to m. Note that all the coefficients in our IP formulation are equal to 1. The IP formulation shown in Section 4.2 is called the *Master Problem (MP)*. In each iteration, the simplex algorithm keeps a subset of columns from all possible columns. This is the *Restricted Master Problem (RMP)*. In step 2 and 3, the current solution and the simplex multipliers are computed. In step 4, a column must be generated by maximizing its objective function. This optimization problem is called the *Pricing Problem (PP)*. The column prices out if its objective function is greater than 1 in step 5. If the optimum is bounded in step 6, a pivot row is determined by the *Minimum Ratio Test* [14]. Then the basis is updated and the algorithm enters a new iteration. The algorithm stops when no column can price out to enter the pivot. At the end, we end up with only a subset of all possible columns whose size is less or equal to $|T|$ if an optimum is found.

5.2.2 Formulation of the Pricing or the Column Generation Problem

In our IP formulation, a column represents a feasible bin. Let $A_s = (a_1\ a_2 \ldots a_{|T|})^T$ be a column whose components a_i are unknown variables. Note that each a_i, $1 \leq i \leq |T|$, represents the number of objects of type T_i packed into a bin. When an object of type T_i is packed into a bin, each bit whose value is equal to 1 in its bit vector is assigned to an unoccupied slot in the partition P_j of the bin. A partition can hold at most m bits whose respective values is each equal to 1. This means that the number of bits from different objects assigned to the slots of a given partition must not exceed m. Let an object of type $T_i = (b_{i1}, b_{i2}, \ldots, b_{iC})$ and b_{ij} be the jth bit in the vector of this object of type T_i. Let $\pi = (\pi_1, \pi_2, \ldots \pi_{|T|})$ be the simplex multipliers as defined in Section 5.2.1. The column generation problem, also known as the pricing problem (PP), is defined as follows:

$$f(a) = \max \sum_{i=1}^{|T|} p_i a_i$$

subject to: $b_{ij}a_i$ £ m, $1 \leq i \leq |T|$, $1 \leq j \leq C$.
a_i ³ 0, a_i integer.

The constraints allow at most m bits in each bin partition where each bit is equal to 1. The PP in our formulation is an integer linear program. The solution of this problem corresponds to step 4 of the simplex algorithm described in Section 5.2.1. In this formulation, $f(a)$ represents $f(A_s)$ in the simplex algorithm of Section 5.2.1.

5.2.3 Solution of the Restricted Master Problem

At each node of the tree, a RMP is solved using column generation as described in Section 5.2.1. At the root node, the first RMP is optimized by relaxing it to an LP. It is not necessary to solve the RMP linear relaxation to optimality to obtain a lower bound [25]. Note that the column generation approach described in Section 5.2.1 terminates when no more columns can be generated, namely when the reduced cost of the last generated column $1 - f(A_s) \geq 0$. Stopping the generation of the columns in this fashion triggers a tailing-off effect, which is characterized by a decrease in the optimization rate of the objective function at the final iteration stages of the simplex approach [4]. Hence, a large number of iterations are needed to reach optimality. To minimize computational effort, one can opt to abort the column generation process early and work with bounds of the final objective function value of the LP. Let z_{IP}, z_B, c_{min} be the IP objective function value, the optimal value of the restricted master problem at the root node, and the reduced cost of the column generated in the pricing problem respectively.

Proposition 1: If $\lceil z_B \rceil = \lceil z_B / (1 - c_{min}) \rceil$, then $z_{IP} \geq \lceil z_B \rceil$.

The proof of this proposition has been presented in [25, 26]. It is a direct consequence of the bounds proposed in [13]. This proposition is used to terminate the optimization of the LP at the current node earlier. The equality of the if clause in proposition 1 is checked at every iteration of the simplex algorithm. The simplex stops as soon as it becomes true. This test is performed at each iteration of the simplex algorithm executed in line 9 of the pseudocode of the proposed approach.

5.3 Description of the Branch-and-Price Approach

In this section, we explain the contents of line 30 through 40 of the pseudocode of the proposed approach. We also show how the pricing problem is modified to account for the branching rules.

5.3.1 Branch-and-Price Rules

The column generation technique used in [14] produces an optimal solution to the LP relaxation of the RMP. However, the solution is not necessarily integral. Note that a solution is integral if every variable in the solution is integer. Recently, many researchers addressed this issue by introducing branching strategies similar to the ones used in branch-and-bound approaches [2, 10, 11, 23, 25-27]. In the branch-and-price approach, the RMP is solved using column generation at the root node of a tree [4]. If the solution is not integral, specific rules are used to bound a subset of the variables from below and above in order to branch out to two new nodes. The left node represents the RMP with an additional constraint, the upper bound used for left branching. The right node is similar except that the additional constraint is the lower bound used for right branching.

After an LP is optimized at a node, branching takes place if $z_B \leq (z_{best} - 1)$. This is shown in line 30 of the pseudocode listed in Section 5. In [27] the authors proposed a branching rule in which a maximal column, whose variable is fractional, is selected for branching. Note that a variable is fractional if it is both real and non-integer. A maximal column is a feasible bin whose unfilled capacity is less than or equal to the smallest weight of any object type in *T*. This rule guarantees the elimination of a fractional variable if one exists in the LP solution at a given node in the tree. Let A_j be a maximal column whose variable $x_j = \alpha$ is fractional. We branch (i) on the left by adding the constraint $x_j \leq \lfloor\alpha\rfloor$ to the LP problem of the current node and placing this new problem in the left node and, (ii) on the right by adding the constraint $x_j \geq \lceil\alpha\rceil$ to the LP problem of the current node and placing this new problem in the right node. It is obvious that the number of constraints increases as we descend the tree. This increase in the number of constraints narrows the search space for the routing solution.

5.3.2 Modification of the Pricing Problem

A branching rule must not exacerbate the complexity of the PP. In addition, the modification of the PP at each node by the branching rule must not result in the generation of infeasible columns [26]. Since each branching decision adds a constraint to the newly created LP, the PP is slightly modified to reflect this change in the LP. As mentioned in Section 5.3.1, each fractional variable x_j represents a maximal column. When solving the PP, we must avoid the generation of a maximal column associated with any left branch constraint. We call these columns *forbidden columns.* This is equivalent to replacing each left branch constraint $x_j \leq \lfloor\alpha\rfloor$ by $x_j = 0$. The PP is not affected with regard to the right branch constraints since the constraints are $\geq$ constraints in the original MP [26]. After solving a PP, we compare the solution column to each forbidden column in the column pool. Assume the object of type T_i is the smallest weight object in the solution column. If a match is found between the solution column and a forbidden column, then we modify the PP by eliminating from it the variable a_i and resolve it again. This modification guarantees that the newly generated column is optimal only with regard to the new PP. This process is repeated until the solution column of the modified PP does not match any forbidden column in the column pool. We found that the probability of generating already existing forbidden columns tend to increase slightly as the algorithm solves RMPs located deeper in the branch-and-price tree.

5.4 Description of the Tree Traversal Strategy

In this section, we explain the strategy adopted to traverse the tree and how it relates to the process of updating the lower and upper bounds of the routing solution. The efficiency of the algorithm depends heavily on updating these two bounds, which is performed at specific instants during the tree traversal.

5.4.1 Tree Traversal

When the algorithm is searching the solution space, it has to traverse the tree in some fashion. Since the modifications introduced in the formulation of the PP to avoid regenerating forbidden columns affect primarily the left nodes in the tree, we favor visiting the left nodes first as we descend the branch-and-price tree. The closest strategy that mimics this scheme is to traverse the tree in a depth-first manner. In [26], the authors present evidence showing that this strategy is very suitable for the branching rules used in the algorithm.

5.4.2 Updating the Lower Bound

Let z_{LB} and z_B be the IP lower bound and the objective function of the LP at the current node respectively. If the current node is the root node, then $z_{LB} = \lceil z_B \rceil$. Otherwise if $z_B > z_{LB}$ at every active node of the tree, then $z_{LB} = \lceil z_{Bmin} \rceil$ where z_{Bmin} is the smallest z_B in the branch-and-price tree. An active node is a node whose LP was already solved. At any time, there may be tree nodes whose RMPs were already formulated but have not yet been solved. Updating the lower bound in this manner brings it closer to the upper bound.

5.4.3 Updating the Upper Bound

Let z_{best} be the best IP solution found so far. When a new integer solution z_B is found, $z_{best} = z_B$ if $z_B < z_{best}$. Note that $z_{best} = z_{ffd}$ initially. As better integer solutions are found deeper in the tree, z_{best} is updated accordingly to reflect how the upper bound moves closer to the lower bound z_{LB}.

6. EXPERIMENTAL RESULTS

The algorithm was implemented in C++ on a Sun Ultra 1 workstation. The optimization software EXPRESS-MP was used to solve the PP [1]. To solve each PP, the matrix of the PP was constructed and packed in a format recognizable by XOSL, which is the optimization library of EXPRESS-MP. Appropriate library routines were called to solve the PP. When the solution is returned by XOSL, it is passed back to our algorithm. Two files are used as input to the algorithm: a board configuration file and a netlist file. The algorithm produces an output file describing the routing results.

	Logic Chips			Crossbar	Chips
Configuration	Chips/board	Pins/chip	Pins/subset	Chips/board	Pins/chip
1	8	96	8	12	64
2	8	96	16	6	128
3	8	96	32	3	256

Table 1 Test Configurations

Config.	Netlist	Nets	Types	Config. Cross-bars	Z_{LB}	z_{best}	Routing Ratio (%)	Pin Sat. (%)	Solved LPs	CPU Time (sec)
1	1	221	112	12	13	13	99	81.25	1	138
	2		115		0	12	100	88.02	0	7
	3		118		13	14	94	75.04	1	118
	4		119		0	12	100	88.02	0	9
	5		121		0	12	100	88.02	0	14
2	6	221	115	6	0	6	100	88.02	0	9
	7		118		0	6	100	88.02	0	8
	8		119		7	7	99	75.44	1	484
	9		119		0	6	100	88.02	0	8
	10		121		0	6	100	88.02	0	11
3	11	221	111	3	4	4	98	66.01	1	450
	12		115		0	3	100	88.02	0	9
	13		120		0	3	100	88.02	0	11
	14		120		4	4	99	66.01	1	220
	15		128		0	4	100	88.02	0	15

Table 2 Experimental Results

Table 1 shows the configurations used to test the algorithms. The parameters used in the test configurations fall within the range of the present technology. Five netlists of different sizes were generated to test the algorithm on each configuration. As in [5, 18, 20], each netlist contains 33% 2-terminal, 38% 3-terminal, 19% 4-terminal and 10% 5-terminal nets.

Table 2 shows the results of the experiments. The first column shows the configuration used for routing. Column 2, 3, and 4 show the netlist, the number of nets in the list, and the number of net types in that list. Column 5 shows the number of routing chips or crossbars in the configuration, while columns 6 and 7 show the lower and the upper bounds as defined in Section 5.4. Column 8 shows the routing ratio, which is the number of routed nets over the total nets in a given netlist. Column 9 shows the average pin saturation of all routing chips. Pin saturation is defined as the ratio of pins used for routing in a routing chip over all the available pins in that routing chip. In our formulation, this is equivalent to the ratio of the number of full slots in a feasible bin over all the available slots in a bin. Column 10 and 11 show the number of linear programs (LPs) solved to reach the solution and the CPU time in seconds.

It is clear that as the size of the netlists increases, it becomes difficult to route each net in the list. The sizes of the netlists used in the experiments were chosen to cover the two possibilities where the algorithm may or may not route the nets. We want to show how the algorithm determines routing success or failure for a given netlist. As the packing density of the feasible bins increases, so does the pin saturation. This in turn allows the routing of larger netlists. Eventually, the size of a netlist will reach a given size for which the number of needed feasible bins to pack each net exceeds the number of routing chips in the configuration. At this point, some nets will fail to route. It is obvious, from the experimental results shown in Table 2, that the branch-and-price tree is very small consisting of only one node. This is due to the tightness of the lower and the upper bounds. The CPU time is very small when a routing solution is found by the heuristic step in the algorithm. However it tends to increase, within reasonable limits, when the algorithm tries to solve an LP. It takes $2|T|$ to $3|T|$ iterations on the average to solve an LP in a given node where $|T|$ is the cardinality of the type set T.

When compared to our algorithm, previously proposed heuristics for this problem do not offer any a priori known bounds on their routing solutions [5, 18, 20]. This uncertainty can be exacerbated by the use of weight coefficients in optimized functions, which can be very sensitive to a variety of inputs. The weight coefficients have to be periodically adjusted to fine-tune the performance of the heuristics. The results of those heuristics can be explained only after extensive experimentation on large sets of data. The bounds of our approach are significant in the sense that they are very tight. This tightness minimizes the search space of the algorithm, which is represented by a branch-and-price tree. Hence, the short execution times of the algorithm. The only proposed exact solution can be very time consuming and is applied only when the heuristic failed to route the nets [18]. Although the formulation in [18] is polynomial in the size of the board configuration, it turns out to be so large that commercial software was inadequate for its implementation. The authors opted to build a tailored solution. Our approach manipulates a $|T|$ x $|T|$ matrix, which is linear in the size of the netlist, regardless of the board configuration size.

In addition, the previously proposed heuristics tend to be sensitive to the architectural parameter *m*. The quality of their routing solutions improves as *m* increases. In [18], *m* was suspected to have an impact on the performance of the proposed routing algorithms. Our algorithm display a good stability in the face of a wide rang of values for *m*. However, the performance of our algorithm may worsen somewhat if *m* is set to an extremal value such as $m = 1$. When $m = 1$, the ratio $\rho = \max\{weight(T_i): T_i \in T\} / mC = 5 / mC$ increases. This is equivalent to increasing the weights of the objects with regard to the bin capacity. It is well documented in the literature that when the object weights increase to half or more the bin capacity, the solution quality in the general version of the bin packing problem produced by both the First-Fit and Best-Fit Decreasing heuristics deteriorates [9]. Since $C = 8$ in our experiments, $\rho = 5 / 8$, which is greater than 0.5. In this case, $|z_{LB} - z_{ffd}|$ will be somewhat larger, which in turn increases the size of the branch-and-price tree. A simple, yet effective way, to remedy this ensuing abnormal behavior of the algorithm is to merge several pin subsets and consider them as one pin subset. This has the effect of increasing the value of *m*. The modeling of the problem will not be affected in any way. Once $m > 1$, the algorithm resumes its fast execution. This extreme case is only of theoretical interest. Most practical architectures use values of *m* that are much higher than 1. Our algorithm shows a consistent behavior regardless of the configuration. This makes it a suitable experimental tool for the synthesis of architecturally efficient board configurations.

7. CONCLUSION

In this paper, we examined the multi-terminal net routing problem of crossbar-based multi-FPGA systems. We proposed an efficient exact algorithm to solve this problem. This algorithm guarantees bounded routing solutions. The bounds used allow this algorithm to handle comfortably problems of reasonable size within acceptable CPU times. In addition, it produces consistent results without being excessively sensitive to the architectural parameter *m*. This algorithm can be integrated as a tool with other layout tools targeted towards the partial crossbar architecture where it can be used as a fast router to estimate the routability of a design. It can also be coupled with a global placement and partitioning tool to guide the partitioning tool such that the partitioned circuit is routable.

REFERENCES

[1] XOSL: EXPRESS-MP optimization subroutine library. Ed. 1998, Leamington Spa, U.K: Dash Associates Ltd.

[2] Anbil, R., Tanga, R., and Johnson, E.L., *A global optimization approach to crew scheduling*, 1991, Georgia Institute of Technology: Atlanta.

[3] Arnold, J.M., Buell, D.A., and Davis, E.G. Splash 2. In *Symp. on Parallel Algorithms and Architectures,* (1992), 316-322.

[4] Barnhart, C., *et al.*, *Branch-and-price: column generation for solving huge integer programs*, in *Mathematical Programming: State of the Art*, Birge, J. R., and Murty, K. G., Editors. 1994, International Symposim on Mathematical Programming.

[5] Butts, M., Batcheller, J., and Varghese, J. An efficient logic emulation system. In *Int. Conf. Computer Design,* (Oct. 1992), 138-141.

[6] Casselman, S. Virtual computing and the virtual computer. In *Proc. Workshop on FPGAs for Custom Computing Machines,*(1993), 43-48.

[7] Chan, P.K., Schlag, M., and Martin, M. BORG: A reconfigurable prototyping board using field-programmable gate arrays. In *First Int. Workshop Field-Programmable Gate Arrays,* (Feb. 1992), 47-51.

[8] Chan, P.K. and Schlag, M.D.F. Architectural tradeoffs in field programmable device based computing systems. In *Workshop on FPGA's Custom Computing Machines,*(Apr. 1993), 152-156.

[9] Coffman, E.G., Garey, M.R., and Johnson, D.S., *Approximation algorithms for bin packing: A survey*, in *Approximation algorithms for NP-hard problems*, Hochbaum, D. S.,Editor. 1997, PWS Publishing Comapny: Boston, MA. p. 46-93.

[10] Desrochers, M., Desrosiers, J., and Solomon, M., A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research* 40 (1992), 342-354.

[11] Desrochers, M. and Soumis, F., A column generation approach to the urban transit crew scheduling problem. *Transporations Science* 23 (1989), 1-13.

[12] Drayer, T.H., *et al.* MORPH: A modular and reprogrammable real-time processing hardware. In *Symp. on FPGAs for Custom Computing Machines* ,(1995), 11-19.

[13] Farley, A.A., A note on bounding a class of linear programming problems, including cutting stock problems. *Operations Research* 38 (1990), 922-923.

[14] Gilmore, P.C. and Gomory, R.E., A linear programming approach to the cutting stock problem. *Operations Research* 9 (1961), 849-859.

[15] Gilmore, P.C. and Gomory, R.E., A linear programming approach to the cutting stock problem: Part II. *Operations Research* 11 (1963), 863-888.

[16] Gokhale, M., *et al.*, Building and using a highly parallel programmable logic array. *Computer* 24, (Jan. 1991), 81-89.

[17] Lewis, D.M., *et al.* The Transmogrifier-2: A 1 million gate

rapid prototyping system. In *Int. Symp. on FPGAs*, (1997), 53-61.

[18] Lin, S.S., Lin, Y.J., and Hwang, T.T., Net assignment for the FPGA-based logic emulation system in the folded-clos network structure. *IEEE Trans. on CAD* 16, 3 (Mar. 1997), 316-320.

[19] Mak, W.K. and Wong, D.F., Board-level multiterminal net routing for FPGA-based logic emulation. *ACM Trans. on Design Automation of Electronic Systems* 2, 2 (Apr. 1997), 151-157.

[20] Mak, W.K. and Wong, D.F., On optimal board-level routing for FPGA-based logic emulation. *IEEE Trans. on CAD* 16, 3 (Mar. 1997), 282-289.

[21] Mak, W.K. and Wong, D.F. On optimal board-level routing for FPGA-based logic emulation. In *DAC*, (1995), 552-556.

[22] Masson, G.M., Gingher, G.C., and Nakamura, S., A sample of circuit switching networks. *IEEE Computer* (June 1979), 32-48.

[23] Savelsberg, M., A branch-and-price algorithm for the generalized assignment problem. *Operations Research* 4, 6 (Dec. 1997), 831-841.

[24] Van Den Bout, D.E., *et al.*, AnyBoard: An FPGA-based, reconfigurable system. *IEEE Design and Test of Computers* (Sept. 1992), 21-30.

[25] Vance, P., *et al.*, Solving binary cutting stock problems by column generation and branch-and-bound. *Computational Optimization and Applications* 3, 2 (May 1994), 111-130.

[26] Vance, P.H., *Branch-and-price algorithms for the one-dimensional cutting stock problem*, . Jul. 1996, Auburn University: Auburn, Alabama.

[27] Vanderbeck, F. and Wolsey, L.A., An exact algorithm for IP column generation. *Operations Research Letters* 19, 4 (Oct. 1996), 151-159.

[28] Varghese, J., Butts, M., and Batcheller, J., An efficient logic emulation system. *IEEE Trans. VLSI Syst.* 1, 2 (June 1993), 171-174.

[29] Vuillemin, J.E., *et al.*, Programmable active memories: Reconfigurable systems come of age. *IEEE Trans. VLSI Syst.* 4, 1 (Mar. 1996), 56-69.

[30] Walters, S., Computer-aided prototyping for ASIC-based systems. *IEEE Design & Test* (June 1991), 4-10.