Algorithms for a Switch Module Routing Problem*

Shashidhar Thakur D.F. Wong
Department of Computer Science,
University of Texas at Austin,
Austin, Texas 78712-1188

S. Muthukrishnan
Courant Institute of Mathematical Sciences,
New York University,
New York, NY 10012

Abstract

We consider a switch module routing problem for symmetric array FPGAs. The work is motivated by two applications. The first is that of efficiently evaluating switch module designs [8]. The second is that of evaluating the routability of global routing paths for a placement on this architecture. Only an approximate algorithm was previously known for this problem. In this paper, we present an optimal algorithm for the problem based on integer linear programming. Experimental results consistently show that our algorithm is very efficient for practical sized switch modules. We further improve this technique, by doing some pre-processing on the given switch module. We also identify interesting special cases of the problem which can be solved optimally in polynomial time.

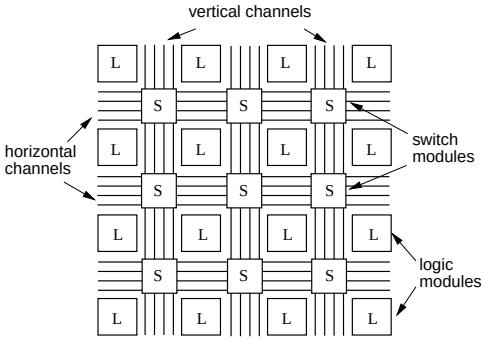

Figure 1: Symmetrical array architecture.

1 Introduction

In the symmetrical-array FPGA architecture [1, 7, 2], routing resources consist of horizontal and vertical channels and their intersecting areas. The layout in such an architecture is shown in Figure 1. An intersecting area of horizontal and vertical channels is referred to as a switch module. A net can change its routing direction via a switch module and such a direction change requires going through at least one programmable switch inside the switch module. Due to the area constraints of switch modules and delay constraints of routing, the number of switches which can be put inside a switch module, usually, is limited. On the other hand, less switches in a switch module would reduce routability. Thus, this presents a problem of designing switch modules to maximize the routability under area and delay constraints. Zhu, Wong, and Chang in [8] presented an algorithm for switch module design. In order to evaluate their designs, they introduced a switch module routing problem, which was the key problem for analyzing the routability of a switch module with respect to various routing instances. This switch module routing problem is the subject of this paper.

Another problem that motivates our work is that of routability analysis of a set of global routing paths for a placement on the FPGA. Typically, routers do a global routing followed by detailed routing. Due to the constraints of the architecture, the channel density in channels does not necessarily give a good measure of the possibility of the existence of a detailed routing [6]. It is therefore important to do some check on the global routing before proceeding with the time-consuming detailed routing. Our work provides a way of doing a local, i.e. at each switch module, feasibility check of the global routing. Thus, if for some switch module the global routing cannot be mapped to a detailed routing, there is no need to attempt a detailed routing for the entire FPGA. Figure 2 illustrates this concept.

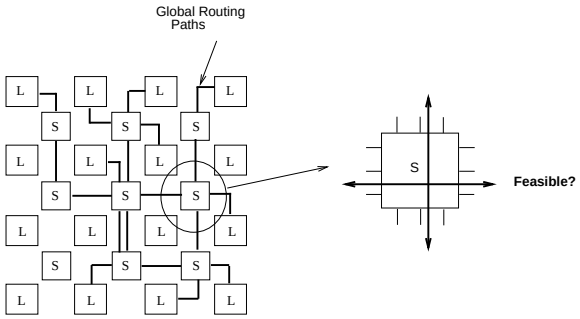

Figure 2: Routability Analysis of Global Routing.

In this way, our technique gives a way of estimating congestion at individual switch modules in a FPGA. This can be used to generate global routing paths which avoid heavily congested channels and switch modules.

A network-flow based algorithm was developed in [8] for the switch module routing problem. But the algorithm was approximate in the sense that it overestimated routability. In this paper, we present an optimal algorithm for the problem, based on integer linear programming. Although the algorithm, in the worst case, does not run in polynomial time, experimental results consistently show that our algorithm is very efficient for practical size switch modules. For example, running times for all the 20×20 switch modules we considered averaged about 0.25 seconds of CPU time. We further improve this approach by proposing a method, that avoids having to recompute solutions to the integer programming problems. This is done by doing some pre-processing on the given switch module. We also

*This work was partially supported by the Texas Advanced Research Program under Grant No. 003658459, by a DAC Design Automation Scholarship, and by a grant from the AT&T Bell Laboratories.

identify interesting special cases of the switch module routing problem which can be solved optimally in polynomial time. This is achieved by reducing them to instances of bipartite matching problems and network flow problems.

2 Problem Specification

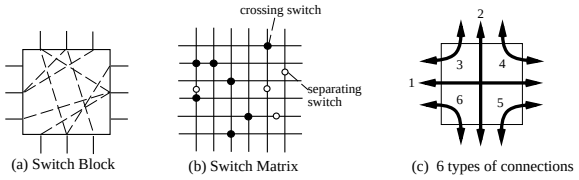

Figure 3: Models for switch module.

A *switch module* is a $W_1 \times W_2$ rectangular box with W_1 terminals on the left and right faces and W_2 terminals on the top and bottom faces. Within a switch module, various terminals are interconnected in some manner dependent on the module. A switch module can be one of two types, namely, a *switch matrix* or a *switch block*.

A *switch matrix* is a rectangular grid of W_1 *horizontal* tracks and W_2 *vertical* tracks. These tracks are electrically non-interacting. The horizontal tracks are numbered top to bottom and the vertical tracks left to right. A switch matrix comprises of two types of switches, namely, *crossing switches* and *separating switches*. These switches are utilized in establishing connections between the tracks. Crossing switches are found between a horizontal track and a vertical track and, when ON, connect those two tracks. Separating switches are found anywhere along a track and, when ON, connects the two track segments. *We assume that each horizontal or vertical track has at most one separating switch.* This is justified by the fact that the routing resources of commercial FPGAs satisfy this constraint. An example switch matrix is shown in Figure 3(b).

A *switch block* is a rectangular box with W_1 terminals on the left and right faces and W_2 on the top and bottom. Some pairs of terminals, on different faces of the box, may have programmable electrical links i.e. these pairs can be programmed to be connected or disconnected. Moreover, these links are electrically non-interacting, unless they share a terminal. An example switch block is shown in Figure 3(a).

Henceforth, a *connection* is a electrical path in the switch module between two terminals on different faces of the switch module. Connections can be of six types as shown in Figure 3(c). The connection labeled i, $1 \leq i \leq 6$, in Figure 3(c), is said to be of *Type* i. Type 1 and Type 2 connections are called the *straight* connections and Types 3,4,5, and 6 are called the *bent* connections. *For a switch matrix, we require that at most one switch be found on the electrical path comprising the connection.* Thus, only straight connections can use a separating switch. This assumption is justified by the fact that each switch introduces a delay. For a switch block, connections have to be chosen from the programmable links specified.

A *routing requirement vector* (*rrv* for short) n is a 6-tuple $(n_1, n_2, n_3, n_4, n_5, n_6)$ where $0 \leq n_1 \leq W_1$, $0 \leq n_2 \leq W_2$, and $0 \leq n_3, n_4, n_5, n_6 \leq \min\{W_1, W_2\}$. For a given switch module and *rrv*, a *routing* is a set of connections which are electrically non-interacting such that there are n_i of Type i connections, for $i \in \{1 \cdots 6\}$. A *rrv* n is said to be *routable* on a switch module S, if there exists a routing for n on S. For example, in Figure 4 a switch matrix and the routing for the *rrv* $(1,1,1,0,0,1)$ on this switch matrix are shown. The *rrv* $(0,0,2,0,0,0)$ is not routable on the same switch matrix as only the two crossing switches on vertical track 2 can be used for a Type 3 routing and not both can be used simultaneously.

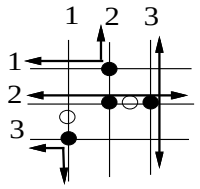

Figure 4: Example of routing.

We consider the following problems.

Routing Decision Problem (RDP): Given a switch module S (either a switch matrix or a switch block) and a *rrv* n, is n routable on S?

Routing Solution Problem (RSP): Given a switch module S (either a switch matrix or a switch block) and a *rrv* n, determine a routing for n on S, if any.

For convenience, we often refer to these problems as simply RDP with n or RSP with n, omitting the input S.

Due to lack of space, we present only the results for the switch matrix model. We have obtained similar results for the switch block case. We refer the interested reader to [5] for these and for proofs of theorems stated in this paper.

3 ILP Formulation

In this section we solve the RDP using an integer linear program (ILP); the solution to the RSP is obtained from the solution to the ILP. We show our formulations for switch matrices and switch blocks separately.

Consider the RDP with the *rrv* $(n_1, \ldots, n_6)$ and switch matrix M. We formulate this problem as an ILP. In the resultant ILP, there are two main sets of constraints. The first set contains at most two constraints for each horizontal or vertical track. For each horizontal track one constraint ensures that the segment of the track to the left of the separating switch, if any, is part of at most one connection; the other constraint ensures this for the segment to the right of the separating switch. Similarly, at most two constraints are generated for each vertical track. Note that if a track does not contain a separating switch, then only one constraint is generated for this track. The second set of constraints and an objective function are generated to ensure that a maximum number of connections specified by the *rrv* are routed in the solution of the ILP. We introduce some notation to succinctly describe the ILP.

Let S_1, S_2, S_3, and S_4 be four *constant* matrices defined as follows. $(S_1)_{ij} = 1$ if a crossing switch is found between horizontal track i and vertical track j such that a separating switch, if any, in this horizontal track is to the right of this crossing switch. Otherwise, it is 0, $i = 1, \ldots, W_1, j = 1, \ldots, W_2$. S_2 is similarly constructed as an indicator matrix of the crossing switches to the right of separating switches in the horizontal tracks. Likewise, S_3 (S_4) is an indicator matrix of the crossing switches above (below) the separating switches in the vertical tracks.

Four $W_1 \times W_2$ *variable* matrices are defined as follows. $(X_k)_{ij} = (x_k)_{ij}$ if M has a crossing switch between horizontal track i and vertical track j and this can be used to achieve a connection of Type k. Otherwise, it is 0, $k = 3,4,5,6, i =$

1,...,W_1, $j = 1, \ldots, W_2$. Variable $(x_k)_{ij}$ is an indicator variable that indicates if the switch between horizontal track i and vertical track j is utilized in a connection of Type k. Note that not every switch can be used for every type of connection. For example, if a crossing switch is above a separating switch for that column, then the switch cannot be used to realize a connection of Type 6. In Figure 3(b), the crossing switch in row 2, column 1 can only route connections of the Types 3,4 and not of Types 5,6. The one in row 4, column 1 can route connections of Type 5,6 and not of Types 3,4.

Define the binary operator $\otimes$ on matrices as the the component-wise multiplication, i.e., $(A \otimes B)_{ij} = A_{ij}B_{ij}$.

Define a *variable* column vector r of dimension W_1 as:

$$r_i = \begin{cases} 1 & \text{if horizontal track } i \text{ is used in a connection} \\ 0 & \text{else} \end{cases}$$

for $i = 1, \ldots, W_1$.

Define a *variable* column vector c of dimension W_2 as:

$$c_j = \begin{cases} 1 & \text{if vertical track } j \text{ is used in a connection} \\ 0 & \text{else} \end{cases}$$

for $j = 1, \ldots, W_2$.

Let e_1, e_2 be two constant column vectors of dimensions W_1, W_2, respectively, with all components 1. The integer programming formulation is shown in Figure 5. The number of variables $\leq W_1 + W_2 + 4$(number of switches) $\leq 4W_1W_2 + W_1 + W_2$ and the number of constraints $\leq 2(W_1 + W_2) + 6 + W_1 + W_2 + 4$(number of switches) $\leq 4W_1W_2 + 3(W_1 + W_2) + 6$.

Problem ILP1.

$$\max \quad e_1^T r + e_2^T c + \sum_{k=3}^{6} e_1^T X_k e_2$$

$$\begin{aligned}
(S_1 \otimes \sum_{i=3}^{6} X_i)e_2 + r &\leq e_1 \\
(S_2 \otimes \sum_{i=3}^{6} X_i)e_2 + r &\leq e_1 \\
(S_3 \otimes \sum_{i=3}^{6} X_i)^T e_1 + c &\leq e_2 \\
(S_4 \otimes \sum_{i=3}^{6} X_i)^T e_1 + c &\leq e_2 \\
e_1^T r &\leq n_1 \\
e_2^T c &\leq n_2 \\
e_1^T X_k e_2 &\leq n_k, \ k = 3, \ldots, 6 \\
r \in \{0,1\}^{W_1}, c \in \{0,1\}^{W_2} & \\
X_k \in \{0,1\}^{W_1 \times W_2}, \quad k = 3, \ldots, 6 &
\end{aligned}$$

Figure 5: ILP Formulation for Switch Matrix.

Theorem 1 *The problem ILP1 has a solution with objective value* $\sum_{i=1}^{6} n_i$ *if and only if the rrv* $(n_1, \ldots, n_6)$ *is routable on* M.

As an example, consider the switch matrix in Figure 6. Figure 7 shows a set of important constraints in the corresponding ILP.

Note that the set of variables that have value 1 assigned to them give a solution to the corresponding RSP.

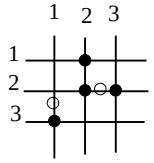

Figure 6: Switch matrix.

$$\begin{aligned}
(x_3)_{12} + (x_4)_{12} + (x_5)_{12} + (x_6)_{12} + r_1 &\leq 1 \\
(x_3)_{22} + (x_6)_{22} + r_2 &\leq 1 \\
(x_4)_{23} + (x_5)_{23} + r_2 &\leq 1 \\
(x_5)_{31} + (x_6)_{31} + r_3 &\leq 1 \\
(x_5)_{31} + (x_6)_{31} + c_1 &\leq 1 \\
(x_3)_{12} + (x_4)_{12} + (x_5)_{12} + (x_6)_{12} + (x_3)_{22} + (x_6)_{22} + c_2 &\leq 1 \\
(x_4)_{23} + (x_5)_{23} + c_3 &\leq 1 \\
r_1 + r_2 + r_3 &\leq n_1 \\
c_1 + c_2 + c_3 &\leq n_2 \\
(x_3)_{12} + (x_3)_{22} &\leq n_3 \\
(x_4)_{12} + (x_4)_{23} &\leq n_4 \\
(x_5)_{12} + (x_5)_{23} + (x_5)_{31} &\leq n_5 \\
(x_6)_{12} + (x_6)_{22} + (x_6)_{31} &\leq n_6
\end{aligned}$$

Figure 7: Example of ILP1 (important constraints).

4 Minimal Dominating Set

For this section, *fix* a switch module S. Consider solving either RDP or RSP on S for various rrv's. Using our algorithm in Section 3, an instance of integer programming problem is solved for each rrv. In this section, we describe a pre-computation on S so that following this pre-computation, either RDP or RSP on S can be solved for any given rrv without resorting to the integer programming problem. For a given S, a set of routing requirement vectors are identified during the pre-computation (this involves solving several integer programs). Following this computation, RDP or RSP on any given rrv can be solved fast by comparing it with this set of rrv's; both the computation of this set and the comparison of a given rrv with the rrv's in this set is now described. First consider solving RDP.

A rrv $(n_1, \ldots, n_6)$ is said to *dominate* another rrv $(m_1, \ldots, m_6)$ if and only if $n_i \geq m_i, i = 1, \ldots, 6$ and for some i, $n_i > m_i$. It is a simple observation that any rrv n is routable if another rrv m is routable on S and m dominates n. Intuitively, we wish to compute the set of all rrv's which dominate all the routable rrv's for S. We formalize this below.

A set R of rrv's is called a *dominating set* for a switch module S, if for a rrv v, v is routable on S if and only if either $v \in R$ or there exists a rrv $w \in R$ such that w dominates v. A dominating set R for S is called *minimal* if $\forall v, w \in R$ neither v dominates w nor w dominates v. The following property is crucial.

Lemma 1 *The minimal dominating set for a switch module S is unique.*

Observe that the set of routable rrvs for S is partially ordered under dominance relation. A rrv v is called a *top element* if it is routable and there exists no other rrv that dominates v. The following lemma is the key in computing the minimal dominating set for S.

Lemma 2 *Let $\Gamma = \{v|\ v$ is a top element $\}$. Then Γ is the minimal dominating set.*

Let $W = \min\{W_1, W_2\}$. Let V be the set of rrvs for S. Define $L(\alpha) = \{v \in V | \sum_{i=1}^{6} v_i = \alpha\}$. A rrv w is a *child* of rrv $v \in L(\alpha)$ if $w \in L(\alpha - 1)$ and w differs from v in exactly one component. Thus, each rrv has six parents and six children, except for the rrvs $(0,0,0,0,0,0)$ and (W_1, W_2, W, W, W, W) which have no children and no parents respectively.

We describe an algorithm to compute the minimal dominating set for a given switch module S. Our algorithm proceeds in levels $1 \cdots W_1 + W_2$. At level β, the set of rrvs in $L(\beta)$ is considered. In particular, only those rrvs in $L(\beta)$, all of whose children in $L(\beta - 1)$ are routable, are considered. For each such rrv, using the integer programming approach in Section 3, it is determined if the rrv is routable. All the rrvs that were considered in level $\beta - 1$ which have the property that none of their parents in level β are routable, are output as top elements. Note that it is sufficient to stop the algorithm after level $W_1 + W_2$, since in succeeding levels, the rrvs are trivially infeasible. From Lemma 2, it is easy to see that the set of top elements in the output of our algorithm is the minimal dominating set. The pseudocode is shown in Figure 8.

```
Output_min_dom_set(S)
/* S is a switch module description */
/* Outputs the elements of the dominating set */
    {
      begin
      insert (0,0,0,0,0,0) in L(0);
      for i = 0 to W1 + W2 - 1
        for each v ∈ L(i)
          found = false;
          for j = 1 to 6
            if j^th parent of v is routable
              insert j^th parent in L(i+1);
              found = true;
            endif
          endfor
          if (not found) output(v);
        endfor
      endfor
      Output each element of L(W1 + W2);
      end
    }
```

Figure 8: Algorithm for computing the dominating set.

Computing the minimal dominating set R for S completes the pre-computation. Following this, consider solving RDP with rrv v. Clearly, v is routable if and only if $v \in R$ or there exists some rrv in R which dominates v. This can be checked quickly by successively performing binary search on the components of the 6-tuples in a straightforward manner. Note in particular that no integer programming problem need be solved.

To solve RSP, we modify the pre-computation described above. Along with each rrv v determined to be in the minimal dominating set R, we determine and store the routing for v. Following this, RSP for any given rrv u can be solved fast. First determine if $u \in R$ or find an element w, if any, in R which dominates u. In the second case, it is easily seen that a routing for u, if any, can be generated from the routing for w (if w exists). Again, no integer programming problem is solved in RSP.

To sum up, by precomputing the minimal dominating set R of S off-line, the need to solve an integer programming problem while solving RDP or RSP on-line is avoided.

5 Special Cases

Since the integer programming problem is NP-Complete. polynomial time algorithms are not known for RDP and RSP using the approach in Section 3. In this section we identify several interesting classes of switch modules for which RDP and RSP can be solved in polynomial time. In what follows, we consider solving the RDP; the solutions to the corresponding RSPs are directly obtained from the proposed solutions to the RDP.

Define a *generic rrv* to be a 6 tuple in which each component is either a number or a special symbol "$*$". For example, $(*,0,0,0,0,*)$ is a generic rrv. Any generic rrv represents the class of all rrv's which differ only in the components marked "$*$". For example, $(*,0,0,0,0,*)$, represents the class containing all rrv's which have 0's in components 2,3,4 and 5. Examples of rrv's in this class include $(10,0,0,0,0,6)$ and $(0,0,0,0,0,6)$.

In what follows, RDP (or RSP) with generic rrv v stands for the problem of RDP on any rrv in the class of rrv's represented by v.

5.1 No separating switches

Suppose that the given switch matrix M contains no separating switches. We characterize the complexity of routing on M in terms of the complexity of the *bipartite matching* problem. The bipartite matching problem is to determine, if a given bipartite graph has a matching of size k.

Let $\mathcal{P}1$ and $\mathcal{P}2$ be two problems. We denote $\mathcal{P}1 \Rightarrow \mathcal{P}2$ if $\mathcal{P}2$ reduces to $\mathcal{P}1$, that is, an efficient algorithm for problem $\mathcal{P}1$ yields an efficient algorithm for $\mathcal{P}2$.[1]

We state the following lemmas. Since Lemmas 3, 4 have important observations, we give the proofs of them here.

Lemma 3 *RDP with $(0,0,*,0,0,0)$ $\Rightarrow$ RDP with $(*,*,*,0,0,0)$.*

Proof: Consider the instance of RDP on M with rrv $(n_1, n_2, n_3, 0, 0, 0)$. If $n_1 + n_3 > W_1$ or $n_2 + n_3 > W_2$ the problem is trivially infeasible. Otherwise $(n_1, n_2, n_3, 0, 0, 0)$ is routable on M if and only if $(0, 0, n_3, 0, 0, 0)$ is routable on M. This is because, given a routing for $(0, 0, n_3, 0, 0, 0)$, we can generate a routing for $(n_1, n_2, n_3, 0, 0, 0)$ by utilizing n_1 horizontal tracks and n_2 vertical tracks disjoint from the connections in the routing of $(0, 0, n_3, 0, 0, 0)$. □

Lemma 4 *RDP with $(*,*,*,0,0,0)$ $\Rightarrow$ RDP with $(*,*,*,*,*,*)$.*

Proof: We claim that rrv $(n_1, n_2, n_3, n_4, n_5, n_6)$ is routable on a switch matrix M if rrv $(n_1, n_2, n_3 + n_4 + n_5 + n_6, 0, 0, 0)$ is routable. We observe that any connection from left to top, in the absence of separating switches, renders the corresponding horizontal and vertical tracks unusable for further connections. Indeed this observation holds for all bent connections. Therefore, a left to top connection can be replaced by any other bent connection without causing any conflicts with the other

[1] Formally, we say $\mathcal{P}1 \Rightarrow \mathcal{P}2$ if an instance of problem $\mathcal{P}2$ can be reduced to an instance of problem $\mathcal{P}1$ in time $O(W_1 + W_2 + |T|)$ where T is the set of crossing switches in M and M is the switch matrix in one of the problems $\mathcal{P}1$ or $\mathcal{P}2$.

connections in routing. In particular, if $n_3 + n_4 + n_5 + n_6$ of Type 3 connections are possible then we can always replace these with n_i connections of Type $i, i = 3, \ldots, 6$. Therefore, if $(n_1, n_2, n_3 + n_4 + n_5 + n_6, 0, 0, 0)$ is routable, so is $(n_1, n_2, n_3, n_4, n_5, n_6)$. □

Lemma 5 *RDP with* $(0, 0, *, 0, 0, 0) \Leftrightarrow$ *bipartite matching problem.*

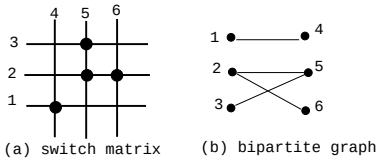

Figure 9: No separating switches.

An example of a switch matrix with the bipartite graph in the transformation of RDP with $(0, 0, *, 0, 0, 0)$ is shown in Figure 9. For any integer k, the rrv $(0, 0, k, 0, 0, 0)$ can be routed on the switch matrix in Figure 9(a) if and only if the graph in Figure 9(b) has a matching of size k.
Now we are ready to state the following theorem.

Theorem 2 *RDP with* $(*, *, *, *, *, *) \Leftrightarrow$ *bipartite matching problem.*

The bipartite matching problem can be solved in time $O(\sqrt{n}m)$ for a bipartite graph with n vertices and m edges [4]. From Theorem 2, it follows that RDP for a switch matrix M with no separating switches can be solved in time $O(\sqrt{(W_1 + W_2)}|T|)$, where $|T|$ is the number of crossing switches in M.

5.2 No separating switches in rows

Consider RDP or RSP with $(*, 0, *, *, *, *)$. Assume that in the switch matrix for these problems, the horizontal tracks do not contain any separating switches - similar results hold for the case where vertical tracks do not have separating switches. Under this condition, it is easy to see, using the technique in the proof of Lemma 4, that $(n_1, 0, n_3, n_4, n_5, n_6)$ is routable if and only if $(n_1, 0, n_3 + n_4, 0, 0, n_5 + n_6)$ is routable. The RDP with generic rrv $(*, 0, *, 0, 0, *)$ is known to be solvable using unit capacity network flows [8]. It follows that under the given condition, RDP or RSP with $(n_1, 0, n_3 + n_4, 0, 0, n_5 + n_6)$ is solvable as well, using unit capacity network flows.

5.3 Problems solvable by flows

Consider the following problem which we call the *non-interfering network flow* problem.

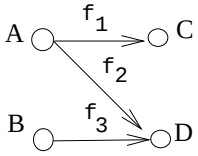

Figure 10: Non-interfering network.

Consider a directed network with four blocks of nodes, namely, A, B, C, and D. In addition there exist special nodes s_1, s_2 and t_1, t_2, respectively, the pair of source nodes and the pair of sink nodes. Arcs between nodes in the blocks, if any, exist between nodes in block A and C or A and D, or between nodes in block B and D. In particular, there are no arcs between nodes in the same block. The source s_1 (s_2) is connected to each node in A (B); every node in C (D) is connected to the sink t_1 (t_2). Each arc has capacity 1. The *non-interfering network flow problem* is the following: given such a network, and integers f_1, f_2 and f_3, does there exist a feasible flow such that source s_1 supplies a flow of $f_1 + f_2$, source s_2 supplies a flow of f_3, sink t_1 receives a flow of f_1 and sink t_2 receives a flow of $f_2 + f_3$? It is easy to see that such a flow exists if and only if there is a matching between the vertex sets $A \cup B$ and $C \cup D$ such that there exist exactly f_1 arcs between nodes in A and C, exactly f_2 arcs between nodes in A and D and exactly f_3 arcs between nodes in B and D.

Following is a list of some RDP's which can be solved using a transformation to the non-interfering network flow problem. In what follows, the switch matrix is assumed to have the following property: each horizontal and vertical track of the matrix has precisely one separating switch.

1. RDP's with generic rrv in which the components corresponding to any three bent connections are marked "$*$" and the remaining components are zero. For example, RDP with $(0, 0, *, 0, *, *)$.
2. RDP's with rrv's in which the components corresponding to any two bent connections which do not share a face of the switch matrix are marked "$*$", and the component of any one straight connection is marked "$*$". The remaining components are zero. For example, RDP with $(*, 0, *, 0, *, 0)$.

We now sketch the transformations from problems listed above to non-interfering network flow problems.

Consider an example of a problem in Category 1 above, say, RDP with $(0, 0, n_3, 0, n_5, n_6)$. We create a node for every terminal of the switch matrix. The nodes of the left face form block A, those on the right form B, those on the top form C and those on the bottom form D. For a crossing switch which is found between the horizontal track i and vertical track j, create an edge from the node in $A(B)$ to the one in $C(D)$ corresponding to the terminals i and j. *It is crucial to note that since each horizontal and vertical track has precisely one separating switch, a crossing switch can be utilized in precisely one bent connection.* It is now easy to observe that $(0, 0, n_3, 0, n_5, n_6)$ is routable if and only if there is a matching between the vertex sets $A \cup B$ and $C \cup D$ such that there exist exactly f_1 arcs between nodes in A and C, exactly f_2 arcs between nodes in A and D and exactly f_3 arcs between nodes in B and D. Thus, RDP with $(0, 0, n_3, 0, n_5, n_6)$ is transformed to the non-interfering network flow problem. See Figure 11 for an example of this transformation.

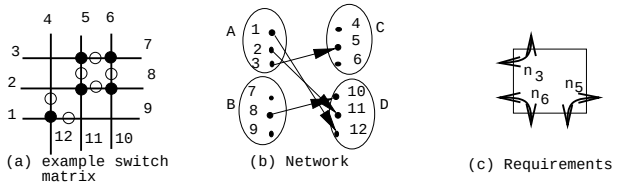

Figure 11: Example of a transformation into the non-interfering network flow problem.

A similar construction suffices for transforming a problem in Category 2 above to the non-interfering network flow problem.

Using standard techniques for computing the max-flow in networks the non-interfering network flow problem on a network of n nodes and m arcs with unit capacity on arcs, can be solved in time $O(nm)$ [3]. Therefore, problems in Category 1 and 2 above can be solved in time $O((W_1 + W_2)|T|)$.

6 Experimental Results

We wrote programs that take in routing problems and switch module descriptions and generate integer programming problems as described in Sections 3(a) and 3(b). We used a popular integer linear programming code called *lp_solve* that uses branch and bound techniques combined with the simplex algorithm for linear programming to generate integer solutions. We ran the program on a Sun Sparc 1 workstation. This was done for switch modules of sizes 10×10 and 20×20. We tested each of the sizes for both switch matrix and switch block models. The results are tabulated in Table 1. The second column gives the size of the switch module (W), the third gives maximum observed running time and the fourth column gives the average running time over 100 experiments. The last three columns give an idea about the size of the ILP. In all cases the *RDP* was being solved. The fast running time of our algorithms makes our approach an attractive one to use in practice for evaluating designs of switch modules as well as for routability analysis of global routing.

Module	W	Run Time		#var.	#constr.	#non-zeros
		Max.	Avg.			
mod40	20	0.3	0.205	97	70	388
mod50	20	0.5	0.220	115	70	460
mod80	20	1.5	0.258	177	70	708
nm40	20	0.3	0.217	120	66	480
nm50	20	0.6	0.236	140	66	560
nm80	20	1.3	0.289	220	66	880

Table 1: Running times for ILP method.

We also compared the routabilities of several switch modules as computed by our exact algorithm with those obtained by the approximate algorithm in [8]. This is shown in Table 2. All experiments used 100 *rrv*s on the switch modules. The extent of overestimation that results from an approximate algorithm justifies the use of our algorithms. The approximate algorithm had an error of about 16%, on an average.

Module	W	% of routable vectors	
		Approx. algo.	Exact algo.
mod40	20	65	58
mod50	20	70	64
mod80	20	94	82
nm40	20	99	76
nm50	20	99	83
nm80	20	99	90
Average	20	87.7	75.5

Table 2: Comparison with approximate algorithm.

We tested the technique mentioned in section 4. We observed a dramatically small search space size (i.e. the size of the minimal dominating set). For example, it was observed that for a 10×10 switch matrix design, the size of the minimal dominating set was 1254 which is just 0.12% of the possible 10^6 possible *rrv*s. As explained before, a binary search could be used on this set of vectors to test for the routability of a specified *rrv*.

7 Conclusions

In this paper, we described an integer programming approach to solving a routing problem on switch modules. The problem was originally proposed in [8] as an important part of their approach to switch module design. The problem is also useful for analyzing the routability of each switch module in a FPGA chip after global routing. Experimental results consistently showed that our algorithm was very efficient for practical size switch modules. We also identified in this paper several special cases of the problem which reduce to well known problems and to which polynomial time algorithms are known.

The techniques proposed provide an efficient way of estimating congestion at switch modules which can be using in computing good global routing paths. We propose to demonstrate this by building a global router based on techniques identified in this paper. The integer programming package we used was general and we did not attempt to customize it to make use of the specific nature of the problem matrix. As can be seen from Table 1, the problem matrix is quite sparse. Exploiting this would further speed up the solution process. Also, whether the routing problem *RDP* is NP-complete is still an open problem.

8 Acknowledgements

We thank Kai Zhu for helpful discussions on several topics in the paper. We also thank Yao-Wen Chang for providing test designs of switch modules.

References

[1] AT&T. *Optimized Reconfigurable Cell Array (ORCA) Series Field Programmable Gate Arrays*, February 1993.

[2] H.C. Hsieh et al. Third generation architecture boosts speed and density of field programmable gate arrays. In *Proceedings of the IEEE CICC*, pages 31.2.1–31.2.7, 1990.

[3] H.N. Gabow. Scaling algorithms for network problems. *Journal of Computing Systems Science*, 31:148–168, 1985.

[4] C. Papadimitriou and Steiglitz. *Combinatorial Optimization - Algorithms and Complexity*. Prentice-Hall Inc., 1982.

[5] S. Thakur, D.F. Wong, and S. Muthukrishnan. Algorithms for fpga switch module routability analysis. Technical Report TR-94-14, Dept. of Computer Science, UT Austin, 1994.

[6] Y. Wu, S. Tsukiyama, and M. Marek-Sadowska. Computational complexity of 2-d fpga routing for arbitrary switch box topologies. In *International Workshop on Field Programmable Gate Arrays*. ACM SIGDA, 1994.

[7] Xilinx Corporation. *The Programmable Logic Data Book*, 1993.

[8] K. Zhu, D.F. Wong, and Y.W. Chang. Switch module design with application to two-dimensional segmentation design. In *International Conference on Computer Aided Design*, pages 480–485. IEEE/ACM, 1993.