Circuit Partitioning with Complex Resource Constraints in FPGAs*

Huiqun Liu[1], Kai Zhu[2] and D. F. Wong[1]
[1] Department of Computer Sciences
University of Texas at Austin, TX 78712
Email: hqliu@cs.utexas.edu, wong@cs.utexas.edu

[2]Actel Corporation
955 East Arques Avenue, Sunnyvale, CA 94086
Email: zhu@actel.com

Abstract

In this paper, we present an algorithm for circuit partitioning with complex resource constraints in large FPGAs. Traditional partitioning methods estimate the capacity of an FPGA device by counting the number of logic blocks, however this is not accurate with the increasing capacity and diverse resource types in the new FPGA architectures. We propose a network flow based method to optimally check whether a circuit or a sub-circuit is feasible for a set of available heterogeneous resources. The feasibility checking procedure is integrated in the FM-based algorithm for circuit partitioning. Incremental flow technique is employed for efficient implementation. Experimental results on the MCNC benchmark circuits show that our partitioning algorithm not only yields good results, but also is efficient. Our algorithm for partitioning with complex resource constraints is applicable for both multiple FPGA designs (*e.g.* logic emulation systems) and partitioning-based placement algorithms for a single large hierarchical FPGA (*e.g.* Actel's ES6500 FPGA family).

1 Introduction

The new generation of large FPGAs are targeted at greater logic capacity and higher system performance. Partitioning heuristics play a fundamental role in addressing the increasing complexity both in multi-FPGA circuit implementation as well as placement on a single large hierarchical FPGA. For example, for placement on a large FPGA of hierarchical architecture such as Actel's ES6500 family, it is necessary to partition the circuit into separate hierarchical blocks first and then do placement on the individual hierarchical blocks.

A popular formulation of the circuit partitioning problem is to minimize the number of cut nets between partitions while satisfying the resource capacity constraints in each partition. Normally the resource constraint is simply calculated as the area or gate count available on the chip in order to avoid overflow of resource usage during placement.

*The work of Liu and Wong was partially supported by the Texas Advanced Research Program under Grant No. 003658288.

However, for circuit partitioning in FPGAs, measuring resource capacity by simply using area or gate count is inaccurate and is no longer adequate for practical purposes. One major reason is due to the increasing number of different resources available in the new generation of FPGAs. Driven by the demand of supporting system level applications, FPGAs are getting larger in terms of capacity and at the same time are also getting more heterogeneous in terms of types of resources available. For example, it is not uncommon to find a commercial FPGA that contains different logic modules (*e.g.* LUTs of different sizes), complex IO modules, various speed grade clocks, embedded SRAM memory arrays, and dedicated architecture resources designed for supporting special functions (*e.g.* wide input gates). This trend of increasing number of different resources will continue as various intellectual property (IP) blocks are integrated with FPGAs. Another major reason for the inaccuracy of simple capacity metric is that normally a node in a netlist can be implemented using different resources in FPGAs. For example, Actel ES6500 FPGA family contains LUT2 and LUT3 [18,19]. A 2-input gate can be implemented using either a LUT2 or a LUT3. Similarly, a resource on FPGA can be used to implement different types of nodes in a netlist. Such multiple choices of implementation of a netlist on an FPGA can not be accurately captured by a simple area or gate count capacity metric.

Though many algorithms have been proposed for circuit partitioning problems [1,2,3,4,5,7,8,9,10,11,12,13], the multiple resource types in an FPGA are not taken into consideration. A partitioning algorithm with simple resource capacity metric may produce partitioning results that actually violate resource constraints and thus render the results unusable. For a partitioning algorithm to be useful for solving practical FPGA partitioning problems, especially for the netlists with high FPGA utilization, it is necessary to employ an accurate resource capacity metric and incorporate it in the partitioning algorithm. None of the published partitioning algorithms that we are aware of meet these requirements.

The above analysis motivates our work in the circuit partitioning problem with complex resource constraints. In this paper, we first give a network flow based feasibility checking algorithm to optimally check whether the amount of logic in a circuit can be implemented by the given set of available resources. Then the feasibility checking method is embedded in the FM-based partitioning method to find a partition that satisfies the resource constraints. For efficiency, incremental flow computation is employed. When moving a node from one subset to another, the constructed flow networks are maintained dynamically and efficiently by the insertion and deletion operations. Our approach can also be applied to hierarchical partitioning and multi-way partitioning algorithms.

Our algorithm is applicable for circuit implementation in multi-FPGAs (*e.g.* logic emulation) and partitioning-based placement algorithms for a single large hierarchical FPGA (*e.g* Actel's ES6500 FPGA family). The organization of the following sections is as follows. We present the problem formulation of partitioning with complex resource constraints in section 2. A network flow based method is proposed for feasibility checking in section 3.1 and an algorithm which integrates the feasibility checking method with an iterative improvement based partitioning is proposed in section 3.2. Section 3.3 discusses the incremental flow computation technique used to make our approach efficient. Section 4 shows the experimental results on the MCNC benchmark circuits.

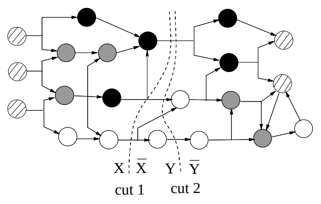
2 Problem Formulation

We consider the problem of partitioning with complex resource constraints for FPGAs. It is different from the previous published algorithms that we are aware of in the following two aspects. First, a target device, such as FPGA, contains multiple types of resources. Secondly, each node in the circuit has multiple choices of implementation by different types of resources. For example, a design system may get an ASIC-like library from FPGA vendors and output the netlist in terms of library cells instead of LUTs. For Actel's ES6500 family, the library cells are classified into one of three categories: basic, hard and soft library cells. A hard or a soft library cell can be decomposed into multiple basic cells. A basic cell can not be further decomposed and represents a distinct logic function which can not be "covered" by any other basic cells. A basic cell can be implemented by one of a few optional resources.

For a device such as FPGA, let $R = \{r_1, ..., r_k\}$ be a set of k resource types and let $n(r_i)$ be the capacity for resource type r_i ($1 \leq i \leq k$). Let C be a set of types of basic cells in the library. Each c in C maps to R_c, where $R_c = \{r_{i_1}, ..., r_{i_s}\}$ is a subset of resources such that a basic cell of type c can be implemented by one of the s alternative resources in R_c.

A circuit can be represented by a netlist $G = (V, E)$ where V is the set of nodes and E is the set of nets. Each net $n_t \in E$ connects two or more nodes together. Each node v in V corresponds to a functional or library cell that can be further decomposed into a set of basic cells, represented as $v = (c_{v_1}, c_{v_2}, ..., c_{v_p})$ where $c_{v_j} \in C$ for $1 \leq j \leq p$. Note that $c_{v_1}, ..., c_{v_p}$ are not necessarily different, and $p = 1$ if v is a basic cell. The decomposition of a functional node v into the basic cells depends on the function of the node. v is implemented only when each c_{v_j} ($1 \leq j \leq p$) is implemented by the available resources.

A circuit G is defined to be *feasible* for a set of resources R if the amount of logic in G can be implemented by the given resources, that is, there exists at least one resource allocation scheme such that $\sum_{v \in V} u(v, i) \leq n(r_i)$ for $1 \leq i \leq k$, where $u(v, i)$ is the number of resource r_i utilized by node v.



cell type	resource needed to implement the cell	resource capacity per partition	
c_1 ●	r_1	r_1	3
c_2 🔘	r_2	r_2	3
c_3 ◍	r_3	r_3	3
c_4 ○	r_4	r_4	3

Figure 1: *A special case of the partitioning with complex resource constraints problem. Each basic cell in the circuit can be implemented by exactly one type of resource.*

Given two sets of resources R_1 and R_2, the problem of two-way partitioning with complex resource constraints is to partition a netlist G into two non-overlapping subsets V_1 and V_2, subject to

1. $V = V_1 \cup V_2$;
2. V_1 is feasible for R_1 and V_2 is feasible for R_2;

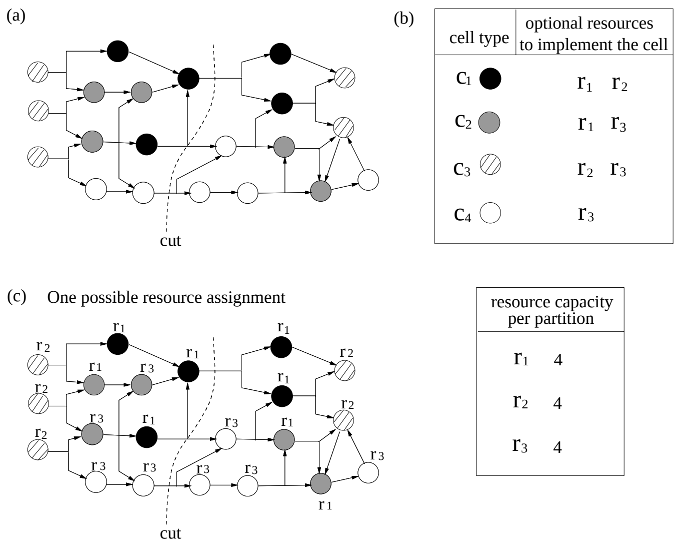
with the objective of minimizing the total number of cut nets $|N_{cut}|$, where $N_{cut} = \{n_t \in E \mid \exists u, v \in n_t, s.t\ u \in V_1, v \in V_2\}$.

We refer to the above partitioning problem as *partitioning with complex resource constraints*. The objective is to minimize the number of cut nets while satisfying the resource constraints simultaneously. A special case of this problem is when there is only one type of resource in both R_1 and R_2. This is the traditional problem of min-cut bipartitioning when the two partitioned subsets are balanced by the total area of each subset, where the area of each cell is the number of resources (*e.g* LUTs) used. Many heuristic approaches have been proposed to solve this problem, such as the iterative improvement method (K&L, FM)[1,2,8,9], simulated annealing[3], spectral-based method[4,7,11] and network flow-based method (FBB)[10]. Even this special case is well known to be a NP-complete problem, so is the general problem stated above. This leads to Lemma 1.

Lemma 1: *The problem of two-way circuit partitioning with complex resource constraints is NP-complete.*

Another special case is when each of the basic cells only maps to one type of resource (*i.e.* $|R_c| = 1$ for each c). The objective of the partitioning problem is to balance the different types of nodes in each subset. In the example shown in Figure 1, each cell can be implemented by one type of resource and the capacity for each resource is given in the table. Cut 2 is a feasible partitioning solution since each type of resources used in each subset does not exceed the capacity. Cut 1 is not a feasible partitioning solution, though it has a smaller cut-size than that of cut 2. This is because there are four cells of type c_4 in $\overline{X}$, which require four r_4 type of resources. Since the capacity for resource type r_4 is only 3, there is not enough resource to implement the four cells of type c_4.

Figure 2 illustrates an example of the partitioning problem with complex resource constraints. Each node in the circuit maps to one basic cell and each basic cell can be implemented by one of the optional resources. For example, c_1 can be implemented by either resource r_1 or r_2, c_2 can be implemented by either r_1 or r_3 and so on. The capacity for each of the three resource types is 4. The cut shown in Figure 2(a) is a feasible partitioning solution as one resource allocation scheme is shown in figure 2(c) to satisfy the resource constraints.



cell type	optional resources to implement the cell
c_1 ●	r_1 r_2
c_2 ◍	r_1 r_3
c_3 ▨	r_2 r_3
c_4 ○	r_3

resource capacity per partition	
r_1	4
r_2	4
r_3	4

Figure 2: *An example of the problem of partitioning with complex resource constraints. Each node in the circuit can be implemented by more than one type of resource.*

Note that R_1 and R_2 can be two different sets of resources, which happens when we hierarchically partition a circuit for placement inside an FPGA device, and the available resources in different blocks of the device may be different. For example, the block near the border of the FPGA device usually has more I/O modules than a block in the middle of the device.

3 Partitioning with Network Flow Based Feasibility Checking

3.1 Network Flow Based Feasibility Checking

With multiple types of resources and multiple choices to implement a basic cell, it is not obvious to see whether there exists a resource allocation for a feasible partitioning solution. In this section, we present a network flow based method to check whether a circuit or a subcircuit is feasible with respect to the resource constraints. In the next section, we will embed this feasibility checking method in the FM-based partitioning algorithm.

In a circuit $G = (V, E)$, each node can be decomposed into a set of basic cells according to its function and the circuit can be implemented only when each of the basic cells is implemented by a resource. Note that the decomposition of a node into basic cells is stored in the library and can be retrieved in $O(1)$ time. Let $C' = \{c_1, ..., c_m\}$ be a set of m types of basic cells decomposed from V and let $n(c_i)$ denote the total number of basic cells of type c_i ($1 \leq i \leq m$) decomposed from V. Typically it is unlikely that every type of basic cell in the library will be used in a circuit, so $C' \subseteq C$ and $|C'|$ is usually less than $|C|$ where C is the set of basic cell types in the library. Given a set of resources $R = \{r_1, ..., r_k\}$, let R_c ($R_c \subseteq R$) be the subset of optional resources to implement basic cell of type c for each $c \in C'$. We construct the flow network $F = (V', E')$ as follows (shown in Figure 3):

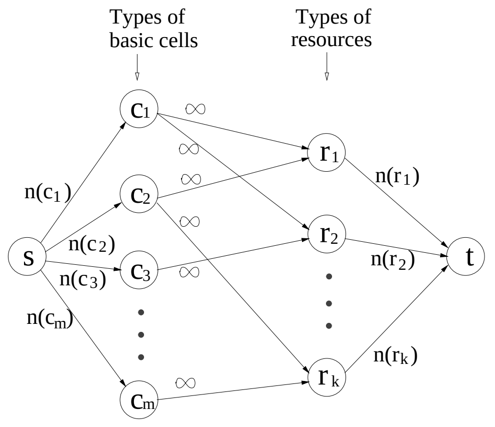


Figure 3: *A flow network is constructed for feasibility checking*

1. $V' = \{s\} \cup C' \cup R \cup \{t\}$, where s and t are the source and sink of the flow network.
2. For each $c \in C'$, add an edge $s \to c$ with capacity $n(c)$, where $n(c)$ is the total number of basic cells of type c in the decomposed circuit.
3. For each $c \in C'$, add an edge $c \to r$ with capacity ∞ for each $r \in R_c$.
4. For each $r \in R$, add an edge $r \to t$ with capacity $n(r)$, where $n(r)$ is the capacity of resource type r.

After the flow network is built, every directed edge is assigned a capacity and initially the flows on all the edges are zero. We can check the feasibility of a circuit by a maximum flow computation on the network. The maximum flow computation pushes flow from the source to the sink until no more flows can be added. Let $cap(v, u)$ and $flow(v, u)$ denote the capacity and flow on edge $v \to u$ respectively. For every edge $v \to u$, $0 \leq flow(v, u) \leq cap(v, u)$. Except the source and sink, for each of the other nodes, the sum of incoming flow is equal to the sum of outgoing flow, *i.e.* for all $v \in V' - \{s, t\}$, $\sum_{u\ (u,v)\in E'} flow(u, v) = \sum_{u\ (v,u)\in E'} flow(v, u)$. An edge $v \to u$ is *saturated* if its capacity is equal to the flow (*i.e* $cap(v, u) = flow(v, u)$). After the maximum flow computation, if every edge $s \to c_i$ is saturated, then the circuit is feasible in that each cell can find the available resource to implement it.

Lemma 2: *A circuit G is feasible if and only if by the maximum flow computation in F, all the out-going edges from the source (i.e $s \to c_i$ for $1 \leq i \leq m$) are saturated, i.e. $\sum_i cap(s, c_i) = \sum_i flow(s, c_i)$.*

Proof If circuit G is feasible, then there exists at least one resource allocation scheme such that each cell can be assigned with an available resource. If a cell of type c is assigned with resource r, then add a flow on the path: $s \to c \to r \to t$. Since each cell is successfully assigned with a resource, it contributes exactly one flow through the network. Thus for each edge $s \to c_i$, $flow(s, c_i) = cap(s, c_i)$ and therefore $\sum_i cap(s, c_i) = \sum_i flow(s, c_i)$.

On the other hand, if after the max-flow computation, every edge $s \to c$ ($c \in C'$) is saturated, then one flow on a path $s \to c \to r \to t$ corresponds to an assignment of resource r to a cell of type c. Since each edge $s \to c$ is

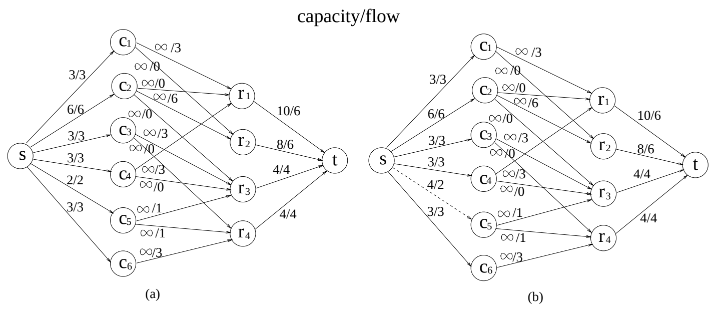


Figure 4: *The circuit is feasible if and only if every edge starting from the source is saturated in the flow network. The circuit corresponds to (a) is feasible. (b) shows an example that even if the total number of cells is less than the total number of resources, the circuit is still not feasible.*

saturated and the capacity on the edge is the total number of basic cells of type c, all the basic cells in the circuit must be assigned with the corresponding resources, therefore the circuit is feasible. From the above analysis, Lemma 2 holds. $\square$

Figure 4 shows two examples. In Figure 4(a), there are 6 types of basic cells and 4 types of resources. $R_{c_1} = \{r_1, r_2\}$, such that there are two edges $c_1 \rightarrow r_1$ and $c_1 \rightarrow r_2$. After the max-flow computation, every out-going edge from the source s is saturated, which means all the cells can be implemented by the available resources. Figure 4(b) is an example showing that even if the total number of basic cells in the circuit (22) is less than the total number of resources (26), the circuit is still not feasible. When the capacity on edge $s \rightarrow c_5$ is changed to four, edge $s \rightarrow c_5$ is not saturated after the max-flow computation ($flow(s, c_5) < cap(s, c_5)$). By Lemma 2, the circuit is not feasible and can not possibly be implemented by the available resources. This is because cell type c_3 and c_5 can be implemented by resource r_3 or r_4, and c_6 can only be implemented by r_4. Besides the three resources of type r_4 used for c_6, there is one resource of type r_4 and four resources of type r_3 left that can be used for cell type c_3 and c_5. However, a total of seven cells of type c_3 and c_5 need to be implemented. Hence, there is not enough resource to implement the logic.

The total number of nodes in the flow network F is $|C'| + |R| + 2$ and the number of edges in the network is $O(|C'| + |C'| \cdot |R| + |R|) = O(|C'| \cdot |R|)$. Note that $|C'|$ is usually much smaller than $|V|$, because C' only contains the types of basic cells decomposed from the circuit, rather than all the nodes in V. C' depends on the size of the library. In practice, many nodes in V are of the same type, and the total number of basic cells of type c_i is reflected on the capacity of the edge $s \rightarrow c_i$ $(1 \leq i \leq m)$. For the Actel's ES6500 family of FPGA, the number of basic cells in the library is around 300 and the average number of resource types is 5. Since the size of the flow network is independent of the size of a circuit, the feasibility checking method is scalable to large circuits.

To make the feasibility checking more efficient, we can further group the basic cells which have the same resource requirement into one class. It is observed that some basic cells, regardless of their logic functions, map to the same subset of resources. We define c_i and c_j to be equivalent (denoted by $c_i \sim c_j$) if $R_{c_i} = R_{c_j}$. Thus we merge all the equivalent basic cells into one node in the flow network, and

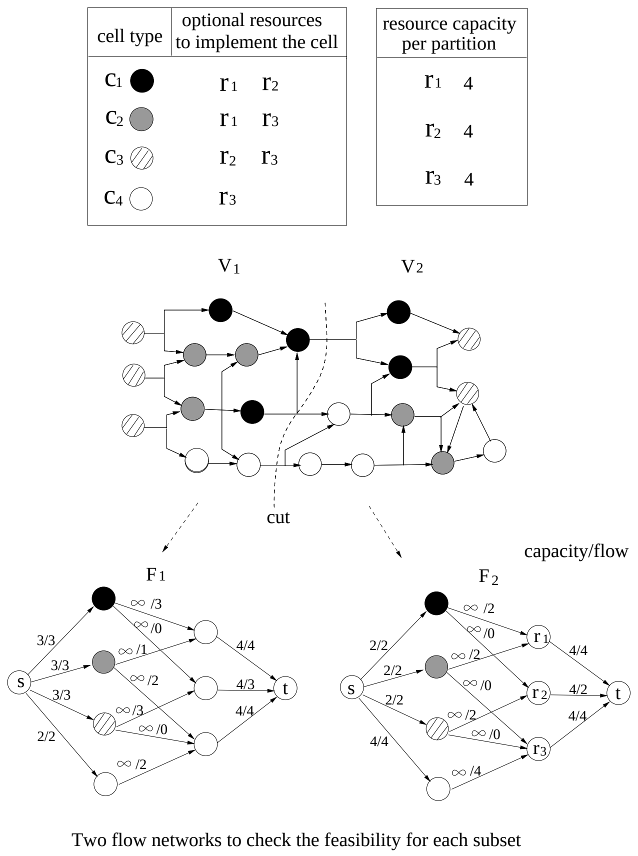

cell type
optional resources to implement the cell
c1 ● r1 r2
c2 ● r1 r3
c3 ▨ r2 r3
c4 ○ r3
resource capacity per partition
r1 4
r2 4
r3 4
V1
V2
cut
F1
F2
capacity/flow
s
t
r1
r2
r3
3/3
2/2
4/4
4/3
4/2
∞/3
∞/0
∞/1
∞/2
∞/4
Two flow networks to check the feasibility for each subset

Figure 5: *Two flow networks F_1 and F_2 are built for the feasibility checking of subsets V_1 and V_2 respectively.*

let C'' be the set of newly merged nodes. We can build the flow network in the same way as before by replacing C' by C''. Note that each node in C'' uniquely maps to one subset of resources. For $|R|$ resource types, the number of non-empty subset is at most $2^{|R|} - 1$, so the number of nodes in C'' is at most $min(2^{|R|} - 1, |C'|)$. In practice, $|R|$ is usually around 5, so $|C''|$ is at most 31. If $|R|$ is 6, then $|C''|$ is at most 63. Therefore, the size of the network can be further reduced.

A two-way partitioning solution is *feasible* only when each partitioned subset is feasible. To check the feasibility of a partitioning solution, we build two networks F_1 and F_2 to check the feasibility of the two subsets V_1 and V_2 respectively. If both subsets are feasible, then the partitioning solution is feasible. Figure 5 shows the two networks which are used to check the feasibility of each subset for the example in Figure 2. Detail is discussed in the next two sections.

3.2 Two Way Partitioning with Feasibility Checking

In this section, we propose a partitioning algorithm by integrating the network flow based feasibility checking in a two way iterative improvement circuit partitioning algorithm. The partitioned subsets satisfy the resource constraints.

Since the two-way balanced circuit partitioning problem is NP-complete, a number of iterative improvement schemes have been proposed [1,2,4,5,8,9,16,17]. In iterative improvement methods, we start with a random two-way partitioning of the circuit, and iteratively improve it by either swapping pairs of nodes between the subsets or moving one node at a time so that the net cut size is reduced.

The FM iterative partitioning process repeatedly move a node from one subset to the other in order to reduce the min cut size. It determines the next best node u_i to move in the ith step as follows. The "unlocked" cell (initially all nodes are unlocked) with the maximum gain in either subset is determined. If the balance criterion on the two subsets can be maintained after moving this node from its current subset to the other one, it is chosen as the node u_i. Otherwise, the unlocked node with the maximum gain in the other subset is chosen as u_i. Node u_i is then moved to the other subset and "locked", and the gains of all its neighbors are updated if a net becomes critical when u_i is moved. The node gain $gain(u_i)$ is inserted in an ordered set S. After all nodes are moved and locked, all prefix sum $S_k = \sum_{t=1}^{k} gain(u_t)$ are computed $(1 \le k \le n)$, and a p is determined for which the partial sum S_p is the maximum. The set of nodes that are actually moved are then $\{u_1, ...u_p\}$. This whole process is called a *pass*. A number of passes are made until the maximum partial sum is zero or negative. The resulting cutset cost is a local minima with respect to the initial partitions V_1 and V_2.

In order to meet the resource constraints, network flow-based feasibility checking discussed in section 3.1 can be integrated in each iteration of the FM partitioning method. Two flow networks F_1 and F_2 (as illustrated in Figure 5) are built for the feasibility checking for each of the two subsets V_1 and V_2.

In each iterative step, when a node with the maximum gain is selected, it is checked whether the subset to which the node is to be moved is still feasible. This is done by the *insertion* operation (more details are given in section 3.3) on the corresponding flow network as follows. The candidate node to be moved is decomposed into a set of basic cells $(c_{i_1}, c_{i_2}, ..., c_{i_p})$, and the capacity on edge $s \to c_{i_j}$ for $1 \le j \le p$ is incremented. Then maximum flow is computed. There are two cases according to the result of the max-flow computation.

Case 1: If all the edges going out of the source s are saturated, by Lemma 2, the subset is still feasible when the node is moved to it. Next, the network of the node's original subset will be modified by the *deletion* operation which deletes the corresponding capacity from edge $s \to c_{i_j}$ for $1 \le j \le p$ and deletes the same amount of flow through the network.

Case 2: If after the maximum flow computation, not every edge out-going from the source is saturated, by Lemma 2, the subset will not be feasible if the node is moved to it. The network is recovered to its previous state by the *deletion* operation which deletes the added capacity and flow on the modified edges. Then another node with the maximum gain will be selected as the new candidate and the feasibility checking is applied again until a node is found which can keep both subsets feasible when it is moved.

We designed the FFC-fm algorithm, which combines the network flow-based feasibility checking method with the FM method for circuit partitioning with complex resource constraints. First, two networks are constructed, one for each subset in the partition. The capacity on the edges from a resource node to the sink (*i.e.* $r_i \to t$) is set to be equal to the capacity of the corresponding resource. Next, a feasible initial partition is found. This is done by randomly distribute all the cells belonging to the same type of basic cells into two subsets according to the resource capacity in each subset. After the initial partition, the capacity on the edges $s \to c_i$ $(1 \le i \le m)$ is set to be the number of basic cells of type c_i decomposed from the subset. Then nodes are iteratively moved from one subset to another while trying to reduce the total cut-size, and the feasibility of the two subsets is checked by the max-flow computation on the two flow networks in each iteration.

3.3 Incremental Flow Computation for Efficient Implementation

The efficiency of the feasibility checking process is of great concern for the practical use of our FFC-fm algorithm. *Incremental flow* technique is employed to make the max-flow computation efficient. In each feasibility checking process, it is not necessary to compute the max-flow from scratch, only additional flow is added to saturate the edges.

After the initial partitioning, maximum flow is computed on the two networks F_1 and F_2. The two networks keep the current status of the resource assignment for each of the subsets, and are dynamically and incrementally changed when a node is moved. In each step, when a node is moved to a subset V_i, the capacity on the related edges in F_i is incremented and additional flow is added in the network. If a node is removed from a subset V_i, then the capacity on the related edges in F_i is decremented and flow is deleted.

Two operations, *insertion* and *deletion* operations, are designed to dynamically maintain the two flow networks (Figure 6). The *insertion* operation is used to check if a subset is still feasible if a node v is moved to it. It works as follows.

Procedure *insertion(v, F)*:

1. Decompose v into a set of basic cells $v = (c_{i_1}, c_{i_2}...c_{i_p})$, then the capacity on edge $s \to c_{i_j} (1 \le j \le p)$ is increased by 1 (*i.e* $cap(s, c_{i_j})$ is increased by 1). Let Δc be the total number of capacities added to these edges.
2. Additional flows are pushed from the source to the sink by the maximum flow computation in F. Let Δf be the total amount of flow added through the network.
3. If $\Delta c = \Delta f$, then every edge going out of the source is saturated. By Lemma 2, the subset is still feasible when node v is moved to it. Otherwise, if $\Delta f < \Delta c$, then the subset is not feasible when node v is moved to it.

Lemma 3: *When a node v is added to a feasible subset V_1, the new subset $V_1 \cup \{v\}$ is still feasible if and only if the newly pushed flow Δf by the max-flow computation is equal to the newly added capacity Δc in F_1.*

Proof If V_1 is feasible and F_1 is the corresponding checking network, then $\sum_{i=1}^{m} flow(s, c_i) = \sum_{i=1}^{m} cap(s, c_i)$ in F_1. Since $V_1 \cup \{v\}$ is also feasible, then every edge $s \to c_i$ is still saturated after the max-flow computation. Therefore, $\Delta f = \Delta c$. On the other hand, if $\Delta f = \Delta c$, then the edges going out of the source are saturated both before and after v is moved, so $V_1 \cup \{v\}$ is feasible. $\square$

The time complexity for one insertion is $O(1)$ in the best case and $O(|C'| \cdot |R|)$ in the worst case, where $|C'|$ is the number of types of basic cells and $|R|$ is the number of resource types. However, the average time for insertion is usually much less than the worst case, which will be shown in our experiments.

The *deletion* operation is used under two situations: (1) If *insertion* operation shows that the subset V_i is feasible when a node v is moved to it, then v is moved to V_i but the capacity and flows should be deleted from the network of its original subset. (2) If after *insertion*, it is found that the subset will become infeasible if the node is moved to it, the inserted capacity and flow should be deleted and the network F_i is restored to its previous state as before *insertion*. The *deletion* operation works as follows.

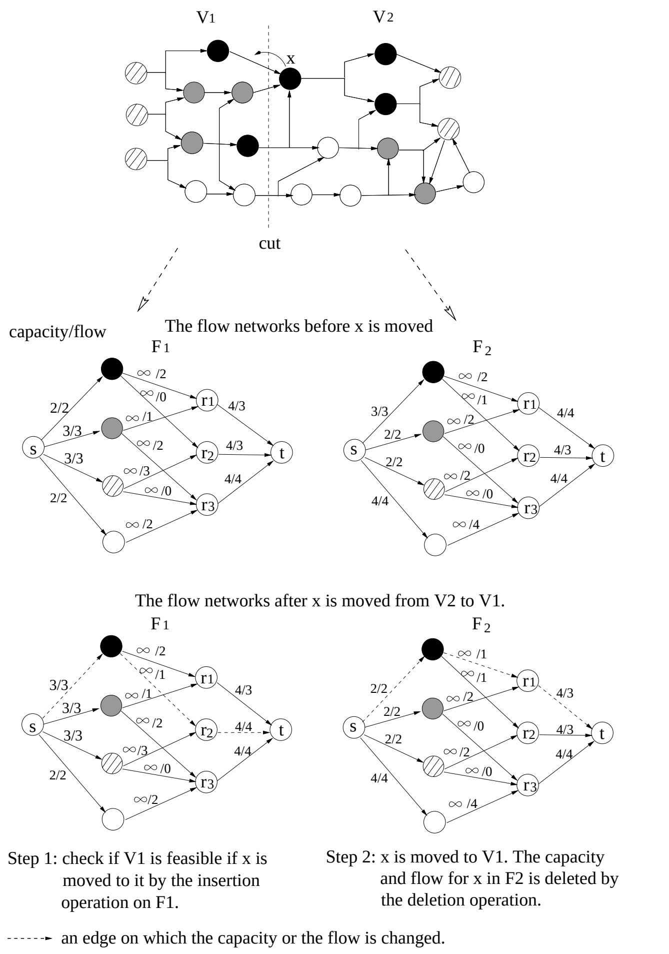
```
Procedure deletion(v, F):

1. Let $v$ be decomposed into a set of basic cells:
   $v = (c_{v_1}, c_{v_2} ... c_{v_p})$;

2. for each $c_{v_j}$ $(1 \le j \le p)$ do
   begin
       $cap(s, c_{v_j}) \leftarrow cap(s, c_{v_j}) - 1$;
       select $r$ incident on $c_{v_j}$ such that $flow(c_{v_j}, r) > 0$;
       $flow(s, c_{v_j}) \leftarrow flow(s, c_{v_j}) - 1$;
       $flow(c_{v_j}, r) \leftarrow flow(c_{v_j}, r) - 1$;
       $flow(r, t) \leftarrow flow(r, t) - 1$;
   end
```

In step 1, the node v to be deleted is decomposed into a set of basic cells $v = (c_{v_1}, c_{v_2} ... c_{v_p})$. This can be done in $O(1)$ time since the information for the decomposition can be directly retrieved from the library. In step 2, for each c_{v_j}, the capacity on edge $s \rightarrow c_{v_j}$ is decremented. Then node r incident on c_{v_j} is selected such that $flow(c_{v_j}, r) > 0$, and flow is deleted from each edge on the path $s \rightarrow c_{v_j} \rightarrow r \rightarrow t$. After the *deletion* operation, every edge outgoing from the source is saturated, and the feature holds that for any node rather than the source and sink, the sum of incoming flow is equal to the sum of the out-going flow.

Lemma 4: *If a node is removed from a feasible subset V_1 and the corresponding capacity and flow are deleted from the flow network F_1 by the deletion operation, the resulting circuit is still feasible and $\sum_i flow(s, c_i) = \sum_i cap(s, c_i)$.*

The time complexity for one deletion operation is $O(|C'| \cdot |R|)$. Let p be the average number of basic cells that can be decomposed from a node and let t be the average number of optional resources for a basic cell, then the average time complexity for one deletion operation is $O(p \cdot t)$. For the Actel's ES6500 family of FPGA, p is around 2 and t is around 1.5, therefore the average time complexity for one deletion operation is $O(p \cdot t) = O(1)$.

In the FFC-fm algorithm, for one feasibility checking, no matter a node is moved successfully or not, it needs one insertion and one deletion on the flow networks, thus the time complexity for one feasibility checking is $O(|C'| \cdot |R|) + O(|C'| \cdot |R|) = O(|C'| \cdot |R|)$. In each iteration of FFC-fm, to find a node to be moved so that the target subset is feasible takes time $O(|C'| \cdot |R| \cdot |V|)$, since in the worst case $|V|$ nodes may be tried. Since the



V1
V2
x
cut
The flow networks before x is moved
capacity/flow
F1
F2
s
r1
r2
r3
t
The flow networks after x is moved from V2 to V1.
Step 1: check if V1 is feasible if x is moved to it by the insertion operation on F1.
Step 2: x is moved to V1. The capacity and flow for x in F2 is deleted by the deletion operation.
an edge on which the capacity or the flow is changed.

Figure 6: *An example of the insertion and deletion operation. Step 1: if x is the node with the maximum gain, the insertion operation checks if x can be moved to V_1. The capacity on edge $s \rightarrow c_1$ is incremented and max-flow is computed on the network F_1. Step 2: since $V_1 \cup \{x\}$ is feasible, x can be successfully moved to V_1. The capacity on edge $s \rightarrow c_1$ in F_2 is decremented and flow is deleted through the network.*

time complexity for one pass of the FM method is $O(P)$, where P is the number of pins of all the nets in the circuit, the time complexity for one pass of FFC-fm algorithm is $O(|C'| \cdot |R| \cdot |V|^2) + O(P)$. For Actel's ES6500 family, $|C'|$ is less than 300 and $|R|$ is around 5, so the time complexity of FFC-fm is $O(|V|^2) + O(P) = O(|V|^2)$.

Our feasibility checking algorithm can be applied to other iterative improvement partitioning methods, such as variations of FM-based method with different tie-breaking strategies, ratio-cut partitioning or simulated annealing method. It can also be applied to hierarchical partitioning and multi-way partitioning methods. For hierarchical partitioning, each cluster can be treated as a single node and decomposed into a set of basic cells when modifying the capacities on the edges of the flow network. Insertion and deletion operations can be applied in the same way. For multi-way partitioning, k networks will be built for each of the k partitions. The flow network corresponding to each subset will be incrementally changed by the insertion and deletion operations when nodes are moved among the subsets, and feasibility is

Table 1: Comparison of the min-cut size

Circuit	#nodes	#nets	Min-cut ($\lvert C'\rvert = 150, \lvert R\rvert = 5$) Static	FFC-fm	impr.%	Min-cut ($\lvert C'\rvert = 250, \lvert R\rvert = 5$) Static	FFC-fm	impr.%
c5315	1778	1655	89	61	31.4	63	41	34.9
c7552	2247	2140	67	39	41.8	59	30	49.1
c6288	2856	2824	98	67	31.6	86	56	34.8
s5378	3225	3176	90	73	18.9	81	62	23.4
s9234	6098	6076	112	87	22.3	93	71	26.7
s13207	9445	9324	101	85	15.2	109	84	23.0
s15850	11071	10984	112	89	20.5	107	81	24.3
s35932	19882	19560	123	101	17.8	133	99	25.6
s38584	22451	20719	107	91	15.0	111	87	21.6
s38417	25589	25483	98	77	21.4	107	82	23.3

checked according to Lemma 3.

4 Experiments and Discussions

We implemented the network flow based feasibility checking algorithm in C language on IBM RS6000 workstation and integrated it into the FM-based partitioning method. We conducted the experiments on the MCNC Partitioning93 benchmark circuits which are shown in Table 1, and with parameters consistent with Actel's ES6500 FPGA family. The library contains around 300 cells. Each library cell is classified into one of three categories: basic, hard and soft cell. A basic cell can not be further decomposed and it represents a distinct logic function which can not be "covered" by any other basic cells. A hard or a soft library cell can be decomposed into multiple basic cells. The average number of basic cells that a library cell consists of is 2 and the average number of resource types that can be used to implement a basic cell is 1.5.

In order to test our algorithm under complex resource constraints, we designed the experiments by using the structure of the netlists in the MCNC benchmark, rather than their actual logic functions. This is because each node in the benchmark circuits is only a simple gate type and each circuit has no more than twelve types of nodes, which is not enough for our purpose. To generate netlists with more types of basic cells, each node in the circuit is randomly mapped to the basic cells in the library.

Since no previous published partitioning algorithms have addressed the same partitioning problem with diverse resources constraints, the comparison of the min-cut size of FFC-fm with that of other papers is not available here. In Table 1, we compare the min-cut size of the FFC-fm algorithm with the *static* partitioning result when the resource allocation for each basic cell is pre-determined before the partitioning. For the *static* partitioning process, a basic cell only maps to one type of resource. In the FFC-fm partitioning process, a basic cell maps to multiple resources and it is necessary to apply the flow-based feasibility checking. In our experiments, both algorithms are run ten times to get the min-cut. The min-cut size closely depends on how tight the resource constraints are. The experimental results not only prove our analysis that the cut size can be substantially reduced, but also show when the resource constraints are tight, the FFC-fm method still produces feasible solutions even when the *static* algorithm fails. Network flow based feasibility checking has the advantage of dynamically adjusting the resource allocation for a subset when each subset is incrementally changed within each iteration of the partitioning process. This gives a node more chance to be moved from one subset to another, thus produces higher probability to reduce the cut size.

Efficiency is also of great concern for practical application of our algorithm. The feasibility checking is kept efficient by the incremental flow computation. As we merge all the equivalent basic cells as discussed in 3.1, the size of the network is almost constant for different netlists since it only depends on the number of types of basic cells and types of diverse resource, regardless of the size of a circuit.

Table 2: Average running time vs. feasibility checking time (*second*)

Circuit	$\lvert C'\rvert = 100, \lvert R\rvert = 5$ T_{tt}	T_{ck}	%$=\frac{T_{ck}}{T_{tt}}$	$\lvert C'\rvert = 150, \lvert R\rvert = 5$ T_{tt}	T_{ck}	%$=\frac{T_{ck}}{T_{tt}}$
c5315	0.52	0.26	45.8	0.45	0.20	45.5
c7552	0.61	0.28	41.9	0.62	0.27	44.2
c6288	0.70	0.31	40.9	0.89	0.44	49.2
s5378	0.80	0.38	42.5	0.81	0.34	42.3
s9234	1.61	0.757	42.8	1.92	0.96	50.2
s13207	2.49	1.19	43.5	3.01	1.41	47.0
s15850	3.03	1.45	43.5	3.11	1.40	44.9
s35932	5.16	2.23	39.4	5.40	2.18	40.5
s38584	7.81	3.51	44.0	6.39	2.63	41.1
s38417	7.44	3.32	45.1	8.20	3.89	47.5

Circuit	$\lvert C'\rvert = 200, \lvert R\rvert = 5$ T_{tt}	T_{ck}	%$=\frac{T_{ck}}{T_{tt}}$	$\lvert C'\rvert = 250, \lvert R\rvert = 5$ T_{tt}	T_{ck}	%$=\frac{T_{ck}}{T_{tt}}$
c5315	0.51	0.23	46.4	0.84	0.50	59.2
c7552	0.65	0.31	48.7	1.07	0.57	51.4
c6288	0.79	0.36	45.1	0.92	0.47	50.9
s5378	0.89	0.41	45.7	1.53	0.78	50.8
s9234	1.73	0.79	45.9	2.03	1.05	51.7
s13207	2.71	1.23	45.2	3.37	1.78	52.7
s15850	3.46	1.60	46.3	5.26	3.07	57.3
s35932	6.28	2.73	43.4	9.59	5.56	57.9
s38584	7.32	3.15	43.1	11.19	6.53	58.1
s38417	7.84	3.45	44.1	8.45	4.16	49.3

We tested our algorithm by assuming there are 5 resource types with varying capacity for each resource type, and a netlist contains 100, 150, 200 and 250 types of basic cells.

Table 2 shows the average total running time (T_{tt}) and the average time for feasibility checking (T_{ck}) in one pass of the FFC-fm algorithm. Recall that one pass in our partitioning method starts with a feasible initial partition and iteratively moves a node from one subset to another until all nodes are locked or no node can be moved. Feasibility checking is employed in each iteration to guarantee that the partitioning solution satisfies the resource constraints. We obtained the running time by running the algorithm with 20 different initial partitions and calculated the average time. The time for finding a feasible initial partition is not counted in T_{tt} in Table 2. From the experiments, the time for feasibility checking ranges from 40% to 60% of the total running time.

Compared with the FM-based method with simple area metric for balancing the two subsets, our algorithm increases the total running time by a reasonable amount, yet yields feasible partitioning results which meet the resource constraints.

5 Conclusion

In this paper, we propose a general problem formulation for circuit partitioning with complex resource constraints in FPGAs. With the emerging new FPGA architectures with diverse resources, the problem formulation for partitioning with complex resource constraint is more accurate than the simple gate count or area metric to estimate both the capacity of an FPGA (or part of an FPGA for a hierarchical partitioning inside one FPGA chip) and the resource requirement of a circuit.

We present a network flow based method for feasibility checking and then integrate the feasibility checking into the FM-based iterative improvement partitioning method, so that the partitioning results satisfy the complex resource constraints. We employ efficient implementation by using the incremental flow technique. Experiments show that our feasibility checking approach is efficient.

Recently, many improvements to the FM-based method have been proposed [8,9,16,17]. Our network flow based checking method can be integrated into those approaches. In our future research, we will explore the strategy of temporarily relaxing the resource constraints to benefit the mincut objective. A node can be moved even if it fails the feasibility checking, and the violations can be corrected in the future moves so that the final solution satisfies the resource constraints.

References

[1] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *Bell System Tech. Journal*, vol. 49, Feb. 1970, pp. 291-307.

[2] C. M. Fiduccia and R. M. Mattheyses, "A linear-time Heuristic for improving network partitions", *Proc. ACM/IEEE Design Automation Conference*, 1982, pp. 175-181.

[3] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, Jr. "Optimization by Simulated Annealing", *Science*, pp.671-680, 1983.

[4] Y. C. Wei and C. K. Cheng, "Towards Efficient Hierarchical Designs by Ratio Cut Partitioning", *Proc. International Conference on Computer-Aided Design*, 1989, pp.298-301.

[5] Y. C. Wei and C. K. Cheng, "An Improved Two-way Partitioning Algorithm with Stable Performance", *IEEE Trans. on Computer-Aided Design*, 1990, pp.1502-1511.

[6] C. J. Alpert and A. B. Kahng, "Recent Directions in Netlist Partitioning: a Survey", *the VLSI Journal*, pp.1-81, 1995.

[7] C. J. Alpert and S. Z. Yao, "Spectral Partitioning: The More Eigenvectors, the Better", *Proc. ACM/IEEE Design Automation Conference*, pp.195-200, 1995.

[8] S. Dutt and W. Deng, "VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques", *Proc. ACM/SIGDA Physical Design Workshop*, pp.92-99, 1996.

[9] S. Dutt and W. Deng, "A Probability-based Approach to VLSI Circuit Partitioning", *Proc. Design Automation Conference*, 1996.

[10] Honghua Yang and D.F. Wong, "Efficient Network Flow Based Min-Cut Balanced Partitioning", *Proc. ICCAD 1994,* pp50-55.

[11] Jianmin Li, John Lillis and Chung-Kuan Cheng, "Linear Decomposition Algorithm for VLSI Design Applications", *ICCAD'95,* pp223-228.

[12] Pak K. Chan, Martin D.F Schlag and Jason Y. Zien, "Spectral-Based Multi-Way FPGA Partitioning", *FPGA'95,* pp133-139, Monterey, CA.

[13] N.C. Chou, L.T. Liu, C.K. Cheung, W.J. Dai and R. Lindelof, "Circuit Partitioning for Huge Logic Emulation Systems", *31th ACM/IEEE Design Automation Conference,* pp244-249, CA, June 19 94.

[14] C. Sechen, *VLSI Placement and Global Routing Using Simulated Annealing*, Kluwer, B.V., Deventer, the Netherlands.

[15] J.R. Ford and D.R. Fulkerson, *Flows in Networks,* Princeton University Press, 1962.

[16] Jason Cong, Honching Peter Li, Sung Kyu Lim, Toshiyuki Shibuya and Dongmin Xu, "Large Scale Circuit Partitioning with Loose/Stable Net Removal and Signal Flow Based Clustering", *International Conference on Computer-Aided Design*, 1997.

[17] Shantanu Dutt and Halim Theny, "Partitioning Around Roadblocks: Tackling Constraints with Intermediate Relaxations", *International Conference on Computer-Aided Design*, 1997.

[18] Actel FPGA Data Book and Design Guide, *Actel Corporation*, 1996.

[19] Actel's Reprogrammable SPGAs, Preliminary Advance Information, *Actel Corporation*, October 10, 1996.