Delay Driven Multi-way Circuit Partitioning

Wong Sze Hon

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science & Engineering

August, 2003

Abstract

Integrated Circuit (IC) is an essential component of many computing facilities today. In order to achieve complicate functionalities, one single chip inside a computer is actually comprised of millions of transistors. This makes the chip design process a lot more difficult and complicated than before. As huge circuits are hard to be managed efficiently, decomposition of complex systems into finer sub-systems is important in the design cycle. After decomposition, each sub-system can be designed and further improved independently and simultaneously to make the design process faster and simpler. Therefore, partitioning is an important technique used in the design cycle.

Circuit partitioning plays an important role in the physical design cycle. A good partitioning of a system will lead to feasible solutions in the succeeding processes such as floorplanning, placement and routing. There are several aspects to be considered in the partitioning process, such as the interface connections between sub-circuits, the delay of the critical path and the size of each partition, etc.

We studied some previous works on circuit partitioning, and focused on their performance on cut-size and delay minimization. We found that acyclic partitioning is an effective way to upper bound the largest number of inter-partition delays along any path. Therefore, in this thesis, two approaches are proposed to solve the acyclic multi-way circuit partitioning problem. The first

one is a clustering based approach. We developed a new acyclic multi-way partitioning algorithm. A modified fanout free cone decomposition is used to pre-cluster a given network, followed by another similar decomposition process to further partition the clustered network. The second one is a network flow based approach. We proposed a net modelling method to ensure an acyclic partitioning when the max-flow min-cut algorithm is applied. The basic idea of these two approaches is to maintain an acyclic partitioning on all the combinational paths. Both of these two approaches aim at minimizing the number of cuts along the critical paths. Experiments are carried out to investigate the performance of these proposed approaches. Results show that both of them are competitive with many current existing algorithms.

摘要

現今很多電子產品都有應用到集成電路的技術。一塊細小的晶片，其實是由數百萬的電晶體所組成的。隨著科技發展，人們對電子產品的要求越來越高，電子產品的內部結構日益複雜，故晶片的體積成為重要的設計考慮。然而，設計複雜的晶片並不容易。由於設計員很難同一時間管理與設計龐大的電路，所以將系統分割成為多個次系統是必要的。這樣，設計隊伍便能同時有效和獨立地設計和優化各次系統。故集成電路分割是在晶片設計中不可或缺的技巧。

集成電路分割技術在整個設計過程當中扮演一個很重要的角色。優良的分割技術能令到往後的步驟如佈局規劃、配置和繞線得到成功的結果。在集成電路分割過程中，設計隊伍須要考慮次系統間的交叉連結、訊號的最大延遲和各次系統的大小等問題。

我們研究過一些已有的電路分割技術，特別集中在縮減次系統間的交叉連結和改善訊號延遲等技術。我們發現非循環式的次系統分割能有效地減少因分割次系統而帶來的訊號延遲。我們在這論文中提出了兩個方法，第一個方法建基於聚類技術。我們開發了一個全新的非循環式多方向分割方法，先用零輸出端圓錐分解來群集系統，然後再用類似的手法更進一步的分割該系統。第二個方法建基於網絡流技術。我們提出的網絡塑型可保証當引用「最大流最小切」方法時，會得到一個非循環式的分割。本論文提出的兩個方法旨在令所有組合式的路徑非循環，而維持最少的臨界路徑被切次數。對比現有的分割技術，實驗結果證明我們提出的方法能更有效地找到優良的分割。

Acknowledgments

I must take this chance to thank Professor Young Fung Yu, Evangeline, who helped a lot in my work. Her constant encouragement and advises made me possible to finish this work. She gave critical remarks and show me directions sometimes. She is more than an advisor to me.

I also wish to express my sincere thanks to my markers in both years, Professor Lee Kin Hong and Professor Wu Yu Liang, David. They have given me invaluable advices and constructive suggestions to improve my work.

Thanks should also be given to my colleague, Steve, who has given me opinions and encouragement time over time. Thanks to my friend, Jill, who proof read my work several times, to improve the quality of the text.

Finally, I wish to thank my parents and my family for supporting me all the time. Thank them for being there when I needed them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Preliminaries

Today, the sizes of computers or electronic appliances are becoming smaller and smaller. It would be difficult to imagine that a tiny chip is actually consisted of billions of transistors. The advanced technology on fabrication of VLSI (Very Large Scale Integration) circuits has made the size of a chip as small as a nail possible. Indeed, it is a complicate process from the design to the fabrication of a single chip. It involves a large number of steps and a huge amount of computational power. In this thesis, we will concentrate on one important aspect in the physical design process, which is, the circuit partitioning problem.

1.2 Motivations

In VLSI system design, it is common that a system is consisted of millions of transistors. Such a huge circuit is hard to be managed efficiently. As a result, decomposition of these complex systems into finer sub-systems is important. Each sub-system can then be designed and further improved independently and simultaneously to make the design process faster and simpler.

Partitioning plays an important role in the physical design cycle of VLSI circuits. A good partitioning of a system will lead to feasible solutions for the succeeding processes such as floorplanning, placement and routing. As the interconnection delay between partitions are relatively large and not preferable, we usually need to reduce the number of interface connections between subsystems, which is called the min-cut problem in the partitioning process. A min-cut partitioning can minimize the number of interconnections. However, one combinational path may be cut by the partitions several times. As the overall delay of a circuit is the delay along the critical paths, multiple cuts on a single combinational path will affect the delay of the whole circuit. Therefore, the number of cuts along a path is also an important aspect to be considered beside the cut size. In order to improve the performance of the circuit, the number of partitioning cuts along each path must be considered.

Acyclic partitioning is an effective way to upper bound the largest number of inter-partition delay along any path. The acyclic multi-way partitioning problem was defined in [6]. It differs from the general partitioning problem because it restricts the edges between different partitions from forming a directed cycle. An acyclic partitioning ensures that all paths are cut by the partitions by at most $k-1$ times only where k is the number of partitions. This can effectively reduce the delay caused by the partition cuts. Although the number of partition cuts along any path is limited, the partitioning solution may be over constrained. As the overall delay of a circuit is defined by the delay along the longest combinational path, some solutions with better cut sizes may be eliminated because of the acyclic constraints. Some move based algorithms like FM can be applied in such a process to improve the cut size as a post-processing step.

1.3 Contributions

We studied some previous works on circuit partitioning. In order to reduce the delay of the circuit, the number of cuts along each path must be considered. We proposed two approaches to solve the problem. One is based on the clustering technique, and the other is based on the network flow technique. The basic idea of these two approaches is to maintain an acyclic partitioning on all the combinational paths. Both of these two approaches aim at minimizing the number of cuts along the critical path.

In the first method, a clustering based approach is used. We developed a new acyclic multi-way partitioning algorithm. A modified fanout free cone decomposition is used to pre-cluster a given network. This decomposition effectively reduces a given network to a smaller and sparser one and maintain the acyclic property of the network. After that, a modified version of this decomposition step will be used again to further partition the clustered network into the desired number of partitions. The size constraint is set to the predefined partition size in the partitioning phase as we want to fill up each partition as much as possible.

The second method is a network flow based approach. We proposed a net modelling method to ensure an acyclic partitioning when the Max-Flow Min-Cut algorithm is applied. We successfully limit the number of cuts of each combinational path to $k-1$ by applying the Max-Flow Min-Cut algorithm where k is the number of partitions. Then a FM post processing improvement step is applied on the resultant partitioning to further improve the overall cut-size. Our proposed algorithm can reduce the delay of the circuit caused by the repeated inter-partition cut efficiently while considering the cut-size at the same time.

1.4 Organization of the Thesis

After this brief introduction, there will be an overview of the VLSI physical design cycle in Chapter 2. It includes some background knowledge about VLSI design cycle and physical design cycle.

In Chapter 3, we will describe some recent approaches on circuit partitioning. It starts with a study of circuit representation, delay modelling and partitioning objectives. Some typical partitioning algorithms will be revised. Delay driven circuit partitioning and acyclic partitioning will also be discussed.

In Chapter 4, our clustering based approach to solve the acyclic multi-way partitioning problem will be presented. We will start with an introduction to some existing approaches and techniques in clustering based partitioning. Then, an overview of our approach will be provided, followed by the details of our algorithm. The experimental results and a conclusion will be presented at the end.

In Chapter 5, we will present our network flow based partitioning algorithm. Previous works on net modelling and network flow based partitioning will be introduced. We will then propose our net modelling for acyclic partitioning. The properties and effects of our net modelling method will be discussed. Lastly, the experimental results and a conclusion will be given.

A conclusion of this thesis will appear in Chapter 6.

Chapter 2

VLSI Physical Design Automation

2.1 Preliminaries

The applications of computing facilities are more and more common today. It is not surprising to find that we are surrounded by a huge number of computer related machines in daily life, such as our personal computers, the ATM machines which were commonly adopted, and many electronic appliances. Integrated Circuits (IC) are one of the essential components of those computing facilities. To perform complicate calculations, the chips inside a computer consists of millions of transistors, which are refereed as Very Large Scale Integration (VLSI) chips. As the complexities of computers increase from time to time, the number of transistors involved increases accordingly. On the other hand, it is favorable to keep the size of the chips as small as possible. Therefore, the problems encountered during the chip design process has become more and more difficult and complicate.

In a VLSI design process, several steps will be involved. Details will be discussed in Section 2.2. As physical design is a very important step in the VLSI design cycle, we will focus on it in Section 2.3, followed by a summary

in Section 2.4.

2.2 VLSI Design Cycle [1]

To produce a packaged chip, we need to go through a series of steps. A flow chart of the VLSI design cycle is shown in Figure 2.1. Details of each step are described as follow:

2.2.1 System Specification

It is necessary to list out important aspects of the system, such as what the system can perform and what constraints are required to be satisfied. These aspects include the functionality of the system, the size constraints of the system and the expected performances, etc. Engineers will work on the subsequent design to fulfill this specification.

2.2.2 Architectural Design

This is a step to design the basic architecture of the system, such as how many ALUs will be used, and what is the size of the cache, etc. Architects can estimate the power consumption and the system performance basing on this architecture, which can help to estimate whether the specification can be met.

2.2.3 Functional Design

This is a step to specify the behavior of the system, without specifying the detailed internal design. This can be done by stating the inputs and outputs of each unit. Besides, interconnections between different units will also be defined.

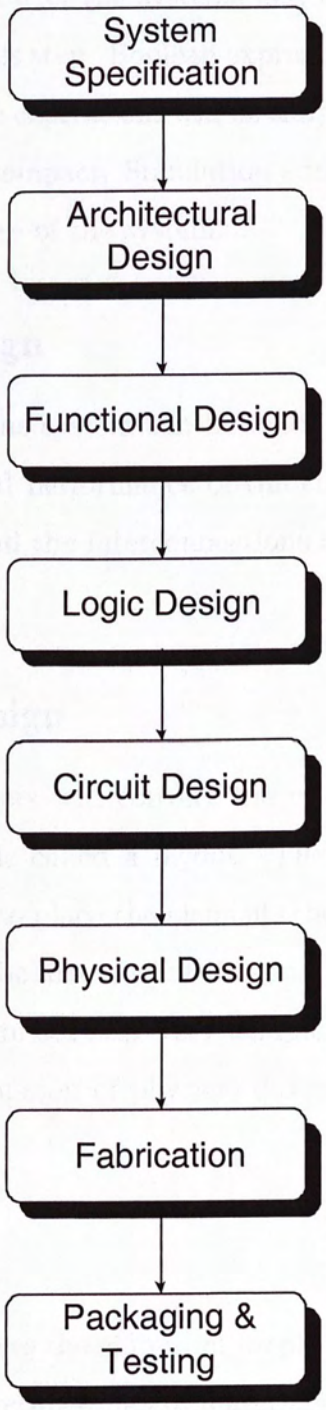


Figure 2.1: VLSI Design Cycle

2.2.4 Logic Design

Aspects like the control flow of the systems and the logic operations in each circuit will be defined in this step. Boolean expressions will be used to describe the logic operations. Those expressions will be simplified as much as possible in order to make the design compact. Simulation and testing will also be carried out to verify the correctness of the system.

2.2.5 Circuit Design

Circuit representation of the system can be made basing on the logic design. In this step, the speed and performance of the circuit will be taken into account. Circuit elements and the interconnections between these elements will be defined.

2.2.6 Physical Design

In physical design, engineers will convert the circuit design into a geometric representation, which is called a layout. There are many aspects to be considered, such as where to place the elements, how the interconnections between the elements should be made, what the most favorable dimensions of the chip is, etc. These steps are actually very complex and each involves several sub-steps. A detailed discussion of physical design can be found in the next section.

2.2.7 Fabrication

This is the step that produces the chip from its physical design. The electronic components are built by layering different materials onto a base made of silicon, which is called a wafer. A large wafer can be used to produce many chips. Prototypes will first be made for testing before mass production of the chips

is carried out.

2.2.8 Packaging and Testing

After testing and checking the prototype, the chips will be mass produced. Before packaging each individual chip, a final verification will be performed to ensure that all the requirements are fulfilled and the chip is functioning properly.

2.3 Physical Design Cycle [1]

Given a circuit representation of the design, engineers will try to produce an exact layout of the design in the physical design step. The steps involved in physical design are shown in Figure 2.2 and details are discussed as follows:

2.3.1 Partitioning

Breaking down a big problem into smaller sub-problems is always a good strategy to solve complex problems. As the complexity of the chip design process increases, it is nearly impossible to design the whole chip all at once. Thus, in the first step of physical design, engineers attempt to partition the circuit into sub-circuits, which are called blocks, in order to make the design process simpler and more efficient. After the decomposition, each sub-circuits can then be designed and managed simultaneously and efficiently, which can greatly reduce the complexity of the subsequent designing steps. Factors like the sizes of the blocks, the dimensions of the blocks and the interconnections between different blocks should be taken into account.

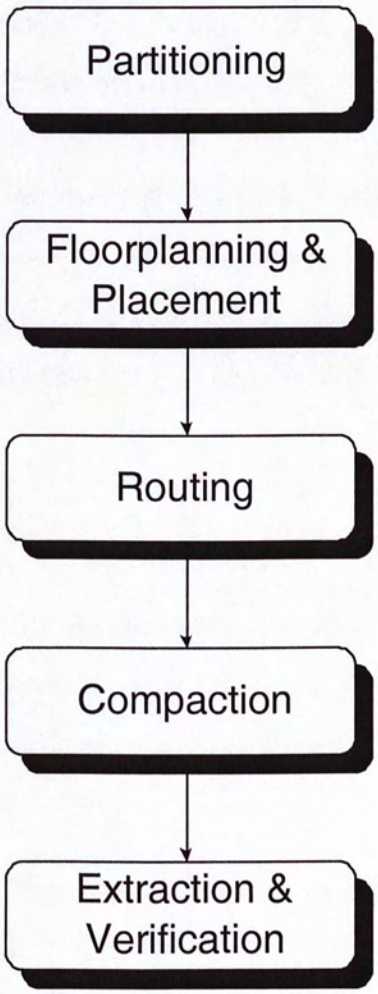


Figure 2.2: Physical Design Cycle

2.3.2 Floorplanning and Placement

Placing the blocks in such a way that all the constraints in area, block dimensions, interconnect length and delay, etc., are satisfied is the main concern in this step. Floorplanning is the planning step to design how to place the blocks. A compact design is favorable, but there are many important aspects that should also be considered. For example, issues like the dimensions of each block and the overall delay should also be taken into account.

After the planning step, the blocks will be placed exactly onto the chips and this step is called placement. After this step, the dimensions of each block and their positions are fixed. Floorplanning and placement are important as it affects the ultimate design significantly and determines whether the required specifications can be met.

2.3.3 Routing

This step aims at completing all the interconnections between the blocks. Objectives like minimizing the total wire length, minimizing the number of vias and minimizing the critical delay etc. should be considered. This step can be further divided into two sub-steps:

1. **Global Routing**: Planning different routes in a global view, without fixing the exact path of each route. It is a rough plan to check whether completing all interconnections is possible.

2. **Detailed Routing**: Complete each connection by giving exact information such as the exact positions of the wires on the metal layers. After detailed routing, the geometric layouts of all the nets will be known.

There may be cases in which some of the connections cannot be routed. In such a situation, the technique rip-up and re-route, i.e. removing some of

the routed connections and re-route them in a different order, will be used. If there are still connections that cannot be routed at the end, engineers may need to go back to the earlier steps in the physical design cycle or even to the login designing step and start the whole process all over again.

2.3.4 Compaction

As mentioned before, it is desirable to have a chip design as small as possible. In this step, the layout will be compressed from different directions in order to make the total area smaller. As the total area is smaller, the wire lengths will be reduced, and thus the delay will also be reduced. During the compaction process, rules regarding the design should be checked to make sure that there is no violations.

2.3.5 Extraction and Verification

As the whole design process is an extremely complex process, verification of each step is a must to ensure that everything works well before proceeding to the fabrication step. It is necessary to make sure that no design rules is violated in the final layout, such as the wire separation rule and the aspect ratio rule etc. Besides, the functionalities of the circuits should also be verified before proceeding to the next step. If any problem is discovered, engineers may need to go back to the earlier designing steps to fix the problem.

2.4 Chapter Summary

Producing a thumb-big chip is a time consuming process. There are many steps to go through, and many of which are computational expensive. Many algorithms have been developed in CAD (Computer Aided Design) tools to help accomplishing the tasks, but there are still many unresolved problems

and new challenges to be explored.

Physical design is a critical stage in the VLSI design process. In the physical design process, circuit partitioning is a crucial step and our research will be focused on performance driven circuit partitioning.

Chapter 3

Recent Approaches on Circuit Partitioning

3.1 Preliminaries

Circuit partitioning is a critical stage in VLSI design. Today, it is usual that a circuit contains several millions of transistors. The huge size makes the circuit designing problem not human manageable. Circuit partitioning is used to break down a complex system into smaller subsystems, such that the designers can manage the simpler subsystems separately. Each subsystem can be designed independently to speed up the design process. Partitioning will affect the performance of the whole circuit, such as the delay caused by inter-partition connections. Good partitioning techniques can improve the overall performance of the circuit. As the size and complexity of VLSI designs has increased rapidly in recent years, the development of good partitioning algorithms and tools are essential.

In this chapter, we will present some background on circuit partitioning like circuit representation and delay modelling. Some commonly used partitioning objectives will be discussed, which is then followed by several well-known partition algorithms.

3.2 Circuit Representation

In order to apply a partitioning algorithm to a given circuit, we must first transform the circuit into a suitable representation. Recent approaches usually attempt to solve the circuit partitioning problem by representing the circuit as a graph. Graph is a straight forward representation of a network of gates or modules, and the circuit partitioning problem can then be solved by some graph partitioning methods in this way. A graph $G(V, E)$, where V is a set of nodes representing the circuit components such as logic gates, flip-flops, inputs and outputs, and E is a set of edges representing the nets in the circuit connecting the circuit components. A feasible graph representation is illustrated in Figure 3.1.

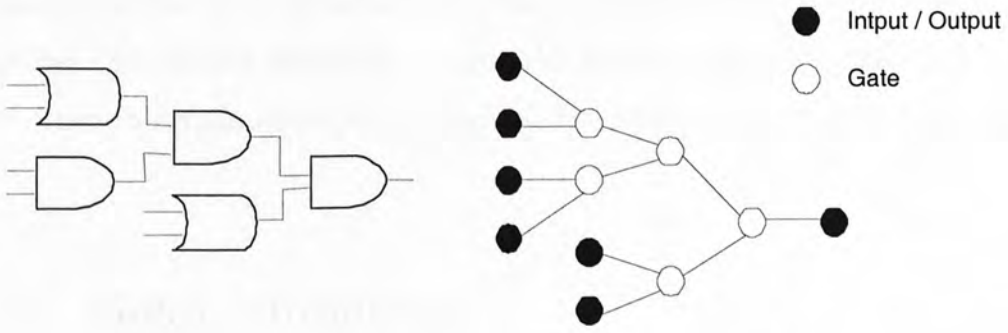


Figure 3.1: Graph Representation of a Circuit

Besides the basic circuit information, the signal flow direction can also be integrated into the graph representation. It can be done by changing the edges in the graph to directed edges. A directed graph representation is essential in some partitioning problems such as unidirectional cut partitioning problem,

acyclic partitioning problem, timing driven partitioning problem, etc. A directed graph representation is illustrated in Figure 3.2.

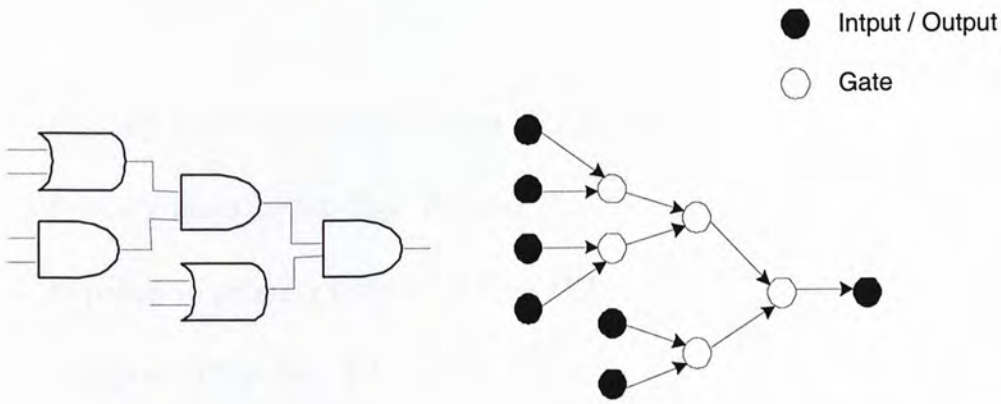


Figure 3.2: Directed Graph Representation of a Circuit

A circuit partitioning problem of aims at separating the set of nodes into two or more disjoint subsets while optimizing the objective function. The most typical objective is to minimize the total number of inter-partition edges. It is called the min-cut objective. Beside the min-cut objective, there are several other objectives which will be discussed in details in the following sections.

3.3 Delay Modelling

The delay of a partitioned circuit is an important aspect to be considered in circuit partitioning. In most high performance systems today, the partitioning algorithm is required to be performance driven, that is, to minimize the overall circuit delay. In order to evaluate the delay of a partitioned circuit, an appropriate delay model is needed. In this section, we will introduce some of them.

The delay of a combinational circuit is the longest delay among all the paths from a primary input to a primary output. The delay of a sequential circuit is the longest delay among all the combinational paths of one of the following four types:

1. Primary input to primary output: $PI \rightarrow PO$
2. Primary input to flip-flop: $PI \rightarrow FF$
3. Flip-flop to primary output: $FF \rightarrow PO$
4. Flip-flop to flip-flop: $FF \rightarrow FF$

There are two basic models for estimating the delay of a circuit. They are the unit delay model [7] and the general delay model [8]. In the unit delay model, all gate delays are assumed to be zero. The interconnections connecting gates in the same partition are assumed to have zero delay. The interconnections connecting gates in two different partitions are assumed to have one time unit delay. In the general delay model, each gate has an intrinsic gate delay. Again, the interconnections connecting gates in the same partition are neglected. A delay of D time units, where D is a constant, is accounted for each interconnection connecting gates in two different partitions. An example is shown in Figure 3.3.

We can deduce the delay of a combinational circuit using the two delay models as discussed in [9]. Each node v has an intrinsic gate delay $i(v)$. For each $v \in V$, $d(v)$ is defined as the maximum delay along any path starting from a primary input node and ending at v. $d(s)$ is zero when $s \in PI$ where PI is the set of primary input nodes. The delay values of all the other nodes can be calculated using the following formula:

$$d(v) = \max_{u \in fanin(v)} \{d'(u)\} + i(v)$$

where $d'(u) = \left\{ \begin{array}{ll} d(u) & \text{if } u \text{ and } v \text{ are in the same partition or } u \text{ is a PI} \\ d(u) + D & \text{if } u \text{ and } v \text{ are not in the same partition} \end{array} \right\}$

The notation $fanin(v)$ represents the fan-in of node v. If one of the nodes is a primary input (PI) or a primary output (PO), there is no need to add the value D to it. It is because we assume that primary input and output nodes can physically be assigned to any partition without affecting the area and delay. By applying this calculation recursively until all nodes are visited, the delay of a combinational circuit can be computed which is equal to the maximum value of $d(v)$ where $v \in PO$.

As mentioned before, the delay of a sequential circuit is the longest delay among all the combinational paths of one of the following four types: $PI \rightarrow PO$, $PI \rightarrow FF$, $FF \rightarrow PO$ and $FF \rightarrow FF$. The delay of a sequential circuit can be computed using the method for combinational circuit by replacing each FF by a PI node and a PO node. Replace each edge that is connected to the FF node originally by an edge connecting to the corresponding PO node and replace each edge that is connected from the FF node originally by an edge connecting from the corresponding PI node. The circuit is reduced to a combinational one in this way. By computing the maximum delay among all the combinational paths, we can deduce the delay of the original sequential circuit.

3.4 Partitioning Objectives

In this section, we are going to introduce some constraints and objectives for the circuit partitioning problem. In real life, we need to deal with one or more of the following parameters at the same time.

3.4.1 Interconnections between Partitions

In a partitioned circuit, the delay between partitions are considered relatively large and not preferred. Also, the number of terminals in each partition is limited. In order to satisfy these constraints and requirements, circuit partitioner must consider the impact of the number of interconnections between partitions. The number of interconnections between partitions should be minimized and this is the most typical objective in circuit partitioning. We call it the min-cut partitioning problem.

3.4.2 Delay Minimization

The performance of a circuit depends on the delay along the critical paths of the whole circuit. As discussed before, the delay between partitions are considered relatively large and not preferred. Although the cut-size minimization objective can reduce the probability that a critical path crosses the partitions multiple times, it does not guarantee any delay minimization. For the objective of delay minimization, we are required to minimize the number of inter-partition cuts along the critical paths.

3.4.3 Area and Number of Partitions

If we had no limitations on the area and the number of partitions, the resulting partitions would be too large or too small, or there would be too many partitions. If a partition is too large, the entire partition cannot be fitted into a chip or a module. If a partition is too small, there will be a wastage of resources. On the other hand, a large number of partitions may result in an increase in the number of inter-partition interconnections. A small number of partitions may leave the design of the sub-circuit still too complex for handling. In circuit level design, partitions of balanced-sizes are preferred, given the required number of partitions.

3.5 Partitioning Algorithms

Many algorithms have been proposed to deal with the partitioning problem. In this section, we will review some well-known algorithms to solve the circuit partitioning problem. we will classify the algorithms according to their partitioning objectives. The first one is cut-size driven partitioning. It is the most typical problem in circuit partitioning. It aims at minimizing the number of interconnections between partitions. The second one is delay driven partitioning. It aims at minimizing the delay of the final partitioned circuit. The third one is acyclic partitioning problem. It aims at producing an acyclic partitioning solution. The delay driven partitioning problem and the acyclic partitioning problem are closely related to our research which will be introduced in Chapter 4 and Chapter 5.

3.5.1 Cut-size Driven Partitioning Algorithm

In this section, we will review several well-known cut-size driven algorithms. The objective of these algorithms is to obtain two balanced partitions with the cut-size minimized or with a ratio balanced cut-size between the two partitions minimized.

Firstly , we will introduce some iterative improvement approaches, the Kernighan-Lin (KL) Algorithm [10] and the Fiduccia-Mattheyses (FM) Algorithm [11]. They are all iterative improvement algorithms using greedy strategy. KL and FM starts with an initial partitioning, and components are then moved between the partitions to improve the cut-size. The process stops when a local minimum is reached. As the solution of the KL and FM algorithm may be trapped in local minima, multiple trials with different initial partitions are needed for higher quality solutions.

After that, we will introduce some stochastic searching approaches. They are the Simulated Annealing approach and the Genetic Algorithm. Simulated Annealing (SA) is a general purpose searching technique. It mimics the process of metal cooling and freezing into crystalline structure with minimum energy *(the annealing process)*. The current solution is updated until a terminating condition is reached. Genetic Algorithm (GA) is a class of searching method inspired by genetic evolution. It simulates the natural selection process in evolution to locate good solutions. It starts with a set of random solutions (*population*). The population evolutes over generation and is replaced by the offspring in the next generation. The population is refined gradually during the evolutional process.

Then, we will introduce the network flow based approaches. The network

flow based approaches use the technique and characteristics of the Max-Flow Min-Cut computation to solve the partitioning problem.

A number of heuristic steps have been developed to improve the partitioning algorithms. They include logic replication [4, 12, 13], multilevel approach [2, 14] and clustering [2]. In logic replication, some nodes are selected and duplicated in two or more partitions in order to reduce the cut-size. In multilevel approach, a sequence of successive grouping steps will be applied to the nodes in the circuit until the number of nodes is smaller than a given threshold. After the grouping steps, a move based partitioning algorithm such as FM will be applied to the clustered circuit to obtain a bisection. The next step is to un-group the nodes at the highest level in the partitioned result and use it as the initial solution for the next FM computation. Clustering [2] is another useful pre-processing step that groups the nodes in a network into clusters. This can significantly reduce the problem size and produce a simpler clustered network before applying a partitioning algorithm.

Kernighan-Lin (KL) Algorithm [10]

The Kernighan-Lin (KL) algorithm is based on pairwise swapping of cells between two partitions in order to maximize the gain after swapping. The gain of a pairwise swapping is the change in the number of inter-partition connections after the swapping. The greater the gain, the smaller the cut-size is resulted after the swapping. The KL Algorithm is the first well known and widely extended partitioning heuristic. It is a local searching algorithm and produces partitions of equal size. The gain of swapping a pair of cells a and b where $a \in A$ and $b \in B$ is defined as follow:

$$G_{ab} = D_a + D_b - 2c_{ab}$$

$$
\begin{aligned}
where \quad c_{ab} &= \text{cost between cell } a \text{ and cell } b \\
D_i &= \text{External cost of cell } i \text{ - Internal cost of cell } i \\
&= \textstyle\sum_{y \in B} c_{iy} - \sum_{x \in A} c_{ix} \\
&\text{(Assuming that cell } i \text{ is in partition } A)
\end{aligned}
$$

The algorithm starts with an edge weighted graph which is partitioned into two subsets (A, B) initially, where $|A| = |B| = n$. The algorithm will work in an iterative manner. The gain value for each pair (a, b) where $a \in A$ and $b \in B$ is first computed. The pair having the maximum gain will be exchanged temporally and locked, so that the two swapped nodes will not be swapped again. The gain value g_i of the selected pair will be recoded and the new gain values for those remaining free nodes will be updated. Then, the next pair of nodes with the maximum gain will be selected and exchanged. This swapping process will be repeated until all nodes are locked. Finally, a k is chosen to maximize the value of $G = \sum_{i=1}^{k} g_i$. If $G > 0$, it means that a reduction in the number of inter-partition connections can be made by swapping the selected nodes between A and B. Permanent swapping will then be performed up to and including step k to maximize the overall gain. A new partition is obtained and this is defined as one pass. In the next pass, the same process will be repeated using the result of the previous pass as the starting partition. These passes will be performed until there is no further improvement in cut-size (i.e., $G \leq 0$). The time complexity of a simple implementation of the KL Algorithm is $O(n^3)$ per pass.

The KL Algorithm has been widely used in industry because of its effectiveness and its simple implementation. However, the time complexity of the

algorithm is quite high. In addition, this approach is not flexible for unbalanced partitioning since the KL algorithm will always result in a balanced partitioning.

Fiduccia-Mattheyses (FM) Algorithm [11]

In view of the drawbacks of the KL algorithm, Fiduccia and Mattheyses presented a new algorithm to modify KL Algorithm in 1982. In the new algorithm, the run time of one pass is reduced to linear. Similar to KL, FM starts with a balanced partition (A, B). It moves a cell to the opposite partition in each step instead of swapping a pair of cells as in the KL algorithm. The algorithm works in an iterative manner. The gain value of each cell is first calculated and the cell with the highest gain value will be moved to the opposite partition. One important feature of the major characteristic of the FM algorithm is that it considers the gain updates of the critical nets only. We will discuss in details the definition of a critical net and the gain calculation process later. A free cell with the highest gain will be chosen to be moved to the opposite partition temporally and locked if the balance ratio constraint is preserved after the move. The value of the balance ratio r is specified by the user and is defined as $|A|/(|A| + |B|)$ for a partitioning (A, B). To maintain the balance ratio constraint, the following situation must be satisfied after each move:

$rW - S_{max} \leq |A| \leq rW + S_{max}$
where $W = |A| + |B|$ and S_{max} *is the maximum size of a cell*

If the movement is allowed, the gain value g_i of such move will be recorded. The process continues until all the cells are locked. Similar to the KL algorithm, a k will be chosen to maximize the value of $G = \sum_{i=1}^{k} g_i$. Permanent swapping will be performed up to and including the step k to maximize the

overall gain. A new partitioning is obtained and this is defined as one pass. In the next pass, the same process will be repeated using the result of the previous pass as the starting partition. These passes will be performed until there is no further improvement in cut-size (i.e., $G \leq 0$).

One important observation of the FM algorithm is that the number of gain update due to a net is limited. For a bipartition (A, B), a net is regarded as a critical net if the move of a cell on it will result in a change in cut-size due to this net. Notice that a net is critical either $A(n)$ or $B(n)$ is equal to 0 or 1, where $A(n)$ and $B(n)$ are the number of cells of net n in partition A and B respectively. Besides, a net will never critical again if it has one locked cell on each partition.

The authors have proved that no more that four update operations are performed for each net in one pass of the algorithm. An example is shown in Figure 3.4.

In KL, the dominant factor in determining the time complexity is the selection process in finding the pair of modules with the largest gain. This takes $O(n^2)$ time. In FM, two sorted bucket lists are maintained to improve the runtime complexity of the cell selection process. Besides, it is proved that the number of updates in one pass due to a net is upper bounded by a constraint. The runtime of one pass of the FM algorithm can be done in $O(m)$ time where m is the number of net. Besides, the single cell movements in FM provides flexibility for unbalanced partitioning.

Simulated Annealing (SA)

Simulated Annealing (SA) is a general purpose searching technique. It mimics the process of metal cooling and freezing into crystalline structure with minimum energy *(the annealing process)*. The algorithm was originally proposed in [15] for finding the equilibrium configuration of a collection of atoms at a given temperature. The idea of using SA as an optimization tool is introduced in [16]. However, it is suggested in [17] and the author proposed to use it as a general technique for different optimization problems.

In [18], the authors apply the Simulated Annealing technique in the circuit partitioning problem. Given a randomly generated partitioning solution, a node is selected randomly to move from one partition to another partition in each iteration of the annealing process . The gain of a move is defined by the following ratio cut formula which is proposed in [19]:

$Gain = cutsize/(|A| \cdot |B|)$
where $|A|$ *and* $|B|$ *is the size of the two partitions*

If the movement gives a better gain ($Gain_{new} - Gain_{old} < 0$), the move will be accepted. However, if the movement does not give a better gain, SA will still accept the move with probability $e^{-\delta/T}$, where δ is the change in the gains of the two solutions, and T is the current temperature value, which is controlled by the cooling rate of the annealing process. This is defined as one iteration. SA can prevent the candidate solution from trapping in a local minima by accepting a worse solution with a certain probability. The temperature value T is a function of the number of moves performed. After each iteration, T will be scaled down by a cooling faction α where $0 < \alpha < 1$. The algorithm stops if there is no changes in the gain of the solution after t iterations.

Genetic Algorithm (GA)

Genetic Algorithm is a class of searching methods inspired by genetic evolution. It is initialized by Darwin's theory of natural selection of evolution [20]. *GA* starts with a set of random solutions (population) of the problem. Unlike other searching techniques, it operates on a population of solutions instead of a single solution. The population evolves over the generations and is iteratively replaced by the offspring in the next generation.

The use of Genetic Algorithm to solve the partitioning problem is proposed in [21]. The authors suggested to encode a partitioning solution as a binary string of C genes where C is the number of nodes in the graph. Each gene represents the partition to which each node belongs. For example, the string [100101] represents a graph of six nodes where node 1, 4 and 6 are assigned to partition one, and node 2,3 and 5 are assigned to partition two. As stated before, GA starts with a set of random solutions (population). In the partitioning problem, we need to ensure that the random solutions in the population represent balanced partitionings.

In each iteration, two solutions are selected from the population as the parents. The probability for selecting an individual as a parent is proportional to its fitness value. The fitness value F_i of a solution i is defined as follow:

$$F_i = (C_w - C_i) + (C_w - C_b)/3$$

where C_w is the largest cut-size in the population
C_b is the smallest cut-size in the population
C_i is the cut-size of solution i.

The two selected solutions, which are called parents, will then crossover and mutate to give a new solution. During the crossover, the parents will be mixed partially to generate a new solution. In such a process, an unbalanced partitioning solution may be resulted (number of ones $\neq$ number of zeros). The mutation process will then be applied to randomly complement some bits of the solution to make it a valid balanced partitioning. The new solution will be added to the population while the solution with the lowest fitness values will be removed from the population. By repeatedly applying this process to the population, the quality of the population will be improved. The algorithm stops when there is no improvement after N generations where N is given by the user and the final solution will be the smallest cut-size solution C_b in the population.

Network Flow Approach

The network flow based approach makes use of the Max-Flow Min-Cut algorithm to partition a circuit. The network flow technique can find a min-cut bipartition which is not necessarily balanced. The time complexity of the Max-Flow Min-Cut computation is $O(|V||E|)$ where $|E|$ is the number of edges and $|V|$ is the number of vertices.

In order to apply the Max-Flow Min-Cut algorithm in circuit partitioning, we must first model the circuit in such a way that the Max-Flow Min-Cut computation can be applied. This is called net modelling. We can model a circuit by a graph G such that when we apply the network flow algorithm, the min-cut in G is equal to the min-cut in the real circuit. Besides, only a two-way partition can be obtained in applying the Max-Flow Min-Cut computation once, a general method to obtain a balanced or multi-way partitioning is by performing the Max-Flow Min-Cut algorithm recursively. However, this

approach may increase the time complexity. Some good heuristics can be used to generate a balanced partition or an r-balanced partition. In the method proposed in [22], after applying the simple Max-Flow Min-Cut computation to obtain a min-cut, the sizes of the two partitions S and T are checked. It tries to obtain a balanced partition in a recursive manner. If the size $|S|$ is too small, all the nodes in S and a node chosen from T are collapsed to the source s. By increasing the flow in the current network, a different cut-size with a larger S will be found. Similarly, if the size $|S|$ is too large, all the nodes in T and a node chosen from S are collapsed to the sink t. The Max-Flow Min-Cut computation will be applied repeatedly until the desired size is obtained.

In this thesis, we have also proposed a network flow based partitioning method, which will be introduced in Chapter 5. Detailed discussions on flow network and net modelling will be given in Chapter 5.

Clustering Approach

The sizes of the partitioning problems grow significantly in recent years. In order to deal with complicated circuit, a pre-processing step, clustering, is proposed and applied to the partitioning problem to improve the efficiency of the subsequent partitioning method and the solution quality. The clustering steps will group the strongly connected nodes in the original circuit together to form clusters. This can reduce the problem size significantly. As the nodes in one cluster will be treated as one node in the subsequent partitioning method, the nodes in the same cluster will be assigned to the same partition in the final solution. Since the aim of the cut-size driven partitioning algorithms is to minimize the cut-size between partitions, it is good to put the strongly connected nodes in the same cluster to reduce the inter-partition cut-size.

However, the clustering technique alone cannot produce a partitioning solution and it is usually applied together with some iterative algorithms like the FM algorithm. Since clustering can reduce the problem size significantly, it is useful in solving large size problems. However, since some nodes are pre-clustered and they will not be separated again in the subsequent partitioning method, some good solutions with small cut-size will be eliminated.

Multilevel Approach

In a multilevel approach, the partitioning process is performed in a hierarchical structure which is divided into two phases, coarsening and uncoarsening. In the coarsening phase, node which are strongly connected will be grouped together recursively. For example, given an original circuit which is represented by a hypergraph, H_0, a new hypergraph H_1 is obtained from H_0 by grouping the strongly connected nodes together. This coarsening step will be applied to the hypergraph H_1 again to give another hypergraph H_2. This step will be repeated until the number of node in the condensed hypergraph is smaller than a given threshold.

In the uncoarsening phase, a move based partitioning algorithm such as the FM algorithm will be applied to the hypergraph H_n. A partitioning solution will then be obtained. The next step is to un-group the nodes in the partitioned hypergraph H_n back to H_{n-1} without changing the partitioning result. This un-grouped partitioning result will be used as the initial solution for the next FM computation. This ungrouping and FM computation steps will be applied to the hypergraph repeatedly until all the nodes in the original hypergraph H_1 are obtained. A partition of the original circuit will then be obtained at the end.

Logic Replication

The logic replication technique can be used to reduce cut-size. This is done by replicating some nodes from one partition to another. An example is illustrated in Figure 3.5. Hwang and El Gamal proposed a min-cut replication algorithm for determining the replication set for a k-way partitioning such that the cut-size of the partition is minimized [12, 13]. However, their algorithm is not optimal in hypergraph. Yang and Wong proposed a Min-Cut Replication Algorithm in [2]. This algorithm provides a method to find the min-area min-cut replication sets in a partitioned hyergraphs optimally.

Given a bi-partitioned circuit, the algorithm first models it as a flow network using the modelling as shown in Figure 3.6. In the bi-partitioned flow network (U, U'), the source s is connected to the primary input nodes with infinite capacity and the primary output nodes are connected to the sink t with infinite capacity. A Max-Flow Min-Cut computation is then applied to the flow network to obtain a $s - t$ min-cut. The replication set C is the set of nodes that belongs to the sink side T except those which are originally in partition U'. The set C is then duplicated in U' and new bi-partitioned circuit becomes $(U, U' \bigcup C)$. This method can be applied to the multi-way partitioning problem. Given a k-way partitioned circuit $(U_1, U_2, ..., U_k)$, the Max-Flow Min-Cut computation will be applied to each pair (S_i, U_i) for all $i \leq k$ where $S_i = \{\bigcup_{j \leq k \wedge j \neq i} U_j\}$. The replication set obtained in each computation will be replicated to the subset U_i. The time complexity is $O(k|V||E|)$ where k is the number of partitions in the circuit, V and E are the numbers of vertices and edges respectively.

3.5.2 Delay Driven Partitioning Algorithm

In this section, we will review some delay driven partitioning algorithms.

Performance Driven Multiway Partitioning[23]

In [23], the author proposed a relaxed acyclic partitioning algorithm for performance driven partitioning problems. This algorithm can minimize the delay and cut-size at the same time. It takes the delay caused by inter-partition edges and the overall cut-size into account by cooperating them in the cost function. Given a hypergraph representation of a circuit $G(V, E)$, a bipartition (B_f, B_t) is obtain by sorting the nodes in topological order. The first half of the nodes will be assigned to the first partition B_f and the other half will be assigned to the second partition B_t. An A-counter and a R-counter are defined for each node as follow:

$$a(x) = \left\{ \begin{array}{ll} |\{y|y \in FO(x) \text{ and } y \in B_f\}| & \text{if } x \in B_f \\ |\{y|y \in FI(x) \text{ and } y \in B_t\}| & \text{if } x \in B_t \end{array} \right\}$$

$$r(x) = \left\{ \begin{array}{ll} |\{y|y \in FI(x) \text{ and } y \in B_t\}| - a(x) & \text{if } x \in B_f \\ |\{y|y \in FO(x) \text{ and } y \in B_f\}| - a(x) & \text{if } x \in B_t \end{array} \right\}$$

$FI(x)$ and $FO(x)$ represent the fan-in and fan-out of the node x respectively. The value $r(x)$ represents the reduction in backward edges if x is moved to the other partition. The R-counter will then be incorporated into the FM gain function as follows:

$$h(x) = \alpha \cdot g(x) + \beta \cdot r(x)$$

where α and β are the weighting constants subject to user adjustment and $g(x)$ is the gain value of cell x as defined in the original FM algorithm for estimating the reduction of the overall cut-size. The cell movements in the algorithm will be based on the value $h(x)$ while the value of $g(x)$ and $r(x)$ will be updated during the FM iterations.

3.5.3 Acyclic Circuit Partitioning Algorithm

The acyclic multi-way partitioning problem was defined in [6]. It differs from the general partitioning problem in the requirement that the edges between different partitions cannot form a directed cycle. As stated in [6], acyclic multi-way partitioning finds applications in pipelining of multi-chip designs, partitioning based logic minimization, and parallel circuit simulations. Beside, acyclic partitioning is an effective way to upper bound the largest number of inter-partition delay along a path. This can generally decrease the number of partition cuts along the critical paths and the delay of the circuit will also be reduced.

Acyclic Multi-way Partitioning of Boolean Networks [6]

Many existing partitioning algorithms have shown that pre-clustering can effectively improve the solution quality. However, most of them do not take signal directions into account. Therefore, it is not possible to apply them directly to obtain acyclic partitionings. An algorithm with pre-clustering followed by a restricted version of the FM Algorithm for acyclic multi-way partitioning was proposed in [6]. As traditional clustering methods do not take signal directions into account, it is not possible to apply them for acyclic clustering. Therefore, a maximum fan-out free cone (MFFC) decomposition clustering technique is used. The MFFC decomposition was proposed in [24] and it can

produce an acyclic clustered network. After clustering the network, a k-way acyclic FM algorithm will be applied on the condensed network and produce a final partitioning. The k-way acyclic FM algorithm is similar to the original FM algorithm. It constructs a k-way acyclic partitioning by applying the two-way algorithm recursively. For example, for a four-way partitioning, a two-way partitioning will be first produced by applying the original two-way algorithm. Then, each of these two partitions will be further partitioned into two partitions. Beside considering the gain of a cell, the movement of a free cell will be accepted only if the move does not violate the area constraints and the acyclic constraints.

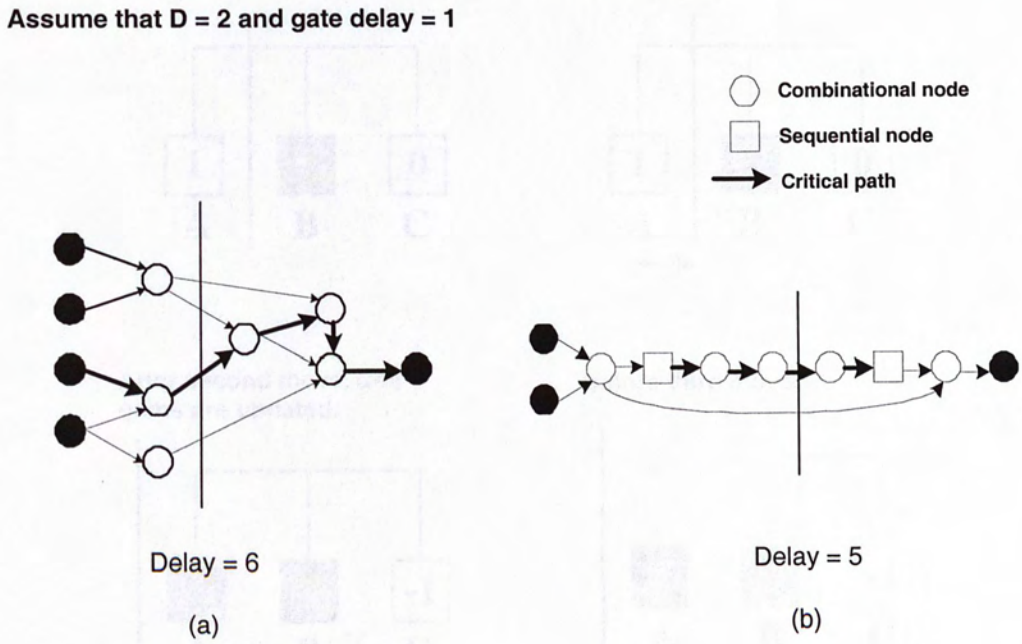


Figure 3.3: General Delay Modelling

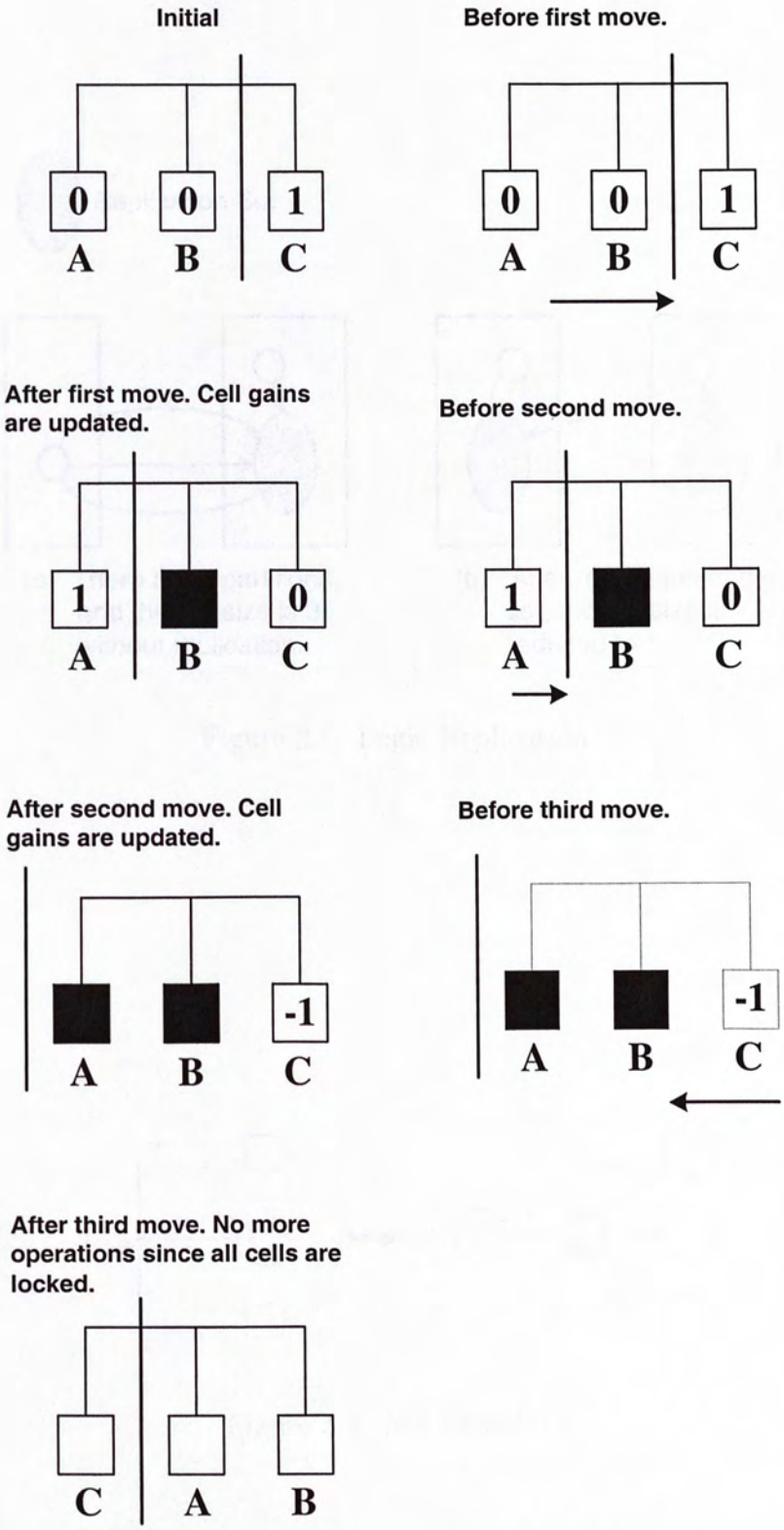


Figure 3.4: Gain Update Operations for a Net in the FM Algorithm

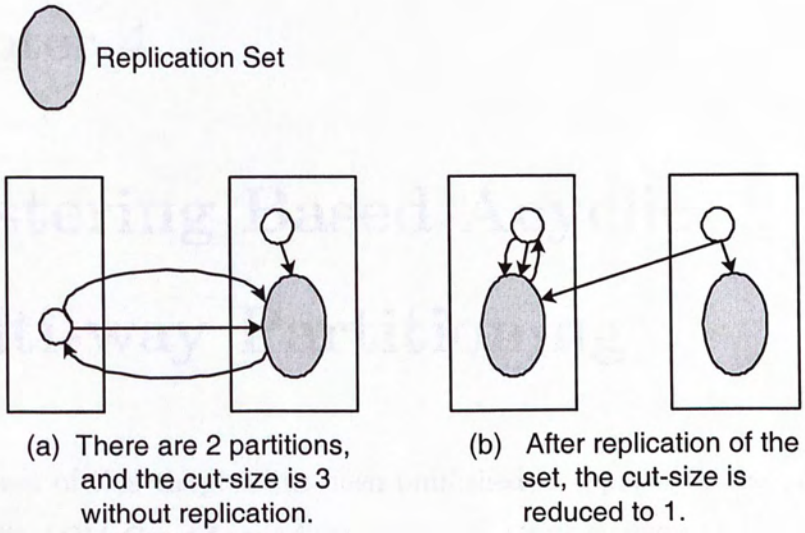


Figure 3.5: Logic Replication

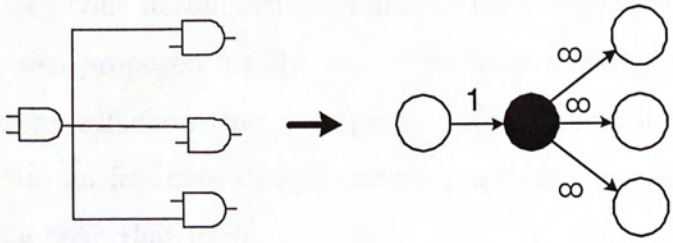


Figure 3.6: Net Modelling

Chapter 4

Clustering Based Acyclic Multi-way Partitioning

The content of this chapter has been published in a paper in the proceedings of the 13th *ACM Great Lakes Symposium on VLSI in 2003 [25].*

4.1 Preliminaries

Many existing partitioning algorithms have shown that clustering can effectively improve the solution quality. However, most of them do not take the signal direction into account. Therefore, it is not possible to apply them directly to obtain acyclic partitionings. In [6], an algorithm based on the maximum fan-out free cone decomposition followed by a restricted version of the FM algorithm was proposed for the acyclic multi-way partitioning problem. In this chapter, we will show that a simple two-phase clustering process based on a modified fan-out free cone decomposition can yield superior acyclic multi-way partitioning than that in [6].

Our algorithm is based on clustering by computing the modified fan-out free cones. Fan-out free cone clustering can be used to reduce a graph to a smaller and sparser one, and maintain the acyclic property at the same time.

Beside cut size consideration, the longest delay of a path is also an important issue to be considered. Acyclic partitioning is an effective way to upper bound the largest number of inter-partition delay along any path. The acyclic multi-way partitioning problem was defined in [6]. It differs from the general partitioning problem because of the restriction that the edges between different partitions of a solution cannot form a directed cycle. Acyclic multi-way partitioning finds applications in pipelining of multi-chip designs, partitioning based logic minimizations, and parallel circuit simulations as described in [6]. Experimental results showed that our algorithm compares favorably with the previous best acyclic multi-way partitioning algorithm in cut-size.

In the following sections, we will discuss the acyclic multi-way partitioning problem and present a clustering based partitioning algorithm to solve the problem. We will first present some previous works on clustering based partitioning in Section 4.2. Then, we will formulate the multi-way acyclic partitioning problem in Section 4.3. Our clustering based acyclic multi-way partitioning method will be introduced in Section 4.4. The details of our algorithm will be discussed in Section 4.5 to 4.8. In Section 4.9, we will present some experimental results. Finally, a summary will be given in Section 4.10.

4.2 Previous Works on Clustering Based Partitioning

When dealing with very large circuits, the performance of the traditional group migration partitioning techniques [10, 11] will be degraded. One solution is to group the cell into a larger cluster. Using cluster techniques, the problem size will be reduced. Most of the clustering methods are done in a bottom up

manner. This means that each cell belongs to its own cluster at first and these clusters are gradually merged to form several larger clusters. In this section, we will introduce some clustering techniques integrated with group migration approaches. Multilevel clustering [2] and Cluster-Oriented Iterative-Improvement Partitioner [3] will be discussed in the following sections.

4.2.1 Multilevel Circuit Partitioning [2]

The multilevel partitioning method continuously clusters the cells into larger clusters until the solution is tractable. Given a hypergraph representation of a circuit, H_0, a new hypergraph H_1 is obtained from H_0 by grouping the strongly connected nodes together. This step is called coarsening. This coarsening step will be applied recursively process, i.e., this coarsening step will be applied to the hypergraph H_1 again and form a hypergraph H_2. This will be repeated until the number of node in the condensed hypergraph is smaller than a given threshold. After the coarsening step, the move based partitioning algorithm FM will be applied to the hypergraph H_n. The FM algorithm is applied to obtain an initial rough partitioning of these clusters. Secondly, the cluster is uncoarening down a level. FM algorithm is applied again to refine the solution. These coarsening and refinement is repeated until the clusters are all uncoarsened. An example is illustrated in Figure 4.1.

The strategy used to cluster the cells is greedy weighted matching algorithm. Connectivity is the aspect used as the matching criterion. The higher the connectivity between the cells, the greater the chance they will form a new cluster. The connectivity is inversely proportional to the cell area and the number of cells in their connected net. Before this multilevel circuit partitioning is proposed, two phase clustering method is used. The basic idea of

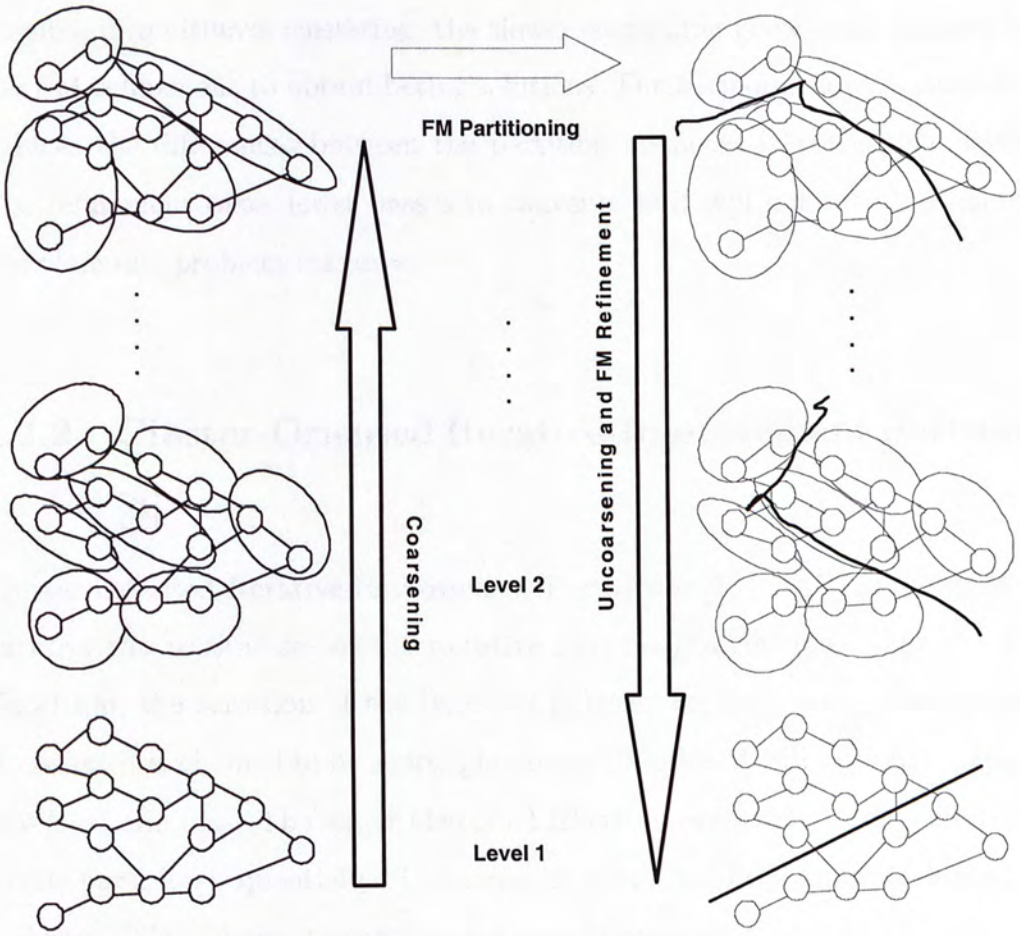


Figure 4.1: Multilevel Circuit Partitioning

two phase clustering are the same as multilevel clustering. The difference is that two phase clustering will only coarsen the cells once from the first level to second level. The FM algorithm is then applied to produce an initial solution. Finally, uncoarsening and refinement is done from second level to first level. Although two phases clustering is faster than multilevel clustering, the result obtained may not be as good as the multilevel one. By using as many levels as possible in multilevel clustering, the slower coarsening gives more chances for the FM refinement to obtain better solutions. Furthermore, slower coarsening reduces the differences between the partition instances in consequent levels. The refinement takes fewer passes to converge so it will not take too long to complete one problem instance.

4.2.2 Cluster-Oriented Iterative-Improvement Partitioner [3]

Cluster-Oriented Iterative-Improvement Partitioner [3] (CLIP) is designed to improve the weaknesses of the iterative partitiong algorithms. For the FM algorithm, the selection of the base cell is based on the previous movement. However, it is claimed to be shortsightedness. Besides, it will be easily trapped into local minimas. The major idea of CLIP is to move strongly connected cells to one partition sequentially. This strongly connected cells group is defined as a cluster. The clusters themselves are weakly connected.

The objective of the algorithm is to move the cells in the same cluster to one side. It means that clustes will not be cut and thus, result in small cutsize after partitioning. Once a cell in a cluster is moved from A to B, more weight is given to the gain of its strongly connected neighbors to increase the probability that they will also be moved to B. An example is shown in Figure 4.2. Like

traditional iterative group migration algorithm, CLIP starts with an initial partition with initial gains computed. The cell with maximum gain is moved first. After moving the first cell, the gains of all cells are reset to zero while maintaining their original ordering in the bucket list structure. Only the gains of the neighbors of the moved cell are updated. This process ensures that the strongly connected cells group (cluster) will be moved to the same side with higher priority. The algorithm repeated until all the cells are moved.

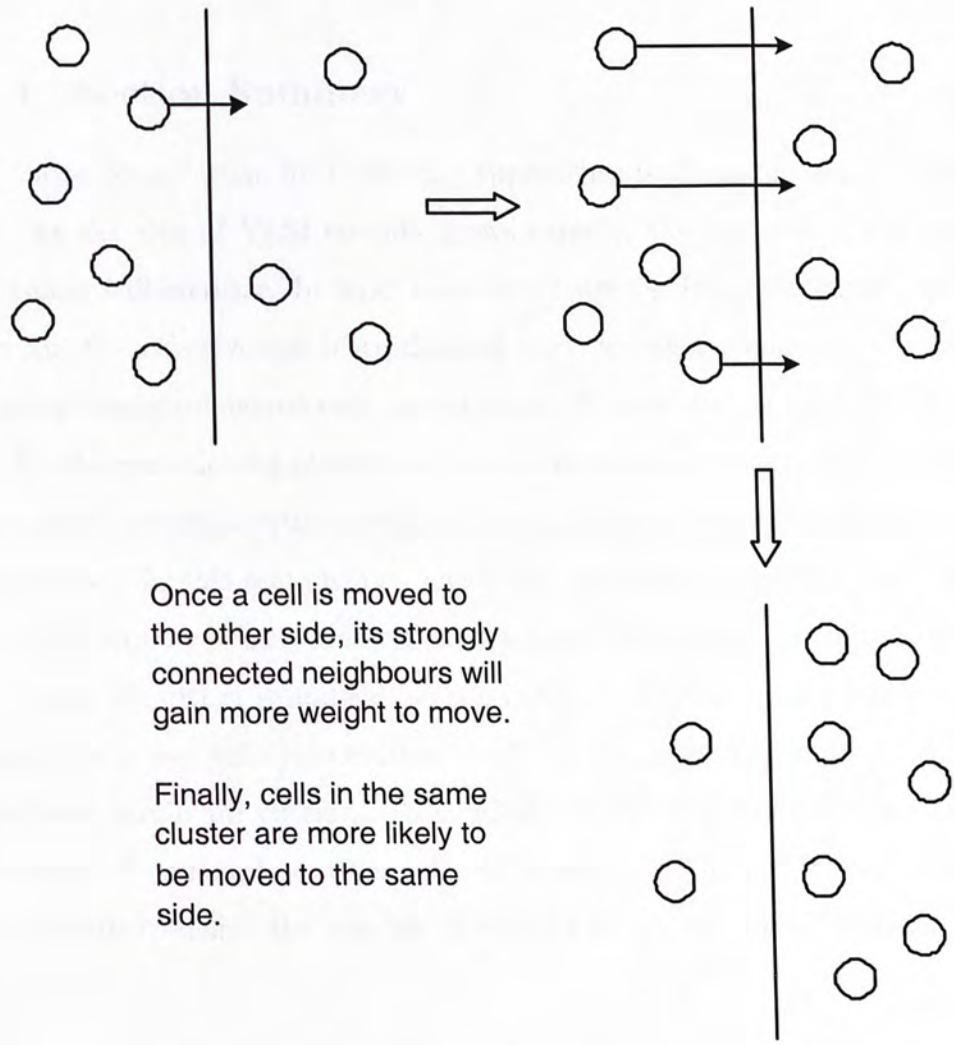


Figure 4.2: CLIP

Unlike the FM algorithm which moves the cells based on the previous cell movement, CLIP concentrates on the strongly connected neighbors. It can explore a wider solution space and get better solution. However, the size of the cluster is difficult to control. If the whole circuit is strongly connected, balanced partitioning is difficult to achieved. After partitioning, the whole clusters will be stayed on one side. Several re-grouping will be applied after each cell movement, forming the new clusters. That is, the cluster sets are changing throughout the algorithm.

4.2.3 Section Summary

The major contribution for clustering approaches is the reduction in problem size. As the size of VLSI circuits grows rapidly, the runtime of the circuit partitioner will increase. In order to adapt future needs, clustering of cells can maintain the effectiveness of traditional partitioning techniques. Clustering groups strongly connected cells into clusters. Cluster is then used as the basic unit for the partitioning algorithm. It can ensure that the strongly connected clusters will not be partitioned while the weakly interconnections between clusters will be. In this section, two clustering algorithms are discussed. In the multilevel approach, cells are first grouped into clusters and then iterative improvement algorithm is applied. While in CLIP, there is no explicit grouping strategy to gruop cells into clustes. Cells can be regrouped during each pass to achieve minimum cutsize. To conclude, CLIP is a flexible algorithm for clustering. It can explore wider solution space for optimal solution. However, it is difficult to define the number of clusters to achieve a good solution.

4.3 Problem Formulation

In this chapter, we want to solve the acyclic multi-way partitioning problem in a combinational network. A combinational circuit is represented by a directed acyclic graph *G(V,E)* where V is a set of nodes representing the gates and E is a set of directed edges representing the interconnections between the gates. The fan-out of a node is the number of edges incident from it and the fan-in of a node is the number of edges incident to it. Primary input is a node with zero fan-in and primary output is a node with zero fan-out. In the given acyclic graph G, each node in V is assigned a unit weight except the primary input and output nodes. The primary input and output nodes are assigned a zero weight each. The weight represents the area occupied by the node. We assume that the areas occupied by the gates are the same which is one unit area. Note that an acyclic partitioning of a sequential circuit can be obtained as follow. We can compute an acyclic partitioning of the combinational network obtained by removing all the sequential elements. Then we can put back the sequential elements into the proper partitions.

Figure 4.3 shows an example of a directed acyclic graph representation of a combinational circuit. The primary input and and output nodes are colored in black and other nodes are colored in white.

Definition 4.1 (An Acyclic K-Way Partitioning Problem) *Given a directed acyclic graph $G(V, E)$, partition the set of nodes V into k disjoint subsets V_1, V_2, ... , V_k, such that the sizes of the subsets do not exceed the size constraints A_1, A_2, ... , A_k, the cut-size is minimized, and the partitioned solution is acyclic, i.e. there is no partitions V_i and V_j such that $i \neq j$, and there are directed paths running from V_i to V_j and from V_j to V_i.*

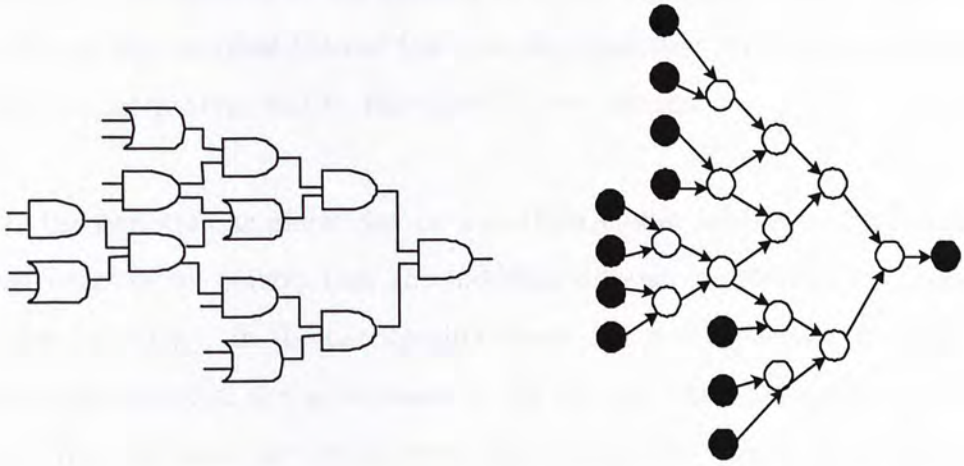


Figure 4.3: A Directed Acyclic Graph Representation of a Combinational Circuit

4.4 Clustering Based Acyclic Multi-Way Partitioning

For clustering based acyclic multi-way partitioning, the system clock is first removed from the given network. Then, the given network is clustered into a sparser network. We use an idea similar to that of the maximum fan-out free cone to cluster the nodes. The maximum fan-out free cone decomposition aims at minimizing the number of edges coming out from a cluster. This is a good strategy since it reduces the total number of edges inside a clustered network globally. After the decomposition, the number of edges is equal to the number of clusters because there is only one edge coming out from each fan-out free cone. This produces a good initial solution for the partitioning process since we aim at minimizing the cut size between the resultant partitions. After the clustering phase, the nodes inside a cluster are collapsed to form one node. As a result, the clustered network becomes simpler and sparser. The number of edges and nodes are fewer comparing with the original network. It is easier to perform the subsequent partitioning task as the size of the solution space

is directly proportional to the number of nodes and edges. In our algorithm, we will use the *modified fan-out free cone decomposition* to perform clustering. Details of the process will be discussed in next section.

In the partitioning phase, we use a method similar to that in the clustering phase because we believe that the modified fan-out free cone decomposition is a good strategy. In the partitioning phase, the size constraint is set to the predefined partition size as we want to fill up each partition as much as possible. We will work on the clustered network in this phase. As a clustered node is actually a collection of nodes, the weight of each clustered node can be large. Therefore, it is hard to obtain an ideal fan-out free cone to fit the size of a partition. In such case, we will first find a maximally fit cone to put into a partition. Then, we will try to fill up the partition as much as possible by taking in smaller cones until no other match is possible. We will discuss this selection process in details in Section 4.7.

4.5 Modified Fan-out Free Cone Decomposition

We use the idea of fan-out free cone to find the clusters and partitions while maintaining the acyclic condition. An input cone of v, denoted by $cone(v)$, is a set of nodes consisting of v and a subset of its predecessors such that any path connecting a node in $cone(v)$ to v lies entirely in $cone(v)$. We can observe that there exists many input cones for a specific node (see Figure 4.4). A fan-out free cone of v, $FFC(v)$, is an input cone of v such that any fan-out of a node in $FFC(v)$, except that of node v, must also be in $FFC(v)$. Fan-out free cone of a node is again not unique. An example is illustrated in Figure 4.5. However,

the number of fan-out edges from any fan-out free cone must be equal to one. We have made use of this size flexibilities to match the size constraints in the clustering and partitioning phases.

Figure 4.4 and 4.5 show the cone and fan-out free cone of a node inside a network. The cones in the figures are shaded in grey. You can observe that there exists many cones and fan-out free cones for a specific node inside the same network.

In our modified fan-out free cone decomposition, after we locate a cone of a node, we will remove the nodes inside the cone and the edges connecting this cone with other nodes in the network before proceeding to form the next cluster. As a result, some nodes connected to the cone will become a primary output node after removal. This step gives a higher probability for later steps to form larger clusters or partitions. A simple example is shown in Figure 4.6. Note that the number of fan-out edges from a cone may be larger than one in this modified decomposition method. Experimental results have shown that it is valuable to do such a modification.

4.6 Clustering Phase

In the clustering phase, the given network is clustered to form a sparser network using the modified maximum fan-out free cone decomposition. Initially, a primary output is selected randomly. It acts as the starting point of the clustering process. We denote this selected node as v and assign it to a cluster C. We will try to fill up the cluster by taking in nodes step by step until the predefined cluster size is reached. We select a fan-in node from C randomly and denote it as u. A testing process is then be performed on u to ensure that

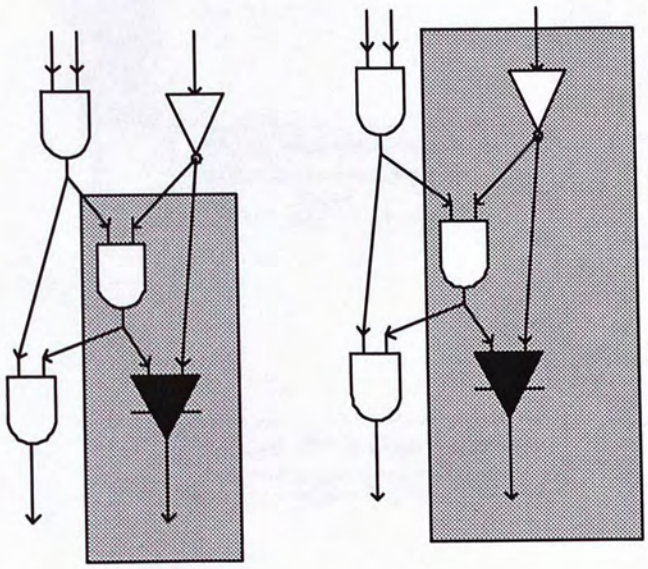


Figure 4.4: Two Different Cones of a Node

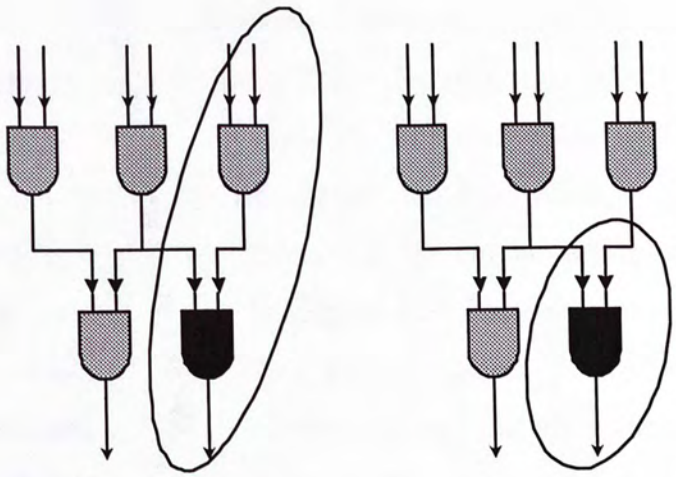


Figure 4.5: Two Different Fan-out Free Cones of a Node

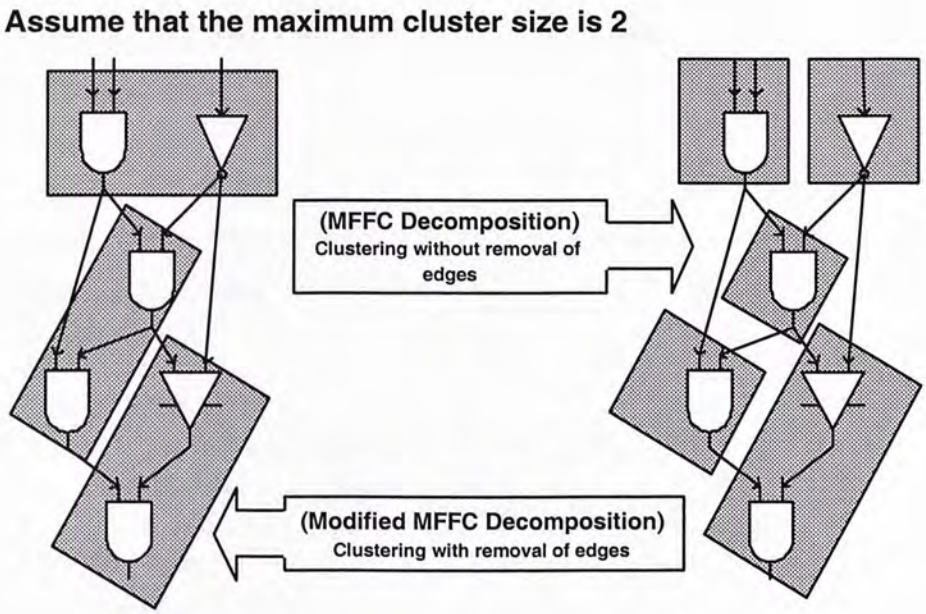


Figure 4.6: Results Obtained from Modified MFFC Decomposition and MFFC Decomposition

the newly clustered nodes do not violate the property of modified fan-out free cone. If we assign node u into cluster C, the node that can be reached from node u must also be assigned into cluster C. Otherwise, the fan-out free property cannot be maintained. Therefore, we need to find the number of nodes that can be reached from node u. If the cluster C can take in the whole set of nodes that can be reached from u, we will cluster all these nodes into C. Otherwise, we will reject this fan-in node u and try another one. If no nodes can be selected, the process will stop and one cluster is formed. An example of the selection process is given in Figure 4.9. The resultant cluster will have a size equal to or smaller than the predefined cluster size. After one cluster is formed, we will remove the clustered nodes and the edges connecting the new cluster with other nodes in the network. We will then work on the remaining network similarly as mentioned before. We will build another cluster by starting from a randomly picked primary output node of the remaining network

and repeat the above process until all the nodes are clustered. The algorithm is given in Figure 4.7 and Figure 4.8, and an example of the whole process is shown in Figure 4.13.

```
Clustering(Network)
1. dolist = 0
2. Do
3.      If dolist>0
4.          i = a randomly selected node in dolist
5.      Else if there is a primary output node
6.          i = a randomly selected primary output node
7.      Else exit
8.      cluster = cluster + 1
9.      Find_Cluster(Network, dolist, i)
10.     Remove the clustered nodes and their edges from the network
11. End_DO
```

Figure 4.7: Clustering Algorithm

4.7 Partitioning Phase

In the partitioning phase, the clustered network is partitioned into desired size (Figure 4.10). Basically, the approach used in partitioning is the same as that in clustering. The main difference is that we work on the clustered network and the cluster size constraint is set to the partition size constraint in this phase. There is no limit for the number of clusters in the clustering phase. In the partitioning phase, we must keep the number of partitions to a predefined value. Therefore, we cannot treat the size of each partition as loosely as in the clustering phase. We must fit each partition as much as possible. In the clustering phase, if no fan-in nodes can be selected to be put into a cluster

```
Find_Cluster(Network, dolist, i)
1. Add the fan-in nodes of i to local_dolist
2. While local_dolist are tested is not empty
3.      If local_dolist>0
4.          i = a randomly selected node from local_dolist
5.          j = numbers of node that can be reached by i
6.          If current_cluster_size + j <max_cluster_size
7.              Assign the set S of nodes that can be reached by i to
                  the current cluster
8.              Add all the fan-in nodes of S to dolist and local_dolist
9.              Remove the clustered nodes from dolist and local_dolist
10.     Else exit
```

Figure 4.8: Find Cluster Algorithm

anymore, we will close the cluster and continue the clustering process with a new empty cluster. However, in the partitioning phase, if no fan-in clusters can be selected to put into a partition anymore, we will first remove the currently partitioned clusters together with their incoming edges from the network and then continue with another cluster that is a primary output of the remaining network to put into the partition until the partition size is reached or no cluster can be fit into it. The algorithm is given in Figure 4.11 and Figure 4.12, and an example of the whole partitioning process is shown in Figure 4.14.

4.8 The Acyclic Constraint

In this section, we are going to discuss how to satisfy the acyclic constraint. As we are performing acyclic partitioning, we must ensure that the acyclic property is not violated in each clustering and partitioning step. We have the following lemmas about the correctness of our algorithm. We have proved the

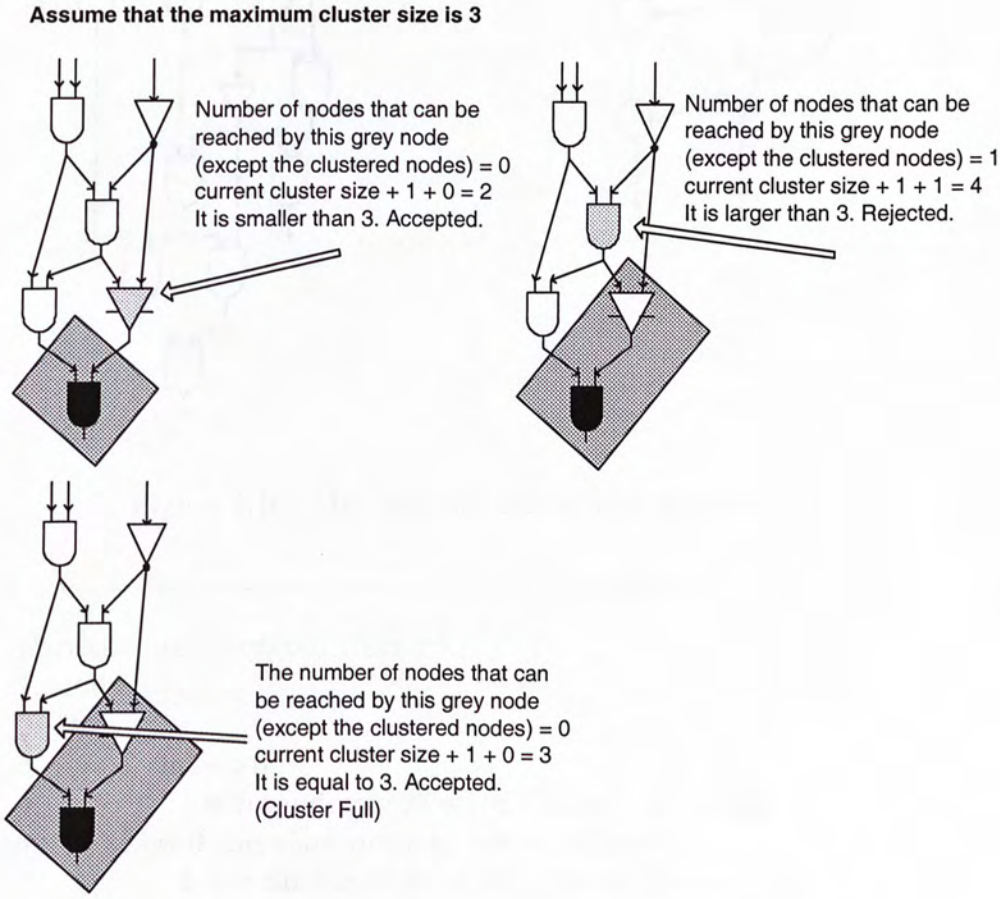


Figure 4.9: Fan-in Node Selection in the Clustering Phase

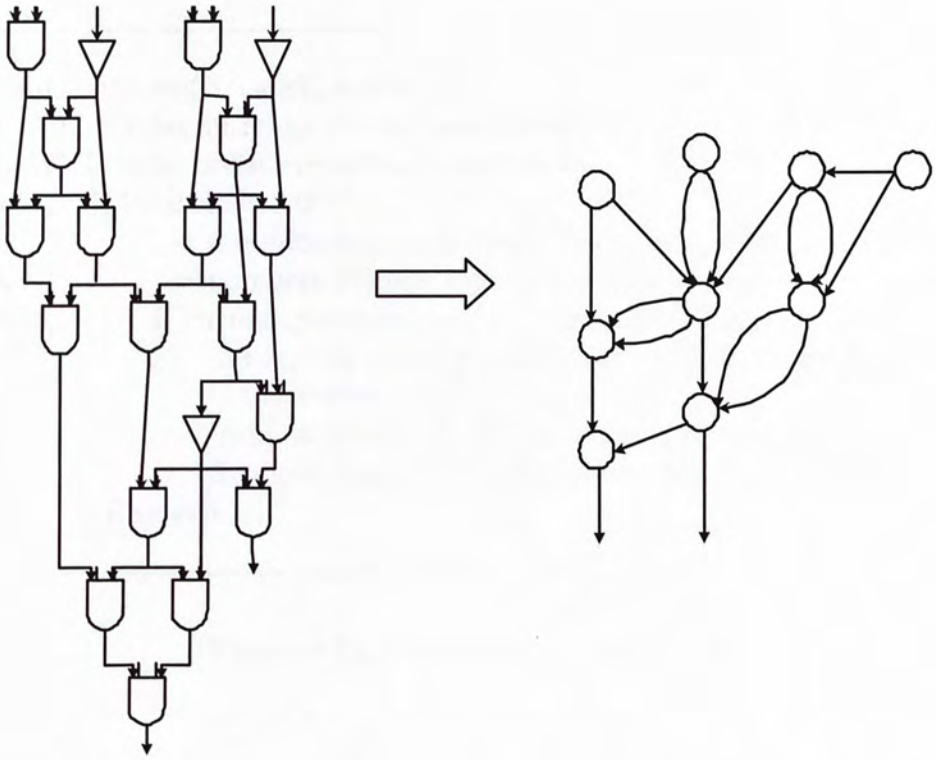


Figure 4.10: The Network Before and After Clustering

```
Partitioning(Clustered_Network)
dolist = 0
Do
    If dolist>0
        i = a randomly selected cluster from dolist
    Else if there are primary output clusters
        i = a randomly selected primary output cluster
    Else exit
    partition = partition + 1
    Find_Partition(Network, dolist, i)
    If current partition is not full yet
        partition = partition - 1
    Remove the partitioned clusters and their edges from the network
End_Do
```

Figure 4.11: Partitioning Algorithm

```
Find_Partition(Network, dolist, i)
1. Add the fan-in nodes of i to local_dolist
2. While local_dolist are tested is not empty
3.      If local_dolist>0
4.          i = a randomly selected node from local_dolist
5.          j = numbers of node that can be reached by i
6.          If current_partition_size + j <max_partition_size
7.              Assign the set S of nodes that can be reached by i to
                  the current cluster
8.              Add all the fan-in nodes of S to dolist and local_dolist
9.              Remove the partitioned clusters from dolist and local_dolist
10.     Else exit
```

Figure 4.12: Find Partition Algorithm

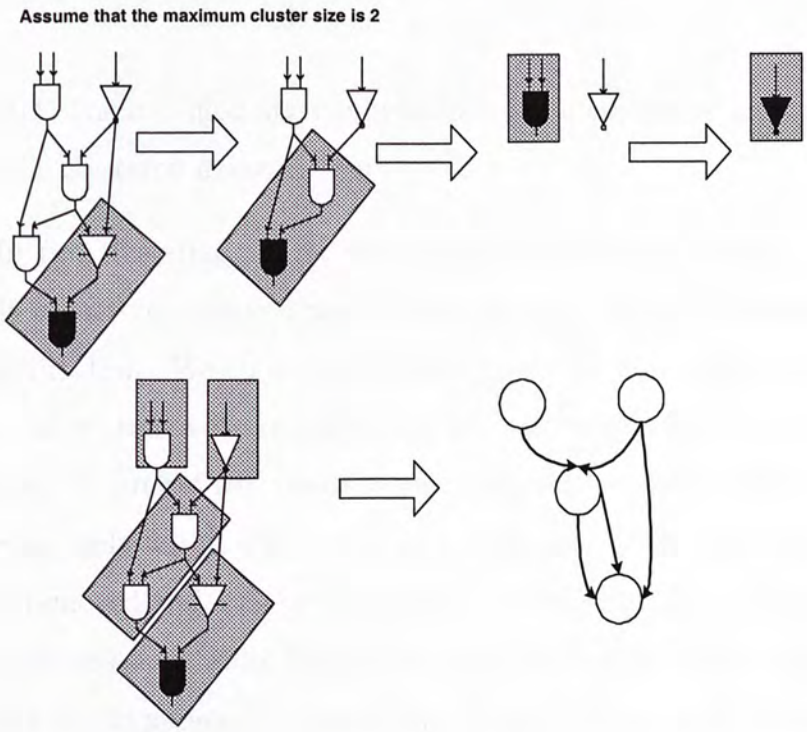


Figure 4.13: An Example of the Clustering Phase

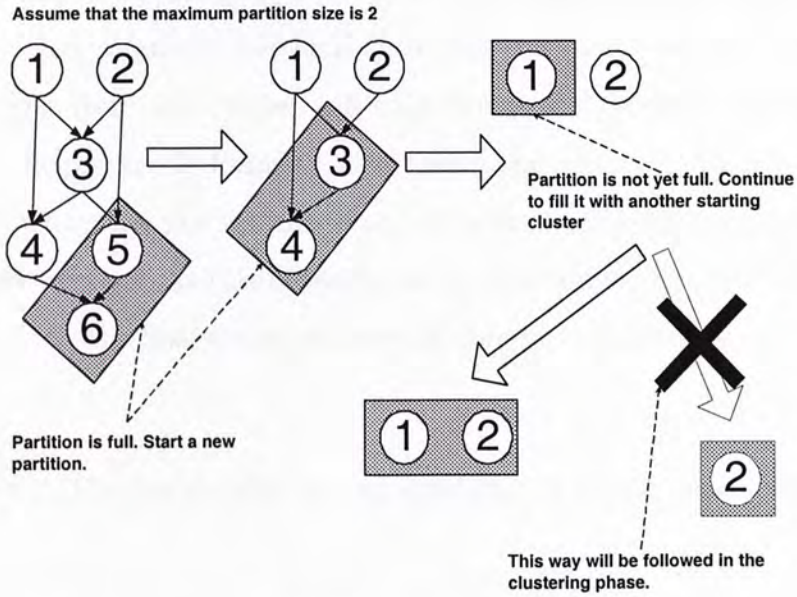


Figure 4.14: An Example of the Partitioning Phase

lemmas to show the correctness of our algorithm.

Lemma 4.1 The modified maximum fan-out free cone clustering process produces acyclic clustered network only.

Proof: In the clustering phase, we use fan-out free cone as our clustering criteria. It is easy to observe that the decomposition keeps the signal flow in one single direction. We start the clustering process from a primary output. It means that we find a fan-out free cone for a primary output. As it is a primary output, its fan-out free cone does not have any outgoing edges. It means that there are only fan-in edges going into that cone. Then the nodes in that cluster are removed and another starting node is selected for the next cluster. The next selected node must be another primary output node in the original network or a fan-in node of the previously formed cluster (thus a primary output node of the remaining network after removing the first cluster). If it is a primary output node of the original network, the case is same as before and

the signal direction can be preserved. If it is a fan-in node of the previously formed cluster, a fan-out free cone including that node will be located. The fan-out edges from this cluster will only flow in a direction from the current cluster to the previously formed clusters while the fan-in edges are coming from the nodes which are not yet clustered. As a result, it keeps the signal flow in one single direction. As this property can be maintained during the formation of each cluster, the final clustered network does not contain any cycle. **Q.E.D.**

Lemma 4.2 The partitioning process produces an acyclic partitioned network only.

Proof: In partitioning phase, we perform the same process as clustering. The main difference is that we do not start a new partition for each cone. This process does not affect the signal direction. The fan-out edges still flow in a direction from the current partition to a previously formed partition while the fan-in edges only come from a cluster which is not yet partitioned. The final partition thus does not contain any cycle. **Q.E.D.**

4.9 Experimental Results

In order to evaluate the performance of our algorithm, we implemented our clustering based partitioning method using C language. The testing platform is Sun Ultra 5/270. The benchmarks are obtained from ISCA85. These data sets contain information of the signal direction. Therefore, we can use it to test our algorithm. Moreover, we can compare our results with that of another clustering based algorithm [6] using the same data suit which is the latest best results of this problem. We compared our results with three algorithms, K-AFM, K-MAFM and hMetis. The experimental results of K-AFM and K-MAFM are obtained from [6]. The experimental results of hMetis is

obtained by using the package provided by the Department of Computer Science & Engineering, University of Minnesota. Note that K-AFM, K-MAFM and our algorithm consider acyclic constraint but hMetis is a general purpose partitioner. For all the experiments, the number of partitions is 8 and each partition allows ±5% deviation from its target size. The maximum cluster size for K-AFM and K-MAFM is 1/2 of the target partition size. The results of K-AFM, K-MAFM and hMetis are shown in Table 4.2 are the best partitioning results obtained by running the program ten times. The results of our algorithm are also the best results obtained by running the program until ten partitioning results are generated. Table 4.1 shows the characteristics of the benchmarks. Table 4.2 shows the cut-size results of different partitioning algorithms. Table 4.3 shows the runtime and cut-size of our algorithm. Note that the runtimes of K-AFM and K-MAFM are not shown because they are not reported in [6].

In comparisons with K-AFM and K-MAFM, our algorithm gives better performance in most of the cases. For smaller circuits, our algorithm is not as good as K-AFM algorithm. However, the results of our algorithm are better for large size circuits. The average improvement to the K-MAFM algorithm is 30%. Our method out-performs K-MAFM when the circuit size increases. This result suggests that the performance of the FM algorithm drops when the size of the circuit increases.

4.10 Chapter Summary

In this chapter, we presented a new acyclic multi-way partitioning algorithm. We first use the modified fanout-free cone decomposition to cluster a given network. This decomposition effectively reduces the given network to a smaller

Circuit	No. of Gates	No. of PIs	No. of Edges
c880	383	60	729
c1355	546	41	1064
c1908	880	32	1064
c2670	1193	233	2076
c3540	1669	50	2939
c5315	2307	178	4386
c6288	2416	32	4800

Table 4.1: Characteristics of the Benchmarks

Circuit K=8	Cluster size	K-AFM (net-cut)	K-MAFM (net-cut)	hMetis (net-cut)	Our Algorithm (net-cut)
c880	28	156	52	60	68
c1355	37	184	23	51	80
c1908	57	327	112	70	84
c2670	89	443	246	51	115
c3540	107	575	232	143	137
c5315	155	866	238	111	218
c6288	153	491	487	128	373

Table 4.2: Results of Different Partitioning Algorithms

Circuit K=8	Cluster size	Our Algorithm (net-cut)	Our Algorithm Runtime (Sec)
c880	28	68	23.6
c1355	37	80	34.3
c1908	57	84	26.7
c2670	89	115	41.3
c3540	107	137	61.8
c5315	155	218	114.4
c6288	153	373	85.2

Table 4.3: Runtime and Cut-size of Our Algorithm

and sparser one while maintaining the acyclic property of the network. Then we use this decomposition again to further partition the clustered network into the desired number of partitions. Our algorithm is able to obtain acyclic multi-way partitioning solutions with smaller cut-sizes comparing with the best algorithm reported previously [6].

Chapter 5

Network Flow Based Multi-way Partitioning

5.1 Preliminaries

In this chapter, we will present a network flow based partitioning algorithm. This network flow based algorithm uses the technique of net modelling and network flow to partition a circuit. As discussed before, the delay of a path is an important issue to be considered in partitioning. Our proposed algorithm aims at minimizing the longest delay along the critical path and the cut sizes between the partitions. Our modelling ensures that all the nodes on a combination path of the circuit will be partitioned in an acyclic manner. We have made use of a specific net modelling to achieve this propose.

We will first review the notations and definitions of a flow network in the next section. The network flow approach can be applied in our acyclic partitioning problem after modelling the circuit as a flow network. We will introduce the net modelling in Section 5.3. Some previous works on network flow based partitioning will be discussed in Section 5.4. Our proposed net modelling will be introduced in Section 5.5 and the properties of our modelling will be discussed in Section 5.6. After that, we will introduce our algorithm in details

from Section 5.7 to 5.8. In Section 5.9, we will present some experimental results.

5.2 Notations and Definitions

Let us review some basic notations and definitions in network flow [26]. A flow network $G = (V, G)$ is a directed graph in which each edge $(u, v) \in E$ has a capacity $c(u, v) \geq 0$. If $(u, v) \notin E$, we assume that $c(u, v) = 0$. There are two distinguished nodes in a flow network G, a source s and a sink t. A flow in G is a real-valued function $f : V \times V \rightarrow \mathbf{R}$ that satisfies the following three properties:

1. **Capacity constraint:** For all $u, v \in V$, $f(u, v) \leq c(u, v)$.
2. **Skew symmetry:** For all $u, v \in V$, $f(u, v) = -f(v, u)$.
3. **Flow conservation:** For all $u, v \in V - \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$.

The term $f(u, v)$ is called the net flow from vertex u to vertex v. It can be positive or negative. The value of a flow f is defined as $|f| = \sum_{v \in V} f(s, v)$. A $s - t$ cut of a flow network $G = (V, E)$ is a partition of V into S and T where $T = V - S$, such that $s \in S$ and $t \in T$. The capacity of a cut is the sum of the capacities on the forward edges, that is, the edges from S to T. An augmenting path is simply a path from the source s to the sink t that can push more flow from s to t . A max-flow f is a flow of maximum value from s to t.

In network flow based partitioning, a well know theorem, Max-Flow Min-Cut Theorem [27], is used. The theorem is given as follows.

Theorem 5.1 (Max-Flow Min-Cut Theorem [27]) *Given a flow network G and a maximum $s-t$ flow in G, let $S = \{v \in V : \exists$ an augmenting path from s to v in $G\}$, and let $T = V - S$. Then the $s-t$ cut (S,T) is a cut of minimum capacity which is equal to $|f|$ and f saturate all forward edges from S to T.*

As the minimum capacity cut can be obtain by a maximum $s-t$ flow, the Max-Flow Min-Cut Theorem can be used to construct a partitioning solution with minimum cut size in a network. However the Max-Flow Min-Cut Algorithm only guarantees that a minimum cut of the network is obtained but there is no constraints in the sizes of the two partitions. Some heuristics have been developed to control the partition sizes. We will introduce them in the following sections.

5.3 Net Modelling

As discussed in Chapter 3, hypergraph is a general representation for circuits. In order to apply the Max-Flow min-cut algorithm in circuit partitioning, we must be able to model a hypergraph such that the Max-Flow Min-Cut algorithm can be applied accordingly. In [28], the authors have showed that the Max-Flow Min-Cut Theorem could be applied to a hypergraph for finding a min-cut for a circuit. Based on the technique discussed in [29, 22], a hypergraph can be transformed into an edge-capacitated network. The basic idea of the method is to model it as a directed graph with addition edges. The transformation is shown in Figure 5.1. The transformation method is as follow: For a net $v \rightarrow w_1, w_2, ...$, add two dummy nodes, d_1 and d_2, with weight 0. Then, add an edge from d_1 to d_2 with unit capacity, and add edges from v and $w_1, w_2, ...$ to d_1 with capacity ∞, and add edges from d_2 to v and $w_1, w_2, ...$ with capacity ∞.

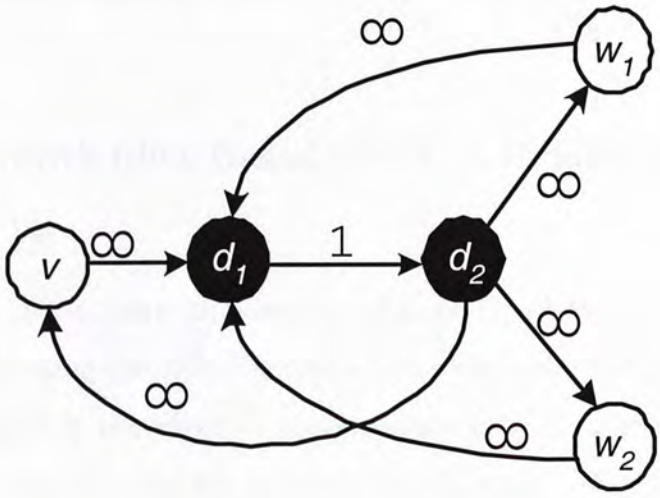


Figure 5.1: Transformation of hypergraph to a edge-capacitated network

The net modelled in such a way contains two dummy nodes with a unit capacity edge connecting them. A minimum cut must occur on the unit edge between the dummy nodes because cutting other edges will lead to infinite cut-size.

5.4 Previous Works on Network Flow Based Partitioning

Since the sizes of the partitions obtained by the Max-Flow Min-Cut algorithm cannot be controlled, a general method to obtain balanced partitions is to apply the Max-Flow Min-Cut algorithm repeatedly to achieve the target size. Although the Max-Flow Min-Cut algorithm can be operated in polynomial time, repeated application may also lead to a long runtime. Some techniques have been proposed to improve it. In this section, we will introduce some network flow based partitioning techniques. We will use some techniques proposed in these previous literatures in our algorithm. The paper on network flow based min-cut balanced partitioning [4] and network flow based circuit

partitioning for time-multiplexed FPGAs [5] will be discussed in the following sections.

5.4.1 Network Flow Based Min-Cut Balanced Partitioning [4]

In this paper, the authors proposed an efficient algorithm to produce a balanced partition using the Max-Flow Min-Cut technique. The major contribution of this paper is the effective implementation of the repeated Max-Flow Min-Cut algorithm to achieve a balanced partitioning. This approach has the same time complexity as one Max-Flow Min-Cut computation rather than n repeated Max-Flow Min-Cut computations.

The algorithm starts with modelling the net using the method discussed in Section 5.3. An example is shown in Figure 5.1. Since the resultant minimum net-cut may not correspond to a balanced partitioning, a balancing heuristic is proposed. It allows the partition sizes to vary from $(1-\varepsilon)rW$ to $(1+\varepsilon)rW$. After applying the simple Max-Flow Min-Cut computation once to obtain a min-cut, the sizes of the two partitions S and T are checked. It tries to obtain a balanced partitioning in a recursive manner. If the size $|S|$ is too small, all the nodes in S and a node chosen from T are collapsed to the source s. By applying the Max-Flow Min-Cut algorithm again, a different cut-size with a larger S will be resulted. Similarly, if the size $|S|$ is too large, all the nodes in T and a node chosen from S are collapsed to the sink t.

The strategy to pick the node from the larger partition to be collapsed is as follows: when the remaining circuit is very large, a node is picked randomly because it is too time consuming to try all possible nodes and locate the one with the minimum cut-size. However, if the remaining circuit is small enough,

it would be better to try all the remaining nodes and pick the one with the minimum cut-size. In the algorithm, a threshold size R is set such that the choices of the two strategies can be made.

After collapsing the nodes, we need to find the cut again using an incremental flow computation. The incremental flow computation can find a new cut with an asymptotic time complexity the same as one Max-Flow Min Cut computation. Instead of invoking the Max-Flow Min-Cut computation from the beginning again, additional flow that saturates the bridging edges of the cut is found and augmented to the flow network. This incremental Max-flow Min-Cut technique makes it practical to handle large circuits. The overall time complexity of the algorithm is the same as that of applying the Max-Flow Min-Cut algorithm once, which is $O(|V||E|)$.

5.4.2 Network Flow Based Circuit Partitioning for Time-multiplexed FPGAs [5]

Time-multiplexed FPGAs is used to reduce complex circuit density by making use of timesharing reconfigurable computing facilities. A large circuit is partitioned into multiple stages and is fitted into a time-multiplexed FPGA such that the same hardware is being used in all the stages. However, there are a lot of constraints in using time-multiplexed FPGAs. For example, the limited number of buffers (interconnection between partitions), the limited area of the FPGA and the precedence constraints that must be satisfied in partitioning a circuit in order to ensure a correct execution order. The algorithm proposed in [5] can give a k-way precedence constrained partitioning for time-multiplexed FPGA.

There are two precedence constraints corresponding to the combinational

nodes and the flip-flop nodes. First, each combinational node must be scheduled at a stage not later than any of its output nodes. The second precedence constraint is related to the flip-flop nodes. As the outputs of a flip-flop node will be used in the next cycle, each flip-flop node must be scheduled at a stage not earlier than any of its output nodes. The modellings for the combinational nodes and the sequential nodes are shown in Figure 5.2 and Figure 5.3 respectively.

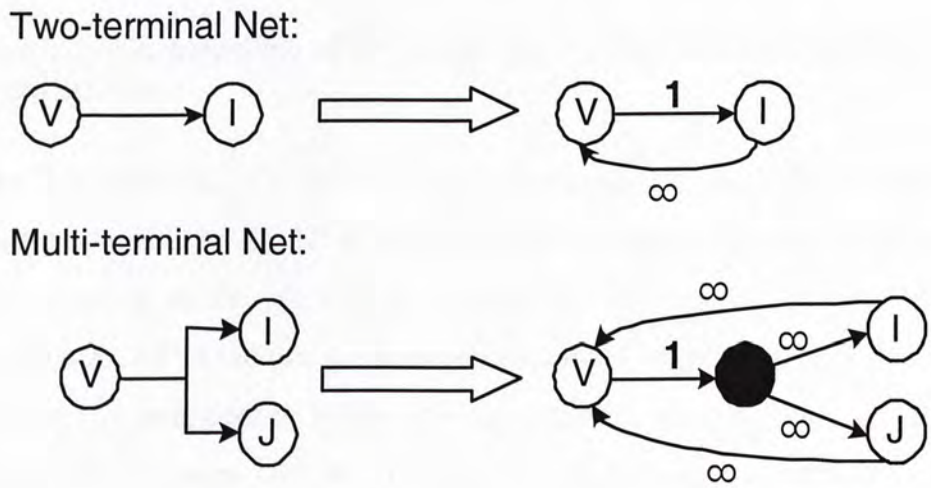


Figure 5.2: Net Modelling of Combinational Net for Time-multiplexed FPGA circuit partitioning

The α-bounded uni-directional bi-partitioning method, which is similar to the network flow based min-cut balanced partitioning method in paper [4] (Section 5.4.1), is used to cut the flow network. A bi-partitioning of the circuit, S and T, will be produced. The number of nodes in S is controlled by the value α which is given by user. It allows the size of S to deviate between $(1-\varepsilon)\alpha$ and $(1+\varepsilon)\alpha$ so that S can be fitted into an FPGA. The algorithm will partition the circuit into k stages iteratively. Firstly, an *As Soon As Possible (ASAP)* and an *As Late As Possible (ALAP)* scheduler will schedule the nodes

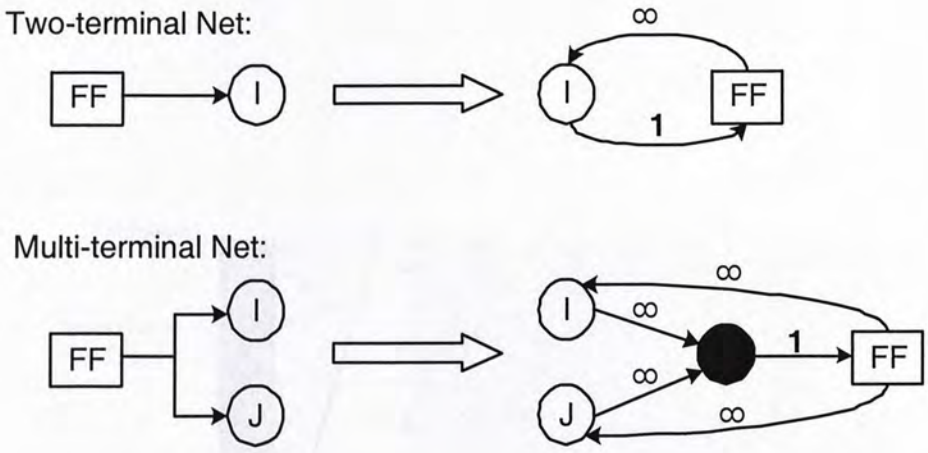


Figure 5.3: Net Modelling of Sequential Net for Time-multiplexed FPGA circuit partitioning

in the flow network. The $ASAP$ scheduler assigns the earliest possible stage to each node and the $ALAP$ scheduler assigns the latest possible stage to each node according to the precedence constraints. Nodes with the same $ASAP$ stage and $ALAP$ stage are pre-assigned as a fixed node to the specified stage to reduce the complexities of the later partitioning process. Other nodes with different $ASAP$ stage and $ALAP$ stage can be assigned to different possible stages and are called flexible nodes.

The algorithm runs in an iterative manner. It starts with assigning nodes to the first stage until all nodes are assigned to the k stages. In the ith iteration, the Max-Flow Min-Cut algorithm is applied to partition the flexible nodes with $ASAP$ stage i and $ALAP$ stage $i+1$. After the computation, the nodes in S are assigned to stage i. All the unassigned nodes with latest possible stage $i+1$ are assigned to stage $i+1$. The algorithm is illustrated in Figure 5.4. The time complexity of this multi-way precedence constrained partitioning algorithm is $O(k|V||E|)$.

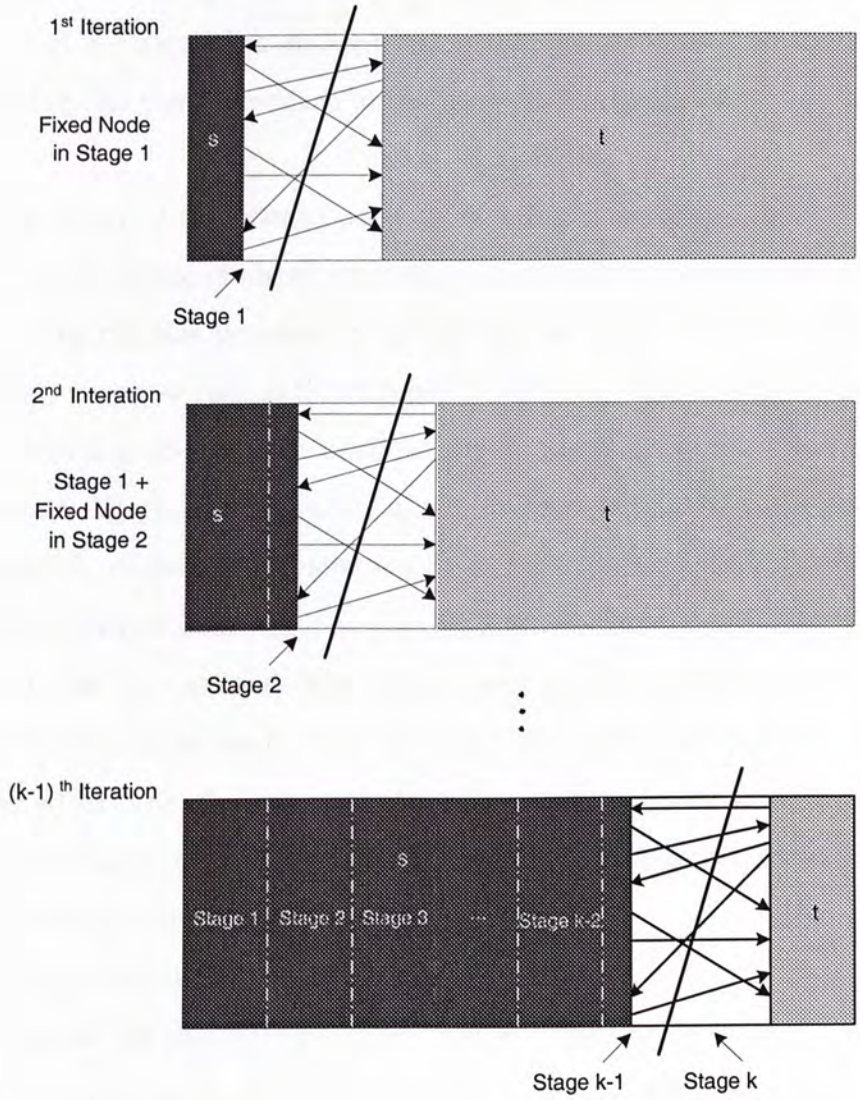


Figure 5.4: Network Flow Based Multi-way Precedence Constrained Partitioning

5.5 Proposed Net Modelling

In Max-Flow Min-Cut based algorithms, the system clock is first removed from the given network. Then, the circuit is transformed into a network flow instant by some net modelling method. The net modelling method can affect the performance of the algorithm and determines the characteristics of the resultant partitioning like the precedence of the nodes in the partitions.

As the delay of the critical path is crucial in determining the circuit performance, our proposed algorithm aims at minimizing the delay of the critical path and the cut size between the partitions. In order to reduce the delay of the circuit, we must reduce the number of partition cuts on the critical path. As introduced in the previous chapter, acyclic partitioning is a general way to reduce the delay caused by partition cut. In order to achieve an acyclic partitioning for the nodes along a combinational path, a net modelling method that can ensure unidirectional cut in the Max-Flow Min-Cut computation is needed. We found that the net modelling method used in time-multiplexed FPGAs [5] (Section 5.4.2) can be applied in our case after some modifications. In time-multiplexed FPGAs, the order of execution and the number of buffers between consecutive stages are constrained. The nodes in a time-multiplexed FPGAs have precedence constraints between them. These precedence constraints are similar to the unidirectional cut constraints in our acyclic partitioning problem . For example, let $input(v) = \{u | u \textit{ is an input node of } v\}$. Given a $s-t$ cut (S, T), using the net modelling in FPGAs, there are only two possible situation: either v and $input(v)$ are in the same partition of v in T and $input(v)$ in S. Both situations are equivalent to producing a unidirectional cut in a bi-partitioning. This modelling feature satisfies the unidirectional constraint in our problem.

In a sequential circuit, we classified the edges into two categories, c-edges and s-edges. S-edges are fan-out edges of the flip-flop nodes, while the remaining edges are c-edges. An example is illustrate in Figure 5.5. The delay of a sequential circuit is defined as the longest delay among all the combinational paths which can be classified into the following four types: (1)$PI \rightarrow PO$, (2)$PI \rightarrow FF$, (3)$FF \rightarrow PO$ and (4)$FF \rightarrow FF$. Our net modelling aims at producing an acyclic partitioning along these four types of paths.

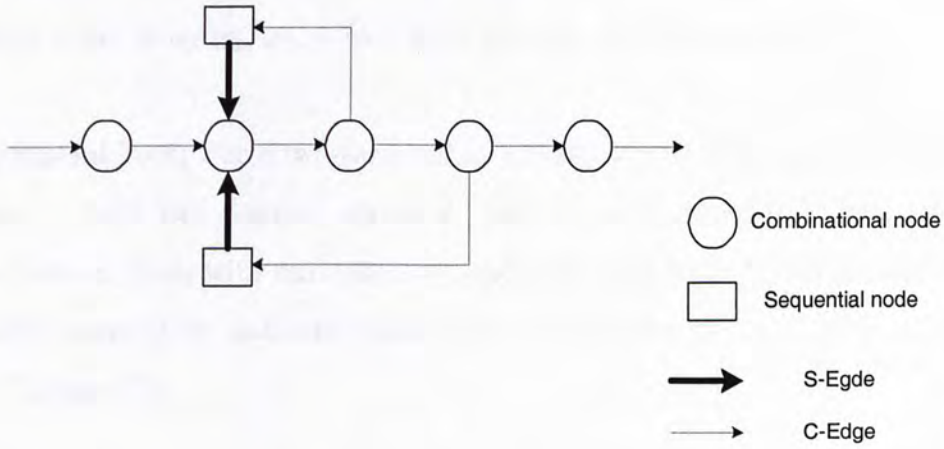


Figure 5.5: C-edges and S-edges

In our proposed modelling, we will enforce the precedence constraints on the combinational paths (a path that consists of c-edges) only because the circuit delay depends on the longest delay among these combinational paths. On the other hand, there is no constraints on the cutting direction of the s-edges. It is because the s-edges are carrying signal for the next cycle and they do not affect the overall delay. S-edge can thus be considered as an input edge for the next clock cycle. This relaxation will allow the Max-Flow Min-Cut computation to find a better solution with a better cut-size without over-constraining the solution. The modelling is described below and examples are shown in Figure 5.6 and Figure 5.7. Notice that a combinational net is a net with a

combinational node as the input and a sequential net is a net with a flip flop node as the input.

Combinational Net: For a two-terminal combinational net $v \rightarrow w$, add an edge from v to w with unit capacity, and add an edge from w to v with capacity ∞. For a multi-terminal combinational net $v \rightarrow w_1, w_2, ...$, add a dummy node with weight 0. Then, add an edge from v to the dummy node with capacity one, and add edges from the dummy node to $w_1, w_2, ...$ with capacity ∞, and an add edges from $w_1, w_2, ...$ to v with capacity ∞. (Figure 5.6)

Sequential Net: For a two-terminal or a multi-terminal sequential net $ff \rightarrow w_1, w_2, ...$, add two dummy nodes, d_1 and d_2, with weight 0. Then, add an edge from d_1 to d_2 with unit capacity, and add edges from ff and $w_1, w_2, ...$ to d_1 with capacity ∞, and add edges from d_2 to ff and $w_1, w_2, ...$ with capacity ∞. (Figure 5.7)

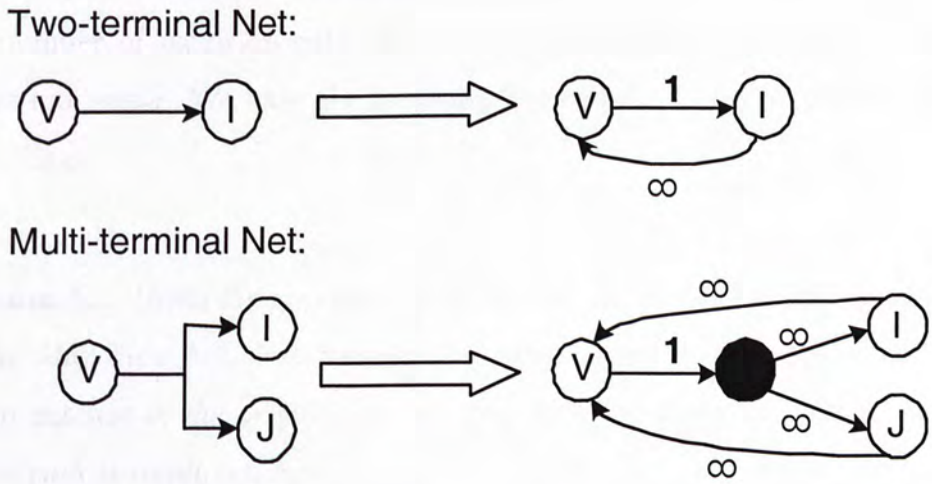


Figure 5.6: Net Modelling of Combinational Nets

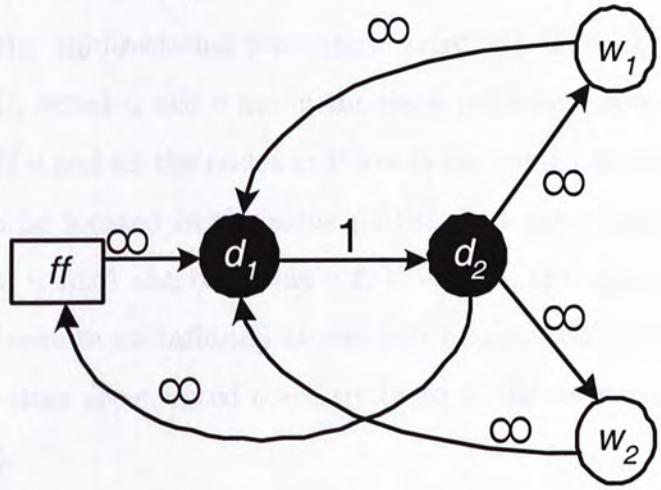


Figure 5.7: Net Modelling of Sequential Nets

5.6 Partitioning Properties Based on the Proposed Net Modelling

By using the above net modelling, we can obtain a bi-partitioning by applying the Max-Flow Min-cut algorithm on the flow network. The size of the min-cut in the flow network will correspond to the min-cut in the real circuit. Also, the number of partition cuts on each combinational path is limited by the above modelling. We have the following lemmas about the correctness of our modelling.

Lemma 5.1 *Using the proposed net modelling, the minimum cut-size obtained by the Max-Flow Min-Cut computation corresponds correctly to the real minimum cut-size of the original circuit, and the cutting edges on each combinational path is unidirectional.*

Proof:

For a combinational net, if u is the input node and V is the set of output nodes, there will be an edge from each node $v \in V$ to u with capacity ∞.

Therefore, it will never be the case that v is in A and u is in $\bar{A}$ in the min-cut solution (So, the unidirectional property is satisfied). Therefore, in a min-cut solution $(A, \bar{A})$, either u and v are in the same partition, or $u \in A$ and $v \in \bar{A}$ (Figure 5.8). If u and all the nodes in V are in the same partition, the dummy node will also be located in the same partition as the cut-size will increase otherwise. If u is in A and one node $v \in V$ is in $\bar{A}$, the dummy node will be located in $\bar{A}$ because an infinite cut-size will be resulted otherwise. In both cases, the cut-sizes are counted correctly (zero in the former case and one in the later case).

For a sequential net, (let ff be the input node and V be the set of output nodes) there will be an edge from dummy node d_1 to d_2 with capacity one. Notice that the nodes in V and the node ff are all symmetric in this modelling. If any two nodes in $\{ff\} \bigcup V$ are separated in two partitions, d_1 must be in A and d_2 must be in $\bar{A}$ because all the nodes in the net have infinite edges connecting to d_1 and have infinite edges connecting from d_2 and infinite cut-size will be resulted otherwise. The count in cut-size is also correct in this case (the count is one). If all the nodes in $\{ff\} \bigcup V$ are in A, d_1 will also be in A because an infinite cut-size will be resulted otherwise. Then d_2 will also be in A because the cut-size will be increased otherwise. The count in cut-size is zero in this case and corresponds correctly to the real cut-size. If all the nodes in $\{ff\} \bigcup V$ are in $\bar{A}$, d_2 will also be in $\bar{A}$ because an infinity cut-size will be resulted. Then d_1 will also be in $\bar{A}$ because the cut-size will be increased otherwise. The count in cut-size is zero in this case which corresponds correctly to the real cut-size. (Figure 5.9) **Q.E.D.**

Lemma 5.2 *Using the proposed net modelling, all combinational paths will be cut by the partitions at most $k-1$ times where k is the number of partitions.*

Proof:

We have proved in Lemma 5.1 that a combinational path must be cut unidirectionally using the proposed modelling. The number of partition cuts along any combinational path is at most $k-1$ where k is the number of partitions because if the number of partition cuts along a combinational path is larger that $k-1$, there must be an edge $e(u,v)$ on this path being cut in backward direction, i.e., $u \in \bar{A}$ and $v \in A$. (Figure 5.10) **Q.E.D.**

5.7 Partitioning Step

After modelling the circuit as a flow network, the Max-Flow Min-Cut algorithm is applied on the network to obtain a min-cut. We implemented the partitioning algorithm using the recursive method proposed in [22] which is already discussed in Section 5.4.1. After applying the simple Max-Flow Min-Cut computation to obtain a min-cut, the sizes of the two partitions S and T are checked. If the size $|S|$ is too small, all the nodes in S and a node chosen from the other partition T are collapsed to the source s. By applying the Max-Flow Min-Cut algorithm again, a different cut-size with a larger S will be resulted. Similarly, if the size $|S|$ is too large, all the nodes in T and a node chosen from S are collapsed to the sink t. The strategy to pick the node from the larger partition to be collapsed is as follows: when the remaining circuit is very large, a node is picked randomly. However, when the remaining circuit becomes small enough, we will try all the remaining nodes and pick the one with the minimum cut-size. Besides, we will not pick a node that will generate an infinite cut-size after collapsing. After selecting and collapsing the nodes, we need to find another cut by using the incremental flow computation until the desired balance ratio is achieved.

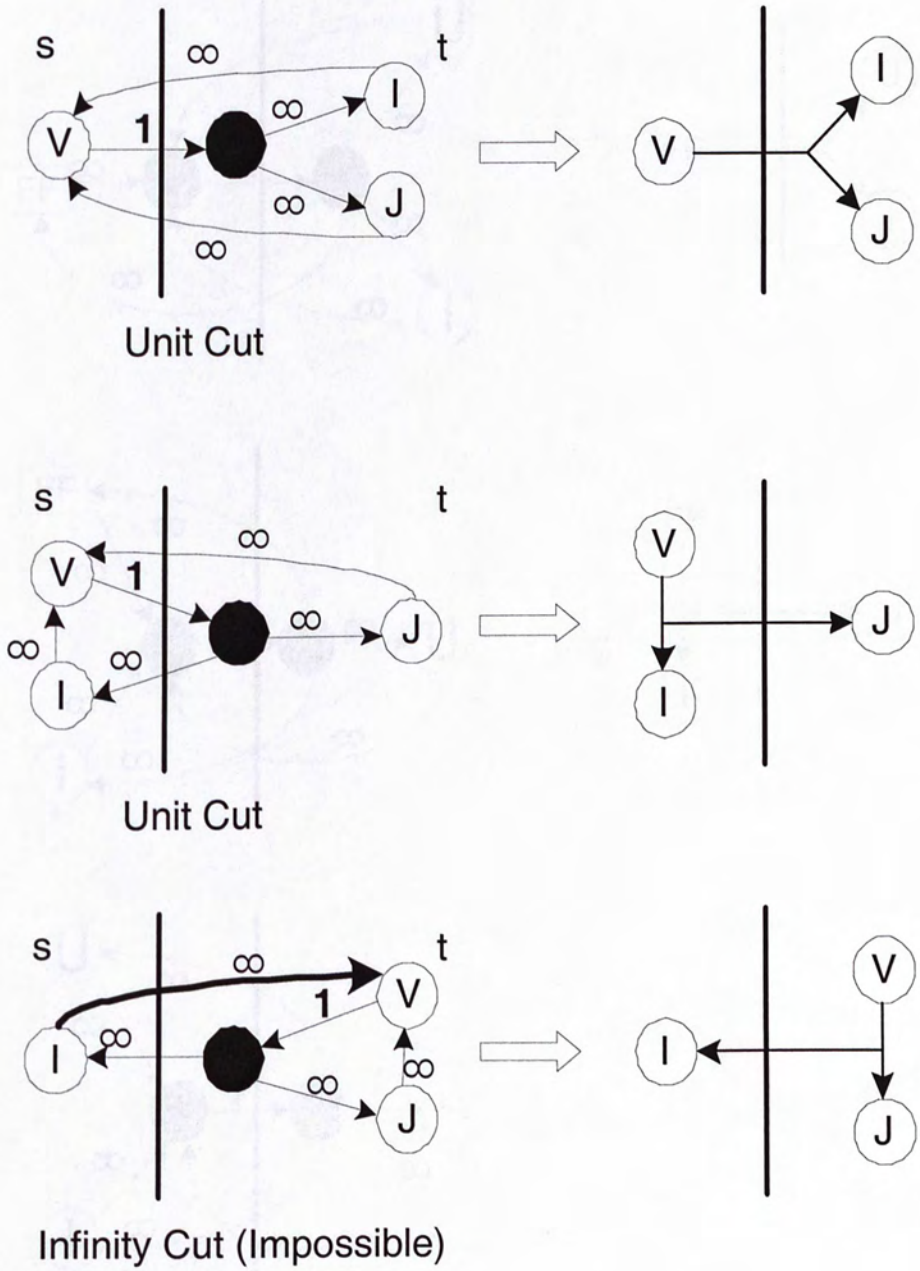


Figure 5.8: Uni-directional Cut

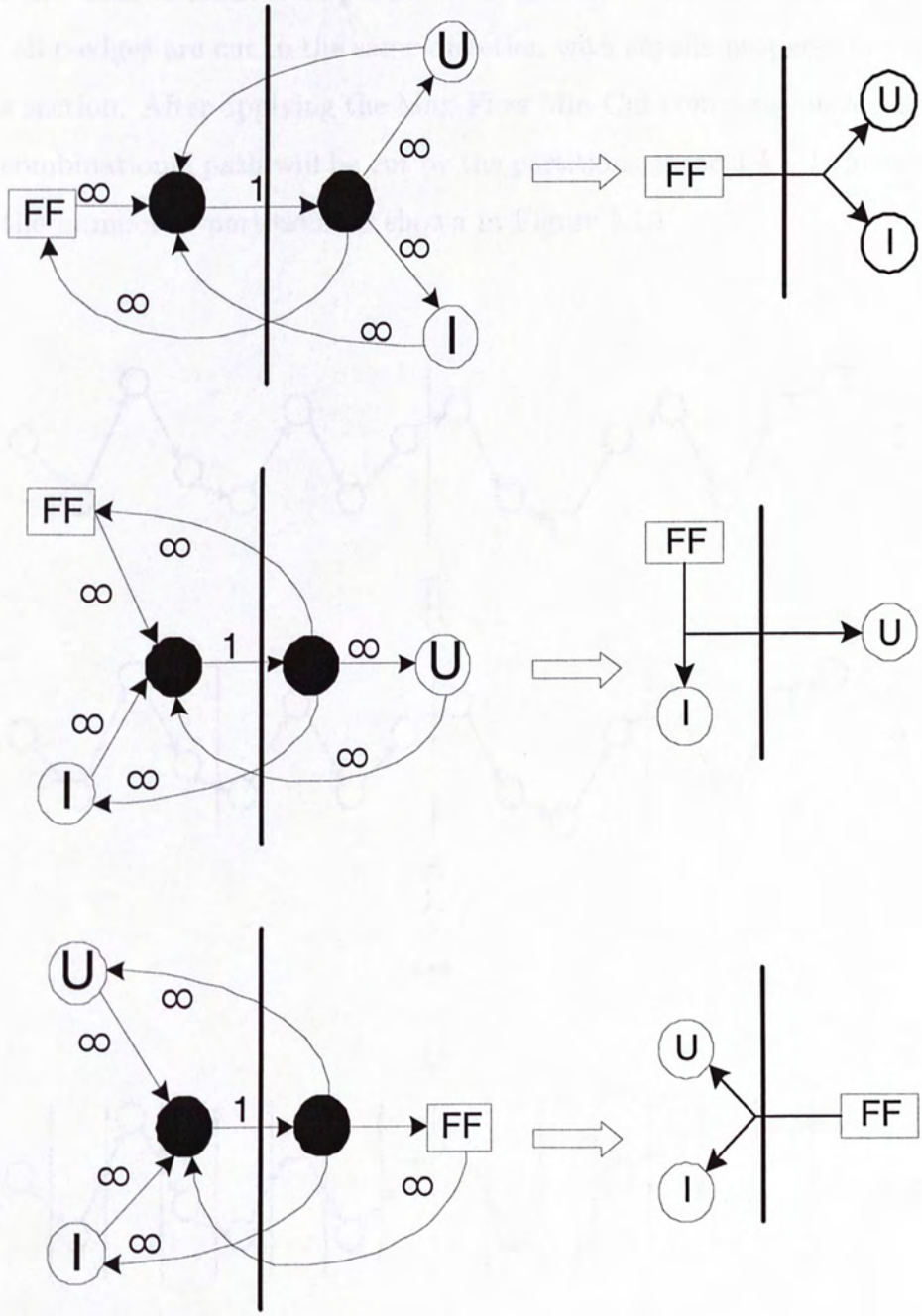


Figure 5.9: Multi-directional Cut

As only a bi-partitioning can be obtained in one Max-Flow Min-Cut computation, we will use a recursive manner to perform the k-way partitioning until the desired number of partitions is obtained. We have already proved that all c-edges are cut in the same direction with acyclic property in the previous section. After applying the Max-Flow Min-Cut computation recursively, any combinational path will be cut by the partitions at most $k-1$ times where k is the number of partitions as shown in Figure 5.10.

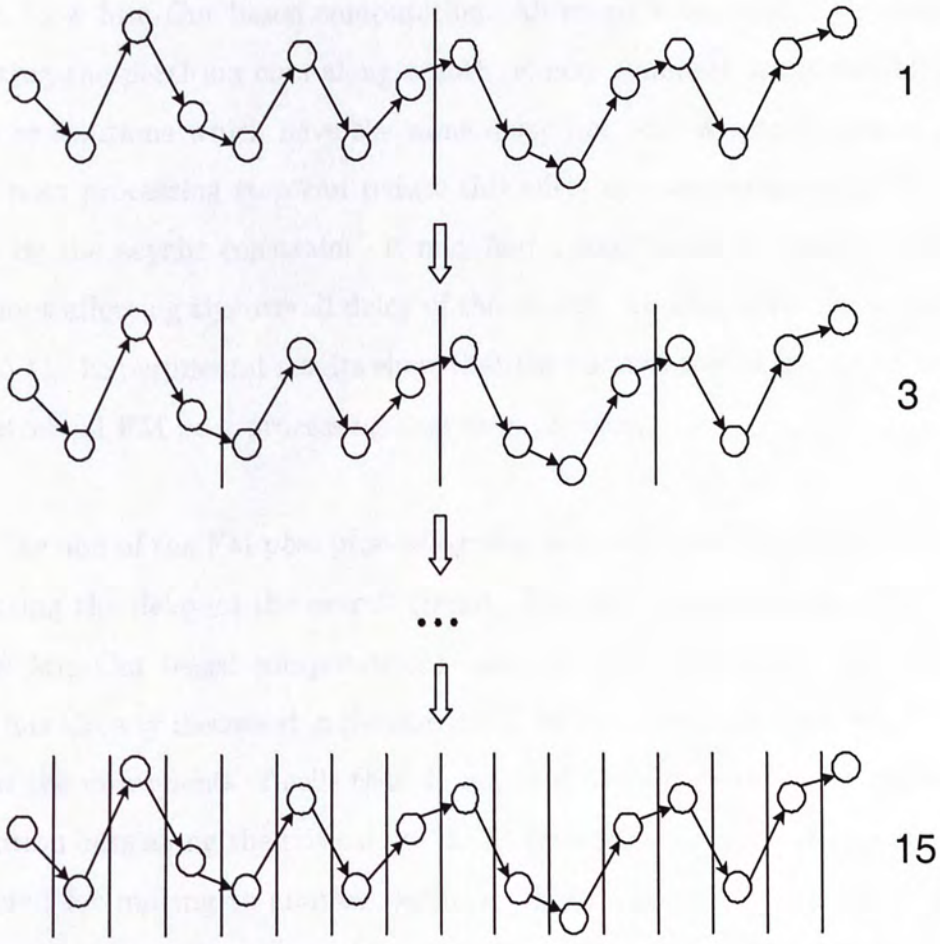


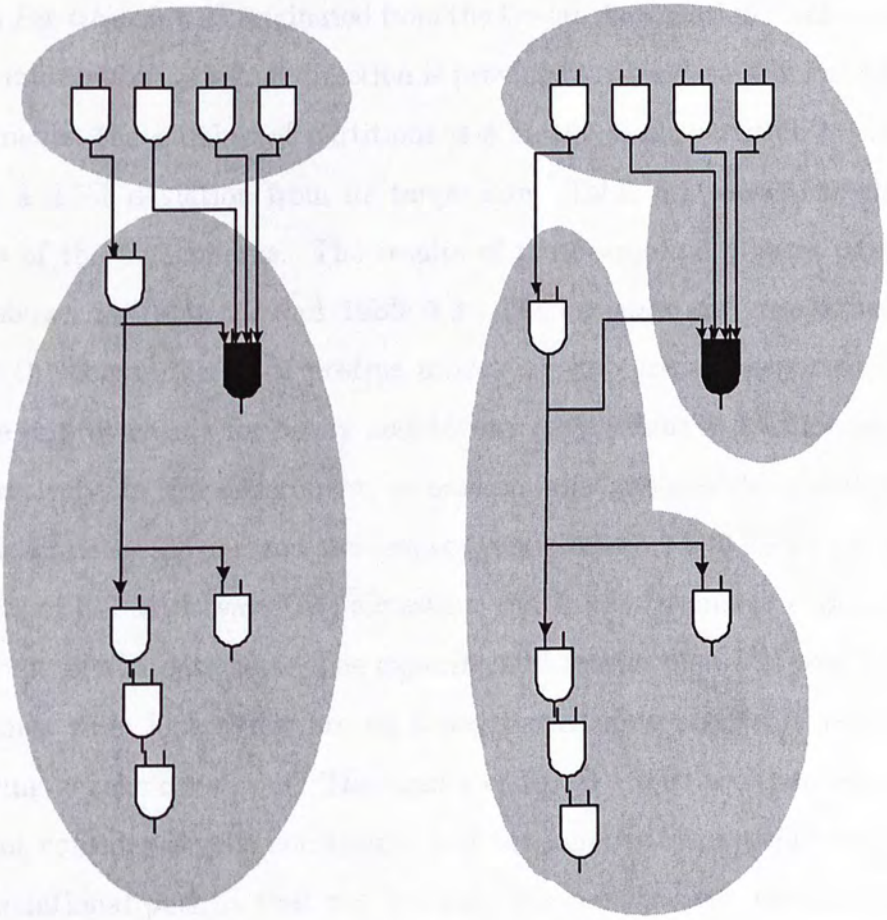
Figure 5.10: Multi-way Partitioning with Combinational Paths Cut by the Partitions at Most $k-1$ times

5.8 Constrained FM Post Processing Step

In order to improve the quality of the partitioning, we will perform a post processing step on the partitioning result after the Max-Flow Min-Cut based computation. A constrained version of the FM algorithm will be applied on the partitioning result. The constrained FM is similar to the original FM, except that some constraints are considered during the iterative moves.

We imposed the unidirectional constraint on combinational paths in the Max-Flow Min-Cut based computation. Although it can reduce the delay by limiting the partition cuts along a path, it may eliminate some possible non-acyclic solutions which have the same delay but with a better cut-size. The FM post processing step can reduce this effect of over-constraining the solution by the acyclic constraint. It may find a solution with a smaller cut-size without affecting the overall delay of the circuit. An example is shown in Figure 5.11. Experimental results show that the cut-size can be improved by this constrained FM post-processing step in most cases.

The aim of the FM post processing step is to minimize the cut-size without affecting the delay of the overall circuit. The partitioning result of the Max-Flow Min-Cut based computation is used as a starting point. The original FM has already discussed in Section 3.5.1. In the constrained version, we only allow the movements of cells that do not lead to an increase in the number of partition cuts along the critical paths. At each step, the maximum gain cell is selected for moving to another partition. If such movement will increase the longest circuit delay, it will be rejected and the movement with the second largest cell gain will be considered. We will apply the constrained FM post processing on each pair of partitions, $(1, 2)$, $(1, 3)$ ... $(k-1, k)$.



Originally, the cut-size is 4.

After moving the gate that shaded in black to another partition, the cut-size is reduced to 2. However, the result is cyclic.

Figure 5.11: Cut-size Reducing in the Post Processing Step

5.9 Experiment Results

We implemented our network flow based partitioning method using the C language. The testing platform is Sun Ultra 5/270. The benchmarks are obtained from *Partitioning* 93 originated from the Design Automation Conference 1993. The information of signal direction is provided in this data set. For all the experiments, the number of partitions is 8 or 16. Each partition is allowed to have a $\pm 5\%$ deviation from its target size. Table 5.1 shows the characteristics of the benchmarks. The results of the 8-way and 16-way partitioning are shown in Table 5.2 and Table 5.3. The cut-sizes and delays before and after the constrained FM posting processing step are also reported. The average improvements for 8-way and 16-way partitioning are 6.81% and 8.24% respectively. In this experiment, we assumed that the gate delay and the inter-partition delay are one and two respectively. Table 5.4 and Table 5.5 show the results of K-FM (k-way FM) algorithm and R-FM (recursive FM) algorithm on some of the data sets. The experimental results of K-FM and R-FM are obtained from [30]. They are all k-way partitioning algorithm without considering acyclic constraint. The results of R-FM are better than ours as they do not consider acyclic constraints and the number of partition cuts along a combinational path is thus not limited. We compare our results with them because they are closest possible in terms of the problem being solved as long as we know. By comparing our results with K-FM and R-FM, we can see how much worse in cut-size is resulted because of the acyclic constraint. We expect to gain in delay but the delays of K-FM and R-FM are not shown because they are not reported in [30]. Table 5.6 compares the results of this network flow based algorithm and the clustering based algorithm proposed in Chapter 4. The results of the clustering based algorithm is generally better than that of the network flow based algorithm.

Circuit	PIs	POs	FFs	Gates	Nets	Delay
s27.xnf	8	1	3	13	24	6
s208.xnf	15	2	8	104	127	14
s298.xnf	7	6	14	133	154	9
s344.xnf	13	11	15	175	203	20
s349.xnf	13	11	15	176	204	20
s382.xnf	7	6	27	179	207	9
s400.xnf	7	6	21	185	213	9
s420.xnf	23	2	16	212	251	28
s444.xnf	7	6	21	202	230	11
s510.xnf	23	7	6	217	246	12
s526.xnf	7	6	21	214	242	9
s526n.xnf	7	6	21	215	243	9
s820.xnf	22	19	5	294	321	10
s832.xnf	22	19	5	292	319	10
s838.xnf	39	2	32	422	493	56
s953.xnf	20	23	29	424	473	16
s1196.xnf	18	14	18	547	583	24
s1238.xnf	18	14	18	526	562	22
s1423.xnf	21	5	74	731	826	59
s5378.xnf	39	49	179	2958	3176	25
s9234.xnf	23	22	228	5825	6076	58
s13207.xnf	35	121	669	8620	9324	58
s15850.xnf	18	87	597	10396	10984	61
s35932.xnf	39	320	1728	17793	19560	29
s38417.xnf	32	106	1636	23815	25483	47
s38584.xnf	16	278	1452	20705	22173	52

Table 5.1: Characteristics of the Benchmarks

Circuit	Before C-FM		After C-FM		% Improvement
	(cut-size)	(Delay)	(cut-size)	(Delay)	(cut-size)
s27.xnf	9	14	9	14	0
s208.xnf	64	24	64	24	0
s298.xnf	72	16	70	16	2.78
s344.xnf	63	28	54	28	14.29
s349.xnf	68	28	60	28	11.76
s382.xnf	71	18	71	18	0
s400.xnf	70	15	52	15	25.71
s420.xnf	66	36	66	36	0
s444.xnf	91	19	85	19	6.59
s510.xnf	161	20	157	20	2.48
s526.xnf	134	15	134	15	0
s526n.xnf	134	15	121	15	9.70
s820.xnf	202	20	179	20	11.39
s832.xnf	180	22	180	22	0
s838.xnf	197	64	177	64	10.15
s953.xnf	261	25	261	25	0
s1196.xnf	271	36	254	36	6.27
s1238.xnf	311	34	259	34	16.72
s1423.xnf	213	71	187	71	12.21
s5378.xnf	502	34	502	34	0
s9234.xnf	584	71	580	71	0.68
s13207.xnf	582	68	507	68	12.89
s15850.xnf	572	65	571	65	0.17
s35932.xnf	806	37	741	37	8.06
s38417.xnf	970	53	820	53	15.46
s38584.xnf	1395	56	1258	56	9.82

Table 5.2: Results of Our Algorithm Before and After the Constrained FM Post Processing (8-Way Partitioning)

Circuit	Before C-FM		After C-FM		% Improvement
	(cut-size)	(Delay)	(cut-size)	(Delay)	(cut-size)
s27.xnf	16	18	14	18	12.50
s208.xnf	75	24	70	24	6.67
s298.xnf	110	17	110	17	0
s344.xnf	102	30	98	30	3.92
s349.xnf	98	30	98	30	0
s382.xnf	112	19	100	19	10.71
s400.xnf	113	18	104	18	7.96
s420.xnf	107	44	90	44	15.89
s444.xnf	131	23	111	23	15.27
s510.xnf	212	23	192	23	9.43
s526.xnf	184	16	167	16	9.24
s526n.xnf	184	16	175	16	4.89
s820.xnf	304	20	257	20	15.46
s832.xnf	269	26	221	26	17.84
s838.xnf	257	68	257	68	0
s953.xnf	339	27	320	27	5.60
s1196.xnf	371	39	331	39	10.78
s1238.xnf	424	40	424	40	0
s1423.xnf	291	77	216	77	25.77
s5378.xnf	707	36	627	36	11.32
s9234.xnf	911	78	900	78	1.21
s13207.xnf	854	70	785	70	8.08
s15850.xnf	883	71	883	71	0
s35932.xnf	1263	39	1190	39	5.78
s38417.xnf	1376	55	1252	55	9.01
s38584.xnf	2019	58	1880	58	6.88

Table 5.3: Results of Our Algorithm Before and After the Constrained FM Post Processing (16-Way Partitioning)

Circuit	K-FM	R-FM	Our Results
s13207.xnf	827	209	507
s15850.xnf	811	228	571
s35932.xnf	1818	294	741
s38417.xnf	2506	449	820
s38584.xnf	2209	314	1258

Table 5.4: Comparison of the Cut-size Results of our Algorithm with K-FM and R-FM (8-Way Partitioning)

Circuit	K-FM	R-FM	Our Results
s13207.xnf	945	270	785
s15850.xnf	1167	320	883
s35932.xnf	3375	373	1190
s38417.xnf	2837	604	1252
s38584.xnf	3825	434	1880

Table 5.5: Comparison of the Cut-size Results of our Algorithm with K-FM and R-FM (16-Way Partitioning)

Circuit	Clustering Based Algorithm	Network Flow Based Algorithm
c880	68	96
c135	80	82
c1908	84	124
c2670	115	146
c3540	137	280
c5315	218	308
c6288	373	301

Table 5.6: Comparison of the Cut-size Results of this Network Flow Based Algorithm and the Clustering Based Algorithm in Chapter 4 (8-Way Partitioning)

Chapter 6

Conclusion

Today, it is common that a system is consisted of millions of transistors. Such a huge circuit is hard to be managed efficiently. As a result, decomposition of these complex systems into finer sub-systems is important in the design process. Each sub-system can then be designed and further improved independently and simultaneously to make the design process faster and simpler.

As the size and complexity of the systems today increase rapidly, more computer aided design tools are required. The development of partitioning algorithms is vital, and it will be essential for the partitioning algorithms to take circuit performance into account. We studied some previous works on circuit partitioning, which include iterative algorithm, network flow based approach, clustering approach, stochastic searching technique, etc. In order to reduce the delay caused by the partition cuts, we take the number of partition cuts along each path into account. We proposed two approaches to solve the problem, of which one is based on an acyclic clustering technique, and the other one is based on the network flow technique.

The basic idea of these two approaches is to maintain an acyclic partitioning of all the combinational paths. Both of them aim at minimizing the

number of partition cuts along the critical paths. In the first method, a modified fanout-free cone decomposition is first used to cluster a given network. This decomposition can effectively reduce the given network to a smaller and sparser one and maintains the acyclic property in the clustering. We then use a similar decomposition to further partition the clustered network into the desired number of partitions. Our algorithm is able to obtain an acyclic multi-way partitioning solution with smaller cut-sizes comparing with the best algorithm reported previously [6]. The second method is a network flow based approach. We proposed a net modelling method to limit the number of partition cuts on a combinational path by maintaining the acyclic property. We make use of the Max-Flow Min-Cut algorithm to generate the min-cut edge and use it in a recursive manner to obtain a multi-way partitioning. An constrained FM post processing step is then applied on the resultant partitioning to further improve the cut-size. Our proposed algorithms can reduce the delay of the circuit caused by the partition cuts efficiently while taking the cut-size into account at the same time.

Bibliography

[1] N. A. Sherwani, *Algorithms for VLSI Physical Design Automation*, pages 1–8, Kluwer Academic Publishers, 1999.

[2] C. J. Alpert, J.-H. Huang, and A. B. Kahng, Multilevel circuit partitioning, in *Proceedings of the 34th annual conference on Design automation conference*, pages 530–533, ACM Press, 1997.

[3] S. Dutt and W. Deng, Vlsi circuit partitioning by cluster-removal using iterative improvement techniques, in *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 194–200, IEEE Computer Society Press, 1996.

[4] H. H. Yang and D. F. Wong, New algorithms for min-cut replication in partitioned circuits, in *Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*, pages 216–222, IEEE Computer Society Press, 1995.

[5] H. Liu and D. F. Wong, Network flow based circuit partitioning for time-multiplexed fpgas, in *Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 497–504, ACM Press, 1998.

[6] J. Cong, Z. Li, and R. Bagrodia, Acyclic multi-way partitioning of boolean networks, in *Proceedings of the 31st annual conference on Design automation conference*, pages 670–675, ACM Press, 1994.

[7] E. L. Lawler, K. N. Levitt, and J. Turner, Module clustering to minimize delay in digital networks, in *IEEE Transactions on Computers*, volume C-18, pages 47–57, 1996.

[8] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vinvrntrlli, On clustering for minimum delay/area, in *Proc. Int. Conf. Computer-Aided Design*, pages 6–9, 1991.

[9] J. D. Huang, J. Y. Jou, W. Z. Shen, and H. S. Chuang, On circuit clustering for area/delay tradeoff under capacity and pin constraints, in *IEEE Transcation on VLSI System*, volume 6, pages 634–642, 1998.

[10] B. Kernighan and S. Lin, An efficient heuristic procedure for partitioning graphs, in *The Bell System Technical Journal*, pages 291–307, 1970.

[11] C. M. Fiduccia and R. M. Mattheyses, A linear-time heuristic for improving network partitions, in *Proceedings of the nineteenth design automation conference*, pages 175–181, 1982.

[12] J. Hwang and A. E. Gamal, Optimal replication for min-cut partitioning, in *ICCAD*, pages 432–435, 1992.

[13] J. Hwang and A. E. Gamal, Min-cut replication in partitioned network, in *Transaction of IEEE on CAD*, pages 96–106, 1995.

[14] G. Karypis and V. Kumar, Multilevel k-way hypergraph partitioning, in *Proceedings of the 36th ACM/IEEE conference on Design automation conference*, pages 343–348, ACM Press, 1999.

[15] N. Metropolis, A. Rosenbluth, M. N. Rosenbluth, A. Teller, and E. Teller, Equations of state calculations by fast computing machines, in *J. Chem. Phys 21*, pages 1087–1092, 1958.

[16] M. Pincus, A monte carlo method for the approximate solution of certain types of constrained optimization problems, in *Oper. Res. 18*, pages 1225–1228, 1970.

[17] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Optimization by simulated annealing, in *Science, Number 4598, 13 May 1983*, volume 220, 4598, pages 671–680, 1983.

[18] T. W. Manikas and J. T. Cain, Genetic algorithms vs. simulated annealing: A comparison of approaches for solving the circuit partitioning problem, in *Technical Report 96-101, Department of Electrical Engineering, The University of Pittsburgh*, 1996.

[19] Y. Wei and C. Cheng, Ratio cut partitioning for hierarchical designs, in *Transaction of IEEE on CAD of IC*, pages 911–921, 1991.

[20] J. H. Holland, Adaptation in natural and artificial systems, in *University of Michigan*, 1975.

[21] T. Bui and B. Moon, Genetic algorithms for graph bisection, in *Technical Report CS-93-07, Department of Computer Science, Pennsylvania State University*, 1993.

[22] H. Yang and D. F. Wong, Efficient network flow based min-cut balanced partitioning, in *1994 IEEE/ACM international conference on Computer-aided design*, pages 50–55, IEEE Computer Society Press, 1994.

[23] J. Cong and S. K. Lim, Performance driven multiway partitioning, in *IEEE/ACM Asia South Pacific Design Automation Conference*, 2000.

[24] J. Cong and Y. Ding, On area/depth trade-off in lut-based fpga technology mapping, in *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pages 213–218, 1993.

[25] E. S. H. Wong, E. F. Y. Young, and W. K. Mak, Clustering based acyclic multi-way partitioning, in *Proceedings of the 13th ACM Great Lakes Symposium on VLSI*, pages 203–206, ACM Press, 2003.

[26] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, page 580, MIT Press, 1990.

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

[28] T. C. Hu and K. Moerder, *VLSI Circuit Design: Theory and Design*, pages 87–93, IEEE Press, 1985.

[29] E. Lawler, *Combinatorial Optimization: Network Flow and Matroids*, Holt, Rinehart, and Winston, 1976.

[30] J. Cong and S. K. Lim, Multiway partitioning with pairwise movement, in *IEEE International Conference on Computer Aided Design*, pages 512–516, 1998.