

VLSI CIRCUIT PARTITIONING FOR SIMULATION AND PLACEMENT

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING AND INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Radwan Tahboub
January 1993

TK
7874
.734
1993

VLSI CIRCUIT PARTITIONING FOR SIMULATION
AND PLACEMENT

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING AND INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Radwan Tahboub

tarafından bağışlanmıştır.

By

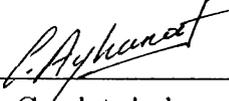
Radwan Tahboub

January 1993

B01426

TK
7874
.T34
1933

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



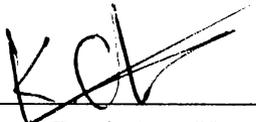
Assoc. Prof. Dr. Cevdet Aykanat(Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



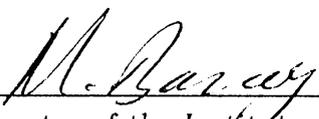
Prof. Dr. Abdullah Atalar

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Assoc. Prof. Dr. Kemal Oflazer

Approved by the Institute of Engineering and Science:



Prof. Mehmet Baray, Director of the Institute of Engineering and Science

ABSTRACT

VLSI CIRCUIT PARTITIONING FOR SIMULATION AND PLACEMENT

Radwan Tahboub

M. S. in Computer Engineering and Information Science

Supervisor: Assoc. Prof. Cevdet Aykanat

January 1993

Simulation time of Very Large Scale Integrated (VLSI) circuits may be improved substantially upon the partitioning of the circuit into several smaller sub-circuits. Node Splitting (NS) is the underlying basis for partitioning of large integrated circuits into several, more manageable, and sometimes similar sub-circuits to enhance computer simulation efficiency. In this thesis, a partitioning scheme based on the NS is used to partition VLSI circuits efficiently. The proposed algorithms will be used as a preprocessing step to increase the efficiency of a VLSI analog circuit simulator designed by the EE Department at Bilkent University. With small modifications, the same algorithms are used to form clusters of transistors based on their interconnections. The clustered circuit will be then partitioned using well known heuristics such as Simulated Annealing and Kernighan-Lin to be used in VLSI placement. The results with this method have been superior to those with the conventional implementations. We have observed a factor of 3-4 speed-up in CPU time, together with 5-10 % improvement in the cut size. Experimental results show that the

proposed algorithms can be efficiently used in VLSI circuit partitioning for simulation and placement.

Keywords: VLSI Circuit Simulation, Placement, Node Splitting, Partitioning, Simulated Annealing, Kernighan-Lin.

ÖZET

BENZETİM VE YERLEŞTİRME İÇİN ÇOK GENİŞ ÖLÇEKLİ TÜMLEŞİK (VLSI) DEVRE PARÇALAMA

Radwan Talboub

Bilgisayar Mühendisliği ve Enformatik Bilimleri Bölümü

Yüksek Lisans

Tez Yöneticisi: Assoc. Prof. Cevdet Aykanat

Ocak 1993

Çok Geniş Ölçekli Tümlleşik (ÇGÖT) devrelerin benzetim süresi devreyi daha küçük birçok alt-devreye parçalayarak oldukça azaltılabilir. Bilgisayar benzetiminin etkinliğini arttırmak için geniş ölçekli devrelerin daha uygun ve bazen de benzer birçok alt-devreye parçalanmasının temelinde Düğüm Bölme (Node Splitting) yatmaktadır. Bu tezde, ÇGÖT devrelerini etkin bir şekilde bölmek için Düğüm Bölme'ye dayalı bir yöntem kullanılmaktadır. Önerilen algoritmalar, Bilkent Üniversitesi Elektrik ve Elektronik Bölümü'nde tasarlanan ÇGÖT analog devre benzetim sisteminin etkinliğini geliştirmek üzere bir ön-işlem adımı olarak kullanılacaklardır. Aynı algoritmalar, küçük değişikliklerle, transistör bağlantılarına göre transistör grupları oluşturmak üzere de kullanılmaktadır. Gruplanan devre daha sonra Tavlama Benzetimi ve Kernighan-Lin gibi bilinen algoritmalar kullanılarak parçalanacaktır. Bu metodun sonuçları, geleneksel metodlarınkine göre daha iyi çıkmıştır. Bağlantı sayısındaki yaklaşık %20'lik azalma ile birlikte CPU zamanında yaklaşık 4

katlık bir hızlanma gözlenmiştir. Deneysel sonuçlar, önerilen algoritmaların benzetim ve yerleştirme için ÇGÖT devre parçalamada etkin bir şekilde kullanılabilceğini göstermektedir.

Anahtar kelimeler : ÇGÖT Devre Benzetimi, Yerleştirme, Düğüm Bölme, Parçalama, Tavlama Benzetimi, Kernighan-Lin.

ACKNOWLEDGEMENT

First I wish to address my special thanks to all helpful people of Turkey. These people made me feel I am living in my own country.

I would like to thank Prof. Dr. Mehmet Baray for his great help since I came to Bilkent University.

To turn now to the actual production of my work, I want to extend special thanks to my supervisor Assoc. Prof. Dr. Cevdet Aykanat for his effective help over a large period, and for encouraging me to choose the best way of research. Also I thank him for providing a lot of support going far beyond technical matters.

I would also like to thank Prof. Dr. Abdullah Atalar and Assoc. Prof. Dr. Kemal Oflazer for their constructive comments on this work.

I take this opportunity to thank all from whom I have learned, both teachers and friends in and out of Turkey. Special thanks goes to Müjdat Pakkan, Satılmış Topçu, and Tevfik Bultan for their great help during my work.

Finally, I wish to extend special thanks to my family , my wonderful wife, Lubna and my sweet little son Rehab (RAMBO). They gave the most for receiving the least.

Contents

| | | |
|----------|---|-----------|
| 1 | INTRODUCTION | 1 |
| 2 | BASIC CONCEPTS | 6 |
| 2.1 | VLSI Circuit Components and Models | 6 |
| 2.2 | Review of Graph Theory | 9 |
| 2.2.1 | Undirected Graphs | 9 |
| 2.2.2 | Directed Graphs | 10 |
| 2.2.3 | Graph Representations | 11 |
| 2.2.4 | Graph Traversals and Connected Components | 14 |
| 2.2.5 | Topological Sort | 16 |
| 2.2.6 | Node Splitting | 18 |
| 3 | VLSI CIRCUIT PARTITIONING FOR SIMULATION | 20 |
| 3.1 | VLSI Circuit Simulation | 20 |
| 3.2 | Graph Representations for Partitioning | 23 |
| 3.3 | Splitting and Clustering of Graph G | 26 |

| | | |
|----------|---|-----------|
| 3.4 | Ordering | 32 |
| 3.4.1 | Constructing Digraph of the Partitioned Circuit | 37 |
| 3.4.2 | Finding the SCCs of the Digraph D | 38 |
| 3.4.3 | Leveling the SCCs | 41 |
| 3.4.4 | VLSI Circuits with Large Feedback Paths | 43 |
| 3.5 | Testing and Experimental Results | 52 |
| 3.6 | Mapping Elements Different from CMOS Transistors to the Partitions | 53 |
| 4 | VLSI PARTITIONING FOR PLACEMENT | 56 |
| 4.1 | VLSI Circuit Placement | 56 |
| 4.1.1 | Review of Min-Cut Partitioning Algorithms | 57 |
| 4.2 | Partitioning Problem | 59 |
| 4.2.1 | The proposed Clustering Approach | 60 |
| 4.3 | Theoretical and Practical Issues | 60 |
| 4.4 | Implementations and Results | 65 |
| 5 | CONCLUSIONS | 69 |

List of Figures

| | | |
|-----|--|----|
| 1.1 | Node Tearing. $N_1, N_2, N_3, N_4,$ and N_5 are the tearing nodes. | 3 |
| 2.1 | An example which shows how to represent a MOSFET transistor in SPICE format. | 7 |
| 2.2 | Functions used in modeling the VLSI circuits. | 8 |
| 2.3 | An example of a graph and its adjacency multilists. | 13 |
| 2.4 | Non-recursive DFS algorithms. | 14 |
| 2.5 | Non-recursive BFS algorithms. | 15 |
| 2.6 | Algorithm to find the topological sort of a dag. | 16 |
| 2.7 | Algorithm to find connected components of a graph. | 17 |
| 2.8 | Algorithm SCC to find the strongly connected components of a graph. | 18 |
| 2.9 | (a) A loop-free graph G (b) The graph obtained by splitting v_1 and v_2 in G | 19 |
| 3.1 | A CMOS example circuit. | 25 |
| 3.2 | (a)The graph G obtained from the example circuit (b) The adjacency multilists of G | 27 |

| | | |
|------|--|----|
| 3.3 | The proposed algorithm for the implementation of the clustering scheme indicated in Definition 2. | 30 |
| 3.4 | The graph $G \circ V_I$ obtained from splitting the input nodes of the graph H | 31 |
| 3.5 | The digraph D constructed for the clustered example circuit. | 34 |
| 3.6 | The In-degree and Out-degree lists of the example circuit . | 36 |
| 3.7 | Reducing the number of edges in the digraph representation. | 38 |
| 3.8 | The steps used in constructing the In-degree lists of D . | 39 |
| 3.9 | (a)The derived digraph D of the example circuit. (b) Its transpose graph D^T | 40 |
| 3.10 | The Leveling algorithm . | 42 |
| 3.11 | The Grouping heuristic . | 43 |
| 3.12 | Illustrating the grouping of levels to find final partitions. | 44 |
| 3.13 | Converting connections between blocks into signal lines (nets). | 46 |
| 3.14 | Reducing the number of nets in the hypergraph of S_i | 47 |
| 3.15 | The steps used in constructing the Net-lists of the hypergraph . | 48 |
| 3.16 | Illustrating the grouping of levels with large SCC to find final partitions. | 50 |
| 3.17 | The Summary of the Partitioning algorithm with both leveling and FM heuristics . | 51 |
| 4.1 | An example of CMOS transistors cluster. | 61 |
| 4.2 | Layout of two transistors.(a) With conventional partitioning. (b) Partitioning with clustering. | 64 |

List of Tables

| | | |
|-----|--|----|
| 3.1 | Results of the partitioning algorithms with leveling tested on 9 real problem instances. | 54 |
| 3.2 | Results of the partitioning algorithms using FM heuristics tested on 9 real problem instances. | 54 |
| 4.1 | Comparing the FM and the CFM results. The algorithms were tested on 9 real problem instances. | 66 |
| 4.2 | Comparing the SA and the CSA results. The algorithms were tested on 9 real problem instances. | 67 |

1. INTRODUCTION

There are several steps involved in the design of a Very Large Scale Integrated (VLSI) circuits, which may consist of several hundreds of thousands of components, mainly transistors. The total time spent in the design loop is usually referred as the *turn-around time*. The main objective of the VLSI designer is to obtain designs with as low a turn-around time as possible. Computer-Aided Design (CAD) tools have become virtually indispensable at various steps in the design process to perform tasks which would, otherwise, take a very large time if they were done by human beings. Thus any serious entry into the VLSI design requires access to suitable CAD tools. There is, however, a bottleneck in speeding up the design process. This bottleneck is in the simulation of the electrical behavior of the circuit due to the unavailability of a simulation tool that is capable of accurately predicting the performance of an entire VLSI circuit at a reasonable cost. The accuracy of the simulator is important, since otherwise the integrated circuit which is fabricated and tested might turn out to perform rather unsatisfactory. For large circuits (typically $> 10K$ transistors), the speed of simulation is equally important so that the entire circuit can be simulated in a reasonably small amount of computation time. However, speed and accuracy of a simulator are often conflicting requirements among existing simulation tools.

Most of the existing simulators for integrated circuits can be classified into two distinct categories, namely, *analog simulators* and *digital simulators*. Analog simulators treat the circuit as a continuous dynamical system with electrical signals such as voltages and currents [23, 2, 5, 6]. Such simulators can

be used to predict the performance of integrated circuits very accurately, however, using these simulators is not compute-effective for VLSI circuits, since they take a large amount of CPU time. Digital simulators, on the other hand, view the circuit as a digital network with signals occupying discrete states such as low (0) and high (1) [3, 4]. Such simulators operate at sufficient speeds. However, these simulators do not model the dynamics of the circuit properly, hence the accuracy of these simulators is not sufficient.

An ideal simulator for VLSI circuits would be one which has the speed and efficiency of digital simulators while providing the accuracy and detail of an analog simulator.

In the EE department at Bilkent University, a new analog circuit simulator is under development [17]. This simulation tool approximates the non-linear elements characteristics by piecewise-linear (PL) functions. This method transforms the set of non-linear algebraic equations describing the system into a set of linear algebraic equations at each time point. These set of equations in turn are solved by using the LU factorization methods. As the size of the circuit grows, the solution time of this set of equations dominates, and the computation time of the solution of the process may reach a point for which the simulator is no longer compute-effective.

Partitioning the circuit into sub-circuits or blocks may substantially reduce the computation time of the solution of the process while using less memory space. If the sizes of the partitioned sub-circuits are adjusted to a point where the simulator works most efficiently then this would make the whole tool compute-effective, i.e., reasonable speed and efficiency while keeping the solution accuracy. But, unfortunately, the sizes of the sub-circuits is not the only restriction that should be taken into account, the way a circuit is partitioned is also very important. For example, if partitioning is done such that strong couplings exists between partitioned sub-circuits, the computation power (i.e., time and memory) could be slowed down to the extent that the simulation process may require more computations and memory than the direct method.

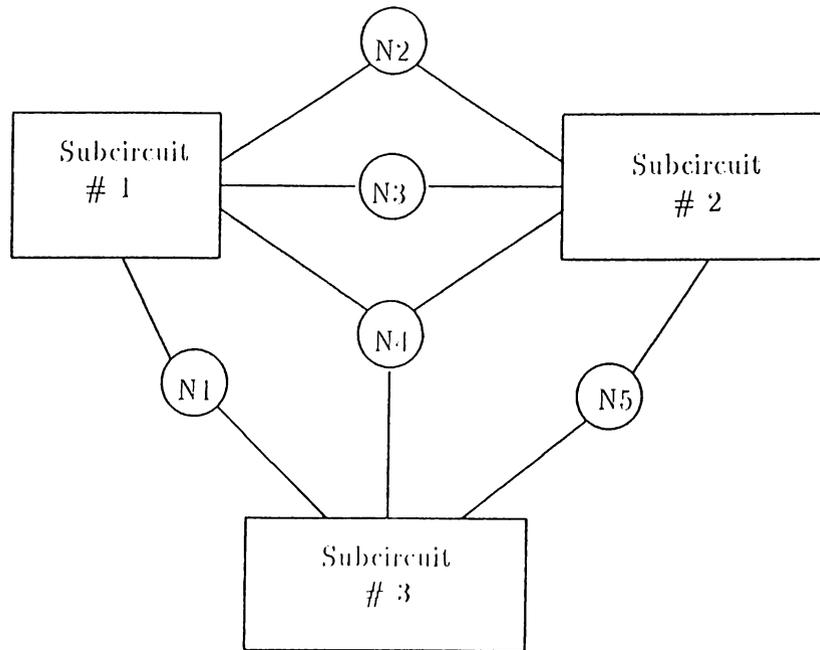


Figure 1.1. Node Tearing. N1, N2, N3, N4, and N5 are the tearing nodes.

Node Splitting (NS) is the underlying basis for partitioning of large integrated circuits into several, more manageable, and sometimes similar subcircuits to enhance computer simulation efficiency. In this work, partitioning algorithms based on the NS scheme will be proposed to be used at the outset as a preprocessing step for the simulation tool being built in the EE Department at Bilkent University. The main aim of this work is to increase the computation power of this simulator by decreasing its speed and memory requirement. Similar work has been done for other simulators. For example a fixed partitioning method depending on the equivalent conductances and capacitances between two adjacent nodes is proposed in [8]. Another dynamic partitioning method is proposed by O.Tejayadi and N.Hajj [11]. Other works is also done for digital simulators [7, 9, 10, 16].

Node tearing (splitting) for circuit simulation has been discussed by Sangiovanni-Vencentelli et.al. [1]. The idea is to divide a network into a set of subnetworks. The nets, or nodes, that are common between them are the tearing nodes, see Figure 1.1. Each subnetwork can then be analyzed separately and the solution is obtained in terms of the tearing nodes.

The proposed partitioning algorithm exploits the inherent partitions or sub-blocks in the circuit. This is done by first splitting the input nodes using a linear time splitting algorithm. After splitting the input nodes, the inherent blocks or clusters which is naturally found in VLSI circuit design can be determined, and the transistors in each sub-block are common channel-connected. Fortunately, and as is expected, the number of such blocks is large enough to adjust the sizes of the final partitions while keeping the interconnections between those partitions within reasonable limits. An algorithm which can determine the feedback paths between the split blocks is used, and all blocks that lies on the same feedback path are collapsed together to form a larger sub-block. If the sizes of those sub-blocks is reasonable, an ordering among those blocks can be found. The simulation of the entire circuit follows an event scheduler similar in many ways to gate-level logic simulators [7], except that now the gates consist of channel-connected transistors. Unfortunately VLSI circuits contains large feedback paths and sometimes most of the circuit blocks turns out to be on the same feedback path, hence a special partitioning procedure should be used. In such cases no ordering can be found and the solution of the circuit becomes more complex and takes more time.

Partitioning VLSI circuits is not only used for speeding the simulation phase in the design process. It also arises in various aspects of VLSI design automation. For example it has direct applications in the placement of components during layout [43]. There are large number of heuristic algorithms¹ that can efficiently partition the given circuit for placement. Most of the existing algorithms take the problem as a pure graph theory problem and model it in an abstract way not taking into account the inherent couplings between the elements. On the contrary, in our work, we are making use of the partitioning algorithms used in the simulation problem and with small modifications we are doing a preprocessing step to form clusters of elements (common-channel connected elements). Then using a well known heuristics such as Kernighan-Lin and Simulated Annealing, a partitioning can be done on those clusters. The results with this proposed method are found to be superior to those with the

¹See Section 4.1.1 for the review of those algorithms.

conventional implementations.

In chapter two, basic definitions relating network and graph theory are reviewed. Node splitting will be also discussed. Chapter 3 presents the partitioning algorithm and the ideas used to make it efficient. In chapter 4 we present how to use the splitting algorithm to speed up the partitioning process for the placement problem. Conclusions and results are drawn in chapter 5.

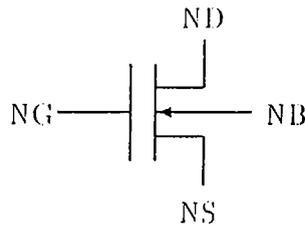
2. BASIC CONCEPTS

In this chapter a basic VLSI circuit model that can be used in partitioning algorithms is presented. Related definitions and theorems from graph theory are also presented.

2.1 VLSI Circuit Components and Models

A VLSI circuit Ω consists of a set of *nodes* N interconnected by a set of *elements*, mainly transistors, M . The circuit description can be extracted directly from the layout using computer programs known as *circuit extractors* [50]. The circuit extractors used at Bilkent University gives a circuit description output in the form of a SPICE file format. In this format, the circuit to be analyzed is described by a set of element cards, which define the circuit topology and element values, and a set of control cards. For example, a MOSFET transistor card can be specified as shown in Figure 2.1. The nodes N_D , N_G , N_S , and N_B denote the *drain*, *gate*, *source*, and *bulk* (substrate) nodes, respectively. MXXXXXXX is the transistor name, and the MNAME is the model name. Here, L and W denote the channel length and width respectively. Other elements cards and parameters are described in details in the SPICE User's Guide [23]. As it will be shown later, the only important parameters for the graph representation of the MOSFET transistors are the N_D , N_G , and N_S nodes. Other parameters are important for simulation and are not used in our models.

In VLSI circuit modeling, there are two types of nodes, namely,



MXXXXXXXX N_D N_G N_S N_B MNAME $\langle L \rangle \langle W \rangle \langle \text{other parameters} \rangle$

Figure 2.1. An example which shows how to represent a MOSFET transistor in SPICE format.

Input Nodes, which are modeled as voltage sources and provide the strongest signals to the network from the outside. Examples of input nodes include the power supply (V_{DD}), and the (GND), as well as the input clock signals.

Normal Nodes or Storage nodes, which are the remaining nodes in the circuit. These are the weakest nodes as they cannot force their signals on a stronger node but are capable of storing a signal dynamically. These nodes, in turn, can be further subdivided into several strength classes depending upon their relative capacitance values.

A MOSFET transistor is modeled as a three-terminal device with a switch between the drain and source terminals and the signal at the gate controlling the state of the switch. Only transistors whose drain and source nodes are different will be taken into consideration. In some technologies, the drain and the source of a transistor might correspond to the same node in the layout as a means of implementing a capacitor. We shall, however, assume that circuit extraction program used at Bilkent University identifies this situation correctly as a capacitor rather than a transistor. Generally, the extraction program provides two types of MOSFET transistors, namely,

| | |
|-------------|---|
| NodeType | : $N \Rightarrow \{\text{input, normal}\}$ |
| ElementType | : $M \Rightarrow \{\text{Transistor, Capacitor, Resistor, ...}\}$ |
| Gate | $M \Rightarrow N$ |
| Source | $M \Rightarrow N$ |
| Drain | $M \Rightarrow N$ |
| Terminal1 | $M \Rightarrow N$ |
| Terminal2 | $M \Rightarrow N$ |

Figure 2.2. Functions used in modeling the VLSI circuits.

n-channel, enhancement type,
 p-channel, enhancement type.

As will be shown later, the type of the transistor has no role in the representation of the transistor in its mathematical model. But, it is very important to know the type of the transistor in the simulation phase.

Mathematically, the VLSI circuit $\Omega(N, M)$ can be specified by giving a listing of nodes in N and elements (mainly transistors) in M and the functions specified as shown in Figure 2.2. In todays technology, typical values of M and N are > 20000 elements and > 15000 nodes, respectively

As mentioned before, the simulator at Bilkent University, approximates the non-linear elements characteristics by piecewise-linear (PL) functions. This method transforms the set of algebraic-differential equations describing the system into a set of algebraic equations at each time point. These equations, in turn, can be solved using the LU factorization method. The size of the system matrix that represents the circuit is proportional to $(2M + N)$, hence solving the circuit at a time point, say t_i , requires the solution of approximately

$(2M + N)$ set of linear algebraic equations. The system matrix is not static and it might change from time to time depending on the state of the non-linear elements involved in the circuit. So the amount of computation carried out in the simulation phase is very large, and special techniques should be used to decrease the computation time of this phase. Simulation time may be reduced substantially by partitioning of a VLSI circuit into several smaller sub-circuits. A detailed description of the mentioned simulator can be found in [17].

2.2 Review of Graph Theory

It is almost impossible to perform operations directly on a VLSI circuits without representing it in a suitable model that can easily provide operations such as search, update, and traverse of the circuit. So, it is much easier to formally present our ideas and concepts if the network is viewed as a *graph*; therefore, some basic fundamentals from graph theory are reviewed in the following subsections.

2.2.1 Undirected Graphs

An *undirected graph* $G(V, E)$ is defined as a non-empty set of *vertices* V , and a set of edges E which connects the vertices in V . If e is an edge and ν and ω are vertices such that $e = (\nu, \omega)$, then e is said to join ν and ω , the vertices ν and ω are called the ends of e and further, ν and ω are said to be adjacent in G .

A *path* of length K from a vertex u to a vertex u' in graph G is a sequence $\langle v_0, v_1, \dots, v_k \rangle$ of vertices such that $u = v_0$, $u' = v_k$ and $(v_{i-1}, v_i) \in E$. The length K of the path is the number of edges in the path. We say that u' is *reachable* from u ($u \sim u'$) via P if there is a path P from u to u' . A path is *simple* if all its vertices, except the origin (v_0) and the terminus (v_k) are distinct. In an undirected graph, a path $\langle v_0, v_1, \dots, v_k \rangle$ forms a *cycle* if $v_0 = v_k$ and v_1, v_2, \dots, v_k are distinct. A *self-loop* is a cycle of length 1. The

vertices of a graph are said to be *loop-free* if the graph does not contain any self-loops. An undirected graph is *connected* if every pair of vertices is connected by a path. The *connected components* of a graph are the equivalence classes of vertices under the “is reachable from” relation.

There are two variants of undirected graphs that are commonly used in the graph theoretical study of VLSI circuits. A *multigraph* is like a loop-free undirected graph with multiple edges between vertices. A *hypergraph* is like an undirected graph, but each *hyperedge (net)* rather than connecting two vertices, connects an arbitrary subset of vertices (*terminals*).

2.2.2 Directed Graphs

A *directed graph* $D(V, E)$, often abbreviated as a *digraph*, is defined as a nonempty set of vertices $V(D)$, and a set of directed arcs $E(D)$ which connects the vertices in $V(D)$. If a is an arc and ν and ω are vertices such that $a = (\nu, \omega)$, then a is said to *join* ν to ω ; ν is the *tail* of a , and ω is its *head* and the arc is usually referred to as simply (ν, ω) .

Path, simple path and reachability definitions given for undirected graphs also apply to the digraphs. In a digraph, a path $\langle v_0, v_1, \dots, v_k \rangle$ forms a *cycle* if $v_0 = v_k$ and the path contains at least one edge. The *cycle* is simple if, in addition, v_1, v_2, \dots, v_k are distinct. A directed graph is *strongly connected* if every two vertices u and v are reachable from each other, i.e., $u \sim v$ and $v \sim u$. The strongly connected components of a graph are the *equivalence* classes of vertices under the “are mutually reachable” relation.

In a digraph, the *in-degree* and *out-degree* of a vertex are the number of edges entering and leaving it respectively. The *degree* of a vertex is the sum of its in-degree and out-degree.

The *transpose* of a digraph $D(V, E)$ is defined as the graph $D^T(V, E^T)$, where $E^T = \{(u, \nu) : (\nu, u) \in E\}$. That is, E^T consists of the edges of D with their directions reversed. It is interesting to observe that D and D^T have

exactly the same strongly connected components: ω and ν are reachable from each other in D if and only if they are reachable from each other in D^T .

2.2.3 Graph Representations

While several representations for graphs are possible, we shall study only the most commonly used: adjacency matrices, adjacency lists, adjacency multilists, in-degree and out-degree lists for directed graphs, and net-lists and connected lists for hypergraphs. The choice of a particular representation depends upon the application and the function to be performed on the graph.

Adjacency Matrix

Let $G = (V, E)$ be a graph with n vertices, $n \geq 1$. The adjacency matrix of G is a 2-dimensional $n \times n$ array, say A , with the property that $A[i, j] = 1$ if the edge (ν_i, ν_j) is in E and $A[i, j] = 0$ if there is no such edge in G . Adjacency matrices can be also used to represent directed graphs. However, VLSI circuits can always be represented by sparse graphs. The degrees of the vertices in a sparse graph are much smaller than n . Hence, adjacency matrix representation is not used in VLSI circuits applications since it introduces $O(n^2)$ space complexity.

Adjacency Lists

In this representation, n rows of the adjacency matrix are represented as n linked lists. There is one list for each vertex in G . The nodes in list i represent the vertices that are adjacent to vertex i . Each node has at least two fields: vertex and link. The vertex fields contain the indices of the vertices adjacent to vertex i . Each list has a headnode. The headnodes are sequential providing easy random access to the adjacency list for any particular vertex.

Adjacency list representation of digraphs necessitates maintaining two lists, *In-degree* and *Out-degree* lists. These lists are very similar to the adjacency

lists used in representing undirected graphs. However there are two lists for each vertex in the directed graph D . For the in-degree lists, the nodes in list i represent the vertices $\nu_k \in V$ such that the edge $(\nu_k, i) \in E$. For the out-degree lists, the nodes in list i represents the vertices $\nu_k \in V$ such that the edge $(i, \nu_k) \in E$.

Adjacency Multilists

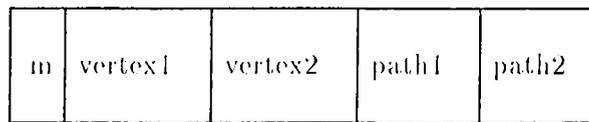
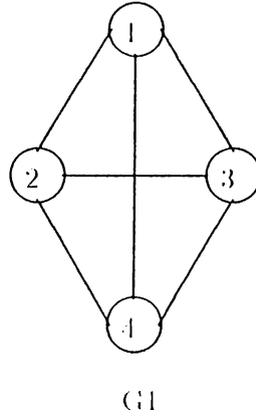
In the adjacency list representation of an undirected graph, each edge (ν_i, ν_j) is represented by two entries, one on the list for ν_i and the other on the list for ν_j . However, in the graph theoretical study of VLSI circuits, each edge corresponds to a component in the VLSI circuit, (e.g. transistors, capacitors, . . . , etc.)¹. Hence, adjacency list representation necessitates the duplication of the circuit information. Furthermore, graph theoretical algorithms developed in this work requires a data structure which enables easy marking of an edge as being processed. This can be accomplished easily if the adjacency lists are mutually maintained as multilists. That is, each edge is represented by a simple node which is in the adjacency lists of the two vertices it is incident to. Unfortunately, adjacency list structure requires the marking of the same edge in two lists. Figure 2.3 illustrates the adjacency multilists of a sample graph. In this scheme, a one bit field, m , is allocated for marking purposes.

Net-lists and Connected Lists

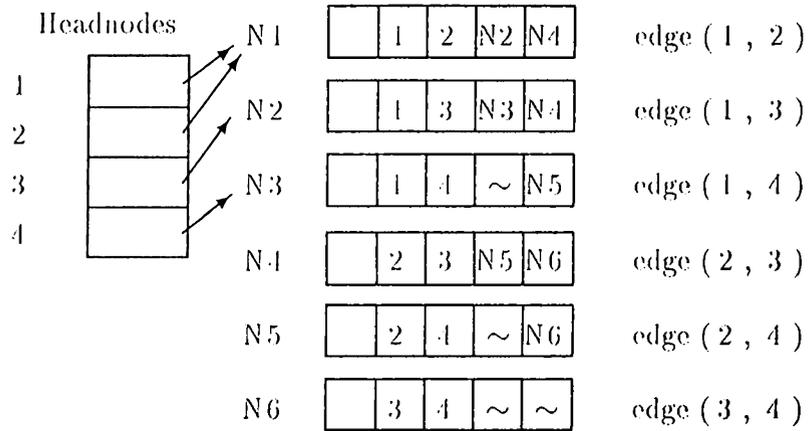
It is very difficult, if not impossible, to represent hypergraphs with any of the previous graph representations. A suitable representation for the hypergraphs can be achieved by using either the *net-list* or *connected list* representations.

In the net-lists representations, there is one list for each net or hyperedge, so all vertices incident to a particular net are found in the list of that net. Each net consists of at least two vertices, and each vertex is contained in at least one

¹See Section 3.2.



Node Structure



The lists are :

- vertex 1 : N 1 → N2 → N3
- vertex 2 : N 1 → N4 → N5
- vertex 3 : N 2 → N4 → N6
- vertex 4 : N 3 → N5 → N6

Adjacency Multilists for G1.

Figure 2.3. An example of a graph and its adjacency multilists.

```
Non-Recursive DFS( $\nu$ )  
  
  Mark vertex  $\nu$  as visited  
  Push  $\nu$  into the stack  
  If the stack is not empty Then  
    Pop a vertex  $\mu$  from the stack  
    For a vertex  $\omega$  adjacent to  $\mu$  do  
      If  $\omega$  is not marked Then  
        Mark vertex  $\omega$  as visited  
        Push  $\omega$  into the stack  
      End If  
    End For  
  End If  
  
End DFS.
```

Figure 2.4. Non-recursive DFS algorithms.

net. In the graph theoretical algorithms developed in this work, it is important to find all nets incident to a particular vertex. It is clear that it would cost too much to find such an information directly from the net-lists, hence another list for each vertex that keeps nets incident to that vertex is needed. This can be achieved by using the connected lists. Generally, the hypergraph is stored as a net-lists, and the connected lists are then constructed, if necessary.

2.2.4 Graph Traversals and Connected Components

Given a graph $G(V, E)$ and a vertex $\nu \in V$ we are interested in visiting all vertices in G that are reachable from ν (i.e, all vertices connected to ν). This can be achieved by using either *depth first search* or *breadth first search*.

```

Non-Recursive BFS( $\nu$ )
  Mark vertex  $\nu$  as visited
  Put  $\nu$  into the queue
  If the queue is not empty Then
    get a vertex  $\mu$  from the queue
    For each vertex  $\omega$  adjacent to  $\mu$  do
      If  $\omega$  is not marked Then
        Mark vertex  $\omega$  as visited
        Put  $\omega$  into the queue
      End If
    End For
  End If

End BFS.

```

Figure 2.5. Non-recursive BFS algorithms.

Depth First Search

Figure 2.4 shows the non-recursive version of the depth first search (*DFS*) algorithm. If the graph is represented by its adjacency multilists, the time complexity of the DFS algorithms is $O(c)$, where c is the number of edges in G . i.e, $c = |E|$.

Breadth First Search

Breadth first search (*BFS*) differs from the depth first search in that all unvisited vertices adjacent to ν are visited next. Then unvisited vertices adjacent to these are visited and so on. The complexity of the *BFS* algorithms is the same with that of the *DFS* algorithms. Figure 2.5 presents the *BFS* algorithm.

We now look at three simple, but very important, applications of graph

```

Tsort( $D$ )
    Call DFS to compute finishing times for each vertex  $\nu$ 
    As each vertex is finished, push it into a stack
End Tsort.

```

Figure 2.6. Algorithm to find the topological sort of a dag.

traversals: (i) topological sort, (ii) finding the connected components of an undirected graph, and (iii) finding the strongly connected components (SCC) of a directed graph.

2.2.5 Topological Sort

A *topological sort* of a directed acyclic graph (dag) $D(V, E)$ is a linear ordering of all its vertices such that if D contains an edge $(u, v) \in E$, then u appears before v in the ordering. If the digraph is not acyclic, then no linear ordering is possible. A topological sort of a graph can be viewed as an ordering of its vertices along a horizontal line such that all directed edges go from left to right. The simple algorithm given in Figure 2.6 performs the topological sort of a dag [49]. The *finishing* time of a vertex ν is a labeling given to the vertex after it leaves the stack of the *DFS* algorithm forever [49]. That is vertex ν is visited and all of its adjacent vertices are either in the stack or marked as visited. When the algorithm finishes a vertex v , this vertex is pushed into a stack. The stack retained by this algorithm denotes the topological sorted ordering.

Connected Components

The connectedness of an undirected graph can be easily determined by making a call to either *DFS* or *BFS* and then checking an unvisited vertex. To find all connected components of the graph, repeated calls to $DFS(\nu)$ or $BFS(\nu)$,

COMP(G)

Mark all vertices as unvisited

While there is an unvisited vertex ν **do** Call $BFS(\nu)$

Mark all the vertices of each component by different labeling

End While**End COMP.**

Figure 2.7. Algorithm to find connected components of a graph.

with ν a vertex not yet visited, is needed. Figure 2.7 shows the algorithm for finding the components of a graph G . The algorithm uses BFS , however, DFS may be used as well.

If G is represented by its adjacency multilists, then the total time taken by BFS is $O(e)$. The total time to generate all connected components is $O(n + e)$. Where $n = |V|$ and $e = |E|$.

Strongly Connected Components

We now consider a second classic application of graph traversals: decomposing a directed graph (digraph) into its strongly connected components. This subsection shows how to do this using one call to the DFS algorithm followed by another call to BFS algorithm [49].

Recall from Section 2.2.2 that strongly connected components (SCC) of a graph $D(V, E)$ is a maximal set of vertices $V' \subseteq V$ such that $\nu \sim \omega$, and $\omega \sim \nu$ holds for each pair of vertices ν and ω in V' , that is, vertices ν and ω are reachable from each other. The algorithm shown in Figure 2.8 is one way of finding the SCC of a directed graph.

SCC(D)

Call *DFS* to compute finishing times for each vertex v
 Construct D^T
 Call *BFS* for each vertex (in D^T) in decreasing
 order of their finishing times
 Mark the vertices of each SCC by different labeling

End SCC.

Figure 2.8. Algorithm SCC to find the strongly connected components of a graph.

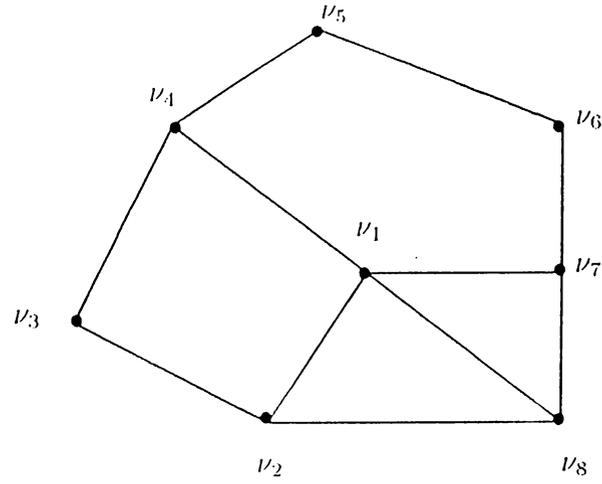
2.2.6 Node Splitting

In this subsection, we will introduce the notion of *splitting* a vertex in a graph G . In the next chapter an efficient algorithm suitable for splitting graphs representing VLSI circuits will be proposed.

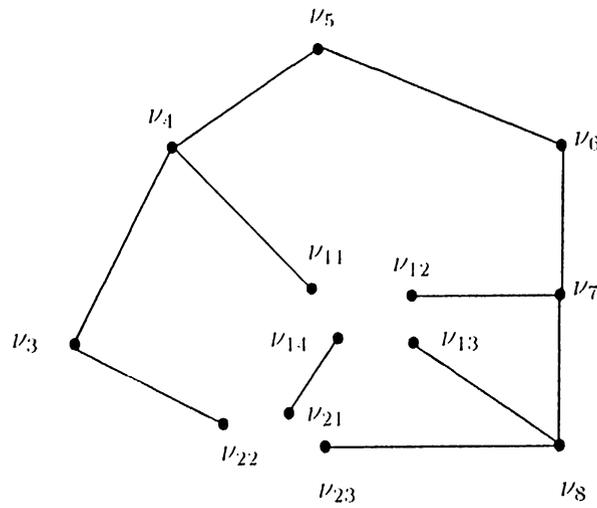
Consider an undirected graph $G(V, E)$ and a vertex $v_i \in G$ of degree $d_{v_i} = k \geq 1$. Let v_i be a loop-free vertex. The v_i -*split* graph or the graph obtained by splitting v_i in G , is a graph obtained by splitting the vertex v_i into k new vertices $v_{i1}, v_{i2}, \dots, v_{ik}$, with each edge formerly joining the vertex v_i to vertex v_j now joining v_{il} to v_j . We denote the v_i -*split* graph as Gov_i . Thus, splitting a vertex with $d_{v_i} = k$ creates a new graph with $k - 1$ more vertices but with the same set of edges. It is obvious to show that if $k = 1$, then splitting the vertex v_i does not alter the graph, so $Gov_i = G$ if $d_{v_i} = 1$. If $V' = \{v_1, v_2, \dots, v_q\}$ is a subset of loop-free vertices in G then the V' -*split* graph of G can be defined as shown in the following recursive equation:

$$HoV' = (\dots((Hov_1)ov_2)\dots)ov_q \quad (2.1)$$

The graph Gov' is well-defined since the order in which the vertices of V' are split does not matter. An example that shows the node splitting is presented in Figure 2.9.



(a)



(b)

Figure 2.9. (a) A loop-free graph G (b) The graph obtained by splitting ν_1 and ν_2 in G .

3. VLSI CIRCUIT PARTITIONING FOR SIMULATION

In this chapter, a partitioning scheme that is used in partitioning a VLSI circuit as a preprocessing step for the efficient simulation phase of the PL-AWE simulation tool is discussed. This partitioning scheme, can be considered as a CAD tool to increase the memory and speed efficiency of the PL-AWE simulation algorithms developed in the EE Department at Bilkent University. In Section 3.1, a general introduction about the need of partitioning for VLSI simulation is stated. Section 3.2 presents how to represent the VLSI circuit using the graph concepts discussed in the previous chapter. Schemes and algorithms used in Partitioning algorithm will be also presented and discussed in details in this chapter.

3.1 VLSI Circuit Simulation

VLSI circuit simulation is one of the most critical and time consuming computational tasks to be performed in VLSI circuit design. State-of-the-art VLSI circuit design requires extensive and accurate simulation. Not only must the circuit be simulated under nominal conditions, but it must also be simulated under a variety of operating conditions. For large circuit designs, SPICE like simulations may require many days on a large, mainframe computers. Such simulators can provide the required accuracy, but, not the throughput to satisfy design requirements.

A new analog circuit simulator, PL-AWE, is under development in the Electrical Engineering Department of Bilkent University [17]. This project involves the development of efficient algorithms which guarantees the convergence of the simulation while maintaining the desired accuracy. However, the speed and memory efficiency of the PL-AWE can be further increased by exploiting the divide-and-conquer strategy since PL-AWE has $O(N^{1.5})$ approximate computational complexity. In divide-and-conquer strategy, the VLSI circuit $\Omega(N, M)$ is partitioned into smaller, not necessarily equal, sub-circuits $\Omega_1(N_1, M_1), \Omega_2(N_2, M_2), \dots, \Omega_K(N_K, M_K)$. The simulations of those sub-circuits should be performed and then a scheme should be found to combine these sub-simulation solutions into a simulation solution of the whole circuit. Assume that $k = 2$ for the sake of clarity of the illustration of the proposed divide-and-conquer strategy. Then, the following three different decomposition cases should be considered in combining the solution of $\Omega_1(N_1, M_1)$ and $\Omega_2(N_2, M_2)$ to obtain the solution of the whole circuit $\Omega(N, M)$.

First, assume that the circuit Ω can be partitioned into two independent sub-circuits Ω_1 and Ω_2 . Then, the solution of each sub-circuit is independent from the solution of the other. It would, of course, take less time and less memory space to simulate Ω_1 and Ω_2 separately than simulating the whole circuit.

Second, a more realistic case, would be the one in which Ω_1 and Ω_2 have some connections between them with the following properties:

- All external connections incident to *the normal nodes* in Ω_1 and Ω_2 are gate (N_G) inputs. Other connections incident to *the input nodes* can be included.
- All gate connections between Ω_1 and Ω_2 are in the same direction, i.e., all connections are either from Ω_1 to Ω_2 or from Ω_2 to Ω_1 .

The first condition is a consequence of an electrical property in which the simulation of a partitioned circuit would be more easier if the connections

between the partitioned blocks are *zero* current links. The second condition guarantees feedback free partitions, for which the simulation of the partitioned circuit might be easier. It is obvious that one could not start simulating Ω_i before having all of its inputs from other partitions defined, hence an ordering among the partitioned sub-circuits should be found for the combination of the simulation results. Once again, if Ω_1 is not taking any inputs from Ω_2 , then simulating Ω_1 and then simulating Ω_2 is expected to take less time and memory space than simulating the whole circuit.

Third, in today's technology, most of the VLSI circuits contains feedback paths between the sub-circuits, so the previous two cases cannot be used in modeling the partitioned sub-circuits with feedback paths. We restate the conditions for this case as follows:

- All external connections incident to *the normal nodes* in Ω_1 and Ω_2 are gate (N_G) inputs. Other connections incident to *the input nodes* can also be included.
- All gate connections between Ω_1 and Ω_2 can be in both directions, i.e. connections are either from Ω_1 to Ω_2 or from Ω_2 to Ω_1 and at least one connection is in the reverse direction.
- The number of gate connections is as small as possible.

It is clear that directly simulating Ω_1 is not possible since some inputs from Ω_2 are not available yet, also simulating Ω_2 before Ω_1 is not possible for the same reason. So special techniques should be used in the simulation of the partitioned circuits with inter-feedback paths. Such techniques cause considerable overhead in the combination of the sub-circuits results. This computational overhead is proportional to the total number of interconnections between the sub-circuits Ω_1 and Ω_2 . Such decomposition cases should be avoided, if possible. Otherwise, decomposition scheme should try to minimize the total number of interconnections as much as possible.

In partitioning VLSI circuits for the simulation problem, the partitioning algorithm should not change the characteristics of the circuit. That is, the final solution of the VLSI circuit obtained by the direct simulation method should be the same with that obtained by simulating the circuit after partitioning. Hence, partitioning the VLSI circuit for simulation can be defined as follows:

Given a VLSI circuit $\Omega(N, M)$, find a partitioning Σ that partitions Ω into several transistor disjoint subnetworks, $\Omega_1, \Omega_2, \dots, \Omega_K$, where each subnetwork or block Ω_i has a certain special configuration that would aid the simulation process. The partitioning strategy is basically to group together a set of transistors to constitute a subnetwork or a block if they have a common-channel-path between their source and drain nodes. However, a suitable graph representation for the VLSI circuits should be devised in order to apply graph theoretical algorithms for partitioning. The next sections presents a suitable graph representation for VLSI circuits and the related algorithms used in partitioning the circuits.

3.2 Graph Representations for Partitioning

The scope of this work is limited to VLSI circuits which contains CMOS transistors as three terminal devices, and capacitors, inductors, resistors, etc. as two terminal devices.

Recall from Section 2.1 that a VLSI circuit $\Omega(N, M)$ consists of a set of nodes N interconnected by a set of elements, mainly transistors, M . Let N_I denote the set of *input* nodes, also, let N_N denote the set of *storage or normal* nodes in $\Omega(N, M)$. Note also that, *GND* or node 0 is treated as an input node. Let

$$M_{CMOS} = \{m \in M : \{ElementType(m) = CMOS\ Transistor\}\} \quad (3.1)$$

denote the set of CMOS transistors in the network. Let $N_o \subseteq N_N$ be the subset of normal or storage nodes at which the user wishes to observe the

output waveforms. Also, let

$$N_G = \{n \in N : n = Gate(m); \quad m \in M_{CMOS}\} \quad (3.2)$$

denote the set of nodes that are gate nodes of the CMOS transistors in the network.

A VLSI circuit $\Omega(N, M)$ can be represented by a sparse undirected multi-graph $G(V, E)$ as follows. Each node in Ω is associated with a vertex in G . Each transistor in Ω is represented by an undirected edge between a pair of vertices in G which corresponds to the drain and source nodes of that transistor. Each two terminal element in Ω is also represented by an undirected edge between a pair of vertices in G which corresponds to the terminal nodes of that element. Hence, this graph representation can be formally stated by the following definition:

Definition 1 : *The undirected graph $G(V, E)$ is said to represent a VLSI circuit $\Omega(N, M)$ if there exist bijections:*

$X : V \Rightarrow N$ and $Y : E \Rightarrow M$, such that $e = (\nu, \omega) \in E$ if and only if,

$$\{X(\nu), X(\omega)\} = \begin{cases} \{Drain(Y(e)), Source(Y(e))\} & \text{if } m \in M_{CMOS} \\ \{Terminal1(Y(e)), Terminal2(Y(e))\} & \text{if } m \notin M_{CMOS} \end{cases}$$

Note that gate interconnections of CMOS transistors in Ω are not included in this graph representation. Hence the graph G does not completely represent the circuit and should be considered as an *intermediate* representation for VLSI circuits. In Section 3.4 we will show how to complete this representation by using directed graphs. So, if the VLSI circuit contains only CMOS transistors, then the undirected graph G representing the circuit Ω has edges that represents the connections of the channels of the CMOS transistors. For the sake of clarity, we will assume that the circuit Ω contains only CMOS transistors. In Section 3.6 we will discuss how to treat the other elements in the circuit. An example circuit that contains CMOS transistors is shown in Figure 3.1 [9].

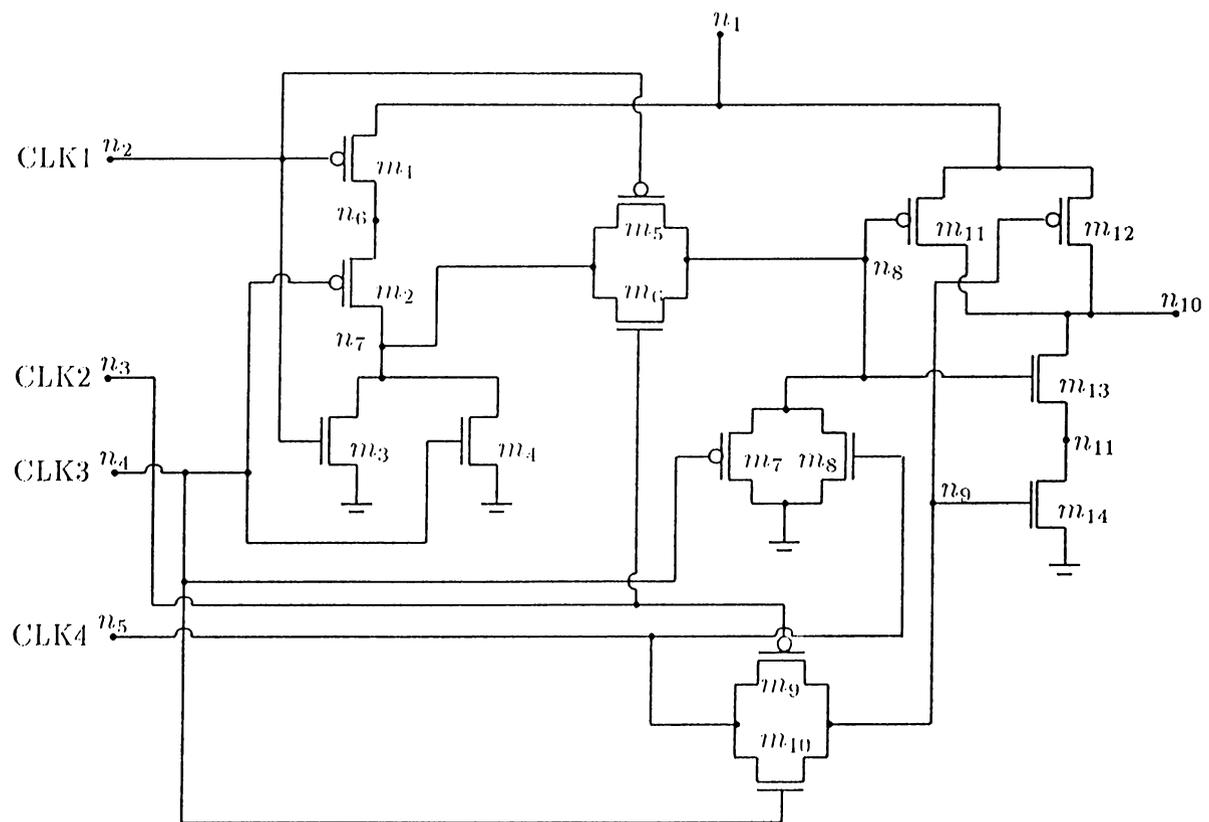


Figure 3.1. A CMOS example circuit.

The undirected graph G can be constructed while reading the circuit description from a SPICE file. The data structure used to represent G is the adjacency multilists. The graph G obtained from the circuit in Figure 3.1 and its adjacency multilists is shown in Figure 3.2. The node structure of the lists consists of :

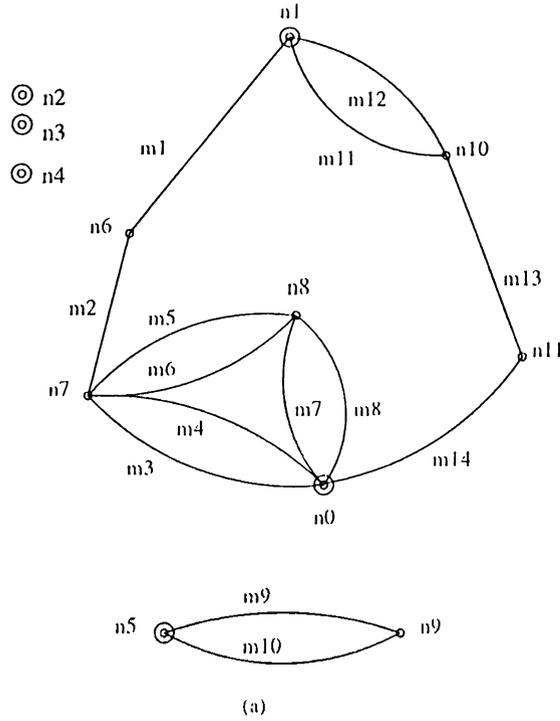
- *Vertex*₁ which represents the drain node N_D of a transistor
- *Vertex*₂ which represents the source node N_S of a transistor
- *Gate* which represents the gate node N_G of a transistor
- *Block* which represents the index i of the partition Ω_i that contains this transistor

Field that contain component information can be included in the node structure. The *block* field is initialized to -1 showing that the element is not yet mapped to any partition. Once again, the size of the G array is N , and the size of the array *edges* is M . In the next Section we describe the splitting algorithm used in splitting the graph G .

3.3 Splitting and Clustering of Graph G

Node Splitting (NS) of graphs has been discussed in general in Section 2.2.6. Here, we will present the concept of *input* node splitting and the algorithm used in implementing the NS. This algorithm is used as a *clustering* scheme of the given VLSI circuit. In later sections, we will discuss how this clustering scheme aids the final partitioning scheme. The *input* vertex splitting and clustering of the graph G can be formally defined as follows:

Definition 2 :Let V_I denote the set of input vertices in G that correspond to the set of input nodes N_I in Ω . The undirected multigraph graph G_I is defined as the multigraph obtained from splitting all the input vertices in V_I



| Block | vertex1 | vertex2 | Gate | path1 | path2 | comp. info. |
|-------|---------|---------|------|-------|-------|-------------|
|-------|---------|---------|------|-------|-------|-------------|

G

| edges | 0 | 1 | 6 | 2 | 9 | 2 | m 1 | 1 |
|-------|---|----|----|---|-----|-----|------|----|
| 1 | 6 | 6 | 7 | 4 | nil | 3 | m 2 | 2 |
| 2 | 6 | 7 | 0 | 2 | 4 | 4 | m 3 | 3 |
| 3 | 6 | 7 | 0 | 4 | 5 | 7 | m 4 | 4 |
| 4 | 6 | 7 | 8 | 3 | 6 | 6 | m 5 | 5 |
| 5 | 6 | 7 | 8 | 2 | nil | 7 | m 6 | 6 |
| 6 | 6 | 8 | 0 | 4 | 8 | 8 | m 7 | 7 |
| 7 | 6 | 8 | 0 | 5 | nil | 12 | m 8 | 8 |
| 8 | 7 | 1 | 10 | 8 | 10 | 10 | m 11 | 9 |
| 9 | 7 | 1 | 10 | 9 | 11 | nil | m 12 | 10 |
| 10 | 7 | 10 | 11 | 8 | nil | 12 | m 13 | 11 |
| 11 | 7 | 11 | 0 | 9 | nil | nil | m 14 | 12 |
| | 8 | 5 | 9 | 3 | 14 | 14 | m 9 | 13 |
| | 8 | 5 | 9 | 4 | nil | nil | m 10 | 14 |

(b)

Figure 3.2. (a)The graph G obtained from the example circuit (b) The adjacency multilists of G .

of the graph G , that is, $G_I = G \circ V_I$. Let C_0, C_1, \dots, C_l , where $l = |V_I|$, denote the blocks (clusters) that correspond to each split vertex in V_I . The connected components of G denoted by $C_{l+1}, C_{l+2}, \dots, C_K$ are defined as the connected components of G_I . Each connected component C_i , for $0 \leq i \leq K$, induces a block $\Omega_i \in \Omega$ which is the subnetwork of elements and/or nodes corresponding to the edges and/or vertices in C_i , respectively.

Recall that, the purpose is to find a partitioning Σ that partitions a given VLSI circuit Ω into several transistor disjoint subnetworks, $\Omega_1, \Omega_2, \dots, \Omega_K$, such that each subnetwork or block Ω_i has a certain special configuration that would aid the simulation process. The simulation of the set of transistors with a common-channel-path between their source and drain nodes have strong interdependencies. This fact necessitates the inclusion of the nodes with common-channel-path in the same partitions. The connected components of the split multigraph G_I identifies such common-channel-path subnetworks in Ω . Hence, the clustering scheme indicated in Definition 2 can be used in finding the common-channel-path subnetworks of the given VLSI circuit. So the configuration of each connected component $C_i, 0 \leq i \leq K$, can now be defined as follows:

- The first $l = |V_I|$, blocks are reserved for the *input* vertices V_I . That is, each *input* vertex $v_i \in V_I$ constitutes an individual block. Clearly, each C_j , for $0 \leq j \leq l$, is a component having exactly one vertex.
- The clusters C_j , for $l < j \leq K$, corresponds to a connected-channel transistors induced by the set of vertices $v_i \in V$ and the edges $e \in E$ that connects those vertices.

The implementation of the clustering scheme indicated in Definition 2 is a two phase process. In the first phase, the split graph $G_I = G \circ V_I$ should be constructed. In the second phase, connected components $C_{l+1}, C_{l+2}, \dots, C_K$ can be found by running the COMP(G) algorithm, defined in Figure 2.7, on graph G_I . Then, first C_0, C_1, \dots, C_l blocks, where $l = |V_I|$ are added to the component

set. The splitting algorithm proposed in [9] is an inefficient algorithm since it increases the size of the split graph G_I . In this algorithm, each split vertex i is duplicated $d_i - 1$ times in the data structure representing the split graph G_I , where d_i denotes the degree of vertex i in the graph G . However, this duplication can be avoided by virtually splitting the input vertices and then finding the connected components directly on the data structure representing the unsplit graph G . This efficient scheme is achieved by modifying the connected component algorithm as proposed in Figure 3.3. This algorithm splits the input vertices of the graph G without expanding the data structures or increasing the number of vertices. The blocks of the VLSI circuit will be found by labeling the edges according to their corresponding components.

The algorithm proposed in Figure 3.3 starts by marking all input vertices $v_i \in V_I$ by different block numbers; $0 \leq \text{block-no} \leq l$, where $l = |V_I|$. Note that, 0 block number is given to the GND block, other input blocks are marked from 1 to l . Then, starting from a vertex v_a adjacent to a split vertex $v_i \in V_I$, initiate a *BFS* search and find all vertices connected to the vertex v_a , mark all edges incident to those vertices by the same block number; $l < \text{block-no} \leq K$, so that a complete connected component is found. The only modification with this *BFS* search is that, when hitting a split vertex $v_j \in V_I$ the search will not add this vertex to the queue. This is because the vertex $v_j \in V_I$ is already marked. After the split algorithm finishes execution, all edges will be mapped to their corresponding connected components C_i ; $0 \leq i \leq K$. Also, another array called *node-block* is constructed which shows the block index i of Ω_i that contains a vertex v . Also, after finding the split graph $G \circ V_I$, we can make one pass through the multiadjacency lists to find the lists of all edges (elements) and vertices (nodes) that belong to each block Ω_i . The split node lists that belongs to each block Ω_i can be found as well. This information will be used when the construction of the fan-in and fan-out lists of the clustered circuit is performed.

Note that, since the gate interconnections are not included in the representation of the intermediate graph G , the graph G might already have more than one connected component before splitting its *input* vertices. For example, the

Split(G)

Block-no = 0
For each split vertex $v_i \in V_I$ **do**
 node-block(v_i) = Block-no
 Increment Block-no
End For

Mark all other nodes with -1 in the node-block array
For each split vertex $v_i \in V_I$ **do**
 For each vertex ω_u adjacent to v_i **do**
 If node-block(ω_u) \neq -1 **then do**
 Call **BFS**(ω_u) { Each vertex and edge is marked with Block-no. }
 Increment Block-no
 End If
 End For
End For

Find all edges (elements) that belongs to each block Ω_i
Find all split nodes that belongs to each block Ω_i

End Split.

Figure 3.3. The proposed algorithm for the implementation of the clustering scheme indicated in Definition 2.

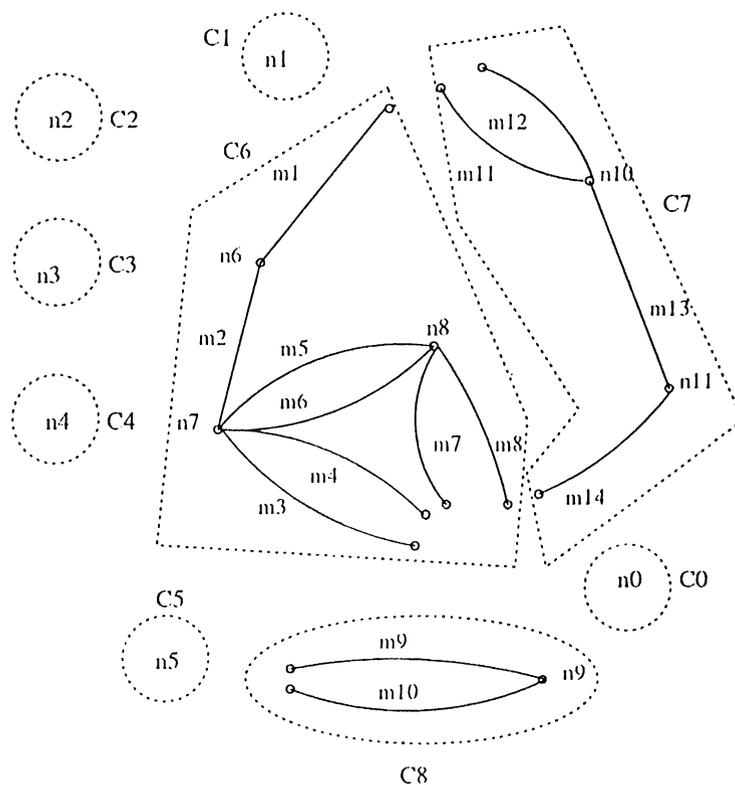


Figure 3.4. The graph G_0V_I obtained from splitting the input nodes of the graph H .

sample graph G given in Figure 3.2 contains two connected components before splitting. However, since the proposed splitting algorithm accesses the graph G starting from its split vertices, each component of G will not be processed unless it contains at least one *input* vertex. The following theorem presents a property in the electrical VLSI circuits, which guarantees that each component G_i of the undirected graph G representing the VLSI circuit Ω has at least one input vertex $v_i \in V_I$ to split.

Theorem 1 : *If the intermediate undirected multigraph $G(V, E)$ that represents the VLSI circuit $\Omega(N, M)$ originally has more than one connected component G_1, G_2, \dots, G_n , then each connected component G_i has at least one vertex $v_i \in V_I$ to be split.*

Proof: From Definition 1, if G_i is a component of G , then G_i induces the sub-network Ω_i in the network Ω which has some common-channel transistors connected to each other. If one of the end terminals of the common-channel

transistors (i.e., the drain or source nodes of the first or the end transistors in the common-channel) is connected to a *normal or storage* node $n_i \notin N_G$ in the network, then due to Definition 1, the subnetwork Ω_i should be connected to another part of the network. Hence, the component G_i will also be connected to another component in G , which contradicts the fact that G_i is a separate connected component. If the end terminals of the common-channel transistors in G_i are not connected to any node then, the connected component G_i is floating, a case which can not happen in VLSI circuits. So one of the end terminals of the common-channel transistors should be connected to an *input* node in the network. Note that the case in which both the end terminals of the common-channel transistors are connected to a *normal or storage* node $n_i \in N_G$ is impossible.

The complexity of the proposed split algorithm is $O(|V| + |E|)$. The graph G_I resulting from splitting the input nodes in the graph G representing the circuit of Figure 3.1 is shown in Figure 3.4. Note that this graph has two components before applying the splitting algorithm. Experimental results show that the number of elements in each cluster varies between 5 to 15 elements. A final partition may be constructed by grouping more than one block of the split blocks in Σ . The number of blocks in Σ is large enough to adjust the final number of elements in each final partition.

3.4 Ordering

A VLSI network Ω can be simulated by processing each of its blocks in a certain order. A block is said to be processed, if given the waveforms at the input nodes to the block, the waveforms at its output nodes are obtained. Thus, in order to process a block, the waveforms at its input nodes must be known. Hence, we must process the blocks in a certain order so that this condition is always satisfied (whenever possible). In this section, we will show when such an ordering exists. However, a suitable graph representation for the partitioned VLSI circuits should be devised in order to determine the relation

between the partitioned blocks such as feedback paths and ordering.

Gate interconnections missing in G and G_I graphs are added as directed edges into the graph D . Information about the interconnections between input nodes and drain/source nodes are lost during the splitting process. These interconnections are also added as directed edges into the graph D . The head of each directed edge in D points to the vertex representing the block that is taking input, and the tail of the edge is incident to the vertex that represents the block giving the output. Hence, this graph representation can be formally stated by the following definition:

Definition 3 : Let χ represent the set of all external interconnections between the blocks of the partitioned circuit $\Omega(N, M, \Sigma)$. The directed graph $D(V, E)$ represents the partitioned VLSI circuit $\Omega(N, M, \Sigma)$ if there exist bijections: $X : \Sigma \Rightarrow V$ and $Y : \chi \Rightarrow E$, such that the directed edge $e = (\nu, \omega) \in E$ if and only if, $\Omega_i = X^{-1}(\omega)$ is taking input from $\Omega_j = X^{-1}(\nu)$, for $0 \leq i, j \leq K$, and $i \neq j$.

The digraph D defined in the definition 3 completely represents the simulation interdependencies between the clusters of the VLSI circuit $\Omega(N, M, \Sigma)$. Note that, the directed graph D has the following properties:

- Each vertex, $v_j \in V$, where $0 \leq j \leq l$, and $l = |N_I|$, corresponds to an input node $n_i \in N_I$ in the circuit $\Omega(N, M)$.
- Each vertex, $v_j \in V$, where $l < j \leq K$, and $l = |N_I|$, corresponds to a sub-circuit of common-channel CMOS transistors Ω_j in the circuit $\Omega(N, M)$.
- Each directed edge $e = (\nu, \omega) \in E$ corresponds to an external connection which is either a gate inter-connection or interconnections that are connecting an input node to a drain or source node of a CMOS transistors.

Figure 3.5 illustrates the digraph constructed for the clustered example circuit. From now on, the terms, *vertex in D* , a *connected component in G_I* ,

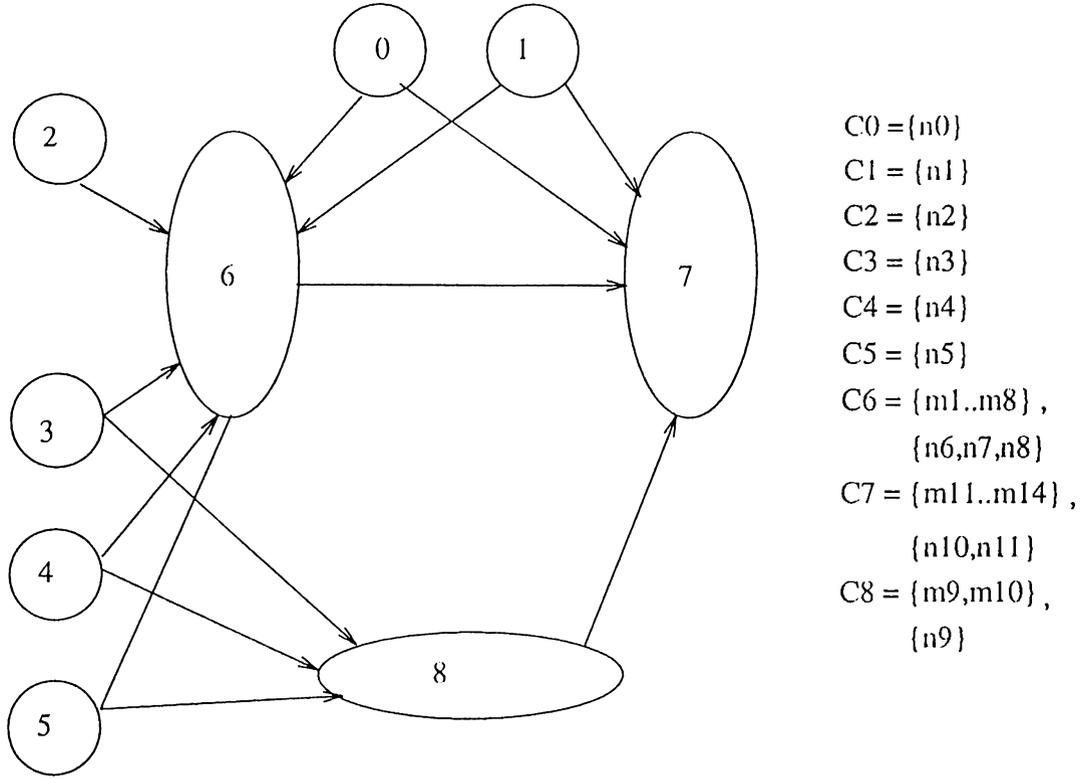


Figure 3.5. The digraph D constructed for the clustered example circuit.

and a *block in Ω* , will be used interchangeably. The input output relations between the blocks and the nodes of the VLSI circuit can be specified as follows:

- The set of *input nodes* for each block Ω_j , where $l < j \leq K$, for $l = |N_l|$, denoted by $INP(\Omega_j)$, is defined as the set of gate nodes of CMOS transistors in Ω_j taking input from a block Ω_i , for $i \neq j$, in addition to the drain and source nodes of CMOS transistors in Ω_j taking inputs from nodes of *input strength*, that is,

$$INP(\Omega_j) = \{N_G^j \cup ((N_D^j \cup N_S^j) \cap N_l)\} \quad (3.3)$$

where N_G^j is the set of gate nodes in Ω_j taking inputs from a block Ω_i , for $i \neq j$. N_D^j , and N_S^j , denotes the drain and source nodes of the CMOS transistors in Ω_j , respectively.

- Similarly, the set of *output nodes* for each block Ω_j , where $l < j \leq K$, for $l = |N_l|$, denoted by $OUT(\Omega_j)$, is defined as the set of *normal or storage* nodes that are drain or source nodes of CMOS transistors in Ω_j driving

the gate nodes of CMOS transistors in Ω_i , where $i \neq j$, in addition to the user defined output nodes, that is,

$$OUT(\Omega_j) = \{(((N_D^j \cup N_S^j) \cap N_N) \cap (N_G^i)) \cup N_o^j\}; \text{ for any } i \neq j \quad (3.4)$$

Where N_D^j , and N_S^j , denotes the drain, and source nodes of the CMOS transistors in Ω_j , respectively, N_o^j , denote the user defined output nodes in Ω_j , and N_G^i , denote the gate nodes in Ω_i , where $i \neq j$.

- Note that, $OUT(\Omega_j)$, for $0 \leq j \leq l$, for $l = |N_I|$, contains the only input node constituting that block.
- The *fanout-list* of each node $n_k \in \Omega_j$ is defined as the set of all blocks Ω_i , $i \neq j$, taking input from node n_k , that is,

$$FOUT(n_k) = \{\Omega_i : n_k \in INP(\Omega_i) \text{ for } i \neq j\} \quad (3.5)$$

- Similarly, the *fanout-list* of a block Ω_j , where $0 \leq j \leq K$, is defined as the set of all blocks Ω_i , and $i \neq j$, taking input from a node $n_k \in \Omega_j$, that is,

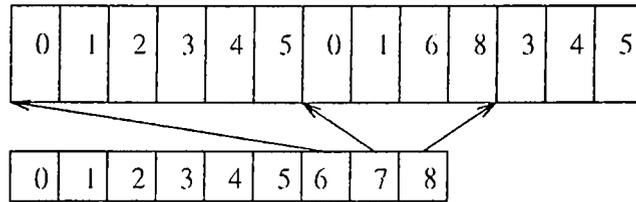
$$FOUT(\Omega_j) = \bigcup_{n_k \in \Omega_j} FOUT(n_k) \quad (3.6)$$

- The *fanin-list* of a block Ω_j , where $l < j \leq K$, for $l = |N_I|$, is defined as the set of all blocks Ω_i , where $0 \leq i \leq K$ and $i \neq j$, giving input to Ω_j , that is,

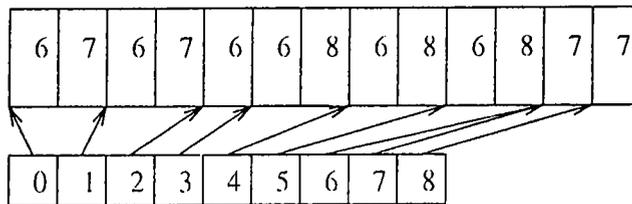
$$FIN(\Omega_j) = \{\Omega_i : \Omega_j \in FOUT(\Omega_i) \text{ for } i \neq j\} \quad (3.7)$$

Note that, the *fanin-list* and *fanout-lists* of the blocks are represented by the *In-degree* and *Out-degree* lists discussed in Chapter 2. The *in-degree and out-degree* list of the digraph given in Figure 3.5 are illustrated in Figure 3.6.

The simulation order of the blocks of a partitioned network is very crucial. Recall from Chapter 2.2.4, that, a linear ordering of a digraph can be easily achieved topological sorting of the digraph. Hence, a linear ordering of the nodes of the digraph representing the partitioned network yield a *good ordering* for the simulation if this digraph is acyclic. In this linear ordering, whenever a



In-degree lists



Out-degree lists

Figure 3.6. The In-degree and Out-degree lists of the example circuit .

block Ω_i is scheduled for processing, all the blocks $\Omega_j \in FIN(\Omega_i)$ are already processed, thus providing input signals to the block Ω_i .

A partitioned VLSI circuit Ω is said to have *feedback* among its blocks if the digraph D representing the circuit Ω has directed cycles. For example, consider a block Ω_l in a network having its output node n_o connected back to a block Ω_j having Ω_l in the $FOUT(\Omega_j)$. Thus, Ω is *feedback-free* if D is acyclic. Recall that, linear ordering for cyclic digraphs does not exist. So, the partitioned network $\Omega(N, M, \Sigma)$ has a good ordering on its partitioned blocks if and only if it is feedback-free. If, however, the network have feedback, the digraph D contains directed cycles, hence, no good ordering exists on its vertices(blocks). In this case, therefore, one must detect the blocks in the network that are within feedback loops, treat this as special blocks and place the rest of the blocks in a good ordering. The directed graph \tilde{D} consisting of vertices which corresponds to the strongly connected components of the digraph D is acyclic and is called the *condensation* digraph of D . It can be proved that the condensation graph \tilde{D} of any digraph D must be acyclic [49].

In the following sections, the algorithms used in constructing the digraph D , finding the SCCs of D , and finding the leveling between the blocks of the condensation dag \tilde{D} will be presented.

3.4.1 Constructing Digraph of the Partitioned Circuit

As mentioned in Section 2.2.4, the running time of the SCC algorithm is linearly proportional to the number of edges in the digraph. Hence, if a particular block Ω_j is taking more than one signal as input from block Ω_i , then this interdependency should be represented by only one directed edge (Ω_i, Ω_j) . Figure 3.7 illustrates this concept. The algorithm given in Figure 3.8 detects such situations very easily by maintaining proper marking arrays. The first for-loop in the pseudo code algorithm detects the gate interconnections between blocks. The second for-loop detects the interconnections between the input blocks and the drain and source nodes in other blocks.

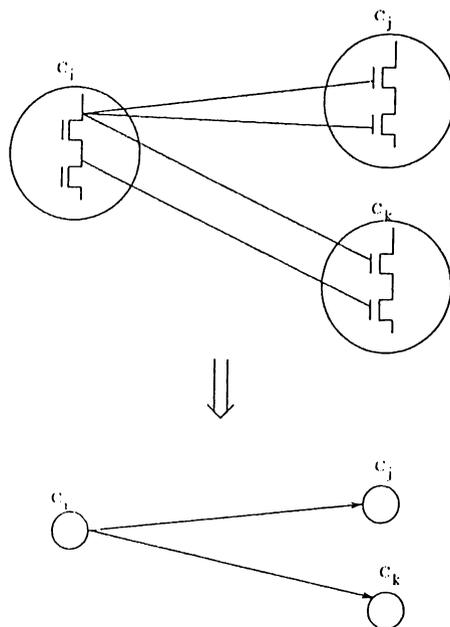


Figure 3.7. Reducing the number of edges in the digraph representation.

The derived digraph D of the circuit of Figure 3.1 is shown in Figure 3.9. Also the D^T is shown. The In-degree and Out-degree lists of the circuit of Figure 3.1 is shown in Figure 3.6.

3.4.2 Finding the SCCs of the Digraph D

As mentioned in the previous section, a good ordering exists only for acyclic graphs. Hence, strongly connected components (SCCs) of the digraph D representing the the partitioned circuit, should be identified. Then, blocks constituting SCCs can be coalesced to form super vertices of a the acyclic condensation digraph \hat{D} .

The algorithm presented in Figure 2.8 can be used to find the SCC components of the derived digraph D . The *DFS* procedure uses the Out-degree lists to compute the finishing times. Since the finishing time is a value between $0..|2V|$, we can keep the sorted vertices directly in an array of length $|2V|$. The vertices is sorted according to their decreasing finishing times. The *BFS*

In-degree(D)

Mark all blocks by -1 in the block-mark array

For each edge $e_t \in \Omega_i$ **do**

Find the block Ω_j that contains the gate node N_G^t of e_t

If $i \neq j$ **Then**

If block-mark (Ω_j) $\neq i$ **Then**

Add block Ω_j to In-degree list of Ω_i

block-mark(Ω_j) = i

End If

End If

End For

For each split node $n_s \in \Omega_i$ **do**

If block-mark (Ω_s) $\neq i$ **Then**

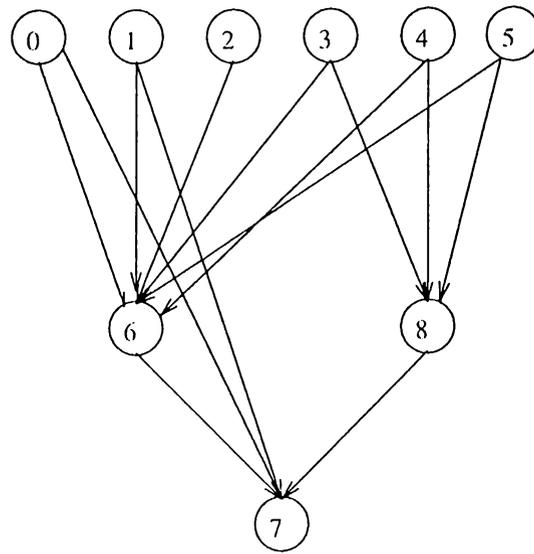
Add block Ω_s to In-degree list of Ω_i

End If

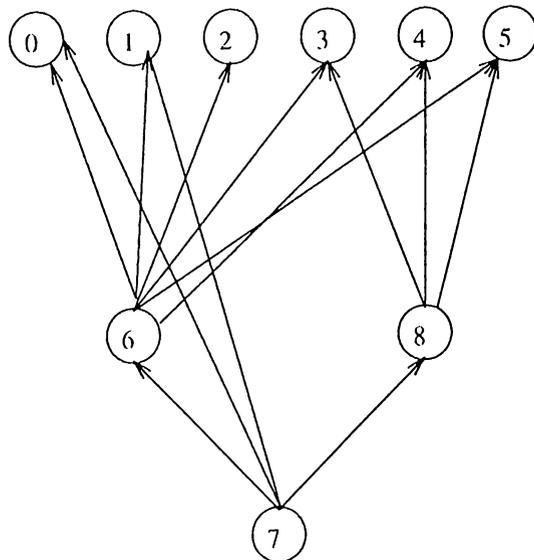
End For

End In-degree.

Figure 3.8. The steps used in constructing the In-degree lists of D .



(a)



(b)

Figure 3.9. (a) The derived digraph D of the example circuit. (b) Its transpose graph D^T .

procedure uses the In-degree lists to label the edges with their corresponding SCC number. Once again, the first $l = |N_I|$ SCCs are reserved for the input node blocks.

After the SCC procedure finishes execution, each edge in G will be marked by an index indicating the SCC to which this element is mapped. The *block* field in the adjacency multilists can be used to carry this labels. Also an array similar to the node-block array is constructed to contain the SCC number for each node. The time complexity of finding the SCC of the graph G is $O(|V| + |E|)$.

Now we are ready to construct the acyclic condensation digraph \tilde{D} which represents the SCCs of the derived digraph G . The In-degree list of \tilde{D} can be constructed by using the same procedure used in constructing the In-degree lists of the digraph D .

3.4.3 Leveling the SCCs

As discussed earlier, the topological sorting on a directed acyclic graph (dag \tilde{D} in our case) yields a good ordering for simulation. Unfortunately, this ordering does not yield any information about the set of SCCs which can be simulated simultaneously and independently. This information can only be obtained by the leveling of the condensation digraph \tilde{D} . In this scheme, each level consists of the SCCs that can be simulated independently and simultaneously. That is, there is no need for intra-level ordering. However, there exists an ordering among the simulation of levels which is referred here as the inter-level ordering. That is, put all blocks $\Omega_i, \dots, \Omega_j$, in a certain level, say l , whenever the blocks in level l is scheduled for processing, all the levels containing any block $\Omega_k \in FIN(\Omega_i), \dots, FIN(\Omega_j)$ have been previously processed, thus, providing input signals to the blocks in level l . The algorithm shown in Figure 3.10 can be used to find the leveling among the SCCs of the condensation digraph \tilde{D} .

The algorithm starts by calling the topological sorting algorithm presented

```
Level( $\tilde{D}$ )  
    Call Tsort( $\tilde{D}$ )  
    Initialize  $\text{level}(\omega) = 0$  for all  $\omega \in \tilde{D}$   
    For each vertex (SCC)  $\omega$  in the topological order do  
        Find all vertices  $v_1, \dots, v_i$ , in the In-degree list of  $\omega$   
         $\text{level}(\omega) = \text{MAX}(\text{level}(v_1), \dots, \text{level}(v_i)) + 1$   
    End For  
  
End Level.
```

Figure 3.10. The Leveling algorithm .

in Section 2.2.5 to find a topological ordering of the dag \tilde{D} . The returned order of the vertices can now be used to find the leveling among the SCCs. This is done by considering each vertex ω in the topological ordering, and then finding all vertices in the in-degree lists adjacent to the vertex ω . The level of the vertex ω is found by adding 1 to the maximum level of those adjacent vertices. Note that, the input nodes blocks are placed in level zero, since they are not taking any input from any other block.

After finding the levels of all the SCCs in the circuit, the next step may be finding the final partitions that can be scheduled for processing using the VLSI simulator. However, the sizes of those final partitions are not random, and should be predetermined such that a good simulator performance is achieved. That is, the size of the final partitions that will be provided to the simulation tool should not exceed a predetermined limit. If the maximum size limit on the final partitions is known, then the simple pseudo-code algorithm presented in Figure 3.11 may be used in finding such final partitions. In this pseudo-code, $|\cdot|$ denotes the total number of elements in that set. The algorithm starts by grouping the SCCs in the level order until the maximum partition limit is reached. Figure 3.12 illustrates this grouping criterion. Note that, the final

Grouping

$$i = 0 \quad P_0 = \{ \}$$

```

For each SCC in level order do
  If ( $|P_i| + |SCC|$ )  $\leq$  Max-limit Then
     $P_i = P_i \cup SCC$ 
  ELSE
     $i = i + 1$ 
     $P_i = SCC$ 
  End If
End For

```

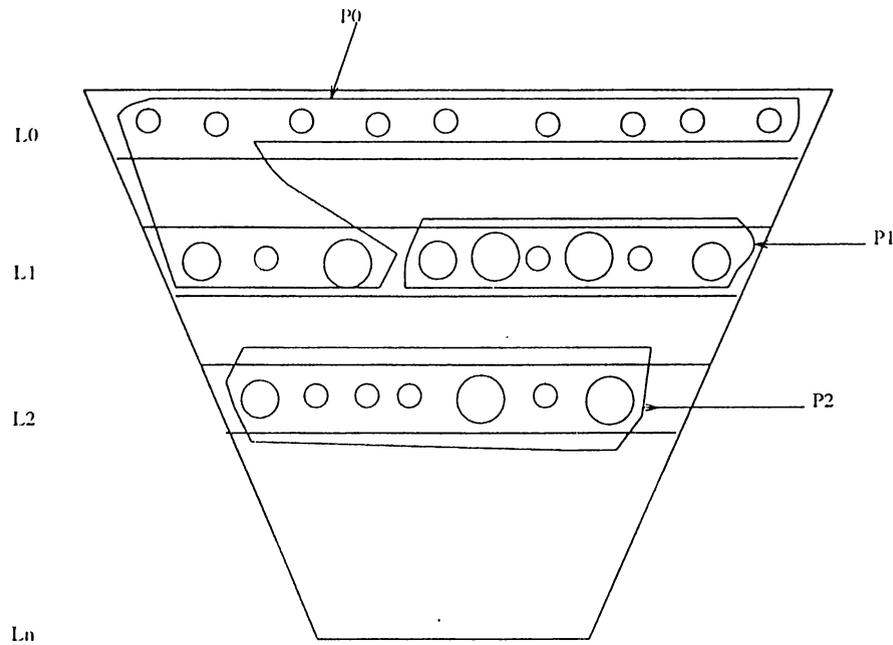
End Grouping.

Figure 3.11. The Grouping heuristic .

partitions may contain SCCs from more than one level, or the SCCs of an individual level may be assigned to more than one partition. The simulation of the SCCs in a particular partition P_i can only be initiated after the completion of the simulation of all partitions $P_0 \leq P_j < P_i$. If the partition P_i contains SCCs from only one level, then the simulation of the SCCs within this partition can be performed in any order. Otherwise, if the partition P_i contains SCCs from multi-consecutive levels, then the simulation of the SCCs within this level can be performed according to their level order. That is, the level information is maintained for the multi-consecutive level partitions.

3.4.4 VLSI Circuits with Large Feedback Paths

As discussed in the previous section, if small feedback paths exist among the partitioned blocks in D , then those blocks can be coalesced to form a SCC in \tilde{D} . However, if the size of a particular SCC S_i in the condensation digraph



* Circles represent SCCs in each level.

Figure 3.12. Illustrating the grouping of levels to find final partitions.

\tilde{D} exceeds the required maximum size limit the grouping algorithm given in Figure 3.11 will fail. Such SCCs should be further divided into a number of partitions until the size of each sub-partition is less than the predetermined limit. Then these sub-partitions can be added as successive vertices into the leveled digraph \hat{D} in place of that large SCC.

Assume that it suffices to divide a SCC S_i into two partitions S_{i1} and S_{i2} such that the size of each sub-partition does not exceed the predetermined limit. Since S_i is a SCC, there should be external feedback path between the partitions S_{i1} and S_{i2} . Hence there exists bidirectional simulation interdependencies between the partitions S_{i1} and S_{i2} . That is, directly simulating S_{i1} is not possible since some input signals from S_{i2} are not available yet, similarly, simulating S_{i2} before S_{i1} is not possible for the same reason. Hence, no good ordering can be found between the partitions S_{i1} and S_{i2} . Special techniques should be used in the simulation of the partitions of the large SCCs. Such techniques causes considerable overhead in the combination of the simulation

results of the partitions. The combination of the final partition simulation results requires the inversion of a dense matrix, called the interconnection matrix, whose size depends on the number of different signals between the partitions S_{i1} and S_{i2} . Hence, it is crucial to minimize the number of signals between the partitions S_{i1} and S_{i2} in order to minimize the computational overhead introduced in the simulation of such large SCCs.

As mentioned earlier, there exist at least one feedback path between any partitions S_{i1} and S_{i2} of the large SCC S_i . Hence, the digraph representing S_i can be considered as a graph in minimizing the interconnections between the partitions S_{i1} and S_{i2} . However, this minimization problem is in fact the well-known graph partitioning problem which is known as an NP-hard problem. Such problems are usually tackled by heuristics. The heuristic proposed by Fiducia and Matheyses (FM) [27] is an efficient linear-time heuristic widely used for graph partitioning.

As mentioned earlier, the computational overhead in the partitioned simulation of large SCCs is proportional to the number of different signals between the sub-partitions. Hence, for a correct modeling of the target optimization problem, large SCCs should be represented as hypergraphs instead of graph. In this hypergraph representation, each net will denote an individual signal between clusters contained in the large SCC.

Constructing the Net-lists of the Blocks in S_i

Like finding the In-degree lists of the derived digraph D , constructing the Net-lists of the hypergraph representing the blocks in the SCC S_i is critical and should give the true information about the connections between the blocks of the partitioned circuit. Figure 3.13 illustrates the net corresponding to a signal from a node in block Ω_i and driving the gate nodes in blocks Ω_l , Ω_k and Ω_j . The first outer for-loop given in Figure 3.15 detects and constructs such nets very easily. However, different cases as illustrated in Figure 3.14 may also exist in VLSI circuits. The first for-loop in Figure 3.15 detects and

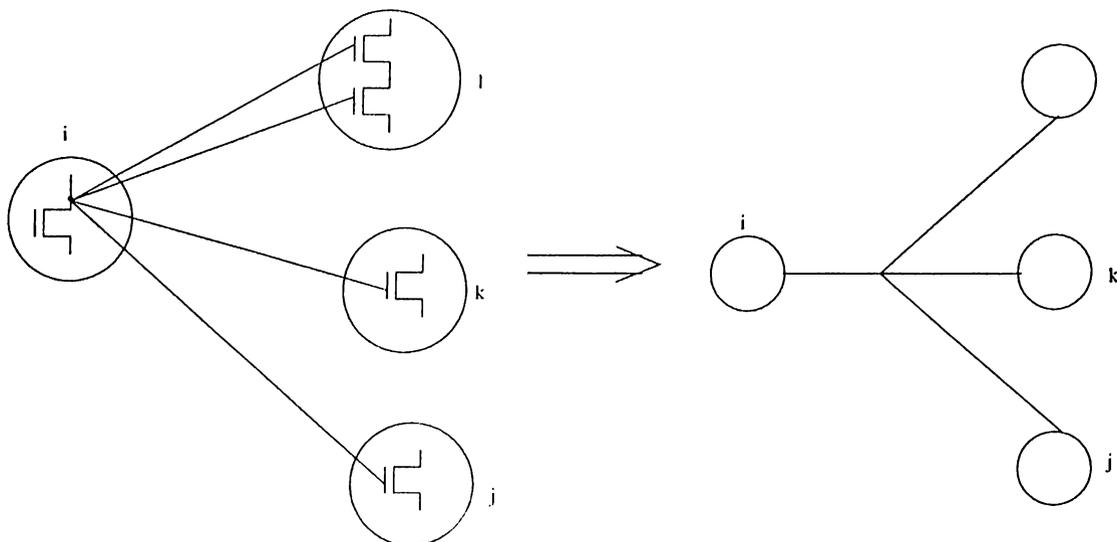
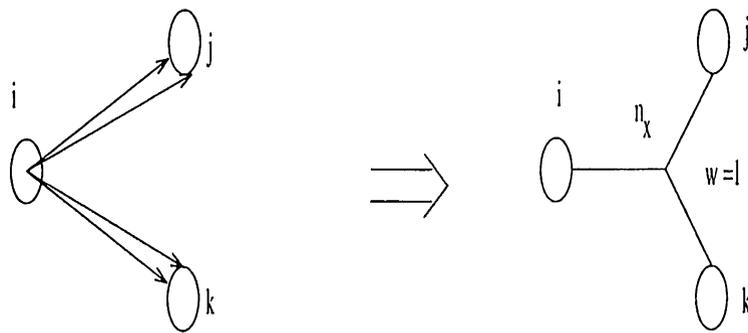


Figure 3.13. Converting connections between blocks into signal lines (nets).

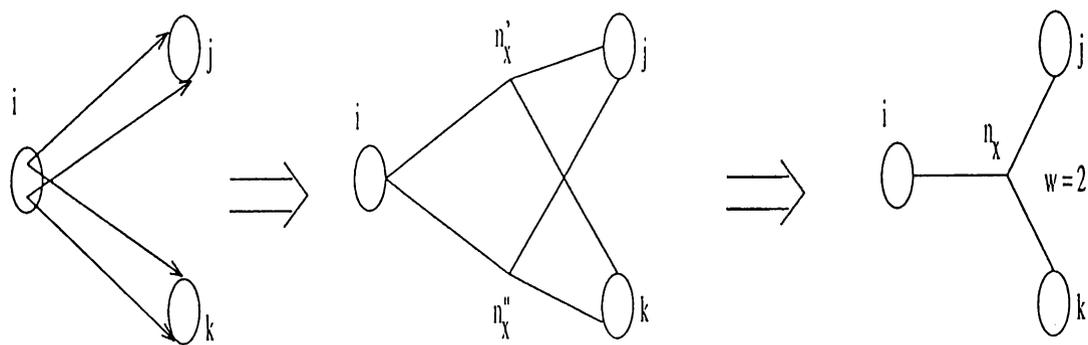
constructs the nets for case one very easily, furthermore, the same loop creates two nets n'_x and n''_x for second case as illustrated in Figure 3.14. In case 1, the net n_x with weight equals to one is constructed directly. While in case 2, the two nets, n'_x and n''_x , can be merged to a single net n_x with weight two since it contains two signals. This merge operation decreases the run time of the FM heuristic since it decreases the number of nets. At the same time, this merge operation increase the run-time of the FM heuristic due the existence of weighted nets. The second nested for-loop in Figure 3.15 achieves such merge operations for case 2 nets. Unfortunately, as is seen in Figure 3.15, this merge operations involves list searches and disturbs the linear complexity of the net construction phase. Furthermore, experimental results show that cases like case 2 is not likely to occur in VLSI design. Hence, such merge operations is not recommended.

Partitioning Using the FM Heuristics

After the construction of the Net-lists of the partitioned blocks of the SCC S_i , the next step is to partition the hypergraph representing S_i . Each cell or vertex in the hypergraph H represents a collection of common-channel connected transistors, the final partition will contain groups of such collections.



Case 1



Case 2

Figure 3.14. Reducing the number of nets in the hypergraph of S_i .

Net-lists($G \circ V_I$)

Mark all nodes by -1 in the node-mark array
For each edge $e_l \in \Omega_i$ **do**
 Find the block Ω_j that contains the gate node N_G^l of e_l
 If $i \neq j$ **Then**
 If node-mark(N_G^l) $\neq i$ **Then**
 Increment the fan-out counter of node N_G^l of e_l by 1
 Add block Ω_i to fan-out-list of node N_G^l of e_l
 node-mark(N_G^l) = i
 End If
 End If
End For

For each node $n_l \in \Omega_i$ **do**
 For each node $n_p \in \Omega_i$ and $l \neq p$ **do**
 If the fan-out counter of n_l = fan-out counter of n_p **Then**
 If fan-out-list(n_l) = fan-out-list(n_p) **Then**
 Delete the fan-out-list(n_p)
 Increment the net weight of node n_l
 End If
 End If
 End For
End For

End Net-lists.

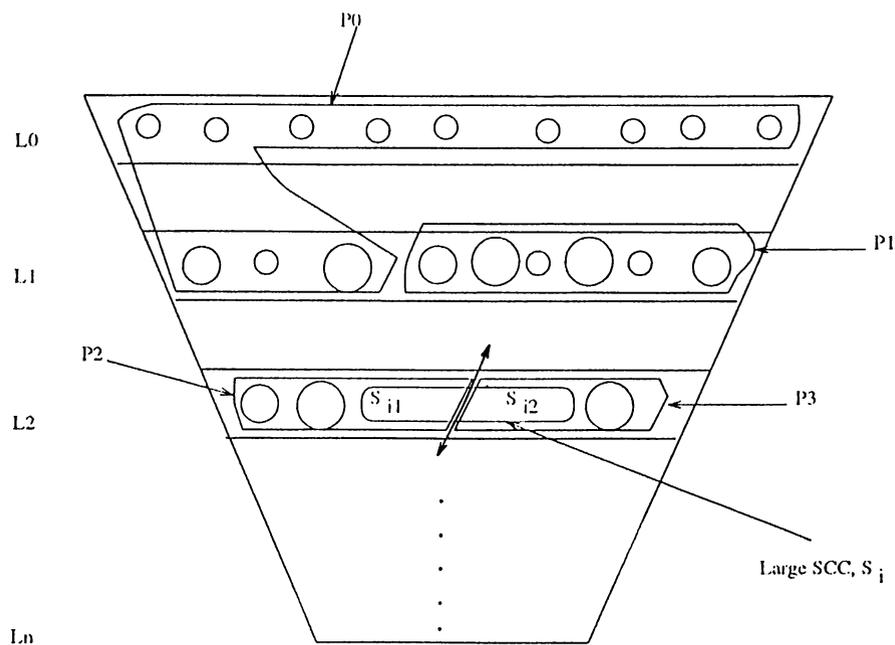
Figure 3.15. The steps used in constructing the Net-lists of the hypergraph .

To achieve this partitioning task, the Kernighan-Lin heuristic is implemented efficiently as described by the Fiducia and Matheyses (FM) [27].

In order to apply the FM to multi-way partitioning two schemes can be used. The first one, partitioning by recursive bisection, recursively partitions the initial graph into two partitions until final partitions are obtained. Other scheme, partitioning by pairwise min-cut, starts with an initial partitioning and then minimizes the cut-size between each pair of partitions until no improvement can be done. The pairwise min-cut scheme yields slightly better solutions compared to the recursive bisection scheme. However, recursive bisection is much faster. Furthermore, pairwise min-cut scheme requires that the final number of partitions is given priori to the execution of the heuristic. Hence, pairwise min-cut scheme is not very suitable in this implementation since a good estimation of the final number of partitions can not be predicted easily. The recursive bisection scheme can be easily applied to our implementation by continuing the bisection operations until the sizes of all partitions decrease below the given maximum limit. Hence, the recursive min-cut scheme is implemented in this work. The exact balance of the partitions during the recursive bisection is not very critical. The partition sizes can be relaxed to have different sizes as long as the cut-size between the partitions is minimized. So the final partitions can have different sizes and the user can define the required load imbalance ratio of the sizes of the partitions.

It should be clear that the FM partitioning is done only when the size of a particular SCC exceeds the maximum limit of the final partitions. The sub-partitions of such large SCCs found by the FM heuristic replace that SCC in the level graph. Other SCCs are still maintained in their levels. Figure 3.16 illustrates the level grouping of circuits with a large SCC S_i . These operations can be summarized by the pseudo-code given in Figure 3.17.

The overall complexity of the proposed partitioning algorithm can be summarized as follows: Constructing the adjacency multilists of G from Ω requires $O(M)$ time, where M is the number of elements in the circuit Ω . Splitting the



* Circles represent SCCs in each level.

Figure 3.16. Illustrating the grouping of levels with large SCC to find final partitions.

graph G requires $O(M + N)$ time, where N is the number of nodes in Ω . Constructing the digraph $D(V, E)$ requires $O(|V| + M)$ operations, where $|V|$ and M denote the number of blocks in the split graph and the number of elements in Ω respectively. Note that, $|V|$ is, in general, considerably smaller than N . Constructing the condensation digraph \hat{D} by finding the SCCs from the digraph D requires $O(|V| + |E'|)$. Note that $|E'| < M$ since multi gate interconnections between each block pair are merged into a single directed edge in the previous step (See Figure 3.6). Experimental results show that $|E'| \approx \frac{1}{2}M$. Leveling of the condensation graph $\hat{D}(\hat{V}, \hat{E})$ requires $O(|\hat{V}| + |\hat{E}|)$ operations. Also note that $|\hat{V}|$ and $|\hat{E}|$ are considerably smaller than $|V|$ and $|E'|$ respectively. Recall that FM heuristics applied to large SCCs is also a linear time algorithm. The grouping algorithm applied to the leveled digraph is a simple algorithm which requires $O(|SCCs|)$. Hence the proposed algorithm is a linear time algorithm.

Partition($\Omega(N, M)$)

Construct the adjacency multilists of G from Ω

Find the C_0, C_1, \dots, C_k blocks of G by using the *Split* algorithm

Construct the In-degree and Out-degree lists of the digraph D

Find the SCCs in the digraph D

Construct the In-degree lists of the SCCs to construct the condensation digraph \tilde{D}

Level the condensation digraph \tilde{D}

For each SCC S_i with $|S_i| > \text{Max-limit}$ **do**

 Construct the Net-lists of S_i

 Run recursive **FM** bisection heuristic on the hypergraph of S_i until the sizes of the sub-partitions S_{i1}, \dots, S_{im} are smaller than Max-limit

 Replace S_i with S_{i1}, \dots, S_{im} in the leveled graph

End For

Run **Grouping** heuristic given in Fig. 3.11 to find the final partitions

End Partition.

Figure 3.17. The Summary of the Partitioning algorithm with both leveling and FM heuristics .

3.5 Testing and Experimental Results

The proposed partitioning algorithm has been implemented and tested on several VLSI circuits with different sizes. The tested circuits are real projects designed in the EE Department at Bilkent University. Tables 3.1, 3.2 shows the results of the tested circuits. In Table 3.1, the problem size is identified by the number of nodes, number of elements and the number of split nodes in circuit respectively. Total number of edges, number of blocks, and the number of elements in the largest block is specified for both D and \tilde{D} graphs. Also, the number of levels and the number of elements in the largest level are specified for each circuit in the table. As can be seen from the table, the sizes of the In-degree lists (number of edges in D) is less than the number of edges (transistors) by approximately a factor of 2. The total number of split blocks is large and each block contains 5 to 110 elements in the average. In general, each SCC block contains 2 to 3 blocks and the average sizes of the SCC blocks is 10 to 12 elements. It is clear that the sizes of the SCCs is suitable to construct the condensation graph and then do the leveling among the SCCs. As mentioned before, the number of levels and the sizes of the levels is suitable to construct groups of consecutive levels as partitions. The grouping of the levels is implemented using the grouping algorithm discussed in the previous section. Partitioning times are included in the table. Note that, the typical simulation time for each circuit is also included in the circuit. All times are measured in CPU seconds.

The sizes of the SCCs encountered in the sample networks are detected to be well below the maximum size limit. Hence, The FM heuristics is tested on the whole VLSI test circuits considering the whole circuit as a large SCC. So after splitting the circuit, the net-lists of the hypergraph representing the partitioned circuit is constructed. Then the FM heuristics is run to find the required number of partitions. The 2-way, 4-way, and 8-way partitioning of the given circuits are as shown in Table 3.2. In general, the cut-size is small for the 2-way partitioning, and becomes larger when the number of required partitions increases. The number of elements for the largest partition is shown in the

table. Partitioning times are also shown together with the typical simulation time for each circuit. All times are measured in CPU seconds.

It is difficult to confirm the acceptability of the shown results since the partitioned circuits is not tested yet in the simulation phase. However, as is seen in Tables 3.1, 3.2 The time taken in partitioning the VLSI circuits is negligible compared to the time taken in the simulation phase and will never cause any overhead in the simulation of the partitioned circuits.

3.6 Mapping Elements Different from CMOS Transistors to the Partitions

From the previous sections, it is clear that only CMOS transistors of the VLSI circuit are taken into consideration while constructing the sub-blocks of the circuit. Real VLSI circuit designs contains elements other than the CMOS transistors, so elements like capacitors, resistors, etc. should be considered in the partitioning of the VLSI circuits. In our partitioning algorithm, the following criteria can be followed while processing the linear elements of the VLSI circuit:

- All linear elements connected between a *normal* node and an *input* (split) node are mapped to the block containing the normal node.
- If a linear element is connecting two sub-blocks and if one of its nodes is a gate node N_G of a CMOS transistor, then the element is mapped to the block containing the N_G node, otherwise, the two sub-blocks are combined to a new sub-block as a SCC.
- If more than one linear element are connected together and are not in the first case, then all connected nonlinear elements are considered as one sub-block.

Other cases can be detected and included in the partitioning algorithm as special cases. The first two cases are very easy to implement, and only

Table 3.1. Results of the partitioning algorithms with leveling tested on 9 real problem instances.

| PROBLEM SIZE | | | SPLIT CIRCUIT | | | | | | | | TIME (Sec) | |
|--------------|-------|---------|---------------|------------|-------------|-----------|-----------|-----------|-----------|-----------|------------|------|
| | | | D | | | \bar{D} | | | Leveling | | | |
| N | M | $ V_I $ | # of Edge | # of Block | M Max Block | # of Edge | # of SCCs | M Max SCC | # of Levs | M Max lev | Sim | Part |
| 64 | 102 | 4 | 50 | 24 | 6 | 30 | 14 | 10 | 11 | 10 | 314.8 | 0.2 |
| 68 | 114 | 12 | 36 | 25 | 14 | 36 | 25 | 14 | 7 | 26 | 17.8 | 0.2 |
| 76 | 122 | 11 | 66 | 32 | 8 | 46 | 22 | 14 | 11 | 14 | 137.8 | 0.2 |
| 97 | 176 | 10 | 87 | 62 | 6 | 87 | 62 | 6 | 7 | 44 | 42.5 | 0.2 |
| 266 | 414 | 20 | 136 | 71 | 14 | 136 | 71 | 14 | 12 | 84 | 612 | 0.2 |
| 405 | 770 | 37 | 432 | 229 | 9 | 432 | 229 | 9 | 35 | 200 | 3584 | 0.3 |
| 1541 | 2560 | 6 | 1279 | 517 | 8 | 767 | 261 | 10 | 182 | 20 | 205687 | 0.9 |
| 2445 | 4616 | 138 | 2326 | 791 | 14 | 2326 | 791 | 14 | 26 | 576 | 36832 | 2.5 |
| 6264 | 11036 | 17 | 5366 | 2050 | 10 | 3702 | 1282 | 14 | 523 | 42 | - | 11.2 |

Table 3.2. Results of the partitioning algorithms using FM heuristics tested on 9 real problem instances.

| SPLIT CIRCUIT | | | FM PARTITIONING | | | | | | | | | |
|---------------|------------|-----------|-----------------|-------|----------------|---------|-------|----------------|---------|-------|----------------|---------|
| D | | | $T(Sec)$ | 2-way | | | 4-way | | | 8-way | | |
| M | # of Block | # of Nets | Sim | Cut | Max Part. Size | T Sec | Cut | Max Part. Size | T sec | Cut | Max Part. Size | T Sec |
| 102 | 24 | 25 | 314.8 | 9 | 54 | 0.1 | 18 | 28 | 0.2 | 27 | 16 | 0.5 |
| 114 | 25 | 20 | 17.8 | 5 | 60 | 0.1 | 7 | 34 | 0.2 | 13 | 16 | 0.5 |
| 122 | 32 | 37 | 137.8 | 16 | 64 | 0.2 | 23 | 34 | 0.25 | 32 | 18 | 0.5 |
| 176 | 62 | 51 | 42.5 | 4 | 90 | 0.3 | 10 | 48 | 0.3 | 23 | 26 | 0.6 |
| 414 | 71 | 67 | 612 | 11 | 212 | 0.3 | 22 | 110 | 0.35 | 54 | 56 | 0.6 |
| 770 | 229 | 213 | 3584 | 3 | 286 | 0.5 | 8 | 194 | 0.8 | 13 | 98 | 1.2 |
| 2560 | 517 | 518 | 205687 | 1 | 1280 | 1.7 | 4 | 660 | 2.5 | 8 | 340 | 3.8 |
| 4616 | 791 | 783 | 36832 | 24 | 2398 | 3.5 | 73 | 1200 | 5.6 | 108 | 598 | 6.5 |
| 11036 | 2050 | 1976 | - | 42 | 5550 | 2.3 | 131 | 2784 | 31.5 | 239 | 1430 | 31.1 |

one pass through the linear elements is enough to map the elements to their corresponding sub-blocks. The third case is under investigation since more examples and cases should be examined.

4. VLSI PARTITIONING FOR PLACEMENT

The main aim of this thesis work was to design and implement an efficient partitioning algorithm to speed up the computation time of the PL-AWE simulation tool. While implementing this partitioning algorithm, we have noticed that the same partitioning scheme can be used as a preprocessing step to form clusters of transistors in partitioning for placement in the layout problem. In this chapter we show how to use the Node Splitting concept to form the clusters and then apply the well known partitioning heuristics, such as Kernighan-Lin and Simulated Annealing, to partition the circuit. In section 4.1 we talk about the placement problem in general and then review previous partitioning algorithms that are used in the problem. In section 4.2 we define the partitioning problem and then discuss the clustering scheme. Theoretical issues and implementation results are discussed in sections 4.3, 4.4 respectively.

4.1 VLSI Circuit Placement

With the advent of high-density VLSI, the chip design process has become a very difficult task. A major issue in VLSI design is that of determining how the circuit components are to be distributed and located on a limited silicon area such that the *interconnect area*, called the *routing area*, is minimized. The whole problem is called the layout problem. The second phase of the layout problem, that is routing, strongly depends on the quality of placement performed in the first phase. Clearly, one would like elements connected more

densely to be placed closer together in the layout. However, determining the optimal placement is an NP-hard problem, which implies that it is impractical to determine the perfect placement. NP-hard problems are generally tackled by heuristic or approximation techniques. These have to be specifically designed for each separate problem. A tradeoff frequently encountered in these techniques is between the speed of the algorithm and the quality of the solution. The placement part of the layout problem is usually done by min-cut partitioning or by a simulated annealing bounding rectangle algorithms. In the following subsection we review works done on the former approach, an example about the bounding rectangle algorithms can be found in [29].

4.1.1 Review of Min-Cut Partitioning Algorithms

In the min-cut partitioning algorithms, the circuit is successively partitioned into two parts based on minimizing the number of connections between the two parts. Kernighan-Lin [24], was originally introduced for partitioning communication networks represented as graphs. Starting with an arbitrary partition, in each iteration the algorithm selects a vertex from each block and exchange the pair so that maximum reduction in the sum of the weighted external edges is achieved. The average complexity of the algorithm is known to be $O(n^2 \log n)$, where $n = |V|$. Schweikert and Kernighan illustrated the deficiencies of using graph models for partitioning circuits [25]. They proposed the *nct-cut* circuit model by representing the true connections relationship among components joined by the same signal line.

Fiduccia and Matheyses [27] proposed a partitioning algorithm with linear run time complexity, where the problem size is now defined as the number of pins in the circuit. Rather than pairwise exchanges, each time only a single component from either block is chosen for position change while keeping the partition roughly in balance. Just like the previous algorithms, to prevent components from thrashing between the blocks, a component is not allowed to change its position twice. The component selection criterion based on *cell gain* is similar to the one identified by Schweikert and Kernighan, with each

single component movement, component selection and cell gains information are updated by manipulating a smart *bucket* list data structure. Thus, the run time complexity is made linearly proportional to the problem size.

Recently, a possible improvement on the linear algorithm is proposed by Krishnamurthy [31], based on the observation that a lot of ties are encountered when selecting a component with the largest cell gain. A multi-level cell gain information look ahead is therefore suggested for tie breaking.

More recently, other efficient min-cut heuristic algorithms are proposed and results of such algorithms are comparable to the previous ones [39, 46, 47]. The mentioned heuristic algorithms tries to find a solution which is good in quality and without spending too much time in searching for this solution. Simulated annealing heuristics can do better in finding the final solution, though at the expense of more computation time.

Simulated annealing is a general purpose combinatorial optimization technique to determine the global minimum of an objective function [30]. Its characteristic feature is the ability to explore the configuration space via configurations that actually increases the cost function being minimized. A set of moves is selected, with which one state of the configuration space can be generated from another. In practical applications to problems such as standard cell placement in integrated circuit layout, simulated annealing gives excellent results at the expense of massive computation time. To remedy this inefficiency, various approaches have been proposed that fall into three categories: parallel implementations of simulated annealing [32], carefully controlled move generation strategies [28, 34, 37, 41, 42, 48], and efficient annealing schedules [36].

The mentioned works are generally used for partitioning the VLSI circuit into different partitions with minimum interconnections between them, Other works that do the physical placement after the partitioning or improves the placement of a placed circuit can be found in [29, 33, 35, 47].

4.2 Partitioning Problem

Most of the VLSI partitioning algorithms discussed in the previous section, model the circuit as a graph or a hypergraph and translate the problem into a pure graph partitioning problem. Some of the mentioned algorithms takes into consideration the physical properties of VLSI circuits, but none of them considers the electrical properties of the circuit. That is, one may consider the dependences between the elements such that elements coupled in the simulation of the electrical behavior of the circuit, should be coupled while placing them on the silicon area. Truly speaking, standard cell placement algorithms can be considered as algorithms that are taking electrical properties into account, but this is not the case in the placement of full custom designs.

Recalling that most of the conventional algorithms view the VLSI circuit as a hypergraph, where cells or modules are the vertices of the hypergraph and connections between those cells as the hyper edges or nets, and defines the min-cut partitioning problem as finding a partition of the set of cells into two blocks A and B such that the number of nets which have cells in both blocks is minimal. A user specified balancing criterion can be used to determine the sizes of the partitions. Each cell or module may be a standard cell or macro cell if semi-custom designs are used, or a group of transistors if full custom designs are used. The efficiency and performance of the partitioning algorithm are proportional to the number of vertices and the nets in the hypergraph representation of the given VLSI circuit. Hence, decreasing the number of vertices and the number of nets is a crucial issue in partitioning for placement. Various automated clustering schemes are proposed in the literature [37, 34] for this purpose. However, these schemes are pure graph theoretic schemes and they do not exploit the electrical properties of the VLSI circuit. As is described in the next section, we propose an automated clustering scheme based on the electrical properties of the given circuit.

4.2.1 The proposed Clustering Approach

In this subsection we propose a clustering scheme that can be efficiently used in forming clusters of elements or CMOS transistors as preprocessing step for the partitioning algorithm. The idea is exactly the same with that of node splitting approach used in partitioning the VLSI circuit for simulation.

Again, we first represent the VLSI circuit by an undirected graph G as defined in Section 3.2. Then we use the node splitting algorithm defined in Section 3.3 to find the common-channel-connected transistors of the circuit. That is we find the sub-blocks $\Omega_1, \Omega_2, \dots, \Omega_k$, of the circuit $\Omega(N, M)$. Then we directly find the hypergraph model of the partitioned circuit $\Omega(N, M, \Sigma)$ by using the Net-list algorithm defined in Figure 3.15. Conventional min-cut partitioning algorithms can then be used on this hypergraph representing the clustered circuit for finding a good placement.

4.3 Theoretical and Practical Issues

The main aim of clustering is to decrease the problem size for the partitioning algorithm, while ensuring that the results are at least as good as those without clustering. Clustering reduces the number of entities that need to be moved from one partition to another. This reduces the search space by reducing the degrees of freedom for the partitioning algorithm, since cells within a cluster are a part and we do not consider cell configurations in such clusters. In addition to a computation speed up, search efficiency is achieved without affecting the solution quality. This follows by observing that without clustering, the partitioning algorithm will try to group highly connected cells, so if one cell is selected to be a move, then other cells would have to somehow find its way close to the partition of the first cell. Several moves would have to be tried, hence more computation is done by the partitioning algorithm.

For the proposed clustering algorithm, to see the difference between the

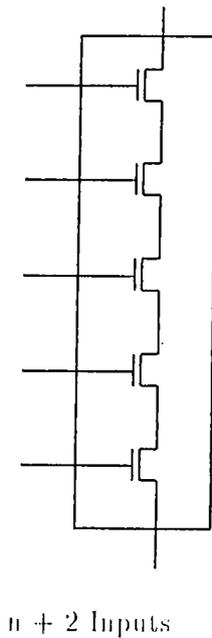


Figure 4.1. An example of CMOS transistors cluster.

amount of computation in non-clustered circuits and clustered circuits, consider a typical cluster as shown in Figure 4.1. This figure shows a common-connected-channel cluster of CMOS transistors obtained after the splitting approach.

Assume that the average number of elements inside a cluster is n . And assume that the partitioning algorithm is implemented using the Fiduccia and Matheyses [27]. Recall that the complexity of this algorithm is $O(P)$ where P is the total number of pins in the circuit. In the case of no clustering, $P = 3 * M$ since each transistor has exactly three pins connected to the nets of the hypergraph. Where M is the total number of CMOS transistors in the circuit. On the other hand, if clustering is used then $P = \frac{M}{n} * (n' + n_i)$, since each cluster has at most n gate inputs in addition to n_i inputs from input nodes to the cluster. Note that the average value of n' is considerably smaller than the number of transistor gates in the clusters, this scheme will be more clear in the next paragraphs. Experimental results showed that the average size of each cluster is $n \cong 10$. Also the average size of inputs from input nodes n_i is about 2 to 3. So it is clear that the average speed up of the proposed scheme compared to the conventional schemes is ~ 3 . In fact the average speed up is

greater than this, because in real VLSI design, each node is supplying input to more than one transistor gate. This causes the average gate inputs n' to be considerably less than n in the clustered circuit. This will be approved by the experimental results discussed in the next section.

While speeding up the computation time, we have to ensure that the complexity of the clustering algorithm is at most equal to the complexity of the partitioning algorithm. This is ensured by our clustering scheme as it was shown in Section 3.3.

Should the size of the clusters be large to combine large number of cells, or small to contain only few number of cells. Large clusters leads to a great reduction in the CPU time for the partitioning algorithm. But, this would affect the solution quality, since the degrees of freedom are greatly reduced. The proposed clustering algorithm exploits the inherent clusters in the circuit while keeping the size of the clusters reasonable so that the solution quality is never affected. Furthermore, experimental results shows that the sizes of the clusters are almost similar to the sizes of the standard cells in semi-custom design. Most of the tested circuits have few different cluster sizes, this will make it easier to do the physical placement and routing in the layout of the VLSI circuit.

Speeding up the computations while improving the solution quality, is not the only advantage of the proposed clustering algorithm. In the following paragraphs we will discuss the main advantages of the proposed clustering scheme.

Since the clustering algorithm is based on the electrical properties used in the simulation of VLSI circuits, exactly two types of interconnections between the final partitions can be found, namely,

Input Node Connections, Such connections are links from input nodes to a drain or source nodes of a CMOS transistor. So each cluster will have at most $|N_I|$ connections from input nodes. Such links have currents greater than zero and should have a specified line widths in the physical layout of the circuit.

Usually, such lines are metal lines and are placed at the borders of the silicon area. Some clusters will have some input links from input nodes connected to a gate node N_G of a CMOS transistor, this links will be considered to be in the second connection type.

Gate Node Connections, all other connections between partitions are inputs to gate nodes of CMOS transistors. Such connections have low currents, hence in the routing phase of the layout of the circuit there may be no need to use wide metal lines. Hence, this may decrease the routing area of the chip. However, In real VLSI layout, the width of such interconnection lines also depends on the fan-out of the node driving the gates.

Another important advantage of the proposed clustering algorithm is that it guarantees the adjacency of similar diffusions. To make this clear, assume that in the conventional partitioning algorithm, two connected transistors are found to be in two different partitions and that the connection link between their source and drain nodes is in the cut set as shown in Figure 4.2(a). Then in the physical layout, there will be two different diffusions for each of the transistors connected by a metal link, this takes more space in the layout area as shown on the right part of the same figure. On the contrary, the proposed clustering algorithm prevents the cut to pass through the connection of the source and drain nodes of the two transistors since they are common-channel connected. This case is explained in part (b) of Figure 4.2 together with the physical layout of the transistors. Of course this should not be the case when one of the transistors is *p-channel* and the other is *n-channel*. Unfortunately, the proposed clustering algorithm is blind and can not differentiate between such cases in the current version. Modifications to the algorithm can be done to detect such cases, or the detection can be left to the physical placement of the clustered cells.

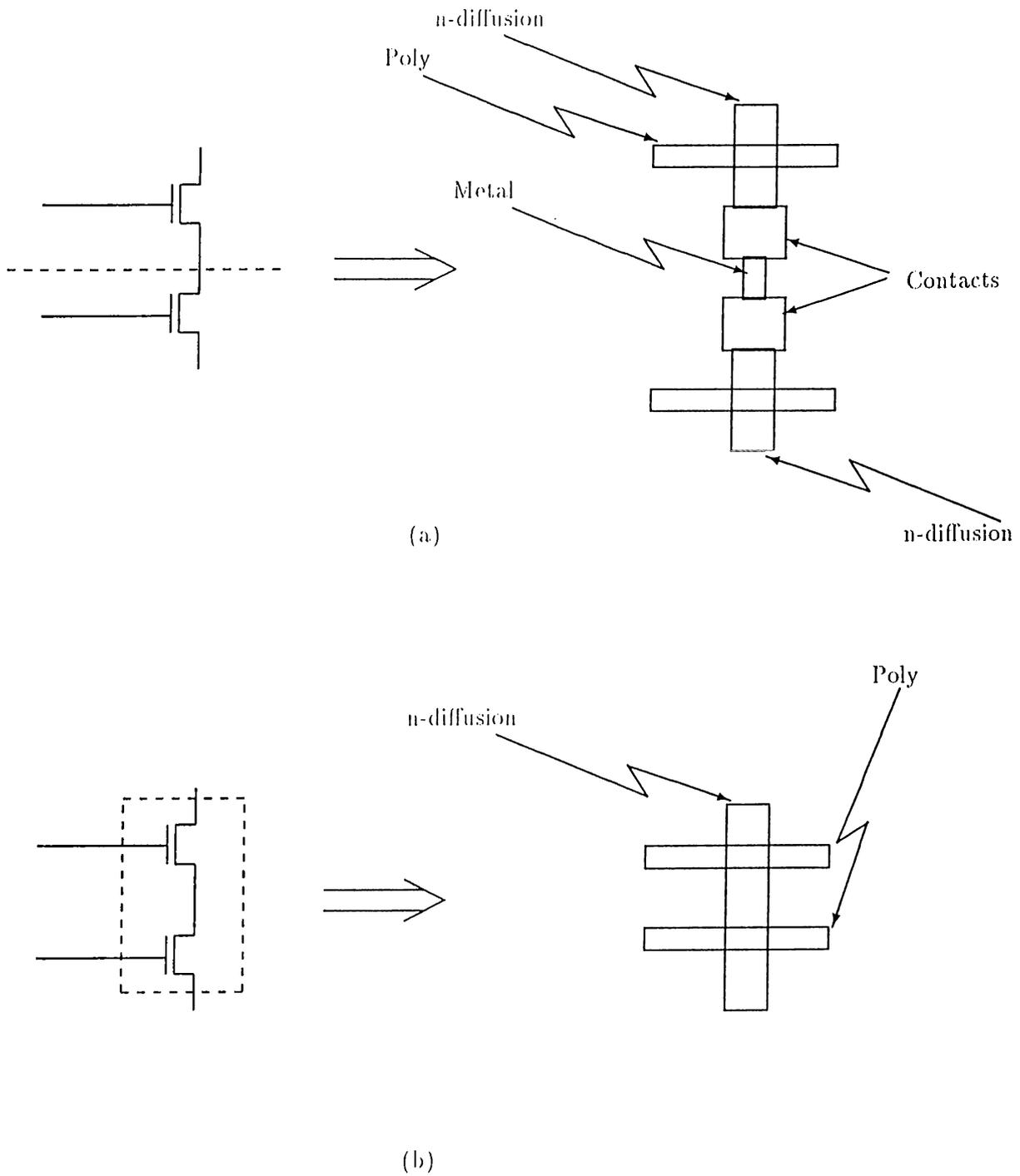


Figure 4.2. Layout of two transistors.(a) With conventional partitioning. (b) Partitioning with clustering.

Table 4.1. Comparing the FM and the CFM results. The algorithms were tested on 9 real problem instances.

| PROBLEM | | | FM PARTITIONING | | | | | CFM PARTITIONING | | | | | DIFF. | |
|---------|---------|-----|-----------------|-------|-----|------|----------|------------------|---------|-----|------|----------|-------|-----|
| M | $ V_I $ | K | V | $ E $ | CS | LI | $T(Sec)$ | V_c | $ E_c $ | CS | LI | $T(Sec)$ | %D | SU |
| 102 | 4 | 2 | 102 | 64 | 6 | 2.0 | 0.2 | 25 | 25 | 5 | 1.9 | 0.1 | 17 | 2.0 |
| 114 | 12 | | 114 | 68 | 7 | 5.3 | 0.3 | 26 | 20 | 4 | 5.2 | 0.1 | 42 | 3.0 |
| 122 | 11 | | 122 | 76 | 8 | 6.6 | 0.3 | 33 | 37 | 7 | 4.9 | 0.1 | 13 | 3.0 |
| 176 | 10 | | 176 | 97 | 12 | 3.5 | 0.4 | 63 | 51 | 9 | 2.3 | 0.2 | 25 | 2.0 |
| 414 | 20 | | 414 | 226 | 11 | 15.4 | 0.7 | 72 | 67 | 10 | 5.3 | 0.2 | 9 | 3.5 |
| 770 | 37 | | 770 | 373 | 12 | 1.3 | 1.3 | 230 | 213 | 7 | 0.3 | 0.4 | 42 | 3.3 |
| 2560 | 6 | | 2560 | 1541 | 9 | 0.3 | 7.5 | 518 | 518 | 7 | 0.0 | 1.9 | 22 | 3.9 |
| 4616 | 138 | | 4616 | 2445 | 40 | 0.3 | 18.5 | 792 | 783 | 43 | 0.2 | 3.5 | -1 | 5.3 |
| 11036 | 17 | | 11036 | 6264 | 119 | 0.3 | 92.5 | 2051 | 1976 | 92 | 0.2 | 20.2 | 23 | 4.6 |
| 122 | 11 | 4 | 122 | 76 | 25 | 31.1 | 0.5 | 33 | 37 | 22 | 22.1 | 0.3 | 12 | 1.8 |
| 176 | 10 | | 176 | 97 | 27 | 20.4 | 0.6 | 63 | 51 | 22 | 7.1 | 0.3 | 19 | 2.0 |
| 414 | 20 | | 414 | 226 | 27 | 19.9 | 1.1 | 72 | 67 | 20 | 12.5 | 0.4 | 26 | 2.8 |
| 770 | 37 | | 770 | 373 | 39 | 2.8 | 2.4 | 230 | 213 | 19 | 0.5 | 0.8 | 51 | 3.0 |
| 2560 | 6 | | 2560 | 1541 | 27 | 0.8 | 12.8 | 518 | 518 | 23 | 0.0 | 2.9 | 15 | 4.4 |
| 4616 | 138 | | 4616 | 2445 | 100 | 0.5 | 26.5 | 792 | 783 | 89 | 0.6 | 4.7 | 11 | 5.6 |
| 11036 | 17 | | 11036 | 6264 | 230 | 0.4 | 135.5 | 2051 | 1976 | 190 | 0.3 | 26.1 | 17 | 5.2 |
| 414 | 20 | 8 | 414 | 226 | 45 | 36.1 | 1.8 | 72 | 67 | 38 | 22.5 | 0.7 | 16 | 2.6 |
| 770 | 37 | | 770 | 373 | 80 | 8.3 | 3.2 | 230 | 213 | 39 | 1.1 | 1.4 | 51 | 2.3 |
| 2560 | 6 | | 2560 | 1541 | 58 | 0.8 | 17.5 | 518 | 518 | 56 | 0.0 | 3.9 | 4 | 4.5 |
| 4616 | 138 | | 4616 | 2445 | 178 | 0.9 | 35.5 | 792 | 783 | 146 | 2.2 | 6.3 | 18 | 5.6 |
| 11036 | 17 | | 11036 | 6264 | 345 | 0.4 | 165.4 | 2051 | 1976 | 290 | 0.9 | 29.6 | 16 | 5.6 |
| 2560 | 6 | 16 | 2560 | 1541 | 117 | 1.9 | 21.20 | 518 | 518 | 116 | 0.0 | 5.2 | 0 | 4.1 |
| 4616 | 138 | | 4616 | 2445 | 255 | 4.3 | 46.50 | 792 | 783 | 221 | 4.5 | 7.9 | 13 | 5.9 |
| 11036 | 17 | | 11036 | 6264 | 555 | 0.7 | 194.60 | 2051 | 1976 | 458 | 1.4 | 34.4 | 18 | 5.7 |
| 11036 | 17 | 32 | 11036 | 6264 | 905 | 0.9 | 205.50 | 2051 | 1976 | 655 | 4.1 | 37.8 | 28 | 5.4 |

Table 4.2. Comparing the SA and the CSA results. The algorithms were tested on 9 real problem instances.

| PROBLEM | | | SA PARTITIONING | | | | | CSA PARTITIONING | | | | | DIFF. | |
|---------|---------|-----|-----------------|-------|-----|------|----------|------------------|---------|----|------|----------|-------|------|
| M | $ V_f $ | K | V | $ E $ | CS | LI | $T(Sec)$ | V_c | $ E_c $ | CS | LI | $T(Sec)$ | %D | SU |
| 102 | 4 | 2 | 102 | 64 | 6 | 2.0 | 60.4 | 25 | 25 | 5 | 1.9 | 17.4 | 17 | 3.5 |
| 114 | 12 | | 114 | 68 | 6 | 5.3 | 3.2 | 26 | 20 | 3 | 5.2 | 17.3 | 42 | 4.1 |
| 122 | 11 | | 122 | 76 | 8 | 3.2 | 72.5 | 33 | 37 | 7 | 1.6 | 23.5 | 13 | 3.1 |
| 176 | 10 | | 176 | 97 | 11 | 12.5 | 57.3 | 63 | 51 | 8 | 11.4 | 17.2 | 27.2 | 3.4 |
| 414 | 20 | | 414 | 226 | 9 | 9.2 | 327.7 | 72 | 67 | 6 | 13.1 | 57.7 | 33.3 | 5.7 |
| 770 | 37 | | 770 | 373 | 12 | 10.1 | 779.8 | 230 | 213 | 6 | 0.3 | 238.2 | 50 | 3.3 |
| 2560 | 6 | | 2560 | 1541 | 24 | 10.0 | 6720.9 | 518 | 518 | 15 | 3.1 | 435.9 | 37.5 | 15.5 |
| 4616 | 138 | | 4616 | 2445 | 72 | 9.9 | 24404.1 | 792 | 783 | 33 | 10.0 | 910.9 | 54 | 26.8 |
| 11036 | 17 | | 11036 | 6264 | 155 | 10.0 | 180157 | 2051 | 1976 | 72 | 7.6 | 6521 | 53 | 27.6 |

4.4 Implementations and Results

In this section, we compare the conventional partitioning methods with the clustered partitioning methods. We have implemented the clustering algorithm proposed in section 4.2.1.

We have also implemented the Fiduccia and Matheyses [27] (FM) heuristic and we used it in partitioning the circuit by first constructing the Net-list of the VLSI circuit. The Net-lists are constructed using the algorithm proposed in Figure 3.15. In this case each block is considered to have exactly one element inside it connected to at most three different nets or hyper edges. We also constructed the Net-lists of the clustered circuit and used the same partitioning algorithm to partition the clustered circuit, we called this method (CFM). Results in the CFM were superior to those in the conventional FM. The average computation speed up is ≈ 4 on the average, and the cut-size is $\approx 20\%$ less. Table 4.1 shows the results of 6 different real VLSI circuit designs using both methods. Results were averaged over 30 times, with a random initial configuration for each time. In this table, V and E denotes the number of vertices and nets of the hypergraph representing the original circuit. Also, V_c and E_c denotes the number of vertices and nets of the hypergraph representing the clustered circuit. Note that, both V_c and E_c are considerably smaller than V and E respectively. The *percent Load Imbalance ratio* (LI) is defined as,

$$LI = \frac{W_{max} - W_{min}}{2 * W_{avg}} * 100\% \quad (4.1)$$

As can be seen from the table, the LI for CFM is in general better than the LI for the conventional FM method.

Table 4.2 shows the results for both the conventional simulated annealing (SA) and the clustered simulated annealing (CSA) methods. Those results are shown for 2-way partitioning only. Once again the performance of the CSA is better than that of conventional SA.

Previous results shows that the proposed clustering method can be efficiently used in the partitioning of VLSI circuits in the full custom designs as a CAD tool in solving the placement problem of VLSI layout.

5. CONCLUSIONS

Partitioning the circuit into sub-circuits or blocks greatly reduces the computation time of the solution of the simulation process while using less memory space. Node Splitting (NS) is the underlying basis for partitioning of large integrated circuits into several, more manageable, sub-circuits to enhance computer simulation efficiency. In this work, partitioning algorithms based on the NS way can be used at the outset as a preprocessing step for the simulation tool being built in the EE Department at Bilkent University, PL-AWE. The proposed partitioning algorithm exploits the inherent partitions or sub-blocks in the circuit.

In chapter 2 of this thesis, we reviewed the basic concepts of graph theory since the core of this work was based on those concepts. Chapter 3 starts by defining the partitioning problem for simulation. Section 3.2 shows how to represent the circuit by an undirected graph. The input node splitting of this graph is done to exploit the inherent clusters of common-channel connected transistors in the circuit. The graph representation of the circuit is then completed by adding the input/output interdependencies between the blocks of the circuit, hence the digraph model representing the clustered circuit were constructed. Finding the SCCs of the digraph representing the partitioned circuit is then constructed by condensing the blocks that have feed-back paths. Leveling and grouping of those SCCs is also considered. Also solving the problem of very large SCCs is discussed in details in the same chapter.

Section 3.5 shows the experimental results after implementing the proposed partitioning algorithm. Once again, it is difficult to confirm the acceptability

of the results shown since the partitioned circuits is not yet tested in the simulation phase. The time taken in partitioning the VLSI circuits is negligible compared to the time taken in the simulation phase and will never cause any overhead in the simulation of the partitioned circuits.

Partitioning VLSI circuits is not only used for speeding the simulation phase in the design process, but also it arises in various aspects of VLSI design automation. It has direct applications in the placement of components during the layout problem of the VLSI circuits.

In our work, we are making use of the partitioning algorithms used in the simulation problem and with small modifications we are doing a preprocessing step to form clusters of elements (common-channel-connected elements). Then we showed that by using well known heuristics, such as Kernighan-Lin and Simulated Annealing, a partitioning can be done on those clusters. The results with this method have been superior to those with the conventional implementations as shown in section 4.4. For real data (with sizes ranging from 400 to 11500 transistors), we have observed a factor of ≈ 4 speed-up in CPU time, together with $\approx 20\%$ improvement in the cut size. Other advantages of the proposed clustering method are discussed in section 4.3. Experimental results show that the proposed algorithms can be efficiently used in VLSI circuit partitioning for simulation and placement.

The interface between the simulation and partitioning algorithm can be the next step to finish this work. Upon the finishing of the whole simulation tool together with the partitioning algorithm, the same simulator will be considered for parallelization as a future work.

Bibliography

- [1] A. Sangiovanni-Vincentelli, L. K. Chen, and L. O. Chua. "A new Tearing approach-Node Tearing Nodal Analysis," in *Proc. IEEE Int. Symp. on Circuits and Systems.*, pp. 113-115, Apr., 1977.
- [2] P. Yang, I. N. Hajj, and T. N. Trick, "SLATE: A Circuit Simulation Program with Latency Exploitation and Node Tearing," *Proc. IEEE Int. Conf. on Circuits and Computers.*, pp. 353-355, October, 1980.
- [3] G. R. Case, "SALOGS - A CDC 6600 Program to Simulate Digital Logic Networks, vol. 1 - User's Manual," Sandia Laboratory Report SAND 74-0441, 1975.
- [4] S. A. Szygenda, "TEGAS2 - Anatomy of a General Purpose Test Generation and Simulation System for Digital Logic," *Proc. of the Ninth ACM Design Auto. Workshop*, June, 1972.
- [5] C. F. Chen, C. Y. Lo, H. N. Nham, and P. Subramaniam, "The Second Generation MOTIS Mixed-Mode Simulator," *Proc. of the 21st Design Auto. Conf., Albuquerque, New Mexico*, pp. 10-17, June, 1984.
- [6] J. White and A. L. Sangiovanni-Vincentelli, "RELAX2.1 : A Waveform Relaxation based Circuit Simulation Program," *Proc. IEEE Custom Integrated Circuits Conf., Rochester, New York*, pp. 232-236, May, 1984.
- [7] Ronald E. Bryant. "A Switch-Level Model and Simulator for MOS Digital Systems," *IEEE Trans. on Computers*, vol. c-33, No. 2, pp. 160-177, Feb., 1984.

- [8] J. White, and A. Sangiovanni-Vincentelli. "Partitioning Algorithms and parallel Implementation of Waveform Relaxation for Circuit Simulation," in *Proc. IEEE Int. Symp. on Circuits and Systems.*, pp. 221-224, June, 1985.
- [9] Vasant B. Rao, and Timothy N. Trich. "Network Partitioning and Ordering for MOS VLSI Circuits," *IEEE Trans. on Computer-Aided Design*, vol. CAD-6, No. 1, pp. 128-144, Jan., 1987.
- [10] Norman P. Jouppi. "Derivation of Signal Flow Direction in MOS VLSI," *IEEE Trans. on Computer-Aided Design*, vol. CAD-6, No. 3, pp. 480-490, May, 1987.
- [11] O. Tejayadi, and I.N. Hajj. "Dynamic Partitioning Methods for Piecewise-Linear VLSI Circuits Simulation," *Int. Journal of Circuit Theory and Applications*, vol. 16, pp. 457-472, 1988.
- [12] Ronald A. Rohrer. "Circuit Partitioning Simplified," *IEEE Trans. on Circuits and Systems*, vol. 35, No. 1, pp. 2-5, Jan., 1988.
- [13] Lena Peterson, and Sven Mattison. "Circuit Simulation On a Hypercube," in *Proc. IEEE Int. Symp. on Circuits and Systems.*, pp. 1119-1122, 1988.
- [14] A.J. van der Hoeven, A.A.J. de Lange, E.F. Deprettere, and P.M. Dewilde. "A Model for High-Level Description and Simulation of VLSI Networks," *IEEE Micro*, pp. 41-47, August, 1990.
- [15] Paul F. Cox, Richard G. Burch, Dale E. Hocevar, Ping Yang, and Berton D. Epler. "Direct Circuit Simulation Algorithms for Parallel Processing," *IEEE Trans. on Computer-Aided Design*, vol. 10, No. 6, pp. 714-725, June, 1991.
- [16] Dan Adler. "Switch-Level Simulation Using Dynamic Graph Algorithms," *IEEE Trans. on Computer-Aided Design*, vol. 10, No. 3, pp. 346-355, March, 1991.
- [17] C.T. Dikmen, M.M. Alaybeyi, S. Topcu, A. Atalar, E. Sezer, M.A. Tan, and R.A. Rohrer. "Piecewise Linear Asymptotic Waveform Evaluation for

- Transient Simulation of Electronic Circuits,” in *Proc. IEEE Int. Symp. on Circuits and Systems.*, vol. 2, pp. 854-857, 1991.
- [18] Masakatsu Nishigaki, Nobuyuki Tanaka, and Hideki ASAI. “Hierarchical Decomposition System and its Availability for Network Solution,” in *Proc. IEEE Int. Symp. on Circuits and Systems.*, vol. 2, pp. 884-887, 1991.
- [19] Chung-Kuan Cheng, Yen-Chuen Wei, and Ze’ev Wurman. “The Mapping of Logic Design into a Very Large Scale Hardware Simulator,” in *Proc. IEEE Int. Symp. on Circuits and Systems.*, vol. 3, pp. 2036-2039, 1991.
- [20] Richard M. M. Chen, W. C. Siu, and Andrew M. Layfield. “Running SPICE in Parallel,” in *Proc. IEEE Int. Symp. on Circuits and Systems.*, vol. 2, pp. 880-883, 1991.
- [21] P. M. Lin. “A circuit Characterization of Principal and Refined Partition of Graphs,” in *Proc. IEEE Int. Symp. on Circuits and Systems.*, vol. 2, pp. 994-997, 1991.
- [22] Sujit Dey, Franc Breglez, and Gershon Kedem. “Circuit Partitioning for Logic Synthesis,” *IEEE Journal of Solid-State Circuits* ., vol. 26, No. 3, pp. 350-363, March, 1991.
- [23] A. Vladimirescu, Kaihe Zhang, A. R. Newton, D. O. Pederson, and A. Sangiovanni-Vincentelli. *SPICE User’s Guide*, University of California, Berkeley, Ca, 94720.
- [24] Kernighan, B. W., and Lin, S. “An efficient heuristic procedure for partitioning graphs,” *Bell Syst. Tech. J.*, vol. 49, pp. 291-307, 1970.
- [25] Schweikert, D. G., and Kernighan, B. W. “A proper model for the partitioning of electrical circuits,” in *Proc. 9th Design Automat. Workshop*, pp. 57-62, 1979.
- [26] Garey, M. R., and Johnson, D. S. *Computers and Intractability*. San Francisco, CA: Freeman, pp. 209-210, 1979.
- [27] Fiduccia, C. M., and Mattheyses, R. M. “A linear heuristic for improving network partitions,” in *Proc. Design Automat. Conf.*, pp. 175-181, 1982.

- [28] S. B. Akers. "Clustering Techniques for VLSI," in *Proc. IEEE Int. Symp on Circuits and Systems*, pp. 172-176, 1982.
- [29] G.J. Wipfler, M. Wiesel, and D.A. Mlynski. "A Combined Force and Cut Algorithm For Hierarchical VLSI Layout," in *Proc. 19th Design Automat. Conf.*, pp. 671-677, 1982.
- [30] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. "Optimization by simulated annealing," *Science*, vol. 220, pp. 671-680, 1983.
- [31] Krishnamurthy, B. "An improved min-cut algorithm for partitioning VLSI networks," *IEEE Trans. Comput.*, vol. C-33, pp. 438-446, 1984.
- [32] P. Banerjee, and M. Jones. "A Parallel Simulated Annealing Algorithm for Standard Cell Placement on a Hypercube Computer," *Proc. IEEE Int. Conf. on Computer-Aided Design*, vol. 20, No. 2, pp. 510-522, 1985.
- [33] Mely Chen Chi. "An Automatic Rectilinear Partitioning Procedure for Standard Cells," in *Proc. 24th ACM/IEEE Design Automat. Conf.*, pp. 50-55, 1987.
- [34] Lov K. Grover. "Clustering based Simulated Annealing for Standard Cell Placement," in *Proc. 24th ACM/IEEE Design Automat. Conf.*, pp. 56-59, 1987.
- [35] Inderpal Bhandari, Mark, and Daniel Slewioerek. "The Min-Cut Shuffle: Toward a Solution for the Global Effect Problem of Min-Cut Placement," in *Proc. 25th ACM/IEEE Design Automat. Conf.*, pp. 681-685, 1988.
- [36] Jimmy Lam, and Jean-Marc Delosme. "Performance of a New Annealing Schedule," in *Proc. 25th ACM/IEEE Design Automat. Conf.*, pp. 306-311, 1988.
- [37] Sivanarayana Mallela, and Lov K. Grover. "Clustering based Simulated Annealing for Standard Cell Placement," in *Proc. 25th ACM/IEEE Design Automat. Conf.*, pp. 312-317, 1988.

- [38] Peterson, C., and Anderson, J. R. "Neural networks and NP-complete optimization problems; a performance study on the graph bisection problem," *Complex Syst.* vol. 2, pp. 59-89, 1988.
- [39] Andrew B. Kahng. "Fast Hypergraph Partition," in *Proc. 26th ACM/IEEE Design Automat. Conf.*, pp. 762-766, 1989.
- [40] Thang Bui, Christopher Heigham, Curt Jones, and Tom Leighton. "Improving the Performance of the Kernighan-Lin and Simulated Annealing Graph Bisection algorithms," in *Proc. 26th ACM/IEEE Design Automat. Conf.*, pp. 775-778, 1989.
- [41] Michael Upton, Khosrow Samii, and Stephen Sugiyama. "Integrated Placement for Mixed Macro Cell and Standard Cell Designs," in *Proc. 27th ACM/IEEE Design Automat. Conf.*, pp. 32-35, 1990.
- [42] Abhijit Chatterjee, and Richard Hartley. "A New Simultaneous Circuit Partitioning and Chip Placement Approach Based on Simulated Annealing," in *Proc. 27th ACM/IEEE Design Automat. Conf.*, pp. 36-39, 1990.
- [43] Lengauer, T. *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley, pp. 251-258, 1990.
- [44] Van den Bout, D. E., and Miller, T. K. "Graph partitioning using annealed neural networks," *IEEE Trans. Neural Networks*, vol. 1, no. 2, pp. 192-203, 1990.
- [45] Yih, J. S., and Mazumder, P. "A neural network design for circuit partitioning," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 1265-1271, 1990.
- [46] Yoko Kamidoi, Shin'ichi Wakabayashi, Jun'ichi Miyao, and Noriyoshi Yoshida. "A Fast Heuristic Algorithm for Hypergraph Bisection," in *Proc. IEEE Int. Symp. on Circuits and Systems.*, vol. 2, pp. 1160-1163, 1991.
- [47] In-Cheol Park, and Chong-Min Kyung. "Vertical Partitioning of Row-Based Circuits with Minimal Net-Crossings," in *Proc. IEEE Int. Symp. on Circuits and Systems.*, vol. 3, pp. 2032-2034, 1991.

- [48] Takeo Hamada, Chung-Kuan Cheng, and Paul M. Chau. "An Efficient Multi-level Placement Technique Using Hierarchical Partitioning," in *Proc. IEEE Int. Symp. on Circuits and Systems.*, vol. 3, pp. 2044-2047, 1991.
- [49] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MacGraw Hill, 1990.
- [50] Wolter Scott. *Magic Tutorial #8: Circuit Extraction*, University of California, Berkeley, Ca, 94720.