California Institute of Technology

Computer Science

5104:TR:83

A Graph Model and the Embedding of MOS Circuits

by

Tak-Kwong Ng

The author is supported by IBM Corp. under its Residency Study Program.

The research described in this report was sponsored in part by the Office of Naval Research under contract number N-00014-84-K-0043.

The research was also sponsored in part by the Defense Advanced Research Projects Agency, ARPA order number 3771, and monitored by the Office of Naval Research under contract number N00014-79-C-0597.

Abstract

The direct automated transformation of a circuit into the "best" physical layout is hard. An alternative is the transformation of a circuit into a suitable intermediate form, the layout topology. Each layout topology defines an equivalence class of physical layouts. A few layout topologies can be chosen according to their likeliness for leading to the "best" design. Each of these layout topologies can then be transformed into a physical layout that will be optimized. The final design can be chosen from the set of optimized physical layouts. Each optimized physical layout corresponds to a unique layout topology.

A circuit is modeled as a graph. The circuit's graph model is analyzed by the embedding algorithm. The embedding algorithm determines the set of layout topologies that will be transformed into the physical layouts for further processing. A layout topology is specified as a graph together with the set of cyclic orders of the vertices, and the layer assignment of the edges.

"Faith is the confident assurance that something we want is going to happen. It is the certainty that what we hope for is waiting for us, even though we cannot see it up ahead."

Hebrews 11:1

Special acknowledgement is due to Professor Lennart Johnsson for his supervision and discussion on the research. I must also thank him for the comments, suggestions, and the many hours he spent in the reading and the editing of the drafts that lead to this manuscript.

Acknowledgements

I want to thank many people who have helped my research, in one way or another. In particular, I want to thank the following:

I want to thank my wife Ya-En for everything and, in particular, for her encouragement.

I want to thank Mr. Ralph Solomon of the IBM Corp. Without his recommendation I would not be participating in the Residency Study program.

I want to thank Dr. William Heller of the IBM Corp. Without his recommendation I would not be studying at Caltech.

I want to thank Dr. Alexander Frey, Professor Randy Bryant, and William Dally for their valuable comments on the drafts.

I want to also mention Dr. Ronald Ming of R.P.I. for his suggestion of adapting the Hopcroft-Tarjan's planarity testing algorithm for the graph embedding problem.

Table of Contents

1 The VLSI Design Process 1

1.1 Overview of the VLSI Design Process 1

1.2 Silicon Compilation and the Physical Design Process 3

1.3 Connection Cost 6

1.4 An Alternative Design Strategy 7

1.5 Remarks 9

1.6 Definitions and Concepts 10

2 Circuit to Graph Transformation 19

2.1 Introduction 19

2.2 Topological Characteristics of MOS Circuits 20

2.3 A Graph Model for MOS Circuits 23

2.4 Summary of the Graph Model for MOS Circuits 27

2.5 Circuit Description 29

2.6 A Circuit to Graph Transformation Algorithm 29

2.7 Algorithm Complexity 32

2.8 Remarks 32

3 An Application of the Graph Model - the Embedding Problem 34

3.1 Introduction 34

3.2 Planarity Testing 35

3.3 Hopcroft and Tarjan's Planarity Testing Algorithm 37

3.4 Extensions of Hopcroft-Tarjan's Algorithm for Graph Embedding

..... 40

3.5 General Graph Embedding 42

3.6 Valid Graph Embeddings 50

3.7 Algorithm Complexity 53

3.8 Remarks 56

4 Circuit Graph Embedding Topology to Layout Topology 57

4.1 Introduction 57

4.2 Creating a Layout Topology 58

4.3 Layout Topology to Physical Layout Transformation 60

4.4 An Algorithm for Drawing Embedded Graphs 62

4.5 Algorithm Complexity 70

4.6 Remarks 70

5 Experience 71

6 Conclusion 94

Bibliography 99

List of Illustrations

Figure 1.2.1 Implementations of an Exclusive OR Circuit. 5

Figure 2.1.1a A Planar Circuit. 21

Figure 2.1.1b Goldstein-Schweikert's Model of the Planar Circuit. 22

Figure 2.1.1c VanCleemput's Model of the Planar Circuit. 23

Figure 2.3.1 Non-Planarity Caused by Using 2 Gate Terminals. 25

Figure 2.3.2 Another Circuit Topology/Models for the Circuit in Figure 2.1.1. 26

Figure 2.6.1a Circuit Specification of an XOR. 31

Figure 2.6.1b A Graph Model of an XOR. 32

Figure 3.2.1 Pairwise Incompatibility Test. 36

Figure 3.3.1 Relations of Cycles and Paths. 38

Figure 3.3.2 Rules for Determining the Orientation of a Cycle. 39

Figure 3.4.1 The Initial Cycle and Exterior Vertex. 43

Figure 3.4.2a A Planar Graph, its Paths and Path Tree. 44

Figure 3.4.2b The Orientation Constraint/Compensation Graph. 45

Figure 3.4.3a K6, its Paths, Path Tree, Orientation Constraint Graph. 46

Figure 3.4.3b The Compensation Graphs for K6. 47

Figure 3.5.1 Orientation Assignment for the Graph of Figure 3.4.2. 48

Figure 3.5.2 Orientation/Plane Assignment for the Graph of Figure 3.4.3. 50

Figure 3.6.1 An Example of an Invalid Cyclic Order. 54

Figure 3.6.2 An Example of a Valid Cyclic Order. 55

Figure 4.2.1 Expansion of a Gate Vertex. 61

Figure 4.4.1 Working Example for the Drawing Algorithm. 65

Figure 4.4.2 Rules to Determine the Drawing Direction of Edges. 66

Figure 4.4.3a The Drawing of a Graph. 67

Figure 4.4.3b The Horizontal Position Constraint Graph. 68

Figure 4.4.3c The Vertical Position Constraint Graph. 69

Figure 5.1.1 Area Penalty Caused by a Contact. 72

Figure 5.1.2a Circuit Schematic and Specification of an XOR. 74

Figure 5.1.2b The XOR's Circuit Graph. 75

Figure 5.1.2c Paths, and Path Tree of the XOR's Circuit graph. 76

Figure 5.1.2d The Orientation Constraint Graph and Compensation Graphs. 78

Figure 5.1.2e An Orientation and Layer Assignment. 79

Figure 5.1.2f The Topological Circuit Graph. 80

Figure 5.1.2g The Transistor Trees. 81

Figure 5.1.2h Result of the Gate Vertex Expansion. 82

Figure 5.1.2i A Rectangular Drawing of the Layout Topology. 83

Figure 5.1.2j The Compacted Layout of the XOR, (Graph Approach). 84

Figure 5.1.2k The Compacted Layout of the XOR, (Manual Approach). 85

Figure 5.1.2l The Compacted Layout of the XOR with "Poor" Topology. 86

Figure 5.1.2m The Compacted Layout of the XOR (G.S. Model). 87

Figure 5.1.2n Summary of the Quality of Various Topology of the XOR.

..... 88

Figure 5.1.3a The Schematic of a Pulse Synchronizer Circuit. 89

Figure 5.1.3b Two Layouts of the Pulse Synchronizer Circuit. 90

Figure 5.1.4a A 4 Bit Carry Chain. 91

Figure 5.1.4b A Compacted Layout of the 4 Bit Carry Chain. 92

Preface

The automated transformation of designs specified in circuit schematics into the "best" physical layout is a problem of great importance. We have taken a graph oriented approach to this problem. Circuit schematics are modeled as graphs, that are embedded in planes. The embedded graphs are then mapped into physical layouts, by mapping abstract planes into physical layers, and transforming the graph description into a description of a layout (geometry). A graph model for circuits implemented by metal oxide semiconductor (MOS) transistors is presented. An algorithm for mapping a circuit into its circuit graph model is described.

Graph embedding topologies are derived in two steps. The graph is initially analyzed and the information about possible crossings between paths is captured in a set of constraint graphs. The second step is the derivation of a solution in which the constraints are satisfied and some criteria are optimized. An algorithm for the first step is described. The second step is usually NP complete. A heuristic algorithm is given.

A circuit graph embedding topology does not necessarily specify a valid physical implementation. This situation and the algorithm for determining the validity of the embedding topology is described. An algorithm for generating a drawing from a layout topology that can be easily expanded into a valid physical layout is also presented.

The graph model and the algorithms are presented in the context of the N-channel MOS (nMOS) technology with a single layer of poly-crystalline silicon and a single layer of metal. The graph model for the MOS transistor is

universal. The cross-over resolving strategy depends on the number of electrically conducting layers of materials and it must be tailored for each specific technology.

Figure 0.1.1 illustrates our approach to the circuit embedding problem as encountered in VLSI design. Logic designs are specified in circuit schematics. Circuit schematics are then transformed into circuit graphs. The specification format of circuit schematic, our graph model and the transformation algorithm are discussed in Chapter 2. Other circuit graph models are also discussed in Chapter 2.

Circuit graphs are analyzed by our embedding algorithm. For each circuit graph, a path tree, an orientation constraint graph and a set of compensation graphs are generated by the algorithm. The embedding problem is then mapped into a graph coloring problem. A graph embedding topology is totally defined by the path tree, the set of orientation assignments (left or right) of the bridges, and the set of plane assignments of the paths . The variations in the orientation and the plane assignment define different graph embedding topologies. The embedding algorithm and the selection of embedding topologies for further processing is discussed in Chapter 3. The algorithm for the drawing of an embedding representing a particular topology is described in Chapter 4.

The embedding algorithm is applicable to any graph. The algorithm, when applied to a circuit graph, generates circuit graph embedding topologies. Additional checking is needed when the algorithm is applied to a circuit graph to guarantee that the generated circuit graph embedding topologies have physical implementations. Furthermore, an expansion step is needed to convert circuit graph embedding topologies into layout topologies. A

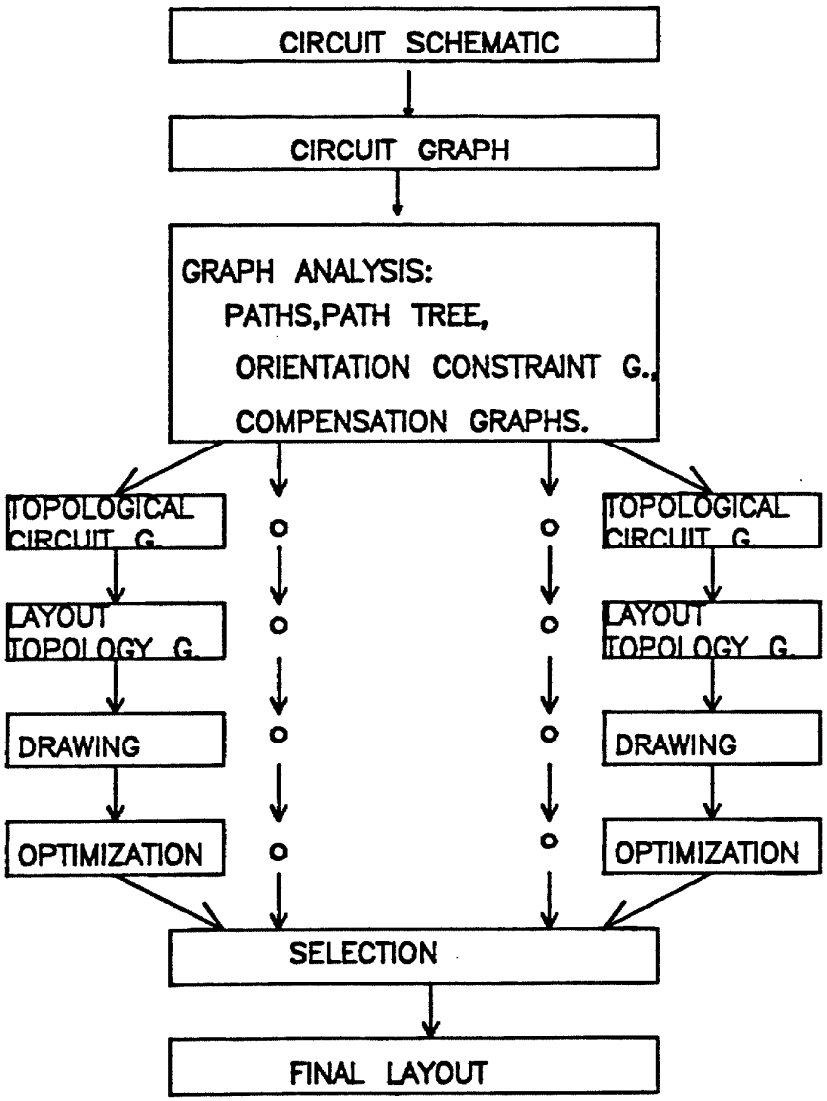


Figure 0.1.1 Our Approach to the Circuit Embedding Problem

definition of the layout topology and the algorithm for transforming graph embedding topologies to layout topologies are presented in Chapter 4. The layout topology is captured in a layout topology graph together with the

topological specification. The expansion is aided by the generation of a transistor tree for each gate vertex.

Finally, a drawing can be generated for each layout topology. The order of the drawing of the paths is defined by the path tree. Directions are assigned to the edges of the layout topology graph based on the orientation assignment of the bridges. Layers are assigned to the edges of the layout topology graph based on the layer assignment of the paths. The physical positions of the vertices of the layout topology graph are computed with the aid of the two position constraint graphs, one is for the horizontal direction, and the other is for the vertical direction. The drawing representing a layout topology can then be optimized. The final layout can be chosen from the set of optimized layouts.

Examples of layouts generated with the described process, some run time data, and the layout of the same circuit corresponding to a "poor" topology are given in Chapter 5. The conclusions of the report is given in Chapter 6.

CHAPTER 1

The VLSI Design Process

1.1. Overview of the VLSI Design Process

With the arrival of the micro-chip age, very large scale integration (VLSI) chips with millions of transistors will be fabricated in the near future. Even though a physical layout is the final product of the VLSI design process, it is undesirable to work at the geometric level for two major reasons. First, the amount of geometric data of a VLSI chip is enormous. A chip with one million transistors may have tens of millions of geometric shapes. This is similar to the situation in programming. Machine code is the only form executable by a computer. It is tedious to program in machine code. The second reason is that there are many decisions made during the design process that do not depend on the physical implementation. For these decisions there is no advantage to work at the geometric level. This is similar to the task of developing an algorithm. The decision on using a certain data structure depends mainly on the problem at hand rather than the programming language in which the algorithm is implemented.

The design process is usually partitioned into two stages. They are the logic design stage and the physical design stage. In the logic design stage, the designer may use a component without specifying its physical implementation or its physical location within the chip. The designer may specify that there should be an electrically conducting path between two terminals of components without specifying the physical implementation of the path. In general, during the logic design stage, the designer need not deal with the

physical aspect of the design. The design can be expressed in an abstract form, the circuit schematic.

Circuit schematic captures the design with a set of components and the connections of components. Components must be transformed into physical layouts. A physical layout is a set of geometric shapes on different layers that will carry adequate information for the fabrication of the component. The physical locations of components within the chip must be specified. Connections must be implemented by providing the necessary electrically conducting paths between terminals of components. This is the physical design stage. The design can be expressed in some symbolic specification for geometry. An example of a symbolic specification language is the Caltech Intermediate Form (CIF) [Mead,Conway 80]. CIF is accepted by many silicon foundries as a specification language for mask information.

A circuit schematic consists of two types of information. It consists of a list of components in the design. Each component has a set of terminals. The schematic also consists of a list of nets. A net is a set of related terminals. There must be at least one electrically conducting path between each pair of terminals belonging to the same net. This is usually accomplished by connecting all terminals belonging to the same net with a strip of conducting material. This process is the net routing process. Terminals belonging to different nets are not to be connected.

The continuous strip of conducting material used in the routing process is a wire. For the nMOS technology, the conducting material can either be metal, diffused silicon, or poly-crystalline silicon. Wires made up of these materials are **metal wires**, **diffusion wires**, and **poly wires** respectively.

The physical design problem is the problem of determining the positions of components on a plane and realizing the nets such that an objective function is optimized and none of the technological or physical constraints are violated. Constraints may consist of maximum layout dimensions and minimum spacing between wires.

1.2. Silicon Compilation and the Physical Design Process

Tremendous advances have been made towards high level design tools. The notion of silicon compilers is discussed widely in the literature [Ayres 83, Bergmann 83, Johannsen 81, Siskind, et al 82, Shrobe 82]. The goal of a silicon compiler is similar to the goal of a software compiler. A design can be specified easily and precisely with a high level language. Procedures are provided for the transformation of the design from its high level description into possibly many levels of circuit schematics. Each circuit schematic specifies a building block. Building blocks can be nested, i.e., a building block may be composed of many other building blocks. The **primitive building blocks** are the standard components with predefined layouts. A building block in this context is equivalent to a procedure in software. Each primitive building block is equivalent to a machine instruction.

Silicon compilers, like software compilers, differ from each other in many ways. The high level languages can be different. The set of primitive building blocks can be different. The procedures for transforming a design from its high level description into the hierarchy of building blocks can be different.

Current silicon compilers have two common shortcomings. First, the functions of the set of the primitive building blocks are few at the level which they are defined, and rigid. In some compilers each primitive building block

defines a class of closely related functions, as in [Johannsen 81], which somewhat alleviates the problem. The second drawback of current silicon compilers is that the layouts of primitive building blocks are rigidly defined. In compilers where primitive building blocks define a class of closely related functions the layouts are also close. The parameterized register cells in Bristle Blocks, [Johannsen 81], illustrates the level of flexibility at the primitive building block level that might be available in state-of-the-art silicon compilers. Such compilers also have a restricted way of composing primitives.

The rigidness of the functions of the set of primitive building blocks, their layout and interconnection hurts the efficiency of a silicon compiler. It only allows a restricted local optimization at best. The importance of suitable primitives is illustrated with the following example that is naive, but emphasizes the point we are making. Let us assume that the only primitive building block is the NAND gate. An exclusive OR circuit can be implemented with four NAND gates. If there is no restriction on the functions of the primitive building blocks, then an exclusive OR circuit can be implemented with seven transistors. Both circuits are shown in Figure 1.2.1. The layout of the second implementation would be more compact. Software compilers have similar shortcomings. A program can be smaller and its execution can be faster if it is compiled into the micro instructions of the machine.

The rigidness of the layouts of the set of primitive building blocks hurts the efficiency of a silicon compiler in another way. Silicon is equivalent to memory in software. Building blocks are equivalent to subroutines. Since the cost of branching is small and uniform in a random access model of computation, and memory is logically a one dimensional entity, the simplest procedure for packing the subroutines into the memory is to place them con-

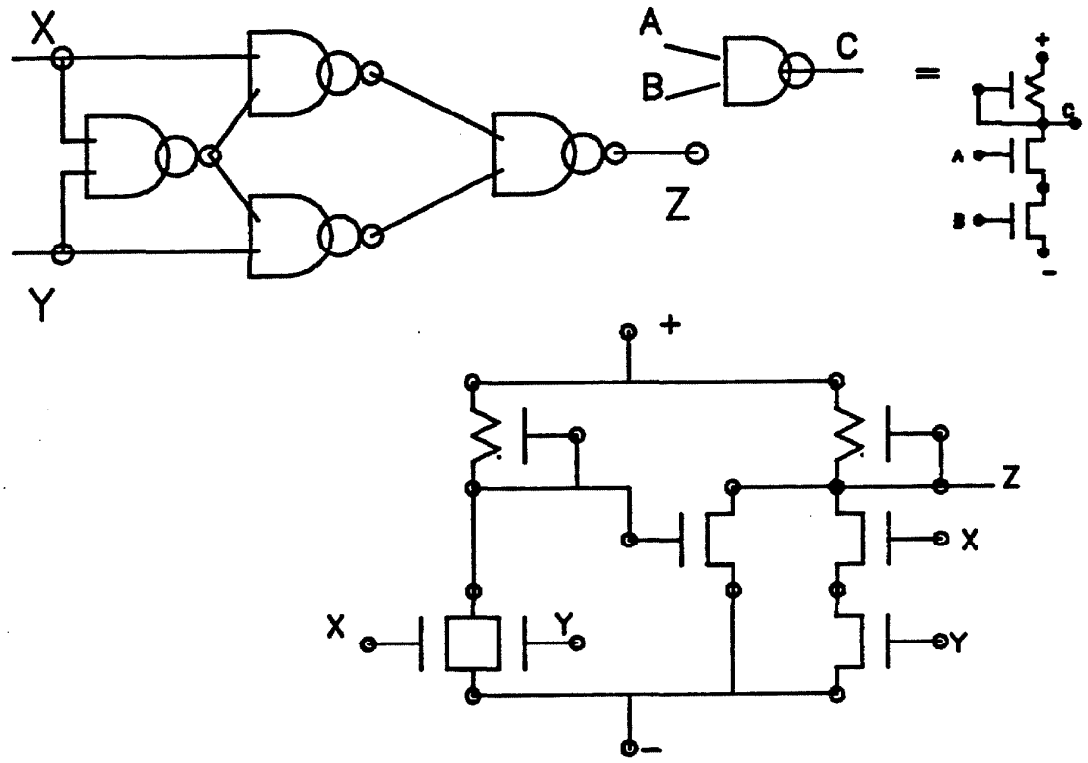


Figure 1.2.1 Implementations of an Exclusive OR Circuit.

secutively in the memory. Hence, there is usually no unusable space in the memory.

Silicon is a two dimensional entity. Building blocks occupy the area. Connection cost is not uniform. It is desirable to place the highly interconnected building blocks close to each other to minimize the connection cost. The dimensions of the building blocks usually are not uniform. It is unlikely that the rigid layouts of the primitive building blocks can be packed tightly in the overall design every time. The only exception is the gate array. The

physical dimensions of the primitive building blocks of the gate array are identical and the locations of the building blocks are fixed. There is no possibility for compaction. However, this does not imply that the area for the logic function is minimized.

1.3. Connection Cost

The rigidness of the layouts of the set of primitive building blocks hurts the efficiency of a silicon compiler in connection cost. This shortcoming is not severe in a software compiler. Connection between building blocks is equivalent to branching to a new memory location in software. Each branch in the random access model of computation is considered as having a uniform cost in program length and performance. Every memory location is logically one unit of cost away from others. Connection cost between building blocks is not uniform and it depends on the length of the actual conducting path. The shapes and the placements of every building block, as well as the locations of the terminals within each building block, affect the connection cost. The rigid layouts of the primitive building blocks will, in general, lead to higher connection cost than layouts with fewer constraints.

Connection cost is composed of two factors, the routing area and the delay. There is a finite resistance and capacitance for each unit of a wire. The propagation delay through a long wire depends on the length and the properties of the wire as well as the driving source and the receiving sink [Seitz 79, Bilardi et al 81]. If we assume that the source and the sink are simple, minimum size, nMOS inverters, then the delay can be approximated by the diffusion equation relating it to time and distance. A simple nMOS inverter is an inverting circuit that uses one depletion mode pull-up transistor and one enhancement mode pull-down transistor. The wire delay increases

quadratically with the length of the wire normalized by a constant that increases linearly with the transit time of a basic switching device.

The propagation delay for a 1 millimeter (mm.) diffusion wire in the six micron nMOS technology is typically 1 nanosecond (ns.). This wire may be used for connecting building blocks with an internal delay of about 25 ns. The connection delay is small compared to the delay of the building blocks. With feature sizes scaled down by a factor of ten in the linear dimension, the length of the 1 mm diffusion wire will be 0.1 mm. The wire delay will remain 1 ns. The transit time of a basic switching device in the new technology will be reduced by a factor of ten. The connection delay will be of the same order as the internal delay of the building blocks. Hence, in the new technology it is important to minimize the lengths of the connections between building blocks.

The above argument is simplistic. The use of an additional metal layer for connection will greatly reduce the wire delay. The scaling of the thickness of the conducting material will increase the wire delay.

1.4. An Alternative Design Strategy

The total transistor area usually accounts for less than two percent of the total area of a chip. The remaining area is either unused or it is taken up by wires. Some improvement in the packing of the building blocks and the reduction in the routing area will not only reduce the total layout area, but may also increase performance. Connection cost can be reduced by changing the logic design or the physical layout.

Manual physical design with some computer aids produce compact designs, usually after a few iterations. Manual physical design is time-consuming and requires experienced personnel to achieve a good result.

An alternative to manual design and silicon compilers with rigid building blocks is to have tools based on transistors and wires as the only primitives. The procedure for transforming a design from its high level description into a circuit of only transistors and wires is non-trivial. The problem of transforming a circuit of only MOS transistors and wires into a layout is the general layout problem. We address the latter problem in this report.

The general layout problem is intractable. Many special cases of the general layout problem, such as the Steiner tree problem [Sahni,Gonzalez 76] and the quadratic assignment problem [Karp 72], are NP-complete. A physical layout is characterized by the specification of the positions and the realization of components, as well as the realization of the nets. Since finding an optimal solution of the physical layout problem is NP-complete, it is in most interesting cases prohibitively expensive to compute because of the enormous size of the solution space.

One possible simplification is to partition the physical layout process into two stages, a topological layout stage, and a layout optimization stage. In the topological layout stage the relative positions of components and the net configurations are determined. The design is then expressed as a layout topology that can be readily transformed into a physical layout. It is during the layout optimization phase that the final positions of components and the physical specification of wires are determined. Optimization algorithms have the option of adjusting the positions of components, and the position and physical dimension of wires for optimal final layout.

The advantage of this approach is that there are some characteristics of the physical layout that do not depend significantly on the actual physical specification of the components and the wires. We have found layout topol-

ogy, defined below, to be a suitable intermediate form between the abstract logic design and the physical layout. The layout topology depends primarily on the architecture and the logic design, and it is readily translatable to a layout. This belief has yet to be supported by empirical data.

1.5. Remarks

Logic designed directly with transistors will result in, at most, the same number of transistors as the same logic implemented with higher level building blocks. In practice, logic designed directly with transistors will result in much fewer transistors. The superior result would, without automated circuit to physical layout transformation, require a substantial manual effort. This is a major reason for not designing directly with transistors. Providing an automated link between circuits and layouts will eliminate most of the manual effort, which in turn will encourage designers to work directly with transistors.

It is unlikely that an automated transformation from circuits to layouts is effective for a "chip full" of transistors because the resource requirement of the process is enormous. One way to manage the complexity is to build a hierarchy of layouts. The transformation can then be applied to only a small portion of the chip. Each piece of the circuit is a building block. This implies that other algorithms are needed to determine the placement of the building blocks and to provide connections between building blocks. The placement and routing problem is usually hard and often ineffective for the case of building blocks with rigid layouts.

The ability to generate a layout from the circuit automatically implies that the shapes of the layouts can be changed quickly to improve the packing of the overall layout. Placement algorithms, such as the min-cut

algorithm[Breuer 77], and the chip planning algorithm [Heller,Sorkin,Maling 82], that depend on the flexibility of the shapes of the building blocks may become workable. Routing may be improved by changing the shapes of the layouts of the building blocks to provide extra routing tracks such that the need to route around building blocks is eliminated.

1.6. Definitions and Concepts

Our approach to the layout problem is based on graphs. In the subsequent sections of this report the following definitions will be used.

Circuit schematic is a set of transistors, a set of ports, a set of nets, and a perimeter. It captures the logic design. Transistors have three terminals and a type. The terminals are traditionally called drain, source, and gate. In nMOS technology the transistor types are enhancement mode and depletion mode. **Ports** are connection points to the exterior of a circuit schematic. Ports are located on the perimeter of the circuit schematic, as well as of the layout. All connections between the exterior and the interior of a circuit schematic must be made through ports. A **net** is a set of terminals and/or ports, connected at all times, by wires and contacts. The layout corresponding to a circuit schematic is confined to some bounded area. The **perimeter** of the layout encloses the implementation of the circuit. The circuit schematic can be assigned a perimeter in an analogous way.

In our graph representation of a circuit schematic, nets are represented by vertices, called **net vertices**. Net vertices are of two types, **gate** and **nogate**. A gate net vertex, or **gate vertex** for short, is a vertex that corresponds to a net with at least one terminal being a gate terminal of a transistor. If none of the terminals in a net is a gate terminal, then the corresponding vertex in the graph representation is of type nogate, and the

vertex labeled **nogate vertex** for short. In a typical design most vertices are gate vertices. The exterior of a circuit is represented by the **exterior vertex.** There are no other vertices in our graph representation of a circuit schematic.

A transistor is in our graph model defined implicitly by a pair of directed edges terminating on the net vertex to which the gate terminal of the transistor belongs. The two edges are named the **drain edge** and the **source edge**, respectively. The drain edge emanates from the vertex representing the net to which the drain terminal belongs. The source edge emanates from the vertex representing the net to which the source terminal belongs.

If the gate is tied to the drain (source), the drain(source) edge will emanate from and terminate on the same vertex. Note that drain and source edges can emanate from a gate vertex of one transistor and terminate on a gate vertex of another transistor. Gate vertices in general have emanating as well as terminating edges. Only terminating edges represent transistors associated with the gate vertex on which they terminate.

Ports are represented by undirected edges in our graph representation of a circuit schematic. A **port edge** has one end incident on the exterior vertex, and one end on a net vertex. There is one port edge for each port. A prescribed order of ports at the perimeter of a circuit schematic is represented by a constraint on the cyclic order of the exterior vertex.

A **circuit graph** is a labeled, connected graph consisting of one exterior vertex, at least one net vertex, and a set of edges. Vertices are labeled with the names of the nets they represent. The exterior vertex is labeled exterior. Drain and source edges are labeled with the name of the transistor to which they belong and the name of the vertex from which they emanate. Port edges

are labeled with the name of the net to which they belong. A constraint on the cyclic order of the exterior vertex may be included.

A net in the circuit schematic is sometimes referred to as the drain, gate or source net if the transistor it is being viewed from has its drain, gate or source terminal belonging to that net. Note, the same net can be labeled drain, source, or gate net depending on from which transistor (terminal) it is viewed.

The following common graph notions are used:

- A **path** is an alternating sequence of distinct vertices and edges, beginning and ending with vertices.

- **Disjoint paths** have no edges in common.

- A **cycle** is an alternating sequence of vertices and edges in which all vertices are distinct, except the beginning and ending vertices that are the same.

- The **direction** of a path or a cycle is determined by the order in which the vertices are given.

- A **connected graph** is a graph in which every pair of vertices are joined by a path.

- A **subgraph** of a graph G has a vertex set that is a subset of the vertex set of G and a set of edges that is a subset of the edge set of G.

- A **connected component** of a graph is a maximal connected subgraph

- A **partition** of a graph G is a set of subgraphs with disjoint edge sets

such that their union is the edge set of the graph G. Each such subgraph is a **component** of the partition. The vertex set of a component is the set of end vertices of its edges.

- A graph is **biconnected** when its edge set can be partitioned into more than one equivalence class by an equivalence relation ~. Two edges are related by ~ iff there is a cycle containing both edges, [Aho,Hopcroft,Ullman 74]. The relation ~ partitions the edge set. The edges in each such set and the set of vertices of these edges forms a connected subgraph. Each subgraph is a **biconnected component** of the graph.

- **Bridges** are defined by a cycle. The bridges and the cycle are components of a partition of a graph, and the vertex sets of the bridges less the vertex set of the cycle are disjoint. Bridges are made up of disjoint paths. A graph can be decomposed into a set of disjoint paths. With the proper choice of paths and bridges it is possible to partially order the bridges under the set inclusion relation.

- The **attachments** of a bridge is the set of vertices defined by the intersection of the vertex set of the cycle and the vertex set of the bridge. The set of attachments are ordered according to their relative positions in the cycle.

- A **drawing** of a graph is any drawing of the graph on a plane.

- The **cyclic order** of a vertex in a drawing is the order of edges incident on the vertex in the counter-clockwise direction.

- A **plane embedding** is an embedding of a graph in a plane such that no

two edges cross in a drawing. The graph is said to be a **plane graph(subgraph)**.

- A **planar** graph has a plane embedding.

- An **embedding** of a graph is a set of plane subgraphs that constitutes a partition of the graph.

- A **region** of a plane graph is a maximal portion of the plane for which any two points may be joined by an arc such that any point of the arc neither corresponds to a vertex of the graph nor lies on an arc corresponding to an edge of the graph.

- **Interior regions** are bounded regions.

- The **exterior region** is unbounded.

- An **interior cycle** encloses an interior region.

- The **exterior cycle** is the perimeter of a plane graph.

- Two plane graphs of the same graph are **equivalent** if there is a bijective mapping of the interior cycles between the two plane graphs, and the exterior cycles are identical.

It is proven [Burstein 80] that two plane graphs of the same graph are equivalent if they have a common exterior cycle, and for each vertex, the cyclic order of the vertices of one plane graph are identical to the cyclic order of the corresponding vertices of the other plane graph.

- Equivalent plane graphs have the same **plane topology**. A plane topology of a plane graph is totally specified by the graph, the exterior

cycle, and the set of cyclic order of vertices of the graph.

- Two graph embeddings are **equivalent** if for each plane, the plane subgraph of one embedding is equivalent to the corresponding plane subgraph of the other embedding.

- Equivalent graph embeddings have the same **embedding topology**. The embedding topology of an embedded graph is totally specified by the graph, and for each plane, the plane topology.

- The **left side** of an embedded cycle is the region to the left of an edge traversed in the direction of the cycle. The **right side** is defined accordingly. The region on the left side of an embedded cycle is the **left region**. Correspondingly, the right side defines the **right region**.

- The **clock-wise** or **counter clock-wise orientation** of an embedded cycle is defined by the direction of the cycle.

- The left side of an embedded counter-clockwise cycle is the **inside region**. The right side of an embedded counter-clockwise cycle is the **outside region**. For an embedded clockwise cycle the left region is the outside region, and the right region is the inside region.

- A **path tree** is a labeled directed tree.

A vertex corresponds to a path in the circuit graph. The paths in the path tree are disjoint. Vertices are labeled with the identification of the path. The number of vertices equals the number of disjoint paths. Each subtree represents a bridge composed of the paths defined by the vertices of the subtree. The subtrees of vertex i represent the bridges of

the cycle formed by the path i and a path from some previous cycles.

- A **left oriented bridge** is a bridge of an embedded graph that is confined to the left region of a cycle, has its attachments adjacent to the perimeter of the bridge embedding, and with the same order of its attachments at the perimeter as the order of the attachments in the cycle.

- A **right oriented bridge** is defined correspondingly.

- The **orientation constraint graph** is an undirected, labeled graph. Each bridge is represented by a vertex in this graph. Vertices are labeled with the identification of the bridges they represent. Each constraint on the orientation of bridges is represented by an edge. There are two types of edges; **s-edges**, and **o-edges**. An s-edge implies that the bridges represented by its end vertices are constrained to have the same orientation. An o-edge implies that the bridges represented by its end vertices are constrained to have opposite orientations in order that the bridges will not cross.

- A **2-coloring** of the orientation constraint graph is a coloring using two colors such that the end vertices of an s-edge have the same color, and the end vertices of an o-edge have different colors. A graph is planar when its orientation constraint graph can be 2-colored.

- A **compensation graph** is a labeled, undirected graph. There is a compensation graph for each edge in the orientation constraint graph. Each disjoint path of the graph being analyzed is represented by a distinct vertex in the compensation graph. A vertex is labeled with the

path it represents. Two paths represented by adjacent vertices would cross each other in drawings if the orientation assignment conflicts the orientation constraint (edge), which the compensation graph represent.

- A **total compensation graph** is the union of the vertex sets and the union of the edge sets of those compensation graphs that corresponds to edges in the orientation constraint graph for which the orientation assignment of bridges conflicts the assignments defined by the edge.

The total compensation graph can be n-colored. A n-coloring of a graph implies that adjacent vertices are assigned different colors. The coloring of the vertices in the total compensation graph assigns paths to planes. The coloring of the total compensation graph corresponds to a partition of the graph. The coloring of the orientation constraint graph specifies a planar embedding of the subgraphs in the partition. Together the two graphs specify an embedding of the circuit graph.

- A **horizontal(vertical) position constraint graph** is a directed, acyclic graph. A vertex in the position constraint graph represents the vertices of the circuit graph that, according to the embedding, (or an explicitly given constraint) must have the same position. An edge between vertex A and its son B with weight C implies that B must be at least C units away from A in the drawing of the circuit graph.

We assign positions to the vertices of the position constraint graph such that the size of the embedding is minimized.

- A **layout topology graph** is a directed, labeled graph. It is derived from

the circuit graph. It has one exterior vertex, nogate vertices, and transistor vertices. A transistor vertex has 4 edges, 2 for the gate of a transistor, and one for each of drain and source. Vertices have cyclic order.

- A **layout topology** is an embedding topology of the layout topology graph.

CHAPTER 2

Circuit to Graph Transformation

2.1. Introduction

The problem of circuit embedding and the problem of graph embedding have many similarities. A transistor can be identified with a vertex in a graph, a wire with an edge. A physical layer corresponds to a topological manifold. The cyclic order of wires to a device corresponds to the cyclic order of edges to a vertex. Space exterior to a layout can be interpreted as the exterior face of a graph. The problem of circuit embedding is equivalent to the problem of graph embedding.

A graph model of a circuit should reflect the circuits topological characteristics. Goldstein and Schweikert [Goldstein,Schweikert 73] have proposed a model for testing the planarity of electrical circuits. Goldstein-Schweikert's model has one vertex for each circuit component, and one vertex for each net. In their model, vertices representing circuit components are called **component vertices** and nets, **net vertices.** A net vertex with connections to k different components will have k incident edges. There will be an edge between a net vertex and each of the vertices representing the components with terminals belonging to this net.

VanCleemput [VanCleemput 76] has proposed a more extensive graph model for the circuit layout problem. A net is represented by a vertex as in Goldstein-Schweikert's model. The model for components is more complex. A component is represented by a partially oriented graph, a **component graph.** A **partially oriented graph** is a graph of one cycle. The vertices in the cycle

represent the terminals of the component. The cyclic order of the terminals is reflected in the cyclic order of the vertices in the cycle. The component graph model may be modified to specify the physically or logically equivalent terminals. Terminals that are physically equivalent are equipotential. A connection can be made to any one of these terminals. Terminals that are logically equivalent have identical logical functions. They may be interchanged to obtain a better layout.

Our primary interest is to provide a graph model for circuits with only MOS transistors. In either model, there are cases that a planar circuit has a non-planar graph model. An example of such a case is shown in Figure 2.1.1a, Figure 2.1.1b, and Figure 2.1.1c. The ports are located along the perimeter of the layout.

2.2. Topological Characteristics of MOS Circuits

A MOS transistor is electrically a three terminals device. It has four terminals physically; the drain terminal, the source terminal, and two gate terminals. The drain terminal and the source terminal are logically equivalent and the two gate terminals are physically equivalent. Any component of a chip with at least three terminals is usually oriented, i.e., the cyclic order of the terminals is prescribed. A MOS transistor, even with all four terminals connected to other MOS transistors, is not oriented because of the equivalences of its terminals.

With the current two dimensional MOS technology, circuit layouts can be adjacent to each other, but not on top of each other. Ports provide connection points between circuit layouts. The cyclic order of ports at the perimeter may be prescribed. The graph model must have provisions for identifying

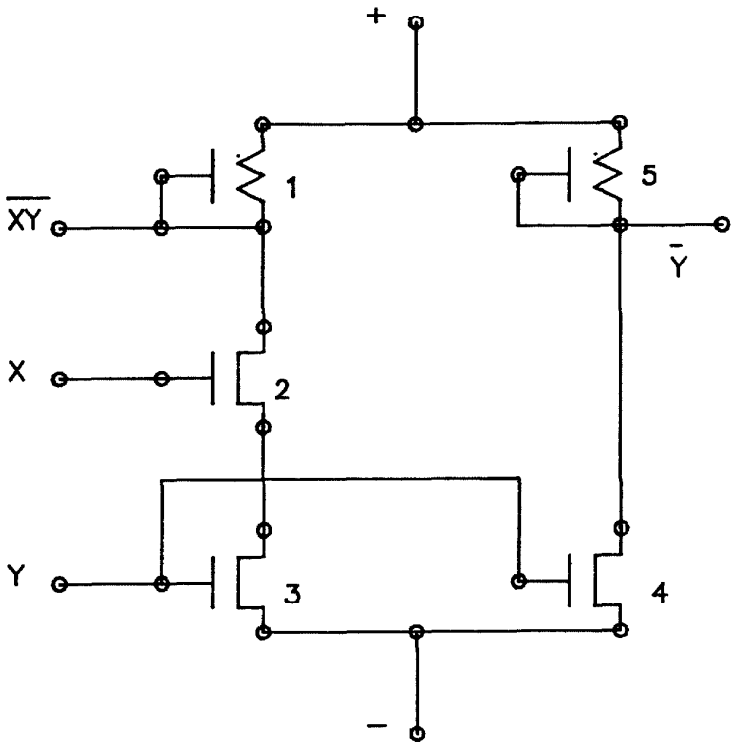


Figure 2.1.1a A Planar Circuit.

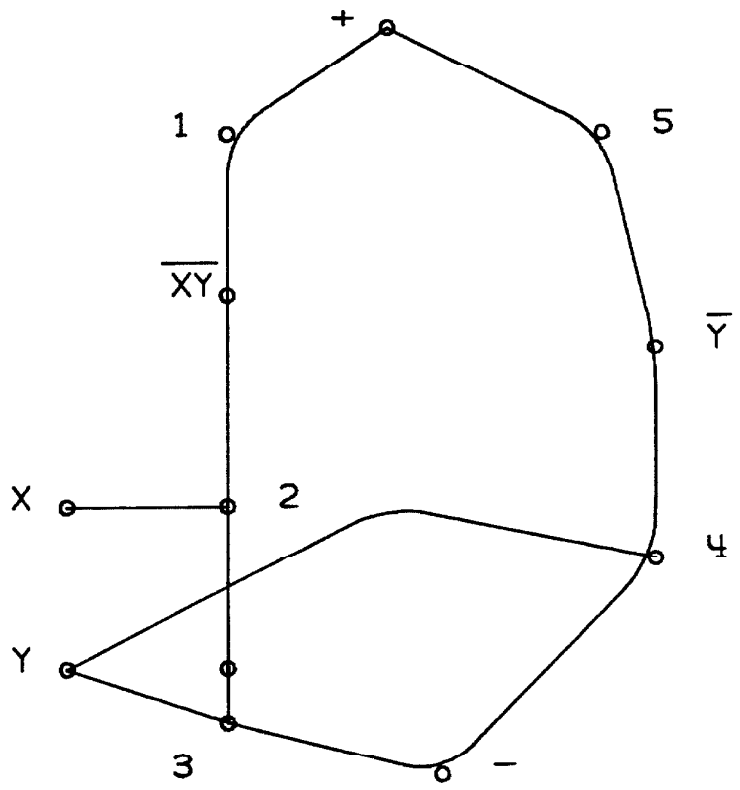


Figure 2.1.1b Goldstein-Schweikert's Model of the Planar Circuit.

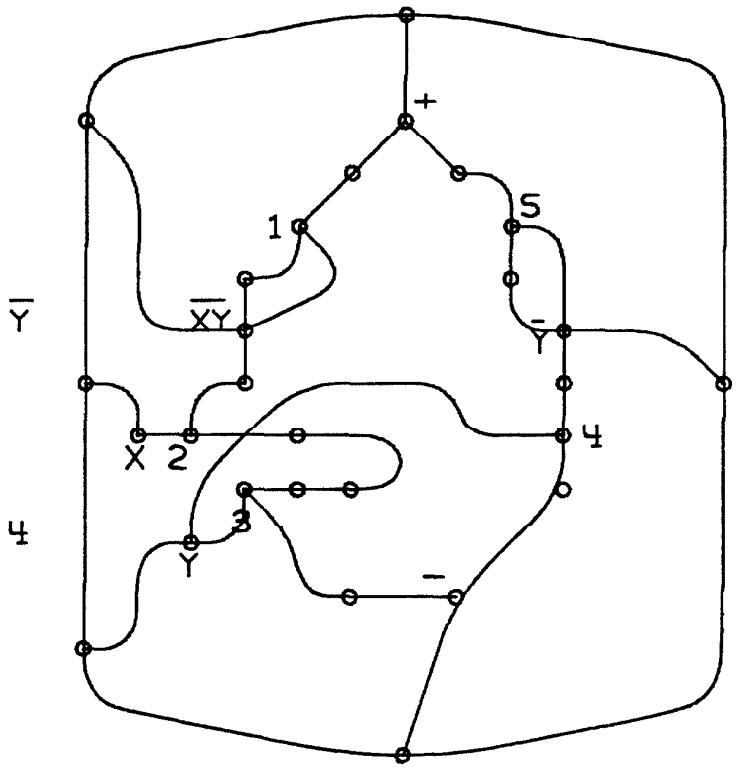


Figure 2.1.1c VanCleemput's Model of the Planar Circuit.

ports and the specification of the prescribed cyclic order. This information is needed for the computation of an optimal layout topology.

2.3. A Graph Model for MOS Circuits

The appropriate model for a net is a vertex as proposed by Goldstein and Schweikert. This appears to be the only feasible solution not requiring the enumeration of all possible combinations of net decompositions.

A MOS transistor is not a complex component. It is not necessary to model transistors explicitly. The transistor type is irrelevant in deriving the layout topology except in the process of presenting a layout topology as a drawing. The types of every transistor in the circuit graph are kept in a separate table. All MOS transistors are topologically identical. A nMOS enhancement mode transistor is defined by a poly wire crossing a diffusion wire. A depletion mode nMOS transistor is created by "covering" the poly diffusion crossing by implant, a separate mask.

As mentioned above, a MOS transistor is electrically a three terminals device, but it has four physical terminals. The two gate terminals are physically equivalent. It is difficult to determine in advance whether the implementation of a MOS transistor with one gate terminal or two gate terminals will lead to the better layout. Besides, even if it can be determined in advance that the two gate terminals implementation will lead to the better layout, it is still difficult to partition optimally the connections to the gate into two subsets, one subset for each terminal. The circuit in Figure 2.1.1 illustrates that the layout would be planar if transistor 3 is implemented with two gate terminals. The layout of the circuit of Figure 2.3.1 becomes nonplanar when transistor 2 is implemented with two gate terminals. Hence, it is hard to determine whether a MOS transistor should be implemented with one or two gate terminals.

It is desirable to have only one graph model for a given circuit, i.e., electrically equivalent circuits shall have the same circuit graph. Otherwise, a number of graphs that grows combinatorially must be explored to find the best layout. The assignment of terminals of a circuit component (three terminals) to the terminals of a physical component (four terminals) is not

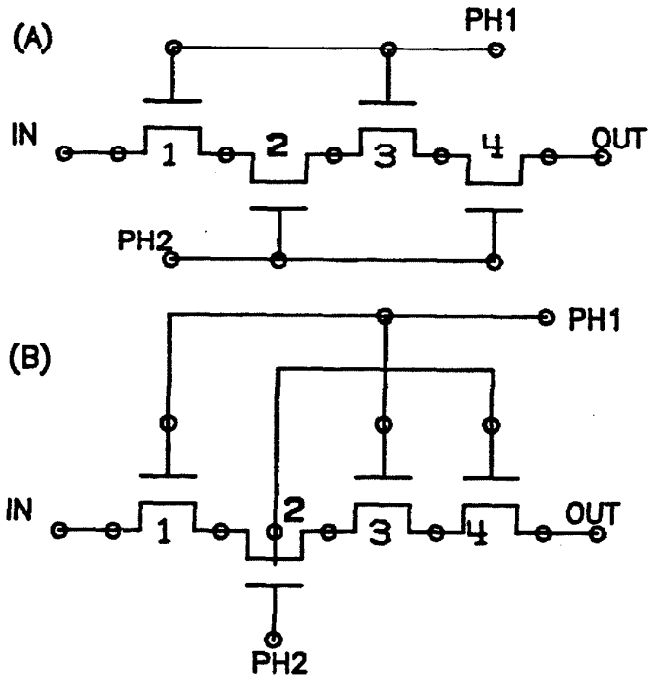


Figure 2.3.1 Non-Planarity Caused by Using 2 Gate Terminals.

trivial in the presence of logically and physically equivalent terminals. Goldstein-Schweikert's graph model depends on the actual number of physical terminals of the MOS transistor. VanCleemput's model is independent of the actual number of physical terminals of the MOS transistor, and so is ours. Figure 2.3.2 shows another circuit topology of the circuit shown in Figure 2.1.1. Goldstein-Schweikert's model of this circuit topology is different from the one shown in Figure 2.1.1. VanCleemput's model remains the same. Our model is also identical for both cases.

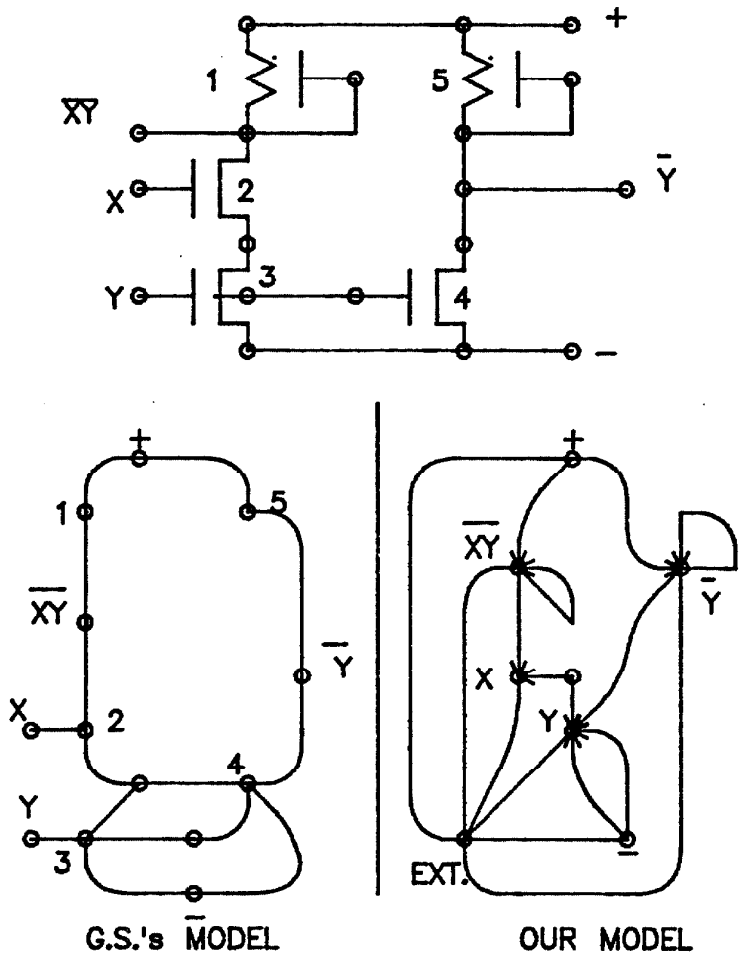


Figure 2.3.2 Another Circuit Topology/Models for the Circuit of Figure 2.1.1.

Our transistor model also resolves the difficulty demonstrated in Figure 2.1.1. The two gate terminals of a transistor, besides being physically equivalent, create a "free" cross-over when it is needed. This "free" cross-

over property is not modeled in VanCleemput's model. Our model implicitly defines the "free" cross-over. This capability is the reason for discarding VanCleemput's model.

The perimeter of a layout has properties similar to a vertex in a graph. A vertex of an embedded graph is accessible from all its adjacent regions. This property is also true for the perimeter of a layout. Hence it is desirable to map the perimeter of the layout to a vertex. In our model this map is accomplished by introducing the exterior vertex. The example shown in Figure 2.3.2 has a vertex identified as the exterior vertex. The prescribed order of the ports can be represented as constraints in the cyclic order of the exterior vertex.

It is not possible to specify the prescribed cyclic order of ports and the requirement of locating ports along the perimeter of the layout with Goldstein-Schweikert's model. VanCleemput models the exterior face of the layout by a cycle. The vertices on the cycle represent the ports. All nets and components must be embedded in the interior region of the cycle. Our model is equivalent to that of VanCleemput's model. We choose our special representation to suit our graph embedding algorithm that will be described in Chapter 3.

2.4. Summary of the Graph Model for MOS Circuits

The exterior face of a layout is represented by the exterior vertex. This vertex has a prescribed cyclic order if there is a prescribed order of ports on the perimeter of the layout. All other vertices are net vertices. There are two types of net vertices, gate and nogate. Nogate vertices have no constraint on the cyclic order. Since transistors are defined implicitly by the source and drain edges incident on the gate vertex, there is a constraint imposed on the

possible cyclic order to the gate vertex. The constraint follows from the electrical characteristics, as discussed in Chapter 3. All vertices are labeled with labels provided in the description of the circuit schematic. Source and drain edges are directed. Every pair of source and drain edges together with the gate vertex on which they terminate defines a transistor. The source and the drain edges are labeled with the transistor they represent as well as the label of the net vertices from which they emanate. Port edges are labeled with the name of the port they represent.

Both Goldstein-Schweikert's model and VanCleemput's model would map each transistor into an unique vertex. The mapping from circuit graph embedding topology to layout topology would be simpler for their models. Transistors in our model are defined implicitly. Since transistors are simple components and they are topologically identical, it is easy to build the transistors by rules embedded in the transformation algorithm. A procedure is described in Chapter 4.

2.5. Circuit Description

We specify a circuit schematic in an informal BNF. Terminals are in lower case. All terminals must be separated by blanks or end-of-lines. {} denotes repetition any number of times including zero; | denotes "or"; and () denotes grouping. Upper and lower case distinctions are ignored.

```
CIRCUIT = { comment } CELL BODY
CELL = HEADER
HEADER = c CELLNAME ;
CELLNAME = name
BODY = PORTS TRANSISTOR END
PORTS = e { ( LAYER ) PORTNET } { ORDER } ;
LAYER = d | p | m
PORTNET = net
ORDER = net
TRANSISTOR = ( T-TYPE ) T-NAME L-W-RATIO SOURCE DRAIN GATE ;
T-TYPE = n | p | d
T-NAME = name
L-W-RATIO = number
GATE = net
SOURCE = net
DRAIN = net
END = .
```

Ports and transistors are specified explicitly. Ports and terminals of transistors that have the same "net identification" belong to the same net. The layer assignment of ports can be specified. Contacts that appear in physical layouts have no logical function and are not specified in circuit schematics. Contacts are introduced later in the layout process. If the order of the ports is prescribed, then they must be listed in the counter-clockwise orientation.

2.6. A Circuit to Graph Transformation Algorithm

A fully labeled, partially ordered and partially directed graph requires the label of all vertices to be specified, the order of ordered vertices to be given, as well as all the edges and the direction of directed edges. To com-

plete the specification of our circuit graph labels are assigned to edges.

The transformation from a circuit schematic to a circuit graph as described in the previous sections is carried out by the algorithm outlined below.

Step 1: Create a vertex for each net. Label it with the name of the net specified by PORTNET, GATE, SOURCE, or DRAIN in the description of the circuit schematic.

Step 2: Add two edges for each transistor, one represents the drain, the other represents the source. Assume that the gate belongs to net G, the drain is connected to net D, and the source is connected to net S. The drain edge has its head on the gate vertex, G, and its tail on vertex D. The source edge has its head on G and its tail on vertex S. Label the edges with the transistor they represent, and the label of the vertex from which they emanate (D and S respectively).

Step 3: Create an edge for each port. One end vertex is labeled "exterior". The other vertex is labeled with the net connected to the port.

The cyclic order of gate vertices is initially given by the order in which transistors and ports are read, i.e., the order is arbitrary. However, drain and source edges always appears in pairs in the cyclic order of gate vertices.

An exclusive OR circuit is coded in our circuit specification language. The specification is shown in Figure 2.6.1 together with the corresponding circuit graph.

```
C XOR
E M + M - P A P B P OUT;
N TR1 1 - N1 A;
N TR2 1 - N1 B;
D TR3 1 N1 + N1;
N TR4 1 - OUT N1;
D TR5 1 OUT + OUT;
N TR6 1 - N2 A;
N TR7 1 N2 OUT B;
```

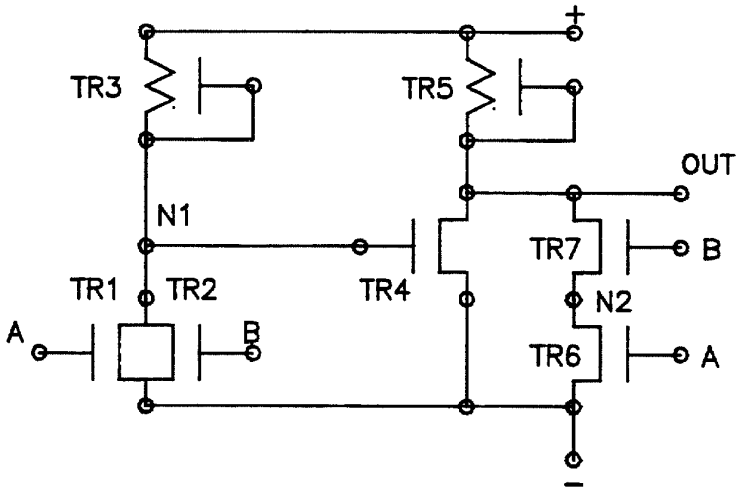
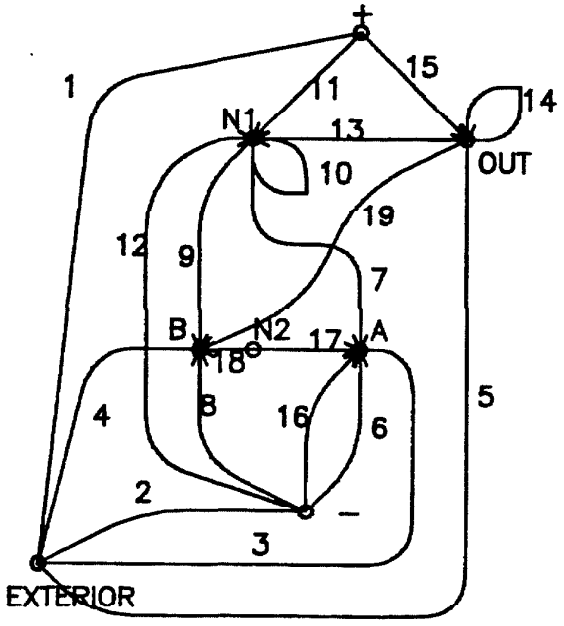


Figure 2.6.1a Circuit Specification of an XOR.



TRANS.	S.	D.
TR1	6	7
TR2	8	9
TR3	10	11
TR4	12	13
TR5	14	15
TR6	16	17
TR7	18	19

Figure 2.6.1b A Graph Model of an XOR.

2.7. Algorithm Complexity

The complexity of the circuit to graph transformation algorithm is linear in the number of transistors and the number of ports, i.e., linear in the number of edges of the circuit graph.

2.8. Remarks

A common configuration of MOS transistors is transistors connected in series, with intermediate nodes in the circuit graph not accessed. The gate

connections may be permuted without changing the function of the circuit. It is desirable to take advantage of the interchangeability of the gate connections to improve the circuit layout.

A direct application of Goldstein-Schweikert's model, or VanCleemput's model, to a circuit schematic containing a series transistor configuration would imply that all transistors are represented explicitly, and that a unique circuit graph is obtained for every permutation of the gates. However, it is desirable to represent all such permutations with a single circuit graph.

The series transistor configuration can be viewed as a single component with multiple gates. Applying VanCleemput's model to such a component results in only one graph for all permutations of the gates, as desired. Nevertheless, VanCleemput's model is discarded since it does not capture the "free" cross-over capability.

Our model also allows the serial transistor configuration to be modeled as a multi-gates transistor. Such a transistor would have two edges incident on a vertex representing all the gate nets. The gate net vertex becomes the gate "nets" vertex.

The multi-gates transistor model is useful if there is an efficient way of decomposing gate nets vertices. An efficient algorithm for this decomposition has not yet been found. Therefore, multi-gates can not be specified in our circuit description notation. A simple change of our BNF is needed, should an efficient decomposition algorithm be found.

CHAPTER 3

An Application of the Graph Model - the Embedding Problem

3.1. Introduction

The proposed circuit graph model is the same for electrically equivalent circuits. The graph embedding topology can be derived from the model by any graph embedding algorithm. The embedding topology can then be expanded into a unique layout topology. In this chapter a graph embedding algorithm is described. This graph embedding algorithm treats all edges as undirected. The expansion from graph embedding topologies to layout topologies is described in Chapter 4.

The major reason for modeling a circuit as a graph is that the problem of embedding a circuit in a set of physical layers can be mapped into the problem of embedding a graph in a set of planes. Each plane in the graph embedding may be equivalent to a layer in the circuit embedding. However, the embedding algorithm use an arbitrary number of layers. The mapping to physical layers, described in Chapter 4, is carried out in the generation of a physical layout. The general graph embedding problem is NP-complete. A problem related to graph embedding is the problem of the determination of the genus of a graph. The genus of a graph is the number of handles needed on a sphere to embed the graph. A planar graph is of genus 0.

The planar graph embedding problem is the least complex embedding problem. It is also a hard problem. But, there exist linear time algorithms for planarity testing [Hopcroft,Tarjan 74, Rubin 75]. We will first describe this algorithm, then our extensions of the algorithm to make it into an algorithm

that can be used as a first step in the generation of a graph embedding topology. Next we generate a partitioning of the graph into a set of planar subgraphs. The partitioning is obtained by coloring graphs derived from the extended planarity test algorithm.

3.2. Planarity Testing

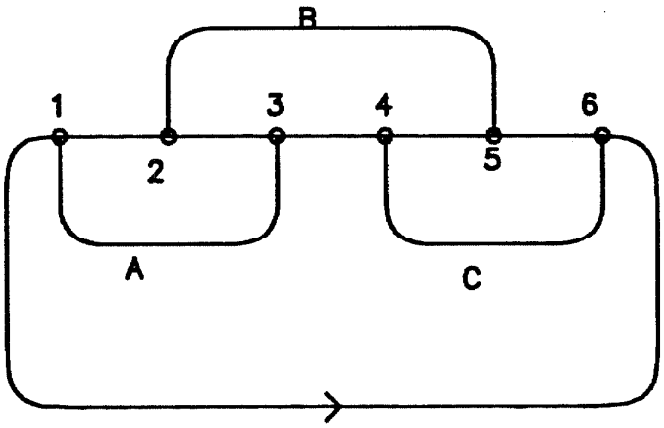
A powerful technique used in planarity testing of a graph is the systematic embedding of edges of a graph. When a planar embedding is found, then the graph must be planar. A graph is non-planar when a planar embedding can not be found. If at some stage of the embedding process, the embedding is found to be non-planar, then there is no need to continue this search path. The bi-product of this testing is a planar embedding for a planar graph.

A graph is planar when all its biconnected components are planar. The testing algorithm may separate the graph into components. The planarity test can then be applied to each component independently. There exist linear time algorithms to partition a graph into its biconnected components [Aho,Hopcroft,Ullman 74]. Without loss of generality the graph is not biconnected.

The testing algorithm is invoked recursively. The first step is to find a cycle in a subgraph. In the initial call the subgraph is the graph. A cycle defines a set of bridges. Bridges must be placed either in the inside region or the outside region of the embedded cycle for the graph to be planar. A pair of **incompatible bridges** are two bridges that will cross when both are assigned to the same region. Incompatible bridges have attachments that interleave in the cycle. Testing for pairwise incompatibility is simple. The example in Figure 3.2.1 helps to illustrate the technique. In Figure 3.2.1

bridge A and bridge B are incompatible. Bridge B and bridge C are incompatible also. The assignment of bridge A and bridge C to the inside region, and bridge B to the outside region resolves the incompatibility. A graph is nonplanar when incompatible bridges cannot be assigned to different regions.

If a planar embedding of a bridge is found it remains to verify that such an embedding together with the original cycle is also planar. This requirement implies that the attachments of the bridge must be adjacent to the



ATTACHMENTS OF A (3,1)
B (5,2)
C (6,4)

Figure 3.2.1 Pairwise Incompatibility Test.

perimeter of the bridge embedding. Furthermore, the order of the attachments of the bridge must agree with the order of the attachments in the cycle. If these conditions are not fulfilled then the bridge embedding is invalid. When a planar embedding of the bridge that will also pass this second test can not be found, then the graph is non-planar.

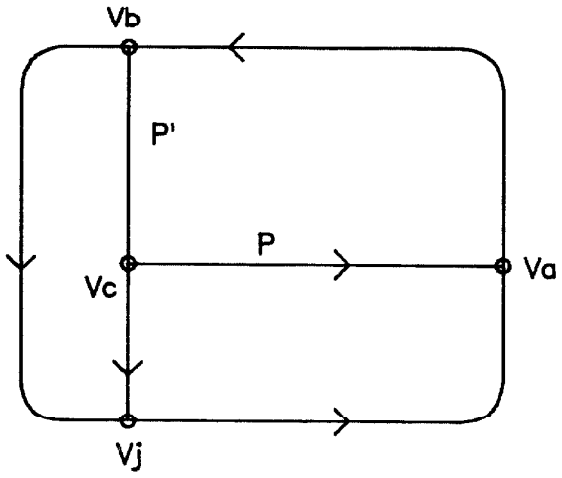
This planarity testing technique is proposed by Auslander,and Parter [Auslander,Parter 61]. Their algorithm is corrected and improved by Hopcroft and Tarjan [Hopcroft,Tarjan 74]. Hopcroft-Tarjan's algorithm runs in linear time.

3.3. Hopcroft and Tarjan's Planarity Testing Algorithm

A new cycle must be found in the subgraph to be tested in each recursive call of the algorithm. The new cycle C can be made up of a path from the previous cycle, C', plus a set of untraversed edges forming a path P with its end vertices attached to the previous cycle. P is part of a bridge of C'. The orientation of B is the same as the orientation of P.

The cycle C' consists of a path P' plus a path of edges in the previously found cycle C''. This situation is illustrated in Figure 3.3.1. The orientation of the new cycle C is a function of the orientation of C', the orientation of P, the last vertex of P, and the last vertex of P'. The table in Figure 3.3.2 defines the rules for determining the orientation of C.

P is assigned to the left region of C'. The algorithm then searches for the bridges of C' that have left orientation and that are incompatible with P. The algorithm attempts to resolve the incompatibility by changing the orientation assignment of the bridges of C'. When a compatible assignment is found then the algorithm will invoke itself to establish a new cycle from C. When a compatible assignment is not found then the algorithm halts and declares



CYCLE C	Va,Vb,Vc,Va
CYCLE C'	Vj,Va,Vb,Vc,Vj
CYCLE C''	Vj,Va,Vb,Vj
PATH C1'	Vj,Vb

Figure 3.3.1 Relations of Cycles and Paths.

the graph non-planar.

The drawing in Figure 3.3.1 is used as a reference. After all the bridges of C' are explored the algorithm will check whether these bridges with their assigned orientation can be combined with C'' and remain planar. This test is the **upward compatibility test.** Each bridge of C' with attachments in C1' except the end vertex Vj must have the same orientation as P'. This type of bridge needs upward compatibility. The algorithm changes the assigned orientation of the bridges of C' such that the upward compatibility is

```
CW  = CLOCKWISE
CCW = COUNTER CLOCKWISE
CASE(1):
   LAST VERTICES OF P AND P' ARE THE SAME
   SET ORIENTATION OF C TO C'
CASE(2):
   LAST VERTICES OF P AND P' NOT THE SAME,
   ORIENTATION OF P = LEFT =>
         SET C TO CCW
   ORIENTATION OF P = RIGHT =>
         SET C TO CW
```

Figure 3.3.2 Rules for Determining the Orientation of a Cycle.

maintained and the incompatible bridges are assigned with different orientation. When such an assignment can be found the algorithm returns to the calling procedure. When such an assignment can not be found, the graph is declared non-planar.

A depth-first search would divide the graph into paths that may be assembled into the cycles needed for planarity testing. In order for the planarity test to be efficient, the adjacent lists are specially ordered such that the needed paths have certain important properties. These properties

are outlined in [Hopcroft,Tarjan 74]. With these special properties the tests of incompatibility and upward compatibility become efficient.

3.4. Extensions of Hopcroft-Tarjan's Algorithm for Graph Embedding

Hopcroft-Tarjan's algorithm is invoked recursively. This invocation scheme implicitly defines a hierarchy of paths. The hierarchy is captured in a path tree. The path tree is directed. Each vertex is labeled with the identification of a path, i.e., the path explored during each recursive invocation of the algorithm. Hence each vertex represents a path that is coming off the cycle formed by the path represented by its father plus a path in the previous cycle. The subtrees of a vertex i represent the bridges of the cycle formed by the path represented by vertex i and a path in the previous cycle.

Hopcroft-Tarjan's algorithm attempts to assign orientations to the bridges of a cycle such that pairwise incompatible bridges have different orientation. The algorithm halts when such an assignment can not be found. In our extension to the algorithm it continues to run, but pairwise incompatibility between pairs of bridges is captured in the orientation constraint graph. In this graph each bridge is represented by a vertex. When two bridges are pairwise incompatible, then a type o-edge is added to the orientation constraint graph. The end vertices are the vertices that represent the two incompatible bridges.

The extended algorithm also generates a compensation graph for each o-edge of the orientation constraint graph. A vertex is created for each disjoint path of the graph being analyzed. Edges are introduced between vertices representing paths that will cross if the bridges related by the o-edge are assigned the same orientation.

After all the bridges of the current cycle are explored Hopcroft-Tarjan's algorithm will do the upward compatibility test as described earlier. The bridge B that needs to be upward compatible is the bridge that must be assigned the same orientation as the orientation of a previously defined path P' as described earlier. The relation of upward compatibility is between a path and a bridge. The orientation of path P' is the orientation of the bridge with P' as the first path. Hence the relation of upward compatibility is also a relation between two bridges. This relation is captured in the same orientation constraint graph. A type s-edge is added to the orientation constraint graph for each bridge that needs to be upward compatible. The end vertices are the vertices that represents bridge B and the bridge with P' as the first path.

A compensation graph is also created for each s-edge in the orientation constraint graph. A vertex is created for each disjoint path of the graph being analyzed. Paths represented by adjacent vertices will cross when the bridges represented by the end vertices of the s-edge are not assigned the same orientation.

In our application of the extended Hopcroft-Tarjan algorithm the initial cycle is always restricted to be a cycle that contains the exterior vertex. Furthermore, the the exterior vertex is always taken to be the first vertex of the first cycle. This particular choice of the initial cycle guarantees that the exterior vertex will be adjacent to the exterior phase in the embedding of the graph. The exterior vertex must be adjacent to the exterior phase in order that a layout can be generated from the embedding.

Figure 3.4.1 is used to illustrate the effects of our choice of initial cycle. The graph has 4 vertices numbered 1,2,3,4. Let the initial cycle C have the

direction 1,2,3, and 4. C has a counter-clockwise orientation. If the path P between 2 and 4 is assigned to the inside region, then vertex 1 is adjacent to the exterior phase. If P is assigned to the outside region, then vertex 1 is still adjacent to the exterior face, because P and a path segment of C must form a cycle with clockwise orientation. The newly formed cycle will not enclose vertex 1.

A planar graph, its paths, its path tree, its orientation constraint graph, and its set of compensation graphs are shown in Figure 3.4.2. The complete graph of six vertices is non-planar. It is shown in Figure 3.4.3 along with its path, its path tree, its orientation constraint graph, and the set of compensation graphs.

In summary, the extended Hopcroft-Tarjan's algorithm creates the path tree, the orientation constraint graph, and the compensation graphs.

3.5. General Graph Embedding

In order to obtain an embedding topology it remains to determine the orientation of bridges, and the plane assignment of paths. The cyclic order of vertices is also needed to completely specify an embedding topology. The cyclic order of vertices in the circuit graph is implicitly specified by the hierarchy of paths as defined by the path tree and the orientation assignment of bridges. The plane assignment of paths partitions the circuit graph into planar subgraphs. Hence, specifying the orientation of bridges and planes of paths suffice to complete the specification of an embedding topology.

The orientation assignment of bridges and plane assignment of paths is treated as a coloring problem of the orientation constraint graph and the compensation graphs. The vertices related by an o-edge must be assigned

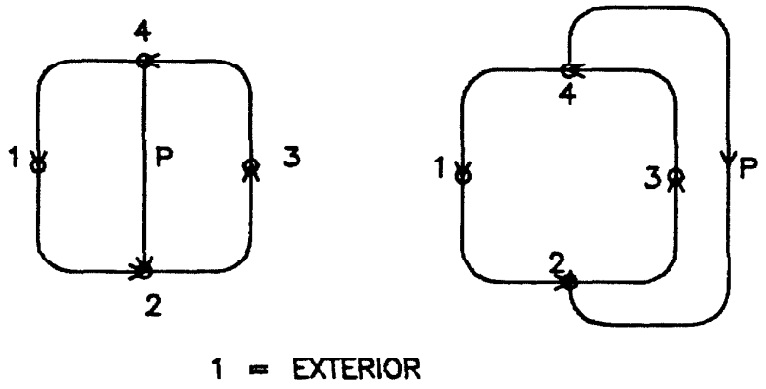
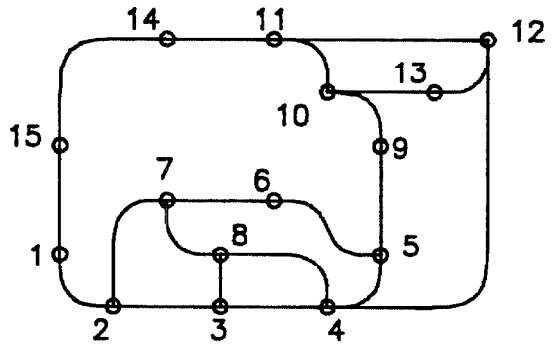


Figure 3.4.1 The Initial Cycle and Exterior Vertex.



PATHS:

C1:	1,2,3,4,5,9,10,11,14,15,1
P2	11,12,4
P3	12,13,10
P4	5,6,7,2
P5	7,8,3
P6	8,4

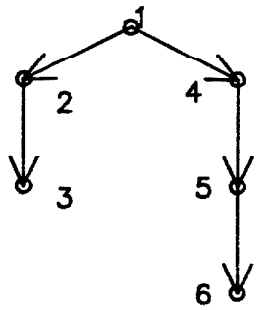


Figure 3.4.2a A Planar Graph, its Paths and Path Tree.

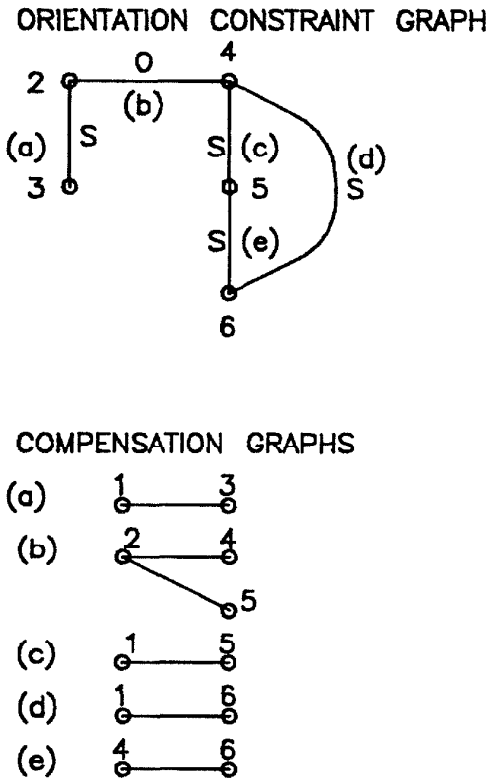
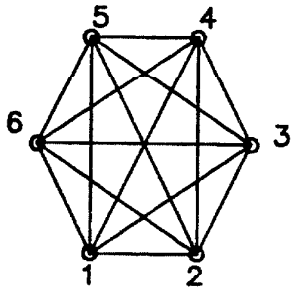
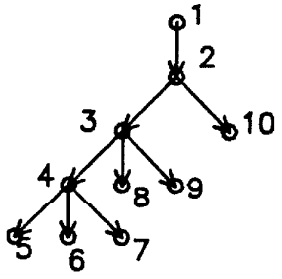


Figure 3.4.2b The Orientation Constraint/Compensation Graph.



PATH TREE



PATHS:

C1: 1,2,3,1

P2: 3,4,1

P3: 4,5,1

P4: 5,6,1

P5: 6,2

P6: 6,3

P7: 6,4

P8: 5,2

P9: 5,3

P10: 4,2

ORIENTATION CONSTRAINT GRAPH

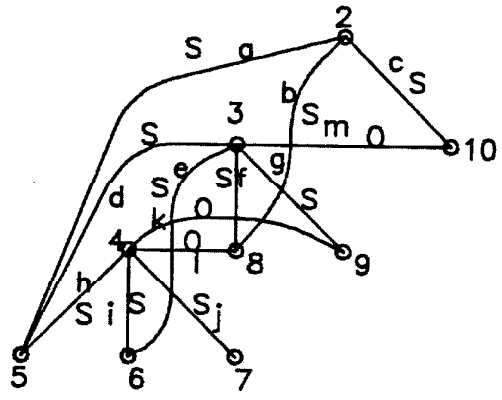


Figure 3.4.3a K6, its Paths, Path Tree, Orientation Constraint Graph.

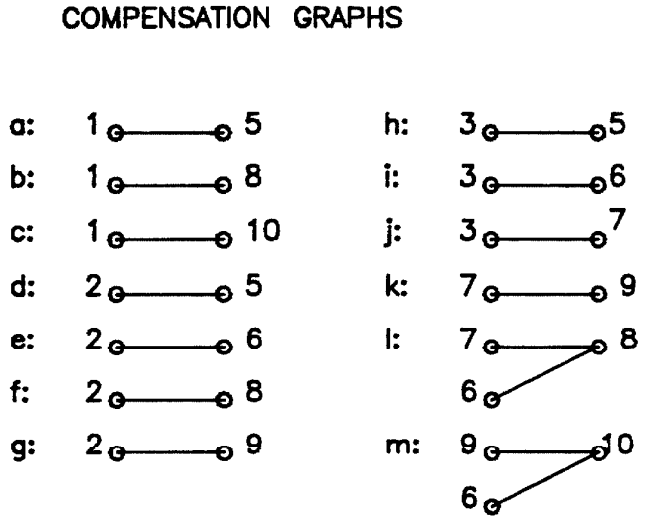


Figure 3.4.3b The Compensation Graphs for K6.

different colors. The vertices related by a s-edge must be assigned the same color. When the constraint graph can be 2-colored according to the above rules, the graph is planar. The orientation constraint graph of the planar graph shown in Figure 3.4.2 can be 2-colored. The coloring is shown in Figure 3.5.1. The two colors used are L and R that represent the left orientation and the right orientation respectively. The orientation constraint graph of the complete graph of six vertices can not be 2-colored.

The general graph embedding algorithm proceeds in the following steps:

1. Color the orientation constraint graph with two colors. Obtain a list of orientation constraints that are not met. A s-edge constraint is not satisfied when the bridges represented by the end vertices of the s-edge are assigned with different orientation. An o-edge constraint is not satisfied when the bridges represented by the end vertices of the o-

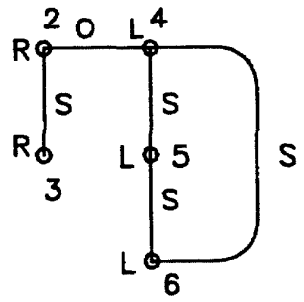


Figure 3.5.1 Orientation Assignment for the Graph of Figure 3.4.2.

edge are assigned with the same orientation.

2. Generate a total compensation graph as the union of the edge sets, and the union of the vertex sets of the individual compensation graphs of the unsatisfied orientation constraints. Color the total compensation graph. Adjacent vertices must be colored with different colors. There is no constraint on the number of colors allowed.

The vertices of the total compensation graph represent paths. A non-planar graph can be embedded in two planes if paths can make transitions between planes. If an entire path is constrained to lie within a single plane, then an arbitrary number of colors may be needed for the embedding of the graph. Coloring of the total compensation graph corresponds to this situation. Allowing an arbitrary number of colors in the coloring of the total com-

pensation graph guarantees that an embedding can always be found. The mapping on to the layers available in the technology being used is made in the generation of a physical layout by introducing contacts.

Note: The edges of the circuit graph are by the plane assignment procedure assigned to unique planes in their entirety, but vertices in the circuit graph are, in general, shared between several different paths and hence often assigned to several planes.

The coloring of the orientation constraint graph and the corresponding total compensation graph can be carried out in many ways. There are several embedding topologies for each path tree and orientation constraint graph. One heuristic algorithm is the following:

Start by assigning an orientation of bridges to satisfy the s-edge constraints. Then assign orientations to the unassigned bridges such that the number of unsatisfied o-edge constraints is small. Build a total compensation graph according to step 2. Assign most of the paths to the first plane. When the graph is a circuit graph, and the first plane(layer) is the diffusion/poly plane, then this heuristic tends to minimize the number of contacts. The poly and diffusion layers in the layout are topologically related and are therefore considered as one plane. A poly wire must not cross a diffusion wire. Figure 3.5.2 shows a valid orientation and plane assignment for the non-planar graph shown in Figure 3.4.3.

In our implementation the cyclic order implied by the path tree and the orientation assignments is made explicit by the drawing algorithm described in Chapter 4.

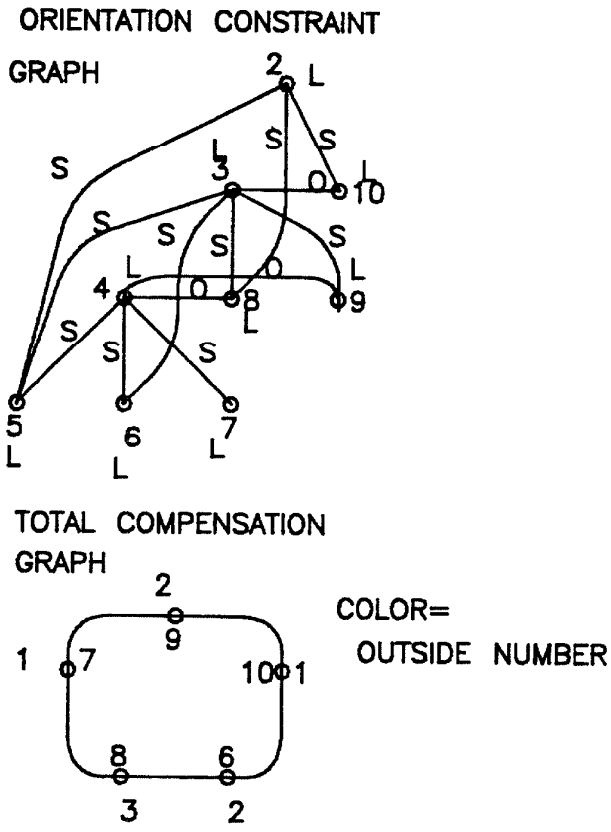


Figure 3.5.2 Orientation/Plane Assignment for the Graph of Figure 3.4.3.

3.6. Valid Graph Embeddings

Not all graph embedding topologies obtained by the general graph embedding procedure are feasible for the generation of a layout. The embedding topology needs to possess the following properties:

1. The exterior vertex must be adjacent to the exterior face of the embedding.

2. The cyclic order of the exterior vertex must equal the reverse of the

cyclic order of the ports on the perimeter of the layout.

3. The cyclic order of every gate vertices must correspond to a valid expansion into individual transistors.

We will first comment on 3. In our circuit graph model a gate vertex will, in general, represent several transistors. Before creating a layout, gate vertices are expanded so that each transistor is explicitly represented. The expansion procedure is described in Chapter 4. The expansion of a gate vertex must preserve the cyclic order of connections to other vertices. This requirement is justified because the transistors defined implicitly by a gate vertex must be implemented within a bounded area. The drains and the sources of the transistors are accessed from the perimeter of the area. The drain and source of a transistor are, in MOS technology, ends of a diffusion wire. Each such wire divides the bounded area into one more region. The diffusion wire of one transistor must not cross the diffusion wire of another transistor. There will be a crossing if the drain and source of a transistor are located in different regions.

MOS transistors are formed by crossing poly and diffusion wires. The ends of the poly wire are the gate terminals. Since the gate is accessible from either region, the gate is accessible from any position around the perimeter. Hence, there is no constraint in the cyclic positions of the gate connections.

We also note that, even though the edges of the circuit graph may be assigned to many different planes, drain and source edges must have their heads on the diffusion/poly layer. Hence, even though formally, edges embedded in different planes are topologically unrelated, this property is not true in the final mapping to a physical layout. In a bounded area around the

heads of drain and source edges the cyclic order has to be valid ignoring their plane assignment. An additional observation is that the transistor type is irrelevant topologically.

Condition 1 is guaranteed by the way the initial cycle is chosen for the invocation of the extended Hopcroft-Tarjan's algorithm.

Condition 2 is verified straightforwardly.

Condition 3 is verified by creating a data structure for each gate vertex. The terminating state discloses whether the cyclic order of the gate vertex is valid.

A **transistor tree** is the data structure chosen to capture the nesting relations of the transistors defined implicitly by the incident edges and the cyclic order of the gate vertex. The edges of the transistor tree are called **arcs** to avoid confusion. The vertices are referred to as the root vertex, the internal vertices, or the leaf vertices. The transistor tree is generated in preorder, [Aho,Hopcroft,Ullman 74]. The shape of the tree is defined by the cyclic order of the gate vertex. The plane assignment of edges is ignored, i.e., all edges incident on the vertex is included in the cyclic order.

The construction of the transistor tree T starts with a null tree and proceeds in the following steps. The **current vertex** and the **new vertex** refers to the transistor tree. After the algorithm is halted and if the current vertex is not the root, then no planar subgraph exists with the cyclic order of the gate vertex. The algorithm proceeds as follows:

1. Add a vertex to T and label it **root**. Mark this vertex current.

2. Process each edge incident on the gate vertex according to the cyclic order. Start with any edge. Call it the current edge. An arc and a

vertex are added to T for each edge incident on the gate vertex. The arc starts from the current vertex and ends at the new vertex. The new vertex is labeled with the label of the current edge. The rules for updating the current vertex are based on the edge being processed as outlined below.

Case a: The edge is a port edge or the edge is emanating from the current vertex, then the current vertex stays the same.

Case b: The edge is terminating on the current vertex, then there are two sub-cases depending on the label of the current vertex.

1. The label of the current vertex and the current edge represent a drain-source pair of a transistor, then the father of the current vertex will become the current vertex.

2. The label of the current vertex and the current edge do not represent a drain-source pair of a transistor, then the new vertex will become the current vertex.

The algorithm halts after all edges in the cyclic order are exhausted. Figure 3.6.1 is an example of an invalid cyclic order with its transistor tree. The last current vertex is identified. Figure 3.6.2 is an example of a valid cyclic order with its transistor tree.

3.7. Algorithm Complexity

The complexity of the graph embedding algorithm is no worse than $O(e^2)$, where e is the number of edges in the graph. The worst case occurs when the path tree has only one level, and every bridge crosses other bridges. In practice, the complexity is closer to O(e).

TRANSISTORS	SOURCE	DRAIN
TR1	1	2
TR2	3	4
TR3	5	6
TR4	7	8

CYCLIC ORDER: G2–1–2–3–G1–5–G3–6–7–4–8

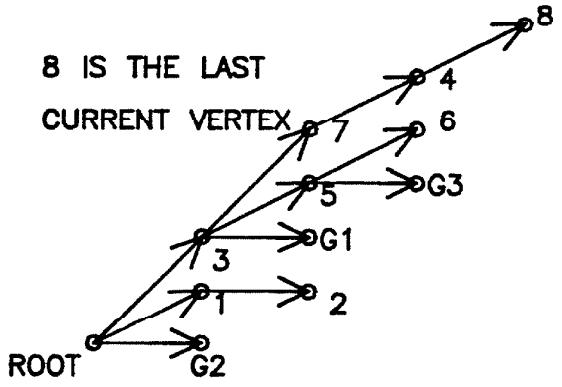


Figure 3.6.1 An Example of an Invalid Cyclic Order.

TRANSISTORS	SOURCE	DRAIN
TR1	1	2
TR2	3	4
TR3	5	6
TR4	7	8

CYCLIC ORDER: G2–1–2–3–G1–5–6–4–7–8

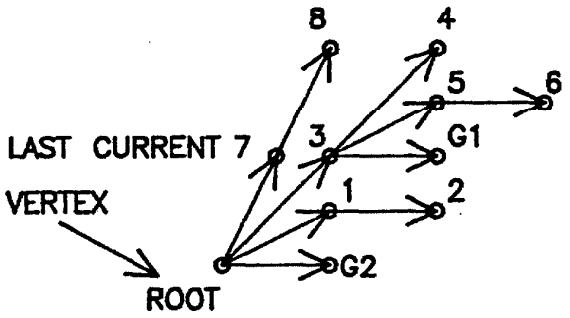


Figure 3.6.2 An Example of a Valid Cyclic Order.

The complexity of the algorithm for checking the validity of an embedding topology is no worse than $O(e^2)$, because the maximum number of edges in the orientation constraint graph and the compensation graphs is less or equal to the square of the number of edges in the original graph. In practice, the complexity is better than O(e), because the number of edges in the orientation constraint graph and the compensation graphs is far less than the number of edges in the original graph.

The algorithm for checking the validity of a topological circuit graph is linear in the number of transistors.

3.8. Remarks

Currently a path assigned to the left(right) of a cycle implies that the entire path must lie to the left(right) of the cycle. This constraint is not needed when the path and the cycle are assigned to different planes. The implication of the elimination of this constraint will be studied.

We do intend to study optimization criteria computable from the orientation assignment of the bridges and the layer assignment of the paths.

The complexity of the assignment of both the orientation constraint graph and the total compensation graph such that some criteria are optimized are usually NP complete.

CHAPTER 4

Circuit Graph Embedding Topology to Layout Topology

4.1. Introduction

The goal of our effort is the generation of circuit layout topologies from circuit schematics. If a valid embedding topology is found by the embedding algorithm the next step is to expand the gate vertices of the circuit graph to obtain a layout topology graph. In the layout topology graph each transistor is represented explicitly as a vertex of degree four. The transistor is modeled as having 2 gate terminals and one terminal for each of source and drain. This transistor model is also used by Goldstein and Scweikert [Goldstein,Scweikert 73]. The layout topology graph together with plane embedding information and cyclic order of vertices define the layout topology.

Once the layout topology is obtained it is straightforward to generate a physical layout. An algorithm is given in section three of this chapter.

The transformation of a layout topology to a physical layout is a one-to-many transformation. It is possible to define an equivalence relation on physical layouts such that each class has a unique layout topology. It is conceivable that the finite number of layout topologies can be ranked according to the likelihood of leading to an optimum design. The optimization can be divided into two phases, classifying layout topologies and finding optimum physical layouts within each class. Different optimization criteria are likely to lead to different classifications and different physical layouts.

We also present an algorithm for drawing an embedding or layout topology on a rectangular grid.

4.2. Creating a Layout Topology

The process of generating a layout topology graph and corresponding layout topology starts from a valid embedding topology. The layout topology graph is obtained by expanding all gate vertices of the circuit graph. Even if a gate vertex only models a single transistor it needs to be "expanded" since the transistor model is different. Vertices in the layout topology graph modeling transistors are referred to as **transistor vertices**. Nogate vertices in the circuit graph remain nogate vertices. The exterior vertex also stays the same. New undirected edges are introduced in the expansion of gate vertices to represent connections (poly wires) between transistor gates.

The cyclic orders of the transistor vertices are null initially. The order of the added edges defines the cyclic order. The end vertices of the edges incident on the gate vertices will be updated since they may be adjacent to the new vertices instead.

The gate vertices are processed one at a time. The cyclic order of the gate vertex is assumed valid. The generation of the layout topology graph use the transistor tree generated in verifying the validity of the embedding topology. There are different rules for processing the root vertex, the internal vertices, and the leaf vertices. The algorithm will be described with the graph G and the transistor tree T. G will equal the layout topology graph on completion of processing the transistor tree T. The cyclic order of vertices in the layout topology graph is also determined so the layout topology is known as well.

1. The root is processed in the following steps. Add a vertex to G to represent the root. The label of this vertex is R. The vertices in T are visited in the order they were created, i.e., preorder. There are two cases:

   a) If the son is a leaf vertex, then identify the edge of G that is represented by the leaf vertex using its label. One end vertex of this edge is the gate vertex being expanded. Change the label of this end vertex to R. Add this edge to the cyclic order of R.

   b) If the son is an internal vertex, then add a vertex to G to represent the transistor. The label of this vertex is denoted as TR. Add an edge to G. The end vertices are R and TR. Add this edge to the cyclic order of R. Add this edge to the cyclic order of TR. The label TR is then stored as part of the information about the son for future use.

2. The internal vertex V is processed in the following steps. A vertex has been added to G to represent this transistor when the father of this vertex is processed. The label of this vertex in G is TR. Identify the edge in G that is represented by the internal vertex V using its label. One end vertex of this edge is the gate vertex being expanded. Change the label of this end vertex to TR. Add this edge to the cyclic order of TR. Add a vertex to G. Label it G2. Add an edge to G. The end vertices of this edge are TR and G2. Add this edge to the cyclic order of TR. Add this edge to the cyclic order of G2. Identify the edge in G that is represented by the last son of the internal vertex V using the label of the last son of the internal vertex V. One end vertex of this edge is the gate vertex. Change the label of this end vertex to TR. Add this edge to

the cyclic order of TR.

The next step is to process the sons of the internal vertex V. The sons are processed according to the order of the descending edges. The last son has been processed already and will not be processed again. The sons are processed the same way as the processing of the sons of the root with *R* replaced by *G2*.

3. The leaf vertices need no further processing.

The algorithm ends after every vertex in T is traversed. An example of the expansion is shown in Figure 4.2.1.

4.3. Layout Topology to Physical Layout Transformation

It is always possible to produce a valid physical layout from a valid layout topology by separating the physical locations of transistors and wires as much as necessary. The transformation from a layout topology to a physical layout is simple. A transistor vertex is transformed into a physical layout of a transistor, i.e., a diffusion wire crossed by a poly wire. The "yellow box" that identify a depletion mode transistor from an enhancement mode transistor in nMOS is added to the physical layout at the last step. A net vertex is transformed into a continuous strip of the proper material. A contact is added where the level assignment of an edge does not agree with the level assignment of other edges incident on the same vertex. For the nMOS process plane 1 of the planes assigned in the embedding process is taken as the poly/diffusion plane (poly and diffusion wires must not cross), plane 2 is taken to be the metal layer. The remaining planes are all mapped into the metal layer with sections on the poly layer to avoid intersecting metal wires.

TRANSISTORS	SOURCE	DRAIN
TR1	1	2
TR2	3	4
TR3	5	6
TR4	7	8

CYCLIC ORDER: G2–1–2–3–G1–5–6–4–7–8

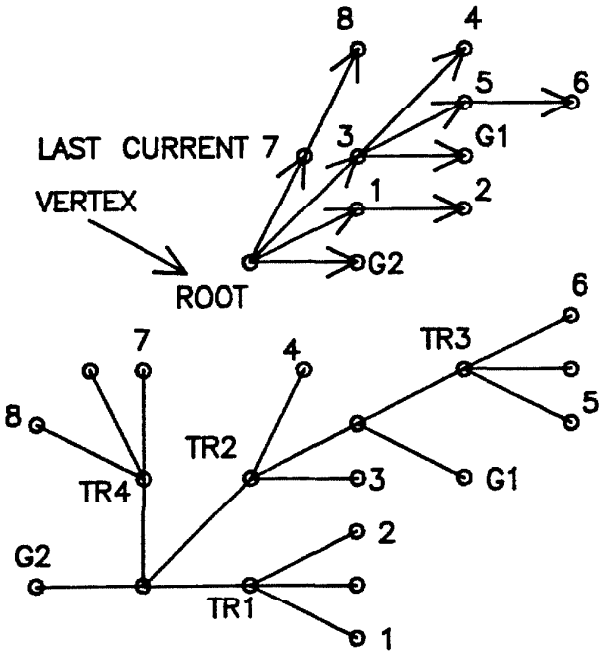


Figure 4.2.1 Expansion of a Gate Vertex.

The cyclic order of connections to a transistor and a net vertex shall agree with the cyclic order of the corresponding vertex in the layout topology.

The drawing algorithm outlined below will efficiently produce drawings corresponding to layout topologies.

4.4. An Algorithm for Drawing Embedded Graphs

The drawing algorithm generates a graph embedding based on the path tree, the orientation assignment of the bridges and the plane assignment of the paths. The drawing algorithm assumes that the orientations of the bridges and the layers of the paths are specified. The drawing is done on a rectangular grid with uniform spacing. All edges start with a length of one. They may be stretched as the situation requires. The drawing procedure is invoked recursively, once for each cycle. The drawing procedure establishes the directions of the edges and the constraints on the positions of the end vertices. These constraints are kept in two constraint graphs, one for the vertical direction and one for the horizontal direction. The constraint graphs are similar to the graphs uses by Liao and Wong [Liao,Wong 83].

The position constraint graph is an acyclic, weighted, directed graph that specifies the minimum spacing between related vertices. The roots of these two graphs are place in position 0 in the respective coordinate directions. The position of a vertex of the embedded graph is computed as the longest path between the root and the corresponding vertices (one for each direction) of the position constraint graphs.

The graphic data needed for the drawing of the embedded graph can be easily derived from the positions of the vertices. The drawing procedure is outlined below. The procedure is invoked by applying it to the initial cycle of the graph G. The vertical position constraint graph is denoted V. The horizontal position constraint graph is denoted H. An edge pointing left or right is a **horizontal edge**. An edge pointing up or down is a **vertical edge**. The

drawing algorithm is explained with the aid of an example in Figure 4.4.1.

1. Denote the path to be added to the drawing by P. P and a path of edges of a previously found cycle C' will form a cycle C. P has two attachments, Va and Vb, to the previous cycle C'. The new path P can be drawn either inside or outside the cycle C'. Directions are assigned to every edge in P such that the first edge and the last edge of P are at a right angle with C'. The directions of the edges of P are assigned based on the orientation of P and C, and the direction of the edges preceding Va and Vb in C'. The difference in the direction of the edges preceding Va and Vb in C' is either 0, 90, 180, or 270 degrees (mod 360). This accounts for eight cases. All other cases can be transformed into one of these cases. The construction rules are demonstrated in Figure 4.4.2.

When P is the initial cycle, then the edges of P will be assigned directions such that P will form a rectangle with counter-clockwise orientation. All the edges but one of the initial rectangle are horizontal with direction 0 degrees. This choice is arbitrary, but simplifies the algorithm. Any other rectangle could have been chosen. Auxiliary vertices may be added to P to ease the drawing. In the case that there are not enough edges in the path to form the pattern, e.g., if there is only one edge in the path, but it must form a "U" in the drawing to connect up to its parent cycle, then two auxiliary vertices are added for the two missing corners, and two edges added to complete the U.

2. The vertical positions of the end vertices of a horizontal edge are the same. Such a constraint can be captured in V by using only one vertex

to represent the positions of the vertices of G that must have the same vertical position. Similarly, only one vertex in H is used to represent the positions of the vertices of G that must have the same horizontal position. The algorithm, based on the directions of the edges of P, adds vertices to H and V according to the rules just described. Figure 4.4.1 shows that vertices 1 and 8 in G are represented by one vertex in H. Similarly the vertices 6 and 9 are represented by one vertex in V.

3. The algorithm then invokes the drawing procedure for the bridges of C.

4. Before the return to the calling procedure, more constraint edges are added to H and V to prevent the bridges of the same cycle from overlapping. This is done by adding edges to H and V to guarantee that parallel edges of different bridges are separated by at least one unit. In the example of Figure 4.4.1, the set of vertices 3,11,12,and 13 should be at least one unit away from the set of vertices 9 and 10 in the horizontal direction. Hence a directed edge is added to H. The edge terminates on the vertex representing the positions of the set of vertices 9 and 10 of G. The edge emanates from the vertex representing the positions of the set of vertices 3,11,12, and 13 of G. Edges are also added to H and V to guarantee the minimum separation of parallel edges between the cycle and its inside bridges. In the same example the set of vertices 13 and 14 should be at least one unit away from the set of vertices 7 and 8 in the vertical direction. Hence a directed edge is added to V. The edge terminates on the vertex representing the positions of the set of vertices 13 and 14 of G. The edge emanates from the vertex representing the positions of the set of vertices 7 and 8 of G.

The drawing algorithm is applied to the planar graph shown in Figure 4.4.1. The orientation assignment of the bridges are shown in Figure 4.4.3. The directions of the edges, the horizontal position graph, and vertical position graph, and the drawing of the embedded graph is shown in Figure 4.4.3.

The drawing algorithm uses the information on layer assignment of edges to draw the different subgraphs in different colors. Neither the drawing sequence nor the resulting drawing depend on the plane assignment.

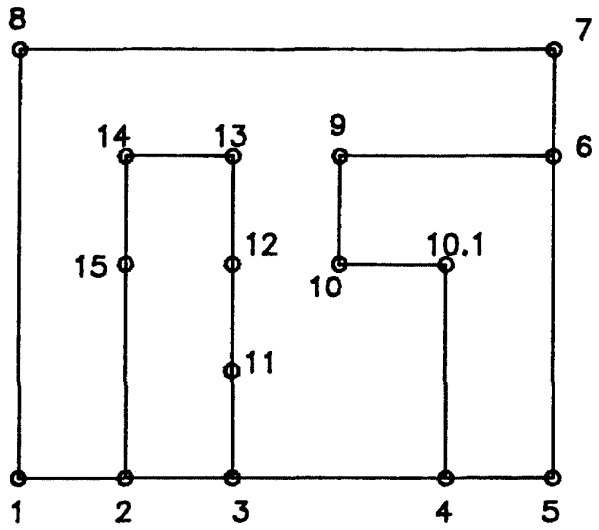


Figure 4.4.1 Working Example for the Drawing Algorithm.

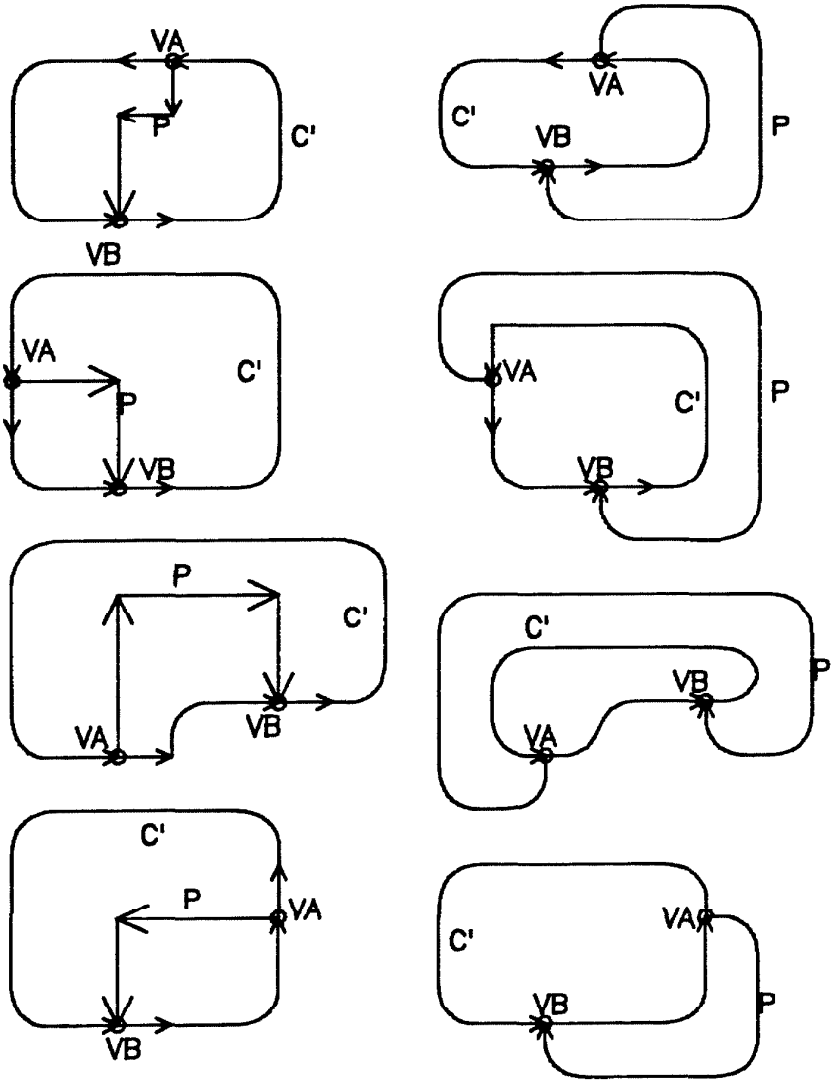


Figure 4.4.2 Rules to Determine the Drawing Direction of Edges.

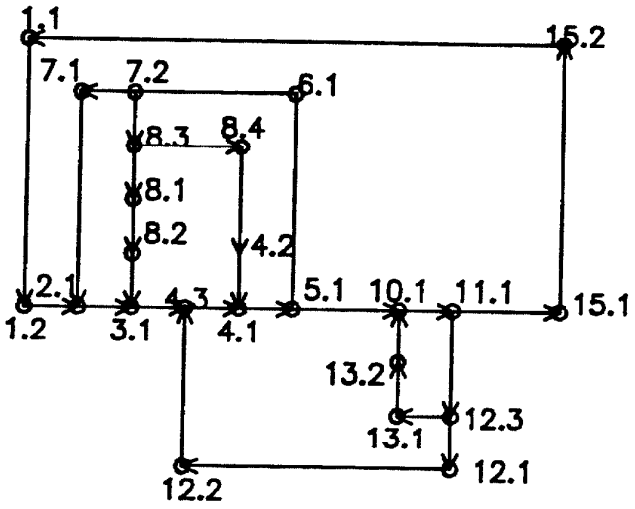
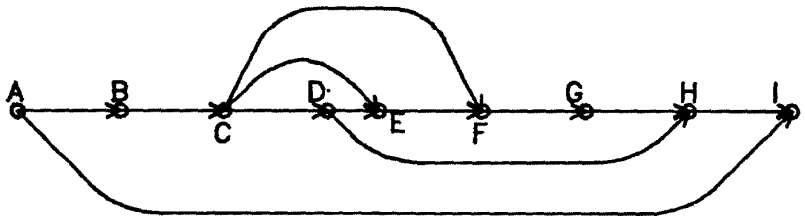


Figure 4.4.3a The Drawing of a Graph.

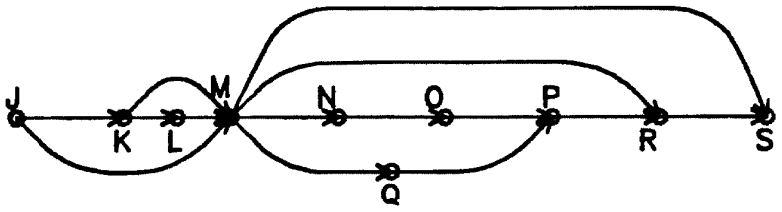
HORIZONTAL POSITION CONSTRAINT GRAPH



		FINAL POSITIONS
A:	1.1,1.2	0
B:	2.1,7.1	1
C:	3.1,8.2,8.1,8.3,7.2	2
D:	12.2,4.3	3
E:	4.1,4.2,8.4	4
F:	5.1,6.1	5
G:	13.1,13.2,10.1	7
H	12.1,12.3,11.1	8
I:	15.1,15.2	10

Figure 4.4.3b The Horizontal Position Constraint Graph.

VERTICAL POSITION CONSTRAINT GRAPH



		FINAL POSITIONS
J:	12.2,12.1	0
K:	13.1,12.3	1
L:	13.2	2
M:	1.2,2.1,3.1,4.3,4.1,5.1,10.1,11.1,15.1	3
N:	8.2	4
O:	8.1	5
P:	8.3,8.4	6
Q:	4.2	4
R:	7.1,7.2,6.1	7
S:	11.1,15.2	8

Figure 4.4.3c The Vertical Position Constraint Graph.

It is easy to obtain the cyclic order of vertices by an extension to the drawing algorithm. The cyclic order of vertices and the exterior cycles are recorded as the edges are drawn. After all the edges are drawn an accurate

description of the cyclic order of vertices and the exterior cycles of the embedding is obtained.

4.5. Algorithm Complexity

The algorithm for decomposing a circuit graph into a layout topology graph is linear in the number of transistors.

The complexity of the algorithm for establishing the two position constraint graphs for the drawing of the layout topology is no worse than $O(e^2)$, where e is the number of edges in the layout topology graph. The worst case occurs when the path tree has only one level. In practice, the complexity is closer to O(e log e).

4.6. Remarks

The reason for modeling a transistor as two edges incident on the gate net vertex is that the "free" cross-over property of the gate of a transistor is captured in this model. When there are more than one transistor with connected gates, the gates of these transistors are required to be connected with an uninterrupted poly wire. This constraint is imposed by the definition of the validity of the circuit graph embedding topology. If there are gates of many transistors connected in a bus-like structure, then the best strategy for connecting the gates may not be the use of an uninterrupted poly wire. The implication of allowing non-planar decomposition of a gate vertex on the embedding algorithm will be studied.

There is no provision for modeling the serial transistor configuration, because of the lack of an efficient algorithm for decomposing the multiple nets represented by a vertex.

CHAPTER 5

Experience

We have tested our algorithms on circuits with up to 36 transistors. It is our experience, from the circuits used to test our algorithms and evaluate our approach as well as other circuit designs, that interconnects are localized. The degree of net vertices are fairly independent of the number of transistors. Hence, a small circuit can serve a macroscopic view of larger circuits.

Our algorithms are coded in Mainsail[1] and executed on a DEC/2060[2] computer. The algorithms are coded for simplicity rather than efficiency. Knowing the time complexity of the algorithms, the embedding cost for a hundred transistors circuit will be less than ten seconds.

For circuits with about 10 transistors the analysis required about a 10th of a second. The orientation and layer assignment, and the expansion step took less than a tenth of a second each. The 32 transistor circuit, a pulse synchronizer, [Johannsen 81], required a total execution time of less than one second.

We have also observed that, for a small circuit, minimizing the number of contacts in the layout would reduce the final layout area significantly. In our approach contacts only enter in the transformation of the layout topology to the physical layout. Contacts do not have any logical function. They only serve to form nets of wires on different layers. The area penalty of a

---

[1]Trademark,Xidak Corp.

[2]Trademark,Digital Equipment Corporation.

contact is largely due to the fact that other features must be separated to make room for it. The physical area of a contact is small, but adding a contact may increase one dimension of a layout by k, where k is the difference in the feature size of a contact and a wire. This is a high penalty. This phenomenon is illustrated in Figure 5.1.1. This is a global effect and it accounts for most of the penalty.

Different strategies for using the different layers are possible. The metal layer is desirable for most functions. It is often determining the area of the layout. In a structured design of Mead-Conway type chip, communication is planned at the floor planning stage and the assignment of layers to, for instance, power, ground, and clock signals is related to how the communication is planned. The attempt is always, in particular in less structured

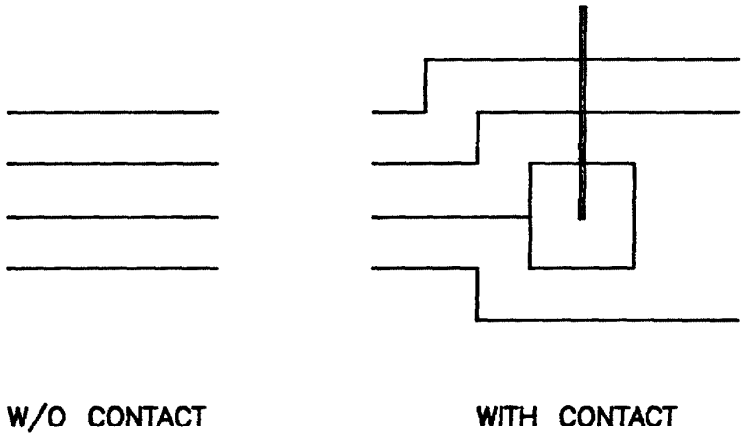


Figure 5.1.1 Area Penalty Caused by a Contact.

designs where the degree of freedom is higher, to assign power, clock, and data buses to the metal layer. For a large circuit, "long distance" wires should also be assigned to the metal layer to improve performance and most likely packing density. Long distance wires connect terminals that are several logic stages apart, or many terminals. It becomes necessary to identify the "long distance" connections and incorporate the result into the circuit specification, and the orientation/layer assignment strategy. In practice, the number of "long distance" nets is only a small portion of the total number of nets.

Our drawing algorithm is adequate for presenting a drawing of a topology, but it does not produce a compact drawing. The inefficiency is caused by the determination of the drawing directions of edges by simple, local, rules. We attempt to improve the ability of the drawing algorithm to generate a compact layout by formulating the problem of assigning drawing directions as a set of constraint equations of binary variables and with an optimizing function.

A picture is worth a thousand words. We present a set of Figures that show the result of various steps of our embedding approach. The circuit is an exclusive OR circuit. The circuit schematic and its specification are shown in Figure 5.1.2a. This circuit specification is the input to our algorithms.

Figure 5.1.2b is a drawing of the circuit graph with the internal numbering of the edges. There are five ports. Edges 1 through 5 are the port edges. Edges 6 through 19 are drain/source edges and these edges are directed. These edges terminate on the net vertices to which the gate terminals of the transistors belong. Edge 10 and 14 are self loops. They are the source edges of two depletion mode pull-up transistors. A depletion mode transistor in the

```
C XOR
E M + M - P A P B P OUT;
N TR1 1 - N1 A;
N TR2 1 - N1 B;
D TR3 1 N1 + N1;
N TR4 1 - OUT N1;
D TR5 1 OUT + OUT;
N TR6 1 - N2 A;
N TR7 1 N2 OUT B;
.
```

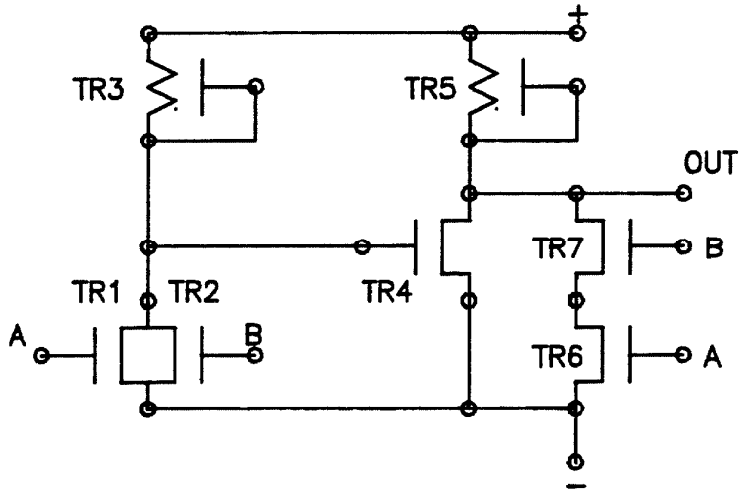
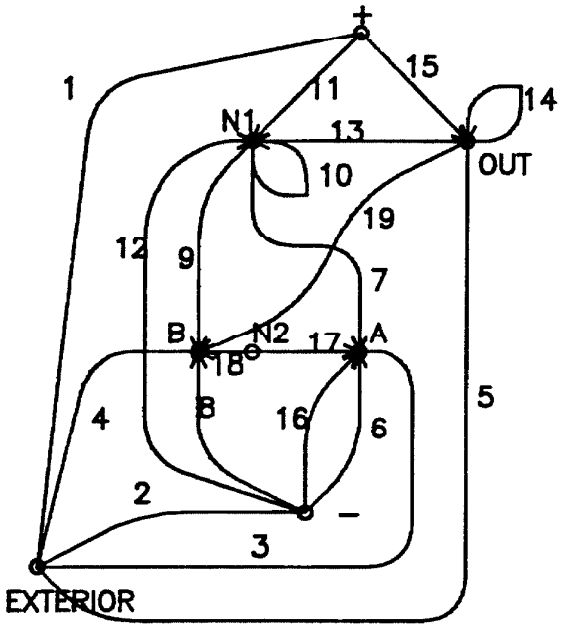


Figure 5.1.2a Circuit Schematic and Specification of an XOR.

pull-up configuration will have the source terminal connected to the gate terminal.



TRANS.	S.	D.
TR1	6	7
TR2	8	9
TR3	10	11
TR4	12	13
TR5	14	15
TR6	16	17
TR7	18	19

Figure 5.1.2b The XOR's Circuit Graph.

The numbering of the vertices is shown in Figure 5.1.2c. The exterior vertex is designated as the starting vertex. Hence, the analysis algorithm assigns it to be vertex 1. Vertex 1 has the special property that it is guaranteed to be adjacent to the exterior region. The paths and the path tree are also shown in Figure 5.1.2c.

Figure 5.1.2d is a drawing of the orientation constraint graph and the set of compensation graphs for the constraints. Figure 5.1.2e shows an orientation assignment of the graph in 5.1.2d made by inspection. If instead the

NET I.D.	VERTEX NO.	PATHS:
EXTERIOR	1	C1: 1,2,3,1
OUT	2	P2: 3,4,5,1
+	3	P3: 5,6,1
N1	4	P4: 6,7,8,1
−	5	P5: 8,2
A	6	P6: 8,4
N2	7	P7: 8,5
B	8	P8: 6,4
		P9: 6,5
PATH TREE		P10: 4,2

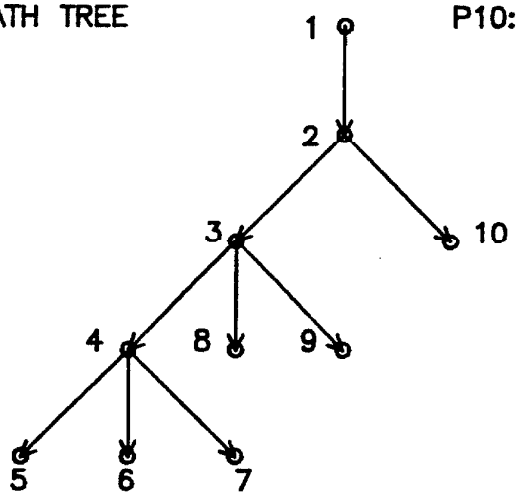


Figure 5.1.2c Paths, and Path Tree of the XOR's Circuit Graph.

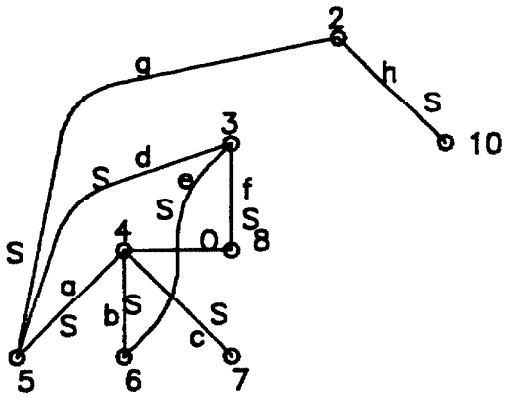
heuristics outlined in Chapter 3 is applied, every bridge would be assigned a left orientation. As a consequence of this orientation assignment one path will be assigned to the metal layer in the layout. The orientation assignment

in Figure 5.1.2e also results in one path being assigned to the metal layer, but it is nevertheless superior. The path in Figure 5.1.2e is actually a connection to the "-" port. Hence, this assignment does not result in an extra contact. The example demonstrates that there is room for improvement in our heuristic. The graphs contain the necessary information for making the proper decisions.

A drawing of the topological circuit graph is shown in Figure 5.1.2f. The transistor trees of the gate vertices A,B,N1,and OUT are also shown in Figure 5.1.2g. The result of the expansion step is shown in Figure 5.1.2h. The unconnected gate terminal is not drawn. Figure 5.1.2i is a rectangular drawing of the topology corresponding to the orientation/layer assignment of Figure 5.1.2e.

Finally, the topology is entered into a sticks design system, [Mosteller 81]. The compacted layout is shown in Figure 5.1.2j. A manually designed XOR circuit is shown in Figure 5.1.2k. It is laid out in the most obvious way, i.e., following a drawing of the circuit schematic. The topology expressed in the drawing of the circuit schematic is not random. A significant time was spent on finding a good topology. The layout corresponding to this topology has one extra contacts and a larger area. The topology generated with our approach is not easily envisioned even by an experienced designer.

ORIENTATION CONSTRAINT GRAPH



COMPENSATION GRAPHS

a: 3 — 5
b: 3 — 6
c: 3 — 7
d: 2 — 5
e: 2 — 6
f: 2 — 8
g: 1 — 5
h: 1 — 10
i: 7 — 8

Vertices of degree 0 are omitted.

Figure 5.1.2d The Orientation Constraint Graph and Compensation Graphs.

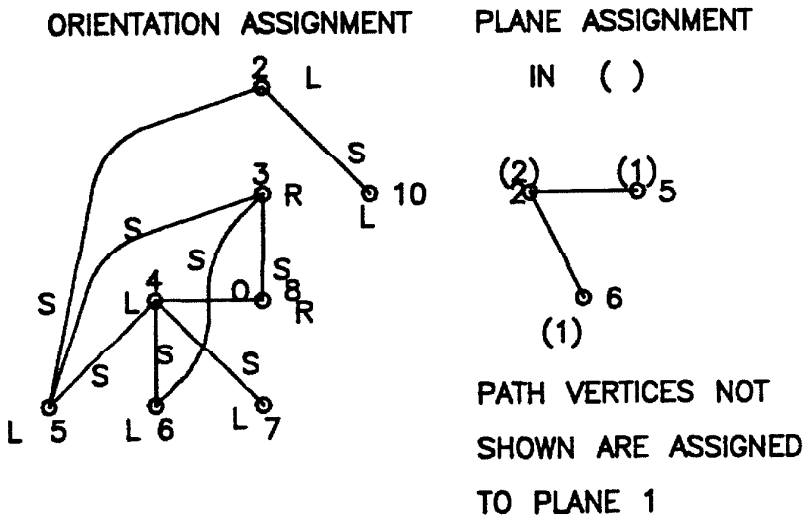


Figure 5.1.2e An Orientation and Layer Assignment.

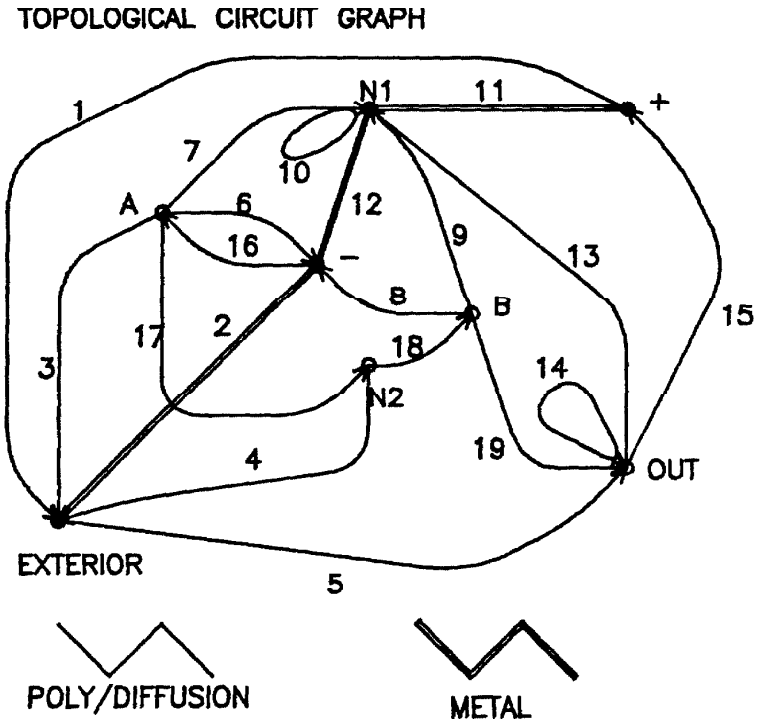


Figure 5.1.2f The Topological Circuit Graph.

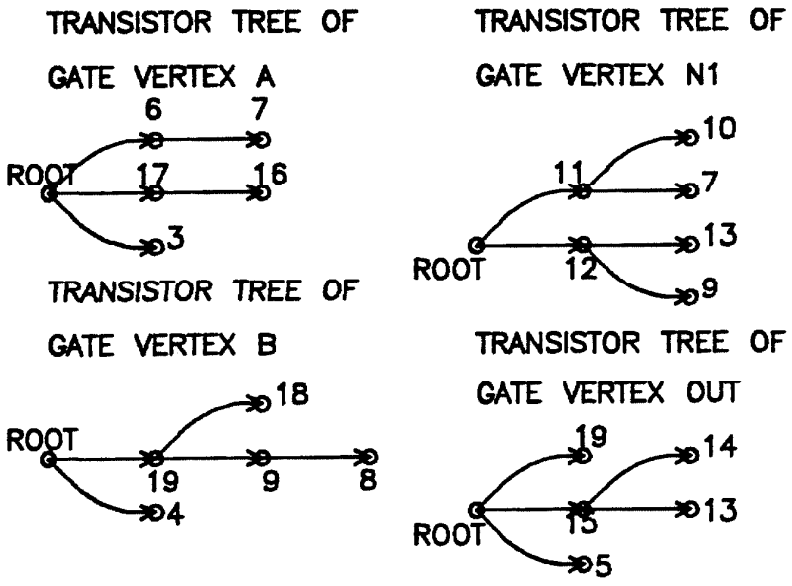


Figure 5.1.2g The Transistor Trees.

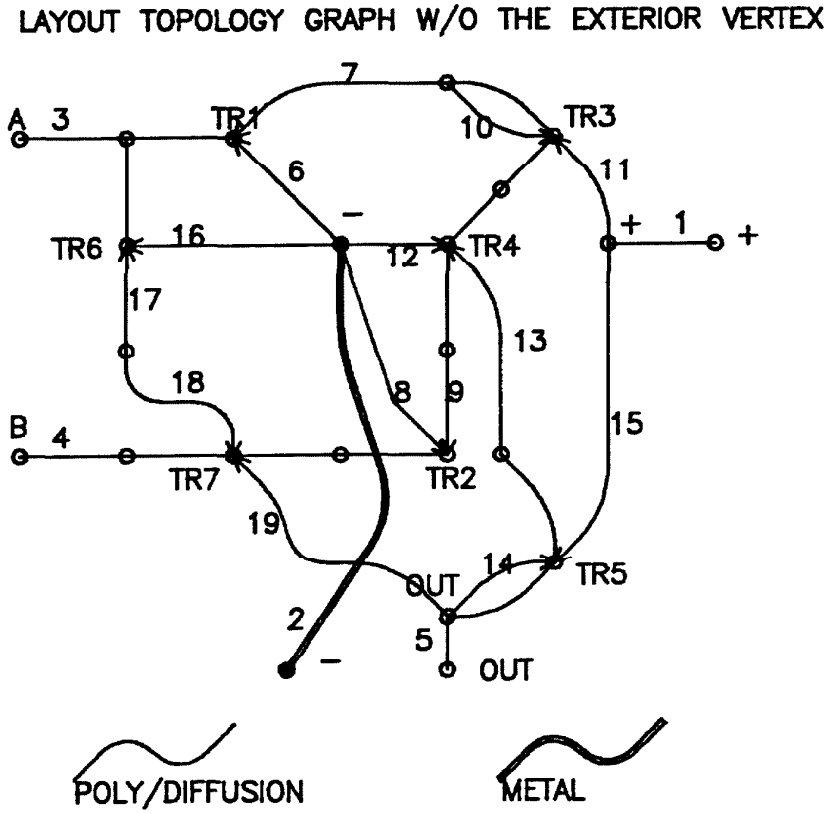


Figure 5.1.2h Result of the Gate Vertex Expansion.

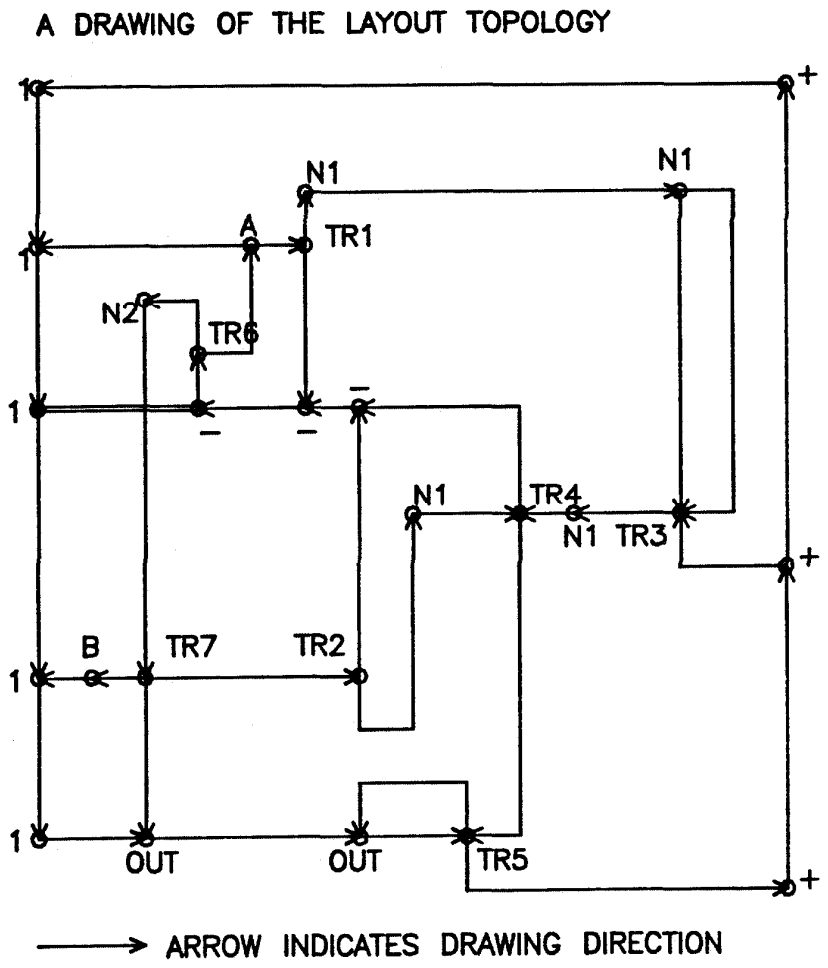


Figure 5.1.2i A Rectangular Drawing of the Layout Topology.

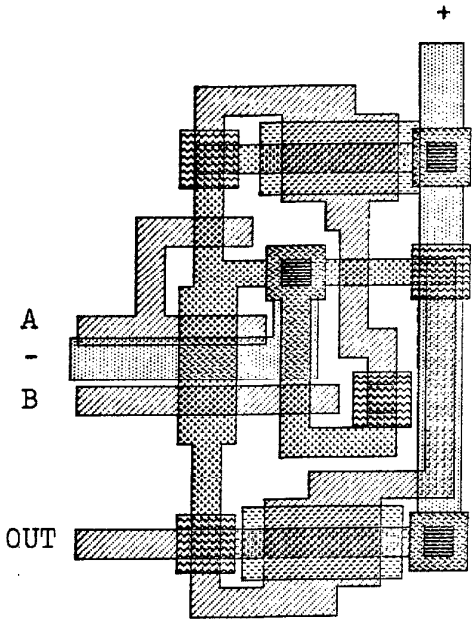


Figure 5.1.2j The Compacted Layout of the XOR, (Graph Approach).

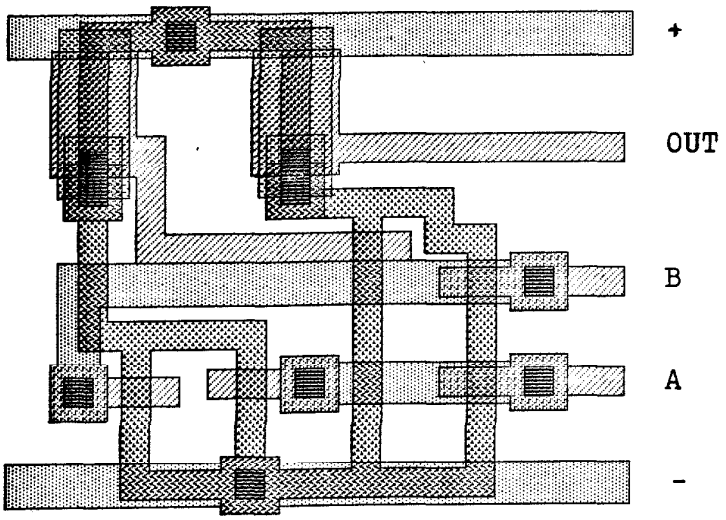


Figure 5.1.2k The Compacted Layout of the XOR, (Manual Approach).

In order to develop an understanding for the properties of layout topologies and their relationship to physical layouts a few experiments were conducted in which our algorithms were "tuned" to render what was believed to be poor layout topologies. The orientation assignment algorithm was modified such that it will choose a topology having the "most" unsatisfied orientation constraints. The compacted layout corresponding to such a topology is shown in Figure 5.1.2l.

We have also created an input specification of the XOR circuit which forces our algorithm to generate a circuit graph that is equivalent to the G.S. model. The compacted layout corresponding to the "best" topology generated by our algorithm is shown in Figure 5.1.2m. The increase in layout area is measurable.

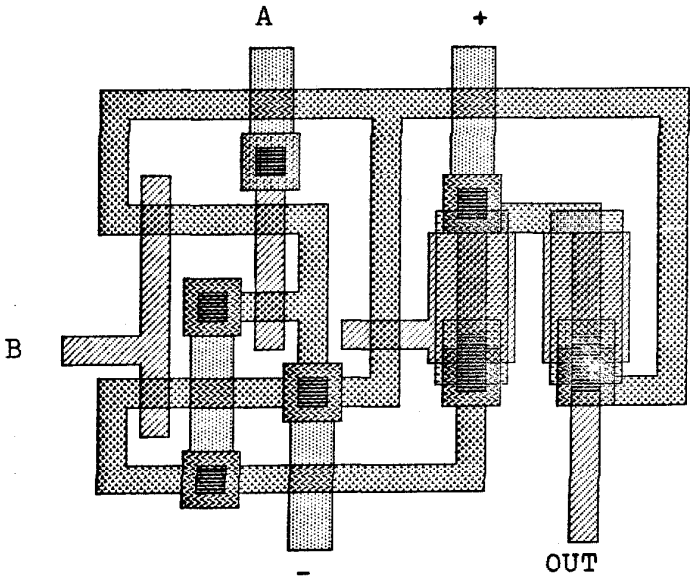


Figure 5.1.2l The Compacted Layout of the XOR with "Poor" Topology.

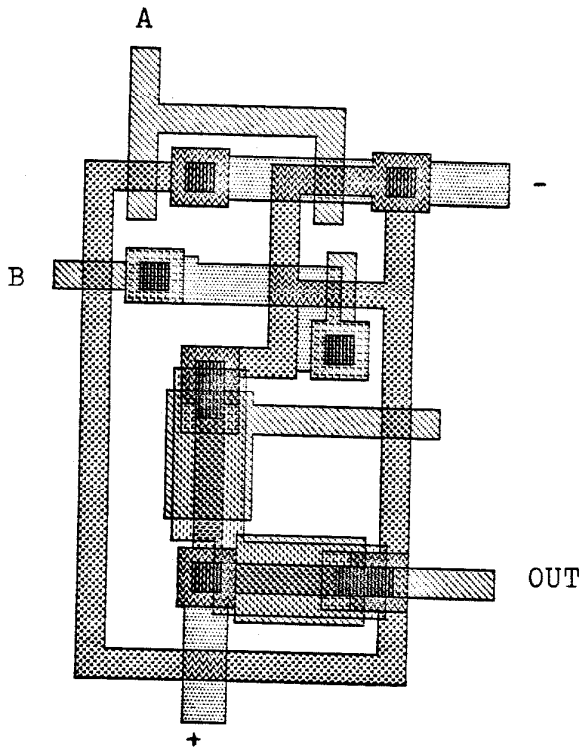


Figure 5.1.2m The Compacted Layout of the XOR (G.S. Model).

The number of contacts and the layout areas of the different topologies is summarized in Figure 5.1.2n.

As a somewhat larger test case for our approach the pulse synchronizer circuit in [Johannsen 81] was analyzed and laid out. Its circuit drawing is shown in Figure 5.1.3a. A layout corresponding to the "best" topology generated by our algorithms is shown as a circuit drawing in Figure 5.1.3b.

TOPOLOGY	#CONTACTS	AREA ($LAMBDA^2$)
GOOD	7	1060
MANUAL	8	1559
POOR	7	1408
G.S.	7	1402

Figure 5.1.2n Summary of the Quality of Various Topology of the XOR.

The two layout topologies are different. The topology of Johannsen's layout implements the power and most signal nets as metal wires across the layout. The vertical dimension of the layout corresponding to this topology is bounded by the space for embedding the metal wires. The positional order of the transistors are fixed by the horizontal power and clocks wires. A layout based on this topology is less "compactable".

The topology generated by our algorithm uses metal wires as jumpers. Most of the interconnects are assigned to the poly/diffusion layer. Transistors are relatively free to be positioned anywhere in the layout. A layout based on this topology is more "compactable".

The exact dimensions of Johannsen's layout are not given. Based on the minimum spacing between metal wires, and the minimum dimension of

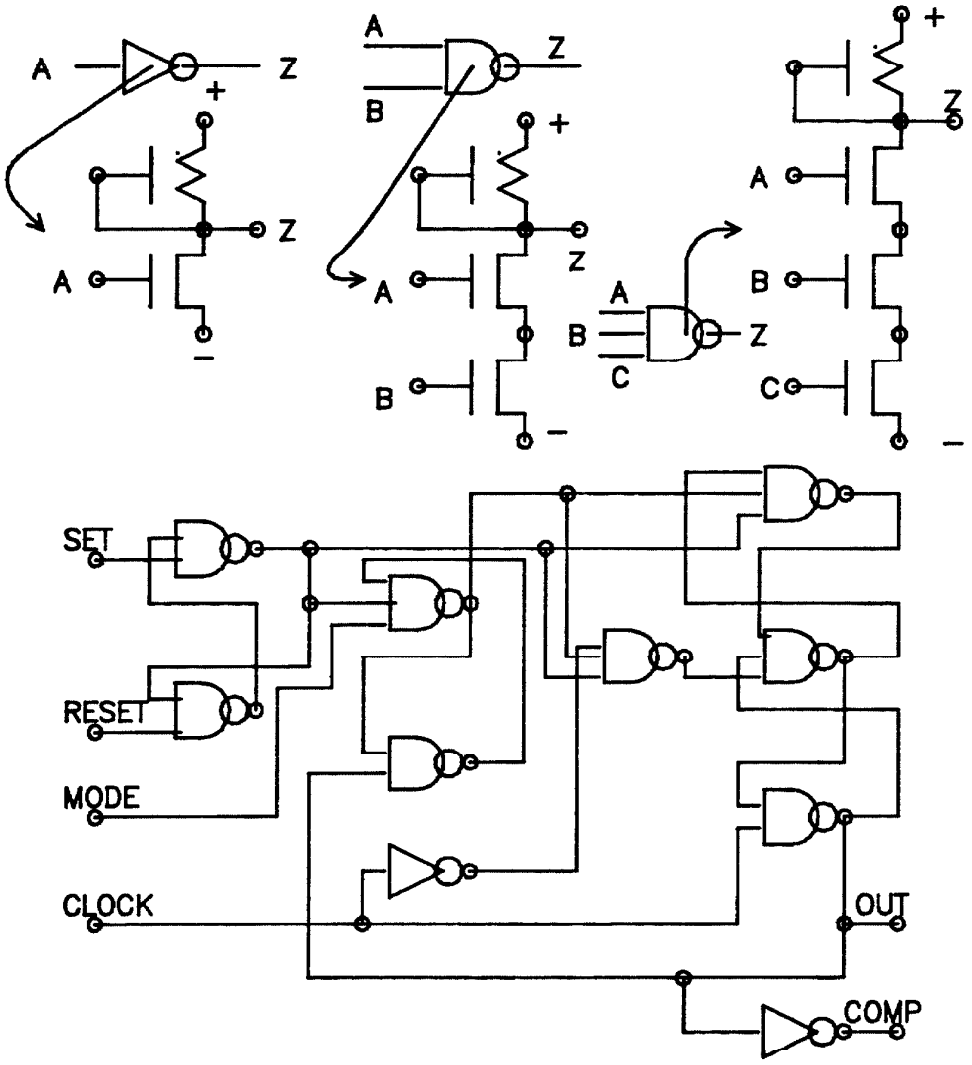


Figure 5.1.3a The Schematic of a Pulse Synchronizer Circuit.

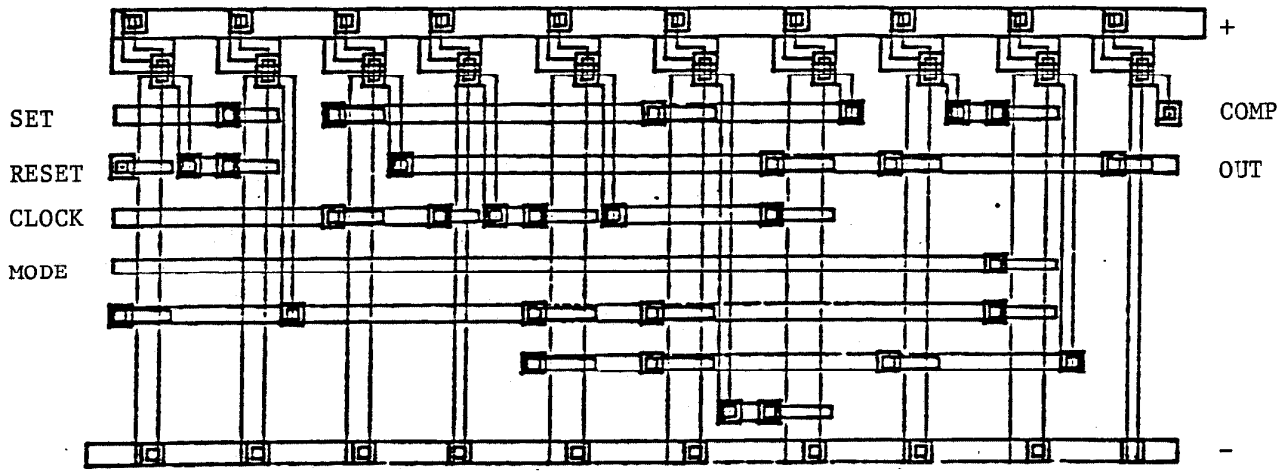


Fig. 5-3: Layout of Pulse Synchronizer

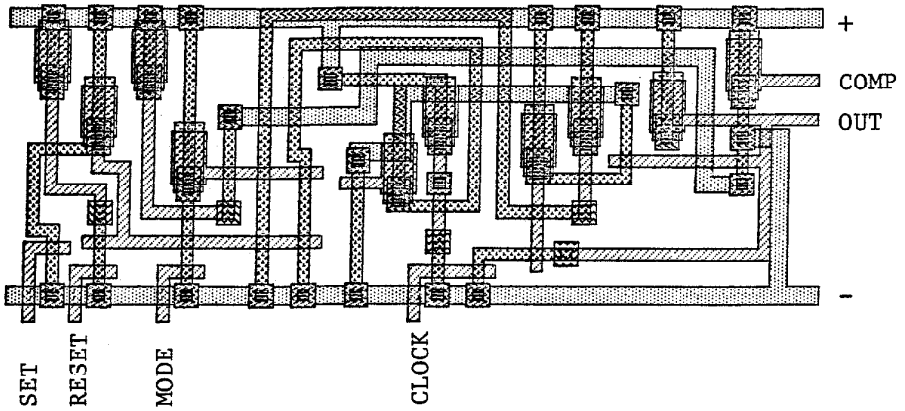


Figure 5.1.3b Two Layouts of the Pulse Synchronizer Circuit.[3]

depletion mode pull-up transistors, its dimension is estimated to be 200 by 80 Lambda. Lambda is a the basic spacing unit used in Mead and Conway's

[3]The top layout is from Figure 5-3 of Johannsen's [Johannsen 81] report.

design style[Mead,Conway 80]. Our layout is 130 by 55 Lambda, about half the area of Johannsen's regular layout.

We have included another example that has different interconnect properties than the previous examples. This example is a 4 bit carry chain circuit from Mead and Conway [Mead,Conway 80] with a total of 36 transistors. The circuit is shown in Figure 5.1.4a.

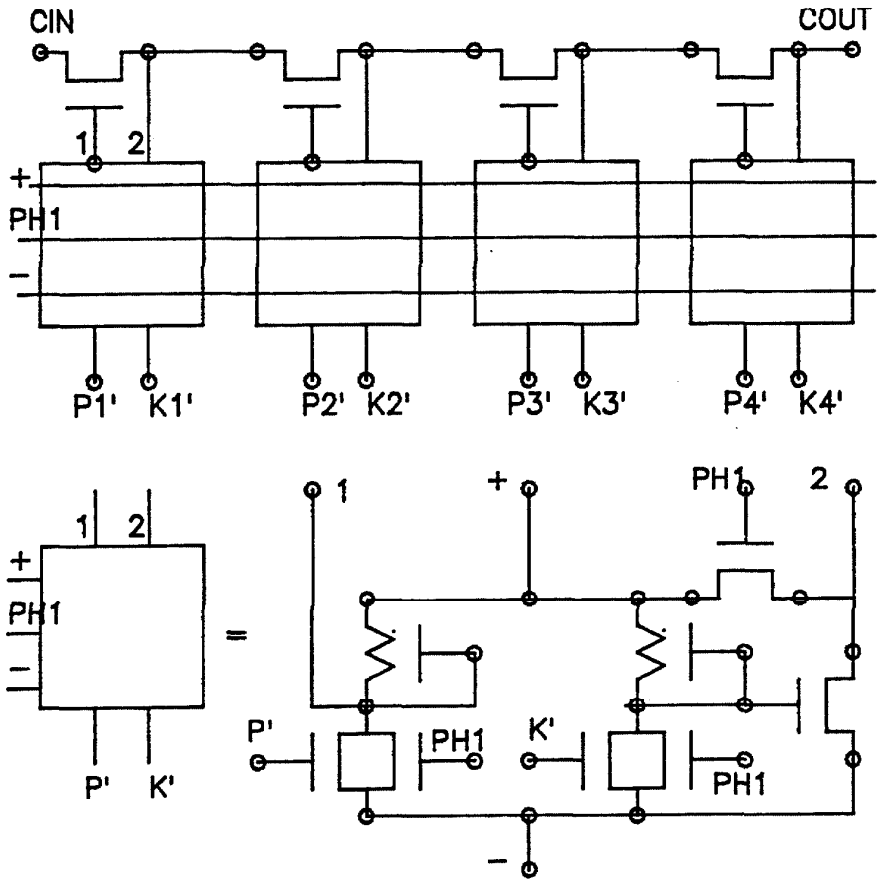


Figure 5.1.4a A 4 Bit Carry Chain.

When the power and clock wires are assigned to the metal layer, the penalty for connecting to these wires via jumpers are small. We demonstrate a method for treating the power and clock wires differently. The power and clock wires are assigned to the metal layer a priori. The circuit specification was modified to exclude the connections for the power and clock. We initially applied our embedding algorithms to the circuit. Starting with this decomposition of the circuit graph, we then added the edges for the power and clock connections. The orientation assignment and layer assignment were carried out in the same way except that cross-overs of the power and clock wires by other wires are assessed with no penalty. The layout of the 4 bit carry chain circuit is shown in Figure 5.1.4b. Our layout is about 80 percent the size of a compatible layout shown in Mead and Conway [Mead,Conway 80].

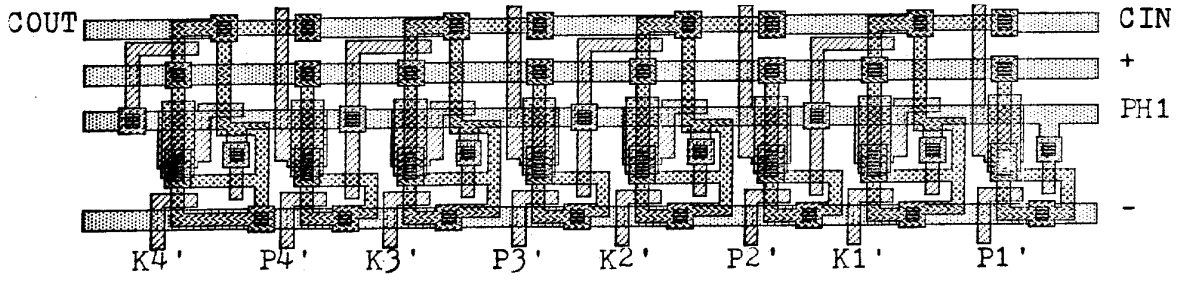


Figure 5.1.4b A Compacted Layout of the 4 Bit Carry Chain.

The detail of the extension of our algorithms has yet to be worked out. There is no difficulty to distinguish the power and clock nets from others because they are special in any design. We expect that our approach, with the extension, is capable of embedding circuits with several hundred transistors. The "bus"-type nets can be treated the same way.

CHAPTER 6

Conclusion

We have taken a graph theoretic approach to the circuit embedding problem. Circuits are modeled as graphs. The circuit embedding problem is solved as the graph embedding problem. Though the graph model and the graph embedding algorithm are two separate issues, they must complement each other to form a total viable solution.

We have discussed two other graph models; one proposed by VanCleemput, the other by Goldstein and Schweikert. We were not successful in locating any embedding algorithm that complements their models. A fundamental difficulty with their models is that there is a unique model for each decomposition of gate nets. The number of models increases exponentially with the number of transistors in the circuit. There is no obvious criteria for pruning the number of models that must be analyzed by the embedding algorithm.

With our graph model, there is only one circuit graph for each circuit. We achieve this result by being less explicit, i.e., transistors are not represented explicitly. We have assumed that the "quality" of the topology of transistors with common gate signal is independent of the valid nesting structure of the transistors. It has been observed that the "quality" of layout topology has direct bearing on the quality of the corresponding physical layout. The definition of quality is subjective and it varies between designs.

Others [Wolf, et al 83] have observed that jumpers have an adverse effect on the overall layout area of small circuits. A jumper allows two wires that belong to different nets to share the same planar space. The two wires

are in different layers. There are contacts at the ends of the jumper to assure electrical connectivity.

With the current fabrication technique, the minimum center to center distance of contacts is always larger than the minimum center to center distance of wires. The ratio of these two numbers, K, is typically 2 in today's NMOS technology. We can assert that the area of the "best" layout of a circuit corresponding to the three different graph models differ by less than a constant factor R**2, where R is the larger of 2 or K. Since the set of topologies of our model includes those of the G.S. and VanCleemput's models, there is a potential to generate a "better" layout from our model (topologies). This assertion is true because we can convert a layout that corresponds to a topology from our model to a layout that corresponds to a topology from the other two models by converting each four terminals transistor to a three terminals transistor with a pseudo gate terminal implemented as a jumper. This requires the expansion of each dimension of the layout by at most a factor of R.

In general, we have observed that the compacted result of a layout with "poor" topology can be much larger than the compacted result of a layout with "good" topology. An attractive feature of the graph theoretic approach is the systematic emulation of different layout topologies. A decision may be made after many different topologies are considered. A hand designer usually considers only a few topologies before focusing on one. It is likely that a superior topology may not be envisioned by the designer, but it can be found by systematic emulation. The quality of the result generated with the graph approach can be superior to the traditional placement and routing approaches because the traditional approaches lack the ability to systemati-

cally consider different topologies. We have observed that some layouts derived from the topologies generated with our approach have 50 percent less area compared to the corresponding hand layouts. Our samples are too small to be statistically significant.

Our current embedding algorithm, with a fixed starting vertex, does not emulate all topologies. It will, in general, generate a different hierarchy of paths with a different starting vertex. A different hierarchy of paths will lead to additional topologies. It has not been proven that it is possible to emulate all topologies with our algorithms.

The computational complexity of deriving the graph model and analyzing it is better than $O(n^2)$, where n is the maximum of the number of vertices and edges in the circuit graph (which equals the number of edges except for degenerate cases). In practice, the complexity is closer to O(n log n). The intent is that the embeddings generated with our algorithm will be compacted with some compaction algorithm. Hence, the computational complexity of the overall process is dominated by the assignment of orientations to paths and planes to edges, and the compaction process.

Denote C as the average number of transistors in a circuit that have gate terminals belonging to the same net. C is typically 3. The number of vertices of a circuit graph based on the G.S. model is about C times the number of vertices of a circuit graph based on our model. The number of edges of a circuit graph based on the G.S. model is about 1.5 times the number of edges of a circuit graph based on our model. The execution time of our embedding algorithm, in the best case, can be a factor of 3 faster than a circuit graph based on the G.S. model.

The compaction problem is NP complete. Most compaction heuristics are based on the iterative improvement technique. The number of iterations is usually linearly dependent on the number of transistors. The computational complexity for each iteration usually varies from linear to quadratic with the number of transistors. Hence, compaction heuristics have quadratic to cubic complexity.

The complexity of traditional circuit embedding heuristics is the product of the number of topologies for a given circuit and the compaction algorithm complexity. Since the complexity for evaluating the quality of a topology in our domain is (a small) constant, the overall computational complexity of our approach is proportional to the sum of the number of topologies and the compaction algorithm complexity. Our approach can be several orders more efficient than the traditional approaches. The efficiency of our approach depends on the assumption that there is a correlation between the quality of a topology with the quality of the corresponding physical layout.

An important aspect of the layout topology problem that is not fully addressed is the optimal assignment of orientation and planes. The domain of the optimizing metric is the set of regions and exterior cycle of each plane. Hence the optimizing metric is easy to compute. It is reasonable to expect that the computation for the optimal topology can be based on the branch-and-bound technique in which the validity test can be incorporated as a criterion for pruning the decision tree. The on-line validity verification has constant computational complexity.

The complexity of the planar compaction algorithm is about $O(n^3)$, where n is the number of transistors. It is expected that efficient heuristics can be developed for the orientation and layer assignment problem such that

the complexity of the automated transformation from circuit to layout is no worse than $O(n^3)$.

Bibliography

Aho, A., Hopcroft, J., and Ullman, J., [1974], The Design and Analysis of Computer Algorithms, Addison-Wesley, Mass., 179-187.

Auslander, L., and Parter, S., [1961], "On Imbedding Graphs in the Sphere," J of Math. and Mech., vol. 10, No. 3, 517-523.

Ayres, R.F., [1983], VLSI Silicon Compilation and the Art of Automatic Microchip Design, Prentice-Hall, NJ.

Bergmann, N., [1983], "A Case Study of the F.I.R.S.T. Silicon Compiler," in Bryant, R.,(eds), Proc., Third Caltech Conference on VLSI, Computer Science Press, 413-430.

Bilardi, G., Pracchi, M., Preparata, F., [1981], "A Critique and an Appraisal of VLSI Models of Computation," in Kung, H., Sproull, B., Steele, G., (eds), VLSI Systems and Computations, Computer Science Press, 81-88.

Breuer, M., [1977], "A Class of Min-Cut Placement Algorithms," Proc. of the IEEE/ACM 14th DA Conf., 284-290.

Burstein, M., [1980], "An Approach to Design Automation of Custom LSI Chip Layout Based on Heuristic Planarization and Annular Imbedding," IEEE ICCC, 1056-1059.

Goldstein, A.J., and Schweikert, D.G. [1973], "A Proper Model for Testing the Planarity of Electrical Circuits," The Bell Sys. Tech. J., Vol. 52, No. 1, 135-142.

Heller, W., Sorkin, G., and Maling, K., [1982], "The Planar Package Planner for

System Designers," Proc. of the IEEE/ACM 19th DA Conf., 253-260.

Hopcroft, J., and Tarjan, R., [1974], "Efficient Planarity Testing," J. of ACM 21, 4, 549-568.

Johannsen, D.L. [1981], Silicon Compilation, T.R. 4530, Computer Science Department, California Institute of Technology.

Karp, R.M. [1972], "Reducibility Among Combinatorial Problems," in Miller, R.E. and Thatcher J.W. (eds.), Complexity of Computer Computations, Plenum Press, New York, 85-103.

Liao, Y., and Wong, C., [1983], "An Algorithm to Compact a VLSI Symbolic Layout with Mixed Constraints," IEEE Trans. on CAD of IC and Sys., Vol. CAD-2, No. 2, 62-69.

Mead,C., and Conway,L., [1980], Introduction to VLSI System,Addison-Wesley.

Mosteller, R. [1981], Rest - A Leaf Cell Design System, Silicon Structures Project Report 4317, California Institute of Technology, CA.

Rubin,F., [1975], "An Improved Algorithm for Testing the Planarity of a Graph," IEEE Trans. on Computers, Vol. C-24, No. 2, 113-121.

Sahni, S., and Gonzalez, T., [1976], "P-complete Approximation Problems," JACM, 23, 555-565.

Seitz, C., [1979], "Self-Timed VLSI Systems," The Caltech Conference on VLSI, 345-355.

Shrobe, H., [1982], "The Data Path Generator," Proc., Conf. on Advanced Research in VLSI., Artech House, 175-181.

Siskind, J., Southard, J., Crouch, K., [1982], "Generating Custom High Performance VLSI Designs From Succinct Algorithmic Descriptions," Proc., Conf. on Advanced Research in VLSI., Artech House, 28-40.

Wolf, W., Newkirk, J., Mathews, R., and Dutton, R., [1983], "Dumbo, A Schematic-To-Layout Compiler," in Bryant, R., (eds), Proc, Third Caltech Conference on VLSI, Computer Science Press, 379-394.

VanCleemput, W.M. [1976], "Mathematical Models for the Circuit Layout Problem," IEEE Trans. Circuits and Systems, Vol. CAS-23, No. 12, 759-767.