Prepared for the 22nd IEEE/ACM Design Automation Conference. June 23-26, Las Vegas, NV

Generation of Layouts From Circuit Schematics; A Graph Theoretic Approach

Tak Ng and Lennart Johnsson

Research Report YALEU/DCS/RR-378
March 1985

Work supported by the Office of Naval Research under Contract No. N00014-84-K-0043

Generation of Layouts From Circuit Schematics; A Graph Theoretic Approach[1]

Tak-Kwong Ng
International Business Machines Corporation
Dept. A74, Bldg. 707-1, P.O. Box 390
Poughkeepsie, NY 12602

S. Lennart Johnsson
Department of Computer Science
and Electrical Engineering
Yale University
New Haven, CT 06520-2158

Abstract

The topological properties of metal-oxide semiconductor (MOS) transistors and the interconnections thereof are captured in a graph model. A set of algorithms is devised for the enumeration of layout topologies of a circuit from its graph model. Layout topologies are presented in stick diagrams. The algorithms select a set of embedded layout topologies with the "fewest" number of jumpers for layout generation and compaction. Layouts for circuits with up to 36 transistors have been generated successfully. The layouts corresponding to the topologies generated and selected by the algorithms are, in most cases, smaller than compacted hand layouts. The worst case computational complexity is $O(n^2)$, where n is the number of transistors in the circuit.

I. Introduction

The transformation of designs specified as circuit schematics into the "best" physical layout is a problem of great importance and unfortunately, great computational complexity. Computer generated layout topologies can lead to circuits that require less area, and that are faster than hand designs. Designs using transistors and wires as the primitive components in circuit schematics and their implementation usually yield circuits that are faster and that have fewer transistors than functionally equivalent designs using higher function primitives.

The computational complexity of layout generation from a given circuit description can be reduced by restricting either the functional domain or the layout domain or both. Many programmable logic array (PLA) generators achieve a low computational complexity by restricting the layout domain. A standard cells approach has low computational complexity due to restrictions on the layout and functional

---

[1]The research described in this report was carried out while the authors were with the Computer Science Department of the California Institute of Technology

domains. The approach we propose achieves an acceptable computational complexity for a large class of circuits by reducing the dimensionality of the layout generation problem. The reduction in dimensionality is obtained through the identification of equivalence classes. No constraint is imposed on the functional or layout domains.

Figure 1 illustrates our approach to the circuit embedding problem. A logic design is specified as circuit schematics. The circuit schematics are then transformed into a **circuit graph.**

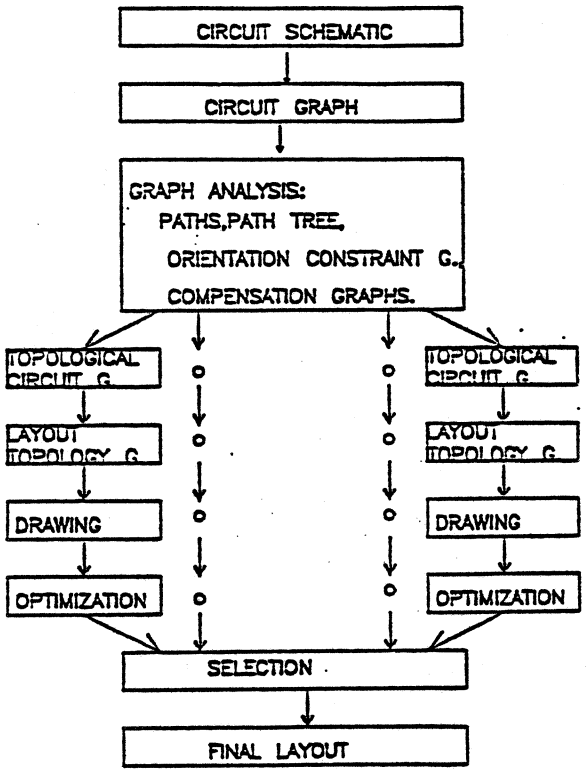


Figure 1: Our Approach to the Circuit Embedding Problem.

The circuit graph is analyzed by our analysis algorithms. They generate a set of data graphs and constraint graphs for each circuit graph. The embedding problem is converted to the problem of coloring the vertices of two constraint graphs. A coloring defines an **embedding topology**. The number of vertices of the constraint graphs is equal to the number of edge-disjoint paths in the circuit graph. One of the constraint graphs, the **total compensation graph** is used for plane assignment of paths. The other constraint graph, the **orientation constraint graph**, is used for the assignment of orientations of paths relative to each other for all the planes. For a planar circuit graph the total compensation graph has no edges, and the coloring is trivial. However, even for large classes of non-planar graphs the number of edges in the total compensation graph is few and the coloring is computationally feasible. The orientation constraint graph is always 2-colored. The computational complexity of enumerating all

embedding topologies of a given circuit graph is acceptable for large classes of circuits. A heuristic selection mechanism is included in the enumeration algorithm in order to generate an embedding topology with the "fewest" number of jumpers. A circuit graph embedding topology is expanded into a **layout topology graph**. A stick diagram is generated for each layout topology graph. An **ideal compactor** produces the final layout.

An ideal compactor transforms different stick diagram representations of a given layout topology into identical layouts. The compactor needs information about the minimum bounding box, and the cyclic order of the ports around the perimeter of the layout. Hence, for a given set of boundary conditions, only different layout topologies lead to different layouts. By choosing layout topologies as the intermediate representations of circuits, the circuit layout problem becomes managable. Our embedding algorithm can efficiently generate and evaluate many different topologies for a given circuit schematic.

The use of transistors and wires as the primitive components for circuit schematics and general layout generators is not new. DUMBO [11] is another general layout generator using these primitives. Layouts generated by DUMBO on the average are 120% larger than hand designs. Wolf has identified two possible reasons for the inefficiencies: 1) the placer is insufficiently attentive to the planarity of the wiring implied by the placement; 2) and the wirer sometimes chooses very bad locations for the contacts it adds to the cells.

A graph theoretic approach to circuit embedding was applied by Rose and Oldfield [8] for the embedding of bipolar analog circuits in single layer circuit boards. Engl, Mlynski, and Pernards [2] used a graph theoretic approach to embed bipolar analog circuits into integrated circuits, and Van Lier and Otten [10] used a similar approach to solve the problem of embedding bipolar analog circuits in a monolithic technology.

Goldstein and Schweikert [3] have proposed a model for testing the planarity of electrical circuits. VanCleemput [9] extended Goldstein and Schweikert's model with the capability of specifying the cyclic order of terminals of components. Physically or logically equivalent terminals can also be specified.

The graph models mentioned previously do not capture the topological properties of gate terminals of MOS transistors. The two physical gate terminals of a transistor, besides being physically equivalent, create a "free" crossover that can be exploited. The connections to the gate can be partitioned into two subsets, one subset connects to each terminal. With our proposed model, there is only one graph for each circuit schematic, independent of whether there are connections to one or both gate terminals. An exponential growth in the number of graphs for a circuit is prevented thereby.

Our graph model is discussed in section II, and the analysis algorithms are described in section III.

Examples of layouts generated by our approach, and some run-time data, are given in section IV. The conclusions and the recommendations for future research appear in section V.

II. The Graph Model

Our primary interest is in MOS circuits. The graph model must capture the essential characteristics of the physical layout. The layout consists of transistors, wires, and contacts. A MOS transistor is logically a three terminals device. It physically has four terminals: the *drain* terminal; the *source* terminal; and the two *gate* terminals. The model for a MOS transistor should be independent of the number of physical gate terminals used. Otherwise, the number of graph models for a circuit grows exponentially. The transistor type is irrelevant in deriving the layout topology, except in presenting a layout topology as a stick diagram. The transistor type is defined by a separate mask, and need not be considered in the layout optimization. All MOS transistors are topologically identical. Wires can be assigned to any of three layers: *diffusion*, *poly*, or *metal*. Contacts serve to connect wires on different layers.

A **net** is a set of terminals and/or ports, connected at all times, by wires and contacts. The appropriate model for a net is a vertex as proposed by Goldstein and Schweikert. 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, or **nogate vertex** for short.

A transistor in our graph model is 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.

In 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 identification of ports and the specification of the prescribed cyclic order.

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. In our model the perimeter of the layout is represented by the **exterior vertex**.

Ports are represented by undirected edges in our graph model. A **port edge** has one end incident on the exterior vertex, and one end on a net vertex. 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.

An exclusive OR circuit and its graph model is shown in Figures 2 and 3. A detailed description of the circuit to graph transformation algorithm can be found in [7]. The computational complexity of the 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.

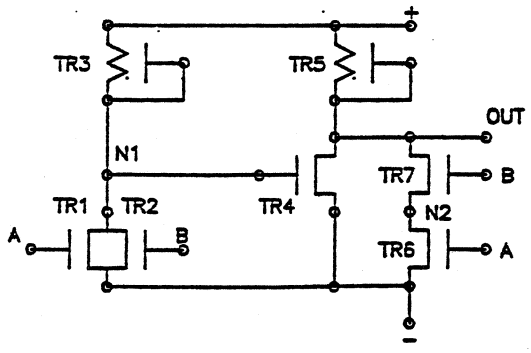
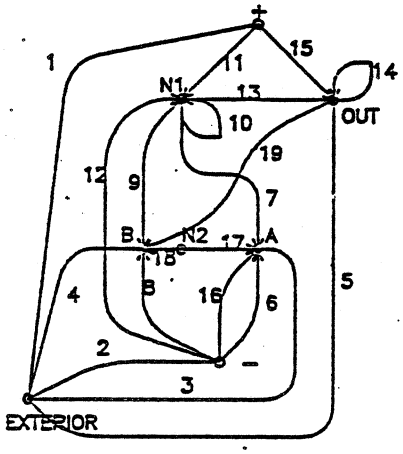


Figure 2: Circuit Specification of an XOR.



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 3: A Circuit Graph Model of an XOR.

III. The Embedding Algorithm

By modeling a circuit as a graph the problem of embedding a circuit in a set of physical layers is converted into the problem of embedding a graph in a set of planes. The general graph embedding problem, i.e., the embedding of a graph in the fewest number of planes, is NP-complete. The planar graph embedding problem is the least complex embedding problem. It is also a hard problem, but there exists a linear time algorithm for planarity testing [4]. For non-planar graphs it is desirable with respect to computational complexity that the embedding algorithm is capable of enumerating many embedding topologies without processing the graph repeatedly. We capture all the information required for the enumeration in three types of graphs, the **path tree**, the **orientation constraint graph**, and the set of **compensation graphs**, during one traversal of the circuit graph.

The first stage of the embedding of the circuit graph is to find a partitioning of this graph into *maximal biconnected* components. A biconnected component is a connected component which stays connected under the removal of any single vertex and the edges incident thereupon [1]. A maximal biconnected component is a biconnected component which cannot be enlarged with additional nodes and associated edges of the connected component of which it is a subgraph, without losing the property of being biconnected. The different biconnected components can be embedded independently. In the following we assume that the circuit graphs are biconnected.

Our algorithm that analyzes the biconnected components creates a data structure such that the enumeration of embedding topologies can be carried out efficiently. The analysis algorithm is a modified PATHFINDER procedure of Hopcroft and Tarjan's (HT) planarity testing algorithm. The PATHFINDER procedure adds paths to a subgraph already tested for planarity. Our analysis algorithm uses the same set of paths as those generated by HT's algorithm. The analysis algorithm is embedded in HT's path-finding algorithm. HT's algorithm is modified such that it halts only after all edges are processed. This is accomplished by deleting the "go to non-planar" clauses from the procedure EMBED. The terminology for the description of the graph analysis algorithms is defined in [7].

HT's algorithm is also modified to capture enough data to enumerate embedding topologies and to efficiently draw the embeddings. The procedure PATHFINDER is invoked recursively. It decomposes the graph into a set of edge-disjoint paths. This defines a hierarchy of paths. The hierarchy is captured in a **path tree** where a vertex corresponds to a path in the circuit graph. The initial cycle selected by HT's algorithm is represented by the root of the path tree. The algorithm selects a sequence of edge-disjoint paths such that each new path forms a cycle with either the preceeding path alone, or the preceeding path and any of the previously selected paths.

The algorithm treats the paths as directed in the direction of increasing index of the vertices of the path, except for the last edge of the path. It is directed to a vertex of a preceeding path or for the initial cycle the first vertex of the path itself. Let P1 and P2 be two edge-disjoint paths of a biconnected component, and the corresponding vertices in the path tree be V1, and V2. Then V1 is the *parent* of V2 if and only if the first vertex of P2 is one of the vertices, but not the first or last vertex, of P1. P1 is referred to as the *parent path* of P2. The *parent cycle of a path* is a cycle formed by the parent path, and sections of some other paths, with the obvious exception for the few paths where the parent path is the initial cycle.

Each subtree of the path tree is a *bridge*. The number of bridges is equal to the number of nodes in the path tree. The *parent path* of a bridge is the parent path of the path forming the root of the subtree defining the bridge. Similarly, the *parent cycle of a bridge* is the parent cycle of the path forming the

root of the subtree defining the bridge. The *parent bridge* of a bridge is the smallest subtree of the path tree that has the bridge as a proper subtree. In the graph embedding bridges with the same parent path must be assigned either to the *left* or to the *right* with respect to their parent cycle for each plane. We refer to the left/right assignment of a bridge as its *orientation*. For each plane the orientation of bridges must be made such that bridges do not cross. For a planar graph there exists an orientation (not unique) such that no crossing occurs. We separate the orientation assignment from the plane assignment.

A pair of *incompatible bridges* are two bridges with the same parent cycle that will cross when both are given the same orientation. The incompatibility is detected by comparing the end vertices of the paths constituting the bridge with the attachments of other bridges having the same parent cycle. The incompatibility in the orientation of bridges with the same parent cycle is captured in the **orientation constraint graph**. Each bridge is represented by a vertex in this graph. The number of nodes in the orientation constraint graph is the same as in the path tree. Each incompatibility is represented by an *o-edge*. An o-edge implies that the bridges represented by its end vertices are required to have opposite orientations as they would cross each other otherwise. The orientation constraint graph with all o-edges inserted is a disconnected graph. The number of connected components, o-components, is at least equal to the height of the path tree. The maximum number of nodes in an o-component is bounded by the largest fanout in the path tree, since o-edges only exist between bridges with the same parent bridge.

There also may exist a constraint on the orientation of a bridge relative to the orientation of its parent bridge. If a bridge and its parent are given opposite orientations the bridge may cross either its parent cycle or the parent cycle of its parent path. This type of constraint is represented by an *s-edge* in the orientation constraint graph. S-edges connect distinct o-components. The orientation constraint graph with both o-edges and s-edges inserted may be connected.

Finally, the analysis algorithm also records the paths that would cross each other when a constraint represented by an edge in the orientation constraint graph is not met. The constraints represented in the orientation constraint graph are between bridges, but in general, only a few of the constituent paths will cross each other if the constraint is not satisifed. The information regarding which paths will cross is recorded in the set of **compensation graphs**. There is one compensation graph for each edge in the orientation constraint graph. Each disjoint path of the circuit graph (i.e., each vertex of the path tree) is represented by a vertex in this graph. Two paths represented by adjacent vertices would cross each other if the orientation assignment conflicts with the orientation constraint (edge) represented by the compensation graph.

The path tree, the orientation constraint graph, and the compensation graphs are the main results of the analysis algorithm. A 2-coloring of the orientation constraint graph is a coloring using two colors,

left and right. The orientation constraint graph is *2-colorable* if there exists a 2-coloring such that the end vertices of every s-edge have the same color, and the end vertices of every o-edge have different colors. A graph is planar when its orientation constraint graph is 2-colorable.

An orientation constraint is not satisfied for a given 2-coloring of the orientation constraint graph, if the end vertices of an o-edge are assigned the same color, or the end vertices of an s-edge are assigned different colors. A **total compensation graph** is defined for each 2-coloring of the orientation constraint graph. The vertex set of a total compensation graph is the vertex set of the path tree. The edge set is defined as the union of the individual compensation graphs of the unsatisfied orientation constraints.

A *valid coloring* of a total compensation graph is a coloring for which adjacent vertices are colored differently. The colors of the total compensation graph correspond to the plane assignment of paths. A 2-coloring of the orientation constraint graph, and a valid coloring of the total compensation graph specify an **embedding topology** of a graph. Different embedding topologies are generated by different 2-colorings of the orientation constraint graph and with a valid coloring of the corresponding total compensation graph.

The coloring of the orientation constraint graph and the corresponding total compensation graph can be carried out in many ways. The following heuristic selection mechanism is used to generate one particular embedding topology (for each biconnected component). First, orientations are assigned to bridges such that the s-edge constraints are satisfied. Then, orientations are assigned to the unassigned bridges such that the number of unsatisfied o-edge constraints is small, if possible. The coloring of the total compensation graph is performed with the objective of assigning as many paths as possible to one plane (the first). This procedure tends to minimize the number of contacts.

For a circuit graph, the exterior vertex is always selected as the starting vertex of the path-finding procedure. This choice guarantees that the exterior vertex is adjacent to the exterior face of an embedding. Additionally, accepted 2-colorings must also have the following properties.

1. The resulting cyclic order of the exterior vertex is the reverse of the cyclic order of the ports on the perimeter of the layout, and

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

Condition 2 is justified because the transistor or transistors defined implicitly by the directed edges terminating on a gate vertex are 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 silicon 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.

MOS transistors are formed by crossing a poly-crystalline silicon (poly) wire and a diffused silicon wire. 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.

The cyclic order of each gate vertex can be checked by considering only the edges representing the drains and sources of the transistors implicitly defined by the edges terminating on this vertex. These edges must not be interleaved. The details of the validity checking algorithm are described in [7].

After an embedding topology is found, the next step is to expand the gate vertices of the circuit graph to obtain a **layout topology graph**. The expansion creates a vertex of degree four for each transistor implicitly defined by the edges terminating on the gate vertex. The expansion step also defines the decomposition of each gate net. The expansion is coded in rules rather than templates. The details of the expansion algorithm are described in [7]. The layout topology graph is drawn in sticks by the *drawing* algorithm.

The drawing algorithm uses the orientations of the bridges and the layers of paths. The drawing is done on a uniform rectangular grid. All edges are initially of length 1. They may be stretched as the situation requires. The drawing procedure is invoked recursively, once for each cycle. The drawing procedure establishes the directions and positions of the segments 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. These constraint graphs are similar to the graphs used by Lao and Wong [6]. The main purpose of these vertical and horizontal constraint graphs is to compute the positions of every transistor and every corner of wires, such that they do not overlap. The drawing does not need to be compact. A compact layout is produced by the ideal compactor. A detailed description of the algorithm is contained in [7].

IV. Experience

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

Our algorithms are coded in Mainsail[2] and executed on a DEC/2060[3] computer ($\approx$ 1.5 MIPS). The algorithms are coded for simplicity rather than efficiency. The 32 transistor circuit, a pulse synchronizer, [5] required a total execution time of less than one second. Knowing the time complexity of the algorithms, the computation cost for a hundred transistors circuit will be less than 10 seconds.

[2]Trademark, Xidac Corp.

[3]Trademark, Digital Equipment Corp.

Figures 4, 5, 6, and 7 illustrate the large variety of topologies for a simple XOR circuit. The layout areas vary from 1060 units to 1559 units. As a somewhat larger test case for our approach, Johannsen's pulse synchronizer circuit was analyzed and laid out. The original layout and the layout corresponding to the "best" topology generated by our algorithm are shown in Figures 8 and 9.

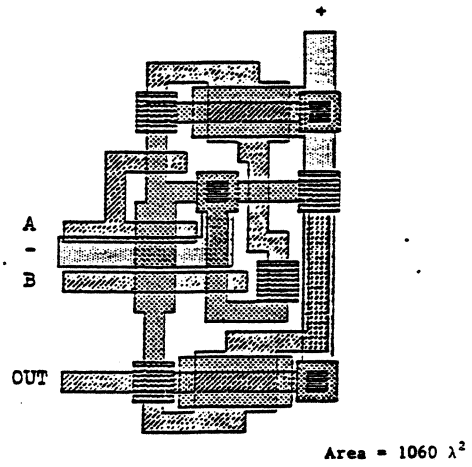


Figure 4: Compacted Layout of the XOR (Graph Approach.)

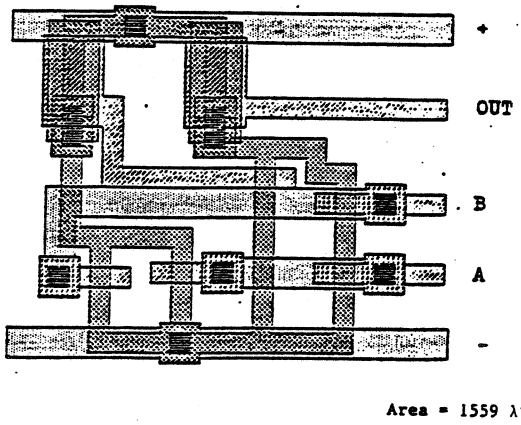


Figure 5: Compacted Layout of the XOR (Manual Approach.)

The layout generated by our algorithms and Johannsen's layout have different topologies. Johannsen implements the power net and most signal nets as metal wires across the layout. The vertical dimension 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."

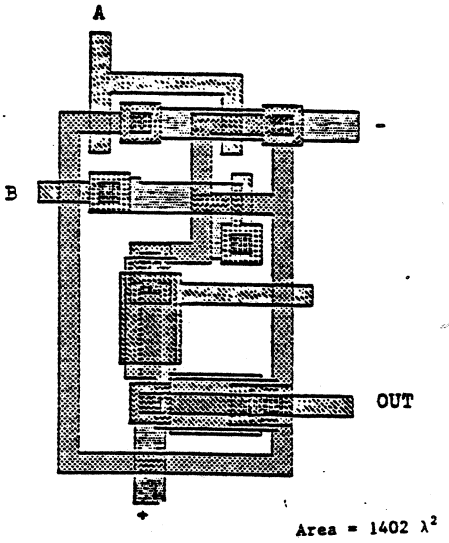


Figure 6: Compacted Layout of the XOR (G. S. Model)

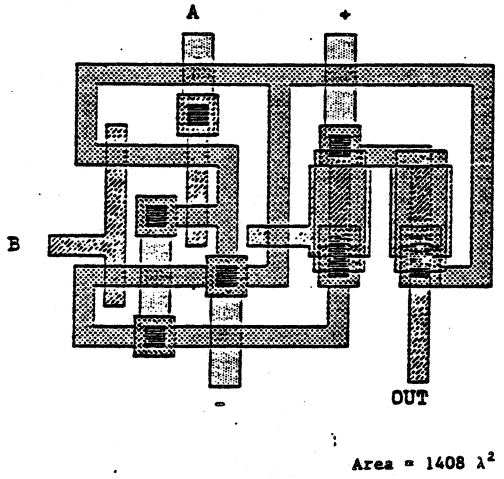


Figure 7: Compacted Layout of the XOR with "Poor" Topology.

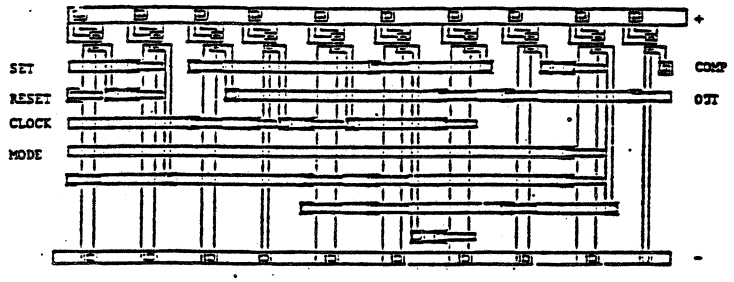


Figure 8: Johannsen's Layout of the Pulse Synchronizer.

We have also observed that, for a small circuit, minimizing the number of contacts in the layout would reduce the final layout area significantly. The area penalty of a contact is largely due to the fact that other features must be separated to make room for it.

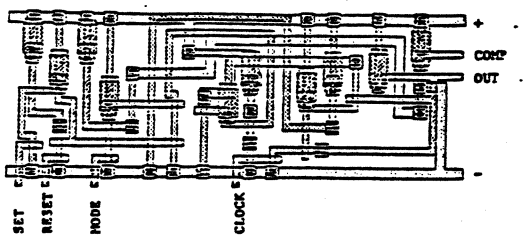


Figure 9: New Layout of the Pulse Synchronizer Circuit.

For a large circuit, "long distance" wires should 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. In practice, the number of long distance nets is only a small portion of the total number of nets.

Our original embedding algorithm does not distinguish long distance nets from other nets. The cost of a jumper to a long distance net is the same as the cost of other jumpers. If long distance nets are assigned to the metal layer, the area penalty for connecting to these wires via jumpers is small. The topology generated by our algorithm was initially poor. We resolved this problem by not assessing any penalty for wires crossing long distance nets.

V. Conclusions

We have applied a graph theoretic approach to the circuit embedding problem. With our graph model, there is only one graph for each circuit schematic. There is no need to decompose gate nets prior to the embedding of the circuit. We achieve this property by being less explicit, i.e., transistors are not represented explicitly. We have assumed that the "quality" of the embedding topology is independent of the valid nesting structure of the transistors with common gate signal.

We have also assumed the existence of an "ideal" layout compactor. Subject to the same boundary constraints, only different layout topologies will result in different compacted layouts. It remains to establish a computable measurement of the quality of layout topologies. Wolf [11] has observed that jumpers have an adverse effect on the overall layout area of small circuits. We have programmed a heuristic selection mechanism to limit the generation of layout topologies to the ones that result in the "fewest" number of crossovers. Different measurements of quality can be accommodated by modifying the heuristic selection mechanism.

Our experience, although limited, suggests that layouts corresponding to a machine generated topology may be more compact than hand layouts. It is a direct result of our graph model and the strategy of considering different topologies. A designer, without any computing aid to generate different topologies, usually considers only a few before focusing on one. It is likely that a superior topology may escape the

designer's vision, but it is considered by our algorithm. Our experience also suggest that the number of crossovers is a first order measurement of the quality of a topology, if area is of primary concern.

Our embedding algorithm, with a fixed starting vertex, does not enumerate all topologies even when the selection heuristic is removed. 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 enumerate all topologies with our algorithms.

An important aspect of the layout topology problem that is not fully addressed is the optimal assignment of orientation and planes. If the domain of the optimizing metric is the set of regions and exterior cycles of each plane, then 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.

Acknowledgement

The support of IBM Corporation and the Defense Advanced Research Project Agency under contract with the California Institute of Technology, and the Office of Naval Research under contract N00014-84-K-0043 with Yale University is gratefully acknowledged.

References

1. Aho A.V., Hopcroft J.E., Ullman J.D.. *Data Structures and Algorithms*. Addison-Wesley, 1983.

2. Engl, W.L., Mlynski, D.A., Pernards, P. "Computer-Aided Topological Design for Integrated Circuits". *IEEE Trans. Circuit Theory CT-20*, 6 (1973), 717-725.

3. Goldstein, A.J., Schweikert, D.G. "A Proper Model for Testing the Planarity of Electrical Circuits". *The Bell Sys. Tech. J. 52*, 1 (1973), 135-142.

4. Hopcroft, J., Tarjan, R. "Efficient Planarity Testing". *J. of the ACM 21*, 4 (1974), 549-568.

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

6. Lao, Y., Wong, C. "An Algorithm to Compact a VLSI Symbolic Layout with Mixed Constraints". *IEEE Trans. on CAD of IC and Sys. CAD-2*, 2 (1983), 62-69.

7. Ng, T. A Graph Model and the Embedding of MOS Circuit. T.R. 5104, Computer Science Department, California Institute of Technology, 1983.

8. Rose, N.A., Oldfield, J.V. "Printed-Wiring-Board Layout by Computer". *Electronics and Power* (Oct 1971), 373-380.

9. VanCleemput W.M. "Mathematical Models for the Circuit Layout Problem". *IEEE Trans. Circuits and Systems CAS-23*, 12 (1976), 759-767.

10. VanLier, M.C., Otten, R.H.J. "Automatic IC Layout: The Model and Technology". *IEEE Trans. on Circuits and Systems CAS-22*, 11 (1975), 845-854.

11. Wolf, W., Newkirk, J., Mathews, R., Dutton, R. Dumbo, A Schematic-To-Layout Compiler. Proc., Third Caltech Conference on VLSI, 1983, pp. 379-394.