Automatic Layout Synthesis of Leaf Cells

Sanjay Rekhi, J. Donald Trotter, and Daniel H. Linder

Microsystems Prototyping Laboratory
NSF Engineering Research Center, 2 Research Blvd.
Mississippi State University, Starkville, MS 39759

Abstract —This paper describes algorithms for automatic layout synthesis of leaf cells in 1–*d* and in a new 1–1/2–*d* layout style, useful for non–dual circuit styles. The graph theory based algorithms use concepts set forth by Euler and Hamilton to achieve two tasks. The transistor placement algorithm uses the Euler's theorem, while the placement of the groups of the transistors is achieved by using Hamiltonian graphs. Results show that the algorithms produce extremely competent layouts when compared to other algorithms in the literature and manual layouts.

I. Introduction

This paper addresses the layout generation of *functional cells*, defined as basic functional entities in a design, but more commonly called "cells." These are classified as one–dimensional (1–*d*) and two–dimensional (2–*d*) cells. A one–dimensional cell is an array of transistors in which all drain and source terminals lie along a single or double horizontal row(s) of diffusion and the growth of the cell is in one direction only, usually along the horizontal axis. Two diffusion arrays are typically used for a CMOS technology, one for the pull–down circuit (Ndiffusion) and the other for pull–up (Pdiffusion). In the two–dimensional cells, the Ndiffusion and Pdiffusion are not restricted to single horizontal lines and growth is in both directions, i.e., along the horizontal and the vertical axes with multiple rows of Pdiffusion and Ndiffusion. Algorithms which automatically generate the symbolic representation of the functional cells from a schematic representation have been described. In a symbolic representation of a cell, various components such as the transistors and the nets are placed on a virtual grid relative to the placement of other devices rather than at fixed locations defined by the process design rules. Different design rule dependent views or physical views can then be compiled from the symbolic view for a given process technology.

The algorithms are capable of generating the functional cells in the one–dimensional layout style and also in a new layout style, which is considered to be one–and–a–half–dimensional (1–1/2–*d*). In 1–1/2–*d* the Pdiffusion is not restricted to be placed in a single row. Some Pdiffusion could be placed in the Ndiffusion row; similarly, the Ndiffusion is also not restricted to be placed in a single row. Thus, there is some restricted growth along the vertical axis. Fig. 1 shows the layout style for the 1–*d* functional cell and the new 1–1/2–*d* functional cell.

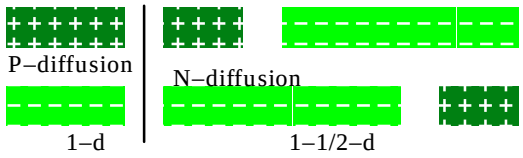

Fig. 1 One and One–and–a–half Dimensional Functional Cell

32nd ACM/IEEE Design Automation Conference ®

The new layout style allows functional cell layout generation usable by standard place–and–route packages for different logic implementations. This paper describes graph theory based algorithms for the task of transistor placement and for the placement of the connected groups of transistors. The transistor placement task is based on the popular Euler's theorem and the task of group placement is influenced by the Hamiltonian graphs.

The important parameters that are used to traverse a graph are trail and path. A trail is defined as an alternating sequence of vertices and edges for which all edges are unique, and a path is defined as a sequence for which no vertices are repeated. A non–trivial closed trail (order $\geq$ 3) is called a circuit and a circuit where all except two vertices (start and end) are distinct is a cycle. Eulerian trail of a graph G is defined for a connected graph G as an open trail containing all the edges of G. A cycle of a graph G containing every vertex of G is called a Hamiltonian cycle of G; thus, a Hamiltonian graph is a graph that possesses a Hamiltonian cycle.

II. Comparison with Other Research

Earlier algorithms developed by Uehara and Cleemput [1], Madsen [2], and Nair *et al.* [3] can be characterized by their use of the Euler's theorem to find the dual–Euler paths in the circuits of series–parallel nature where the pull–up and the pull–down circuits are duals of each other. These algorithms do not always find the optimal results. They ignore routing and heuristically determine the minimum number of dual–Euler paths. The mathematical basis provided by Chakravarty [4] determines that choosing an ordering of transistors such that the number of routing tracks are minimized is an *NP*–hard problem. This mathematical analysis is based on the Euler theorem and assumes series–parallel circuits where the pull–up and the pull–down circuits are duals of each other.

Other algorithms described in [5]– [10] are based on two fundamental operations: transistor pairing and placement of the paired groups, the only differing characteristic being how the transistor pairs are chosen and how the groups are placed. Wimer *et al.* [5] first pair the PMOS and the NMOS transistors to share a vertical column and then form diffusion chains by using an exhaustive search. If the total number of chains is less than or equal to five, the group of chains that minimize the overall circuit cost is found and placed linearly by using an exhaustive search. If the number of chains is greater than five, then a random placement is performed. Chen and Chow [6] also begin by pairing the PMOS and the NMOS transistors in three steps. A weighted bipartite matching operation is performed to pair the transistors that were not paired either in the circuit netlist preprocessing step or in the polysilicon gate based pairing step. A diffusion sharing graph (DSG) is derived from the pairs, where each vertex represents a pair and an edge indicates sharing of diffusions between the pairs. The DSG is modified to obtain an adjacency graph from which the authors derive Hamiltonian paths by using a branch–and–bound algorithm. The orientation of the transistors is determined by scanning the placement from left–to–right followed by a scan from right–to–left.

Ong *et al.* [7] also begin by pairing the two types of transistors, followed by formation of the diffusion lines by using a branch–and–bound algorithm. A diffusion line is formed by a linear placement of the

transistors in the circuit. These lines are then placed linearly; wide transistors are broken into smaller ones and routing is performed. Poirier *et al.* [8] also begin by pairing the two types of transistors to produce a 2–*d* layout of the cell. Hwang *et al.* [10] say that since the height of the cell is user defined, the layout area can only be minimized by minimizing the width of the cell. The authors disagree with this claim [10] since the height is also important in many cases. The authors view that the layout area of the cell is reduced by minimizing the width and the height of the cell, where the most effective way of reducing the width is by minimizing the diffusion gaps with proper placement of the transistors. This also affects the routing density which affects cell height or permissable transistor width. In the synthesis process described by Hwang *et al.*, the authors maintain the user defined height by folding the transistors (which increases the width of the layout) and/or by using cell templates where the horizontal metal–1 or metal–2 [10] power busses are placed in the middle of the cell.

Algorithms based on expert systems have also been devised. These algorithms use a rule based transistor pairing and placement [11]. The exact algorithms described in [9] use various classifications of the dual paths for circuits of height four or less (maximum of four transistors in series) to determine the transistor placement which provides the minimum number of gaps and routing channels.

The algorithm described by Madsen in [2] uses the duality principle to find the Euler's path for various user specified constraints, e.g., fixed i/o pin placement, fixed netlist or changeable netlist. The series and parallel subgraphs are reduced. The path tracing is initiated in the parallel graph, where all possible paths between two end points are found. Then the series graph (dual of the parallel graph) is traced to find the maximum dual–Euler path(s). For the changeable netlist configuration, the edges (transistors) in the series graph (series network) can be swapped as long as the Boolean logic function is preserved. A gap is inserted if the next vertex (circuit node) cannot be reached. The solution which has the minimum number of dual–Euler paths is chosen as the final solution.

Although our transistor placement algorithm appears to be very similar to the algorithm described by Madsen [2], it is fundamentally different; it does not assume duality and does not expect the pull–up and the pull–down circuits to have the same number of transistors, or even the same number of input signals. No graph reduction is possible because of the above assumptions — unlike the algorithm described in [2] which assumes that parallel connected transistors will be series connected in the dual network and vice–versa. Unlike the previous algorithms, which begin with transistor pairing, our algorithm begins with a vertex (circuit node) alignment. The best transistor pair for the selected vertices is then determined. This is different from the delayed binding procedure described in [2] which finds a linear placement of the transistors in the series graph for all possible linear transistor placements in the parallel graph.

The transistor netlist is expected to be non–changeable. No reordering is attempted as the layout generator is expected to be driven by an optimizer described in [12] that produces optimized schematics considering the global constraints, e.g., performance requirements.

III. Process of Layout Synthesis

Fig. 2 shows the block diagram of the entire layout synthesis process. The process consists of three major parts: Placement, Routing and Rendering. The first part is the placement program, which consists of graph determination and partition phase, the placement of transistors for each partition and the placement of the partitions (or subcells). The output of the placement program is a partial symbolic view for which the transistors are placed on a virtual grid. The second part is the routing program which operates on the partial symbolic view and creates the routed symbolic view of the cell. Our goal is to enable the user to edit directly the symbolic view to improve the results of the placement and the routing programs since additional design considerations may be important, e.g., performance vs poly resistance. The optimized symbolic view of the cell is the input to the rendering program, which creates the detailed layout of the cell based on user defined properties, e.g., width of the power busses, height of the cell, etc. The design rules and the final device sizes are also input to the rendering program. The symbolic view is created by assuming a standard device size. If a final device size is larger than the "permissable" device size, the placement and the routing steps are rerun with transistor folding, if necessary.

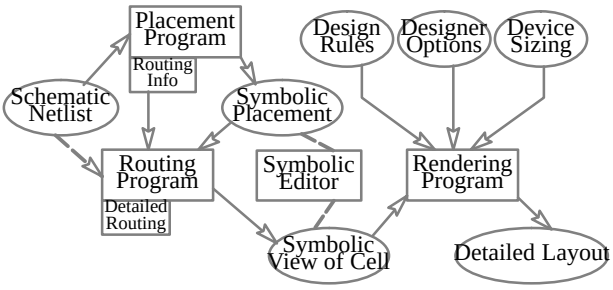

Fig. 2 Layout Synthesis Process

The objective of the transistor placement algorithm is to minimize the number of diffusion gaps in the circuit while trying to minimize the number of routing channels required to form the interconnect among the transistors. Fig. 3 illustrates the diffusion gaps and the alignment of two transistors so that the polysilicon can be on the same vertical column. Other transistor alignments are also possible as shown in the left hand side of the figure, common in multiplexer gates, where the polysilicon gate is not used for alignment. The figure also shows the horizontal routing tracks.

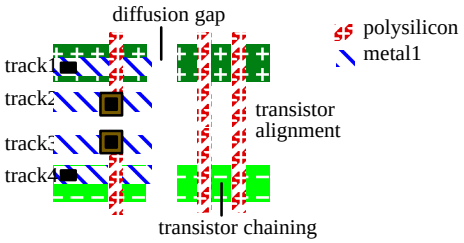

Fig. 3 Related Terminology

IV. Algorithms for Transistor Placement

A. Graph Transformation and Circuit Partitioning

The layout synthesis process begins with a circuit transformation procedure, for which the vertex set of the graph is defined by the circuit nodes and the edge set is defined by transistor labels since there may be more than one edge per vertex pair. For the graph representation, each gate input defines a row which contains two sets of edges, one for the pull–up network and another for the pull–down network. The transistor source and drain connections, the standard size of the transistors etc., are also stored in the representation. The vertex lists are separately formed and are called pverts and nverts for the two networks. The degree of the vertices, the type of the vertex, etc., are stored in the vertex list. Every vertex is classified as internal, external or supply. The supply nodes of the circuit are represented by supply vertices. An external vertex is a circuit node common to both the networks, and a vertex which is neither a supply nor an external vertex is an internal vertex. If an edge exists

between two external vertices, then both the vertices are reclassified as xfer vertices. The common edges, or the transistors, for this arrangement form a transfer gate. This structure has a versatile use and is extremely popular in the design of latches, flip–flops, multiplexers, etc.

For a cell consisting of smaller sub–functions, the circuit is partitioned into subcells. For this phase, unique groups (or sets) of transistors are formed such that the intersection of the vertex lists of the subcells yields either a null set or a set of supply vertices. Fig. 4 shows the circuit schematic of a JK flip–flop cell. The JK flip–flop is made of many different small subcells. The transistors that are connected at source or drain to other transistors are considered as strongly connected transistors. The supply connections are not considered to define the strong connections among transistors since the various subcells may share Vdd/Gnd nodes.

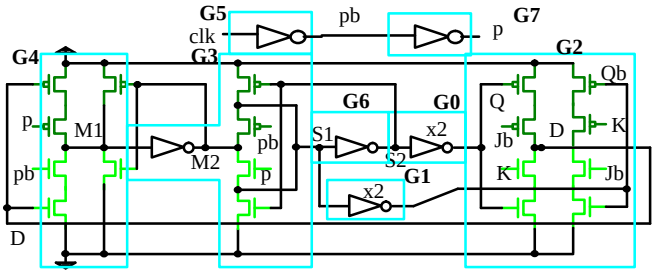

Fig. 4 JK Flip Flop Circuit Schematic

A recursive graph–partitioning algorithm is used to collapse all parallel transistors to form a group of series connected vertices. All vertices that are adjacent to the non–supply vertices in the series–group are found. This provides all the sub–graphs for the subcells that are contained within the cell of interest. All of the sub–graphs in the pull–down circuit set are compared with the sub–graphs in the pull–up circuit set to find the subcells that have common external or xfer vertices. These groups from the pull–up set and the pull–down set that have common external or xfer vertices are merged to form a single subcell and get a unique subcell–id number. The various graphs derived for the JK flip–flop cell after graph partitioning are shown in Fig. 5.

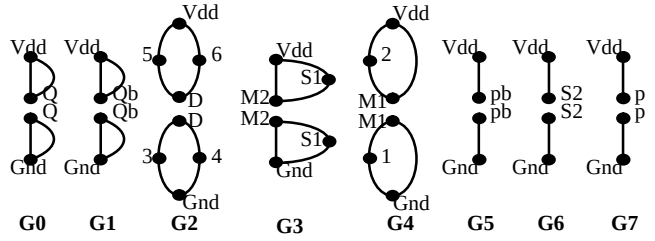

Fig. 5 Various Groups in the JK Flip Flop

B. Transistor Placement Algorithm

The transistor placement algorithm operates on a subcell or a group at one time and begins by finding a set of best vertex pairs (v_i, v_j) that potentially provides the minimum set of trails. The two vertices are usually chosen such that $v_i \in$ {pull–up circuit nodes} and $v_j \in$ {pull–down circuit nodes}, but the selection could be such that $v_i, v_j \in$ {pull–up circuit nodes} or $v_i, v_j \in$ {pull–down circuit nodes}. This flexibility of choice of vertices has not been observed in any of the previous algorithms. This flexibility, coupled with the modularity of the algorithms, provides a unique capability of being able to generate layouts for various logic families. Two representative logic families are chosen to demonstrate this unique capability: the most common and rich static CMOS logic family with non–dual multiplexer gates, series–parallel or non–series–parallel transistor interconnections, and the new CPL logic family. The logic style is derived from the circuit graphs, and the layout architecture is determined, 1–*d* or 1–1/2–*d*. For the 1–*d* layout architecture both the P–graph and N–graph are considered simultaneously; whereas for the 1–1/2–*d* layout architecture, the graphs are considered separately.

The weight of the vertices is calculated based on the degrees of the vertices and circuit configurations. The vertices with a degree of 1 have the highest weight, followed by the remaining odd vertices which have a lower weight in descending order of degrees. The even degree vertices have the lowest weight in the descending order of degrees. The vertex classification, supply, xfer, external and internal also influence the weighting criteria. Depending on the distribution of the vertex weights one of pverts or nverts is chosen as the preferred vertex list. For the 1–*d* layout architecture, all vertices in the preferred list are compared to the vertices in the other vertex list to obtain a set of two vertices. The set contains one vertex from the pull–up circuit and the other from the pull–down circuit, which have to be aligned. For the 1–1/2–*d* layout architecture, the vertex set is derived from a single vertex list. The set of two vertices found above for both the layout architectures is called the seed vertex pair.

A set of transistors (S_T) is found for the seed vertex pair, such that at least one of the source/drain terminals of the transistor pairs corresponds to the vertex pair, and both the transistors have the same subcell–id. The transistor pair that forms a multiplexer gate is preferred. Otherwise, the transistors are aligned based on the gate terminals. If a suitable transistor pair is not found, then all the transistors in S_T are searched to align the vertex pair, and consequently the transistor pair. If a suitable transistor pair is not found, then another seed vertex pair is chosen and the process is repeated.

After the transistors have been aligned, the next vertex pair is derived from the transistor pair and the process is repeated until all the transistors in the group have been covered. If incident edges cannot be determined, then a gap is inserted and the process begins by choosing a new seed vertex pair.

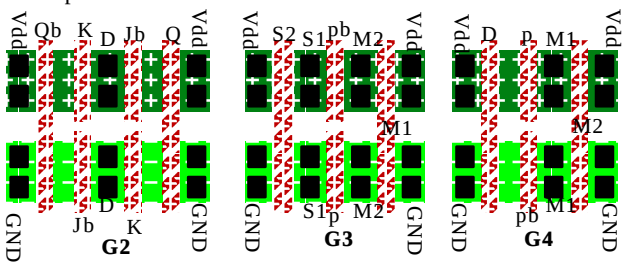

Fig. 6 Transistor Placement for Some Subcells in JK Flip Flop

The trail determination problem for minimum routing tracks is harder when the pull–up and the pull–down circuits are considered simultaneously. If the number of PMOS transistors in a circuit are considerably larger than the number of NMOS transistors or vice–versa, then the pull–up and the pull–down circuits are considered separately, enforcing a 1–1/2–*d* layout style. This style provides an area improvement over the 1–*d* layout style if the difference in the number of transistors is six or more for the MOSIS scalable technologies. No improvement in area is attained if the difference is four. For the difference of less than 4, the 1–1/2–*d* layout style requires more area than the 1–*d* layout style due to the spacing rule between the two types of diffusions. Fig. 6 shows the transistor placement results for a few selected groups from the JK flip–flop. The layouts of the other groups in the JK flip–flop are trivial and are not shown. In the group G_3 the signals S_1 and M_2 form xfer vertices and are aligned as shown. In groups

G_2 and G_4 there are no xfer vertices, but the correct pairs of transistors are aligned.

C. Group Placement Algorithm

The transistor placement algorithm can be repeatedly called for the cells that consist of subcells, as in the JK flip–flop. The transistor placement algorithm results in sets of unbroken diffusion chains called groups. The number of groups may not be the same as the number of subcells because a subcell may result in two or more broken diffusion chains. The re–grouping algorithm is used to find a linear placement for the groups such that for any vertical cut across the cell, the number of nets crossing the cut is minimized. The nets within a group (internal nets) and nets among groups (external nets) are used to calculate the value of the cut. The external nets (excluding the power supply nets) are further classified as a single external net used to interconnect two groups and a multiple external net that connect more than two groups.

A Group Adjacency Graph (GAG) is derived for this algorithm. The various groups form the vertex set and the edge set is defined by the interconnections among the groups. Fig. 7 shows the group adjacency graph for the JK flip–flop. Complete subgraphs are formed by using multiple external nets and then pruned such that the maximum adjacency between groups is preserved. The pruned graph is called the Minimal Group Adjacency Graph (MGAG), as it preserves the maximum adjacency and is minimum with respect to the property of maximum adjacency. Fig. 7 also shows the MGAG for the JK flip–flop.

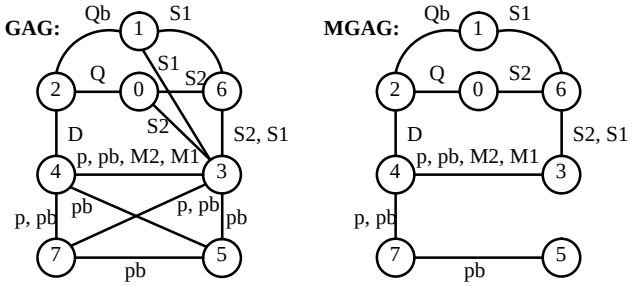

Fig. 7 Group Adjacency Graphs

A linear ordering of the groups is found from the MGAG to minimize the maximum number of net crossings across a group boundary. This task is performed by using a branch–and–bound algorithm which begins by identifying a set of vertices which have the maximum weight (internal nets) and minimum degree (default mode). Alternatively, the user can also force the selection of the seed vertices by defining a global parameter that makes all the vertices in the MGAG the seed vertices.

Each seed vertex starts a solution, the vertices adjacent to the seed vertex are examined to find the one with the maximum degree. If many vertices have the same degree, then weights are examined to find the next solution point. If many vertices have the same weight and degree, then the solution is branched. This recursive process continues until all vertices have been included or until the solution cannot be grown any more (incomplete solution). Each incomplete solution is examined for the remaining vertices to find the complete solutions by using a greedy algorithm, which modifies the incomplete solution. Each non–included vertex is greedily put in the incomplete solution such that the increase in the value of the cut is minimum for each iteration. The final solution for the JK flip–flop is shown in Fig. 8. The solution has been obtained by using the default seed vertex selection. The branch–and–bound algorithm creates only two orderings, ($G_5, G_7, G_4, G_3, G_6, G_0, G_2, G_1$) and ($G_5, G_7, G_4, G_3, G_6, G_1, G_2, G_0$).

A recursive algorithm is used to scan the ordered list to merge the diffusion gaps. For n ($n \geq 3$) groups, the second group is checked against the first and the third. A cost for mirroring group one and/or group two based on the number of nets crossing the boundary is estimated. Then the third group is considered and is compared with the fourth group to calculate the cost for mirroring group three. This process continues for all the n groups in the cell. The recursive algorithm takes into account merging two groups and considers the effect of the new merged group over the entire cost. Thus, many cost values are calculated. The recursive algorithm continues until no more merging can occur. A linear scan follows the previous recursive scan to mirror and merge the groups based on the previously generated cost values. An interactive capability is also provided to control the group placement and merging. The relative position for each group is specified by the user which determines the placement and the routing.

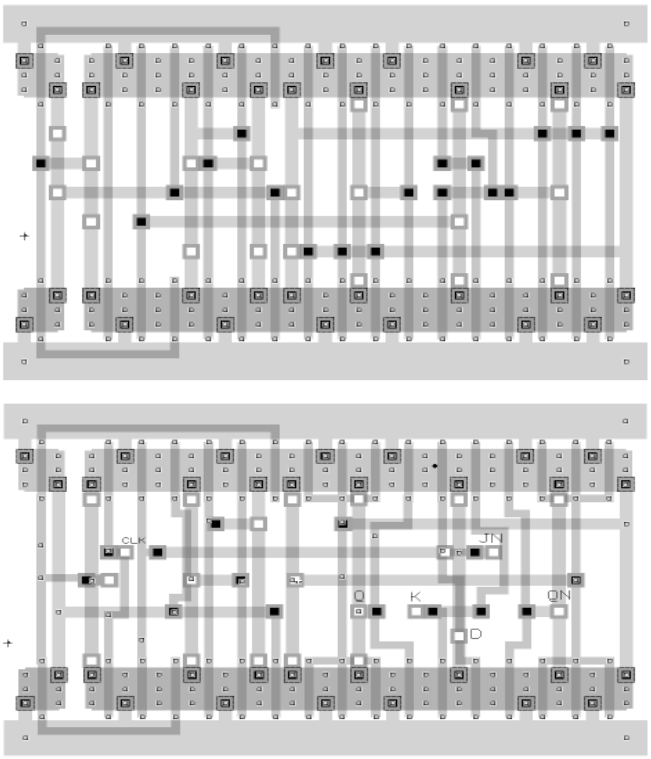

Fig. 8 Layouts of the JK Flip–Flop

V. Routing

Initially, routing was achieved using YACR which is a symbolic channel router. Only a single routing area was used; thus all the routing was in the middle of the two diffusion areas. YACR is a two layer router typically used for routing standard cells with two layers of metal. Some post–processing was added to support the routing in polysilicon layer.

Support for two additional types of routing has been created: metal1 routing over the transistor active areas and poly routing in the corridors under the metal1 supply buses. The former takes advantage of silicided processes. The second is effectively restricted to river routing with three basic connections: poly wire to poly gate, to poly wires running vertically through diffusion gaps, and to poly metal contacts. Since the latter connection cannot be placed under the power bus, it has only limited use. Four general types of wiring have been identified as being reasonable for poly interconnects: low performance asynchronous set and reset connections, high resistance feedbacks for latches, reasonably local internal clocks for multiplexers, and local gate shorts between

neighboring transistors. The extra poly resistance in the local clock coming off of the first clock inverter for a flip–flop is actually an advantage in delaying the signal while the other inverter is responding.

Because the original notion that cell terminals would be placed during the rendering process led to significant rerouting which canceled the benefit of previously generated symbolic views of cells, a program (bipartite matching) was created to place the terminals after transistor placement. This has led to partitioning the channel into two or even three horizontal slices with one or two rows of terminals in the middle. Fundamentally, the routing required violates the basic notion of YACR's two layers on a uniform grid. However, effort is continuing relative to extending and controlling YACR, e.g., track assignment and the layer assignment algorithms.

Other routers, e.g., Mentor Graphics' maze router based Microroute, are also being investigated. Our initial Microroute results for complex cells are less than satisfactory for creating the symbolic view since there appears to be little "strategic planning"; however, the maze router results which are design rule correct may fit in fine for the rendering problem, given the specific design rules.

Fig. 8a shows the symbolic view for the JK flip–flop created with YACR. The symbolic view has only one diffusion gap and contains five horizontal metal–1 routing tracks. Fig. 8b shows a manually edited version with four routing tracks.

VI. Results

These algorithms have been implemented in GENIE, which provides an excellent prototyping environment, but like any other interpretive language, has large execution time. The comparison between execution times for different GENIE and C programs to implement the same task was performed with the GENIE programs averaging a factor of 7.41e7 times slower than the C programs. The GENIE programs are executed from within the Mentor Graphics design environment. The maximum recorded time to obtain the complete solutions for the cells described in the previous algorithms while running Led on a Sun Sparcstation 2 was 20 seconds. For a ring oscillator circuit consisting of 51 two–input NAND gates (204 transistors), where each NAND gate is enabled with a common signal, the algorithm takes 17 minutes to find the solution. This circuit implementation style was chosen so that the worst case branch–and–bound results can be measured. For the same ring–oscillator where only the first gate is enabled and the remaining gates have both the inputs tied together, the algorithm produces the complete solution in 9.4 minutes.

The results produced by the new algorithms have been compared with the results from the existing algorithms described earlier. TABLE I shows the comparison of the solutions obtained from our algorithm and the algorithms in the literature. The algorithms are divided in the following classes, which represent the type of circuits the algorithms expect as input.

1. Single Series–Parallel CMOS with dual pull–up and pull–down circuits and equal number of transistors in both the circuits.
2. Single Series–Parallel CMOS with dual pull–up and pull–down circuits with transistor folding.
3. Multiple Series–Parallel CMOS with dual pull–up and pull–down circuits and equal number of transistors in both the circuits.
4. Multiple Series–Parallel CMOS with dual pull–up and pull–down circuits with transistor folding.
5. Multiple CMOS with dual or non–dual (pass–gate) pull–up and pull–down circuits with transistor folding.
6. Multiple CMOS with non–planar pull–up and pull–down circuits without transistor folding.
7. Non Series–Parallel CMOS.
8. General Class of CMOS Circuits.

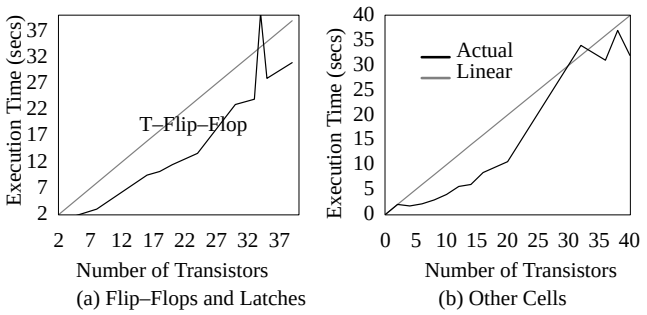

Fig. 9 Execution Time vs Complexity

The algorithms described in this paper implement the category of general class of CMOS circuits, do not assume any particular logic style, and determine the correct layout style, 1–*d* or 1–1/2–*d* for each subcircuit.

TABLE I

Comparison of Results

Reported Algorithm						Our Algorithm				
Author	**CC**	**NG**	**PG**	**RT**	**Q**	**NG**	**PG**	**RT**	**Q**	**ET**
[6]–Fig 10b	5	1	1	4	5	2	1	3	5	8
[6]–Fig 11b	5	2	2	4	6	1	1	3	4	11
[2]–Fig. 5	1	0	0	5	5	0	0	5	5	4
[9]–Pg. 66	3	0	0	4	4	0	0	4	4	6
[9]–Pg. 72	3	0	0	4	4	0	0	4	4	7
[9]–Pg. 78	7	2	2	8	10	3	3	4	7	8
[9]–Pg. 96	3	0	0	5	5	0	0	5	5	3
[9]–Pg. 126	3	1	1	3	4	0	0	5	5	7
[9]–Pg. 151	3	0	0	6	6	0	0	2	2	4
[9]–Pg. 151	3	0	0	9	9	2	2	5	7	12
[9]–Pg. 152	3	2	2	5	7	2	2	5	7	9
[9]–Pg. 127	3	0	0	3	3	0	0	2	2	2
[5]	5	0	0	–	–	0	0	7	7	12
[11]	5	3	3	5	8	2	2	5	7	9
[13]–Fig. 7.7	8	1	1	5	6	0	0	2	2	4
[13]–Fig. 8.5	8	3	3	10	13	3	3	5	8	12
NA	8	–	–	–	–	1	1	6	7	5
NA	8	–	–	–	–	1	1	4	5	6
NA	8	–	–	–	–	1	1	6	7	4
NA	8	–	–	–	–	1	1	4	5	5
NA	8	–	–	–	–	2	2	8	10	9

The table shows the author of the previous algorithm and the classification of the algorithm CC (1–8 above). The number of diffusion gaps (NG, PG) and the number of routing tracks (RT) are used as the basis of comparison. The quality factor Q, is the sum of the routing tracks and the diffusion gaps. The execution time (ET) for our algorithm to obtain the solution is also provided in the table.

On an average, about 40% of the total time to execute the three portions of the synthesis process has been spent on the graph build and circuit partitioning portion. Fig. 9–(a) shows the relationship between

the average execution time in seconds to obtain the layout versus the number of transistors for the latches and the flip–flops in the ITD/MSU standard cell library. Fig. 9–(b) shows the same for the remaining cells in the library, including the full–adder, clock generation cell and other series–parallel and non–series–parallel circuits.

Although a worst case analysis of the algorithms under all of the worst case conditions leads to a worst case timing vs complexity of $O(T^7)$ where T is the transistor count, it has not been observed. For the T–flip–flop with a reset function in the ITD/MSU cell library some of the groups do not have an xfer vertex, and the pull–up and the pull–down circuits do not have common gate signals; but the next vertex pair can be merged. This means that a new seed vertex pair will not have to be found for every transistor pair. The execution time observed from Fig. 9–(a) is therefore worse for this cell than for any other cell. In practice the execution time is roughly linear with complexity.

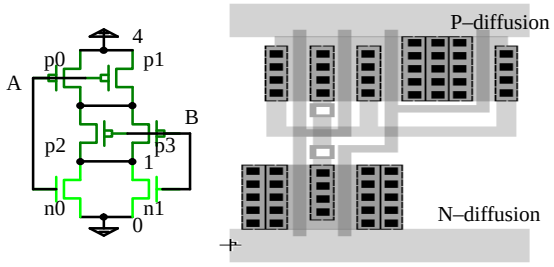

Fig. 10 NOR Gate with Folded PMOS

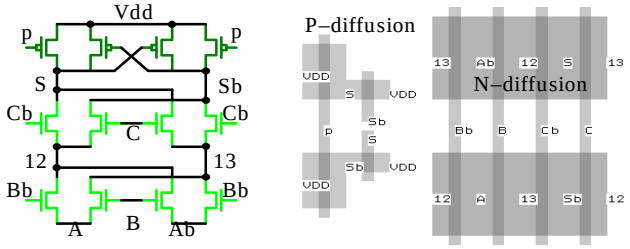

Fig. 11 Half Adder in CPL

VII. Conclusion

The methodology for automatic layout synthesis of a general class of CMOS functional cells and the necessary placement algorithms to implement this process have been described in the paper. The additional support for the 1–1/2–*d* layout style facilitates efficient use of layout area for logic styles with mostly NMOS or mostly PMOS transistors. Fig. 10 shows the placement result for a two input NOR gate where each PMOS transistor is individually folded. Fig. 11 shows the placement result for a half–adder in the CPL logic style and its 1–1/2–*d* solution. Even with all the extra features that are not found in most of the other algorithms, our algorithms provide extremely competitive results when compared to other algorithms in the literature and to handcrafted layouts. While many algorithms have claimed linear order of execution time, their solution spectrum is limited to only static single cell series–parallel circuits. The algorithms described in this paper are not only quite flexible in supporting various circuit styles, but are also run–time efficient, being roughly linear with complexity.

Although the layout generator results are essentially as good as hand layouts for medium or smaller cells, the group placement algorithm needs improvement for the complex cells to compete with handcrafted layouts. As one should expect, the group placement has a tremendous impact on routing. Presently, where handcrafting leads to perhaps four horizontal tracks in a flip–flop, the automatic generation leads to perhaps 5 tracks. Consequently, plans are in place to incorporate an improved weighting scheme based on the following:

1. Reduce solution space consistent with minimum cell width by determining minimum possible number of diffusion gaps and by maximizing power supply sharing between neighboring groups, i.e., strategically place the groups which contribute gaps.
2. Consider intragroup interconnections with a weight based on fractional distances across the group width to support sharing of tracks.
3. Consider the impact of poly routing under the power buses and metal 1 routing over the diffusions relative to the wiring cross–section in the channel with due consideration for performance.
4. Consider terminal placement and associated interconnections.

These considerations lead in the case of the JK flip–flop to the group placement: (G7,G5,G4,G3,G0,G2,G1,G6).

For general usability, the programs will also have to be re–coded from GENIE to a portable and more powerful language, e.g., C^{++}. The transistor placement algorithm lends itself very well to parallel execution if the cell consists of multiple groups. The solution for each group can then be found separately. The branch–and–bound algorithm used for the group placement algorithm also lends itself very well to a parallel program. The different seeds selected in the algorithm can be used to grow the solutions on separate computers to obtain the complete solutions to the group placement problem. Our long–term goal is to utilize parallel programming to obtain the layout for complete row(s) of cells and to extend our 1–1/2–*d* algorithm to a 2–*d* algorithm.

References

[1] Uehara T., and Cleemput W. M., "Optimal layout of CMOS functional arrays," *IEEE Transactions on Computers*, Vol. C–30, p.p., 305–312, May 1981.

[2] Madsen J., "A new approach to optimal cell synthesis," *IEEE International Conference on Computer–Aided Design*, p.p., 336–339, November 1989.

[3] Nair R., Bruss A., and Reif J., Linear time algorithms for optimal CMOS VLSI: algorithms and architectures, (P. Bertolazzi and F. Luccio edition), Amsterdam, Elsevier North–Holland, p.p., 327–338, 1985.

[4] Chakravarty S., He X., and Ravi S. S., "Minimum area layouts of series–parallel transistor Nnetworks is *NP*–hard," *IEEE Transactions on Computer–Aided Design*, Vol. 10, No. 7, p.p., 943–949, July 1991.

[5] Wimer S., Pinter R. Y., and Feldman J., "Optimal chaining of CMOS transistors in functional cells," *IEEE Trans. on Computer–Aided Design*, Vol. 6, No. 5, p.p. 795–801, September 1987.

[6] Chen C. C. and Chow S.–L., "The layout synthesizer: an automatic netlist–to–layout system," *IEEE 26th Design Automation Conference*, p.p. 232–238, June 1989.

[7] Ong C.–L., Li J.–T., and Lo C.–Y., "GENAC: An automatic cell synthesis tool," *IEEE 26th Design Automation Conference*, p.p. 239–244, June 1989.

[8] Poirier C. J., "Excellerator: custom CMOS leaf cell layout generator," *IEEE Trans. on Computer–Aided Design*, Vol. 8, No. 7, p.p. 744–755, July 1989.

[9] Maziasz R. L. and Hayes J. P., Layout minimization of CMOS cells, Kluwer Academic Publishers, Boston MA, 1992.

[10] Hwang C. Y., Hsieh Y. –C., Lin Y. –L., and Hsu Y.–C., "An efficient layout style for two–metal CMOS leaf cells and its automatic synthesis," *IEEE Transactions on Computer–Aided Design*, Vol. 12, No. 3, p.p., 410–424, March 1993.

[11] Kollaritsch P. W., and Weste N. H. E., "TOPOLIZER: An expert system translator of transistor connectivity to symbolic cell layout," *IEEE Journal of Solid–State Circuits*, Vol. SC–20, p.p., 799–804, June 1985.

[12] Rekhi S.,"Automatic layout synthesis of leaf cells," Ph. D. Thesis, Mississippi State University, December 1994.

[13] Baltus D. G., and Allen J., "SOLO: A generator for efficient layouts from optimized MOS circuit schematics," *IEEE 25th Design Automation Conference*, p.p., 445–453, June 1988.