Optimal Chaining of CMOS Transistors in a Functional Cell

SHMUEL WIMER, RON Y. PINTER, MEMBER, IEEE, AND JACK A. FELDMAN

Abstract—**We describe an algorithm that maps a CMOS circuit diagram into an area-efficient, high-performance layout in the style of a transistor chain. It is superior to other published algorithms of this kind in terms of the class of input circuits it accepts, its efficiency, and the quality of the results it produces. This algorithm is intended for the automatic generation of basic cells in a custom or semicustom design environment, thereby removing the burden of arduous mask definition from the designer. We show how our method was used to compose cells in a row into a functional slice (e.g. an adder) that can be used in, say, a data path.**

I. Introduction

IN THEIR SEMINAL paper [10], Uehara and van Cleemput suggested a layout paradigm for CMOS circuits that is geared towards minimizing the diffusion required in the artwork. Their primary concern was to *chain* both the p-type and n-type transistors in such a way that their source or drain ports abut as much as possible, thereby also minimizing the need for additional intracell routing. In order to make the most of this method, good algorithms are needed to generate chains that maximize the occurrences of diffusion ports adjacencies, and then find the smallest number of chains that realize the circuit.

The functionality of such an algorithm is to map a circuit diagram (topology) into a layout (geometry). Normally, the circuit is given as a graph, describing the connectivity of the transistors. The layout is produced as a stick diagram which can be then instantiated into mask data by a circuit compactor. Both the algorithm suggested in the original paper and later work (e.g., [8]) require that the n-part and the p-part of the circuit graph be mutual duals. Moreover, the algorithms of [7] and [8] require that it be a series–parallel graph. On the other hand, the recent work reported in [3] and [6] does not guarantee optimal results in terms of minimum diffusion or any other optimization criterion (although it tries to cut down on interconnect in general).

In this paper, we relax any constraining conditions on the circuit graph and provide a general algorithm that can handle *arbitrary* graphs *optimally*. This is done by allowing a p-type and an n-type transistor to share a *column* (i.e., have the same position in a chain) not only when they are complementary, but also when they have a source or drain port in common. We then formulate a sequence of combinatorial problems that faithfully represent all the optimization considerations, and devise efficient algorithms for their solution.

The results of our technique are longer chains with sparser intra- and interchain routing problems. For example, the CMOS full-added circuit shown in Fig. 1 (from [5, p. 92]) was laid out in *one* chain, as shown in Fig. 2, which is not possible under the constraints of the other algorithms. Moreover, our framework can handle secondary optimization criteria, such as minimizing intracell routing density, gracefully and efficiently. Our algorithm has been analyzed and implemented; it runs almost instantaneously on circuits with up to a few dozen transistors.

The rest of this paper is organized as follows. Section II gives a precise formulation to the problem (including the target layout style), and Section III provides comparison to other work. Then we describe our algorithm in Section IV, exemplify its usage in the context of a functional slice generator (Section V), and conclude with results and possible extensions in Section VI.

II. Definitions

A CMOS circuit is given by a list of its constituent transistors. Each transistor has a *type*, **p** or **n**, determining whether it is to be realized in an n channel or a p channel, respectively. It also has three ports: a *gate*, a *source*, and a *drain*. The interconnection between transistors is defined in terms of labels that are attached to ports. Each such label defines a subset of ports, called a *net*. All ports in a net must be electrically connected, and ports with disjoint labels must be isolated. Fig. 1(b) shows a textual description of the circuit given in Fig. 1(a), where the labels in each transistor description are attached to the gate, source, and drain, in this order.

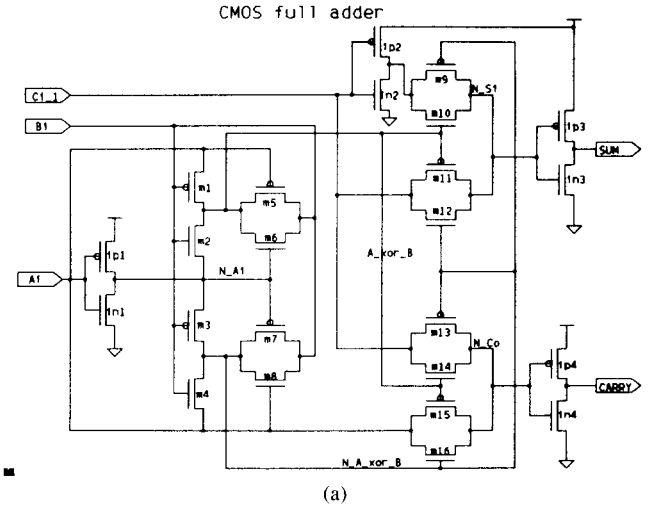
The layout generated by our algorithm is in the style proposed by Uehara and vanCleemput [10]; namely, the transistors are arranged in a *row* (or an array) of transistor pairs. Each pair consists of one p-type and one n-type transistor, occupying a *column*. The two transistors are aligned vertically at the centers of their gates which are realized by vertical strips of polysilicon. If the gates of the two paired transistors belong to the same net (which is not necessarily the case), then the two strips are connected straight through the column.

Diffusion runs horizontally along each side of the row: the n-doped on the top (with p-type transistors along it) and the p-doped on the bottom. Whenever they belong to the same net, adjacent diffusion ports (i.e., the source or

Manuscript received October 20, 1986; revised March 9, 1987.

The authors are with the IBM Israel Scientific Center, Technion City, Haifa 32 000, Israel.

IEEE Log Number 8715657.



(a)

name	type	gate	source	drain
IP4	P	N_CO	VDD	CARRY
IN4	N	N_CO	CARRY	GND
IP3	P	N_SI	VDD	SUM
IN3	N	N_SI	SUM	GND
M1	P	BI	AI	A_XOR_B
M3	P	BI	N_AI	N_A_XOR_B
M9	P	N_A_XOR_B	1	N_SI
M10	N	A_XOR_B	1	N_SI
M11	P	A_XOR_B	CI_1	N_SI
M12	N	N_A_XOR_B	CI_1	N_SI
M13	P	N_A_XOR_B	CI_1	N_CO
M14	N	A_XOR_B	CI_1	N_CO
M15	P	A_XOR_B	AI	N_CO
M16	N	N_A_XOR_B	AI	N_CO
M5	P	AI	A_XOR_B	BI
M6	N	N_AI	A_XOR_B	BI
M7	P	N_AI	N_A_XOR_B	BI
M8	N	AI	N_A_XOR_B	BI
M2	N	BI	A_XOR_B	N_AI
M4	N	BI	N_A_XOR_B	AI
IP2	P	CI_1	VDD	1
IN2	N	CI_1	1	GND
IP1	P	AI	VDD	N_AI
IN1	N	AI	N_AI	GND

(b)

Fig. 1. Circuit diagram for a CMOS full adder (from [5, p. 92]). Each inverter is instantiated as a pair of p-type and n-type transistors. (a) Schematic capture. (b) Textual description (transistors' list).

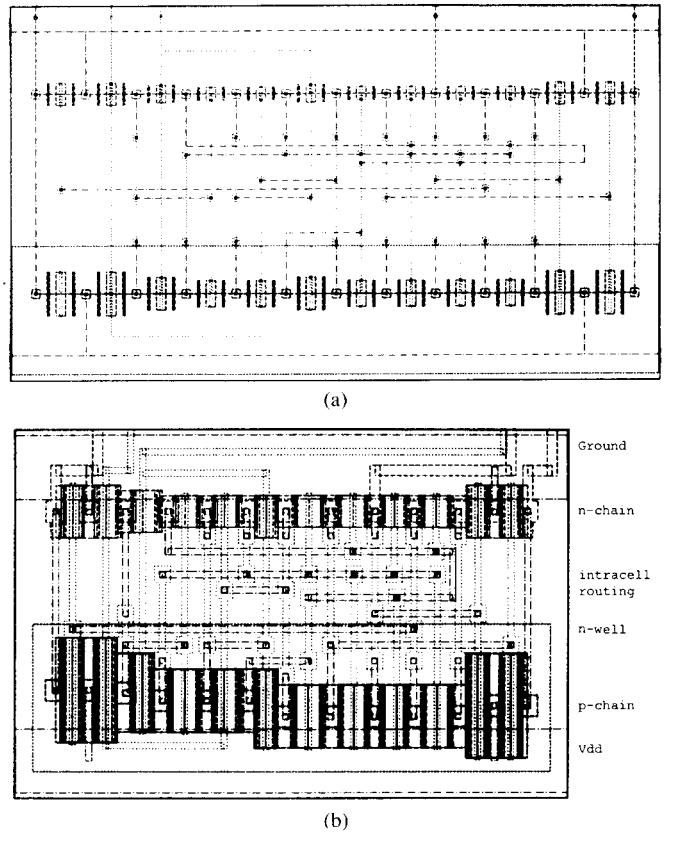
the drain) are connected as part of the row; otherwise, a gap[1] must be left. No diffusion is used anywhere else in in the cell.

Additional wiring is usually required to complete the realization of the circuit's internal connectivity as well as its connections to the I/O signals. We use only one layer of metal in the cells we generate. Power supplies run horizontally along the whole cell in metal: V_{dd} on the top, and V_{ss} (or *ground*) along the bottom. Metal for intracell routing can be used anywhere in the channel that is created between the power rails (which may entail routing over the devices). Polysilicon is also used in places free from both devices and previously routed gate connections. Fig. 2(a) shows a stick diagram of the circuit described in Fig. 1(a).

III. Comparison to Other Work

Our circuit description is purely topological. Some algorithms, such as in [7] and [8], require additional information as part of their input, which is then used to guide the layout algorithms. Moreover, special conditions are imposed on the structure of the input circuit so that the algorithms that are provided would apply, most commonly that the circuit be series-parallel (corresponding to and/or logic), which is rather restrictive.

[1]Gaps between adjacent transistors can be implemented either by a physical break in the diffusion run, or by inserting an "isolation device" [7], i.e., a transistor whose gate is connected either to V_{dd} or ground (depending on its type). In either case, a gap takes up considerably more space than a regular diffusion connection between two adjacent transistors.



(a)

(b)

Fig. 2. Results of applying our algorithm to the circuit diagram of Fig. 1. (a) Stick diagram. (b) Mask data.

Other algorithms, such as [3] and [6], use the same input as ours, and—like us—can handle arbitrary circuits, but then the resulting layout is not necessarily optimal. In this section, we first survey algorithms that are restricted to series-parallel circuits, and guarantee optimality based on this structure, and then describe algorithms of the more general kind.

A. Algorithms for Series-Parallel Circuits

Following [10], two circuit graphs are defined, one for the n-type transistors and the other for the p-type devices. Then, in each graph, the nets (consisting purely of source or drain ports) are the nodes, and one edge corresponds to each transistor, connecting its source and drain. The gate identification is an additional label, and edges representing complementary transistors have the same label in the two graphs. By definition, the graphs representing the p and the n portions must be duals of each other, and edges bearing the same label are mutual duals.

Algorithms in this class attempt to find a sequence of labels (i.e., transistor pairs) such that tracing the edges in sequence will result in Euler paths in both graphs. Needless to say, this approach uses the circuit's logical struc-

ture as a suggestive layout, thereby reducing the complexity of the layout algorithms at the cost of limiting their potential solution space considerably.

When the desired sequence of labels is found, a layout which utilizes one piece diffusion for the p side and one for the n side can be deduced. Since it is not guaranteed that such a sequence does exist, a number of solutions were devised to overcome this difficulty.

1) The first of the two algorithms reported in [8] finds (in linear time) the smallest number of paths covering the circuit graph. The number of diffusion gaps is then uniquely determined by the way the input circuit is described.

2) In [10], it is suggested to introduce additional "dummy" devices which have no functional role. This may result in wasted area and have an adverse effect on the performance of the resulting circuit.

3) Another approach is to permute the devices, i.e., the edges in the graph, in a way that does not change a circuit's functionality but that may facilitate more diffusion adjacencies. Evidently, the components that are combined in series fashion can be permuted at each level, taking advantage of the commutativity of the **or** function, and likewise the parallel components. These permutations are, of course, limited to the series-parallel structure of the graph, but they allow efficient examination of the thus-allowed search space. In the second algorithm presented in [8], the authors study this approach to the extent of trying to find exactly one Euler path to cover the whole circuit; if no single such path exists, the algorithm fails to produce a layout.

4) The study reported in [7] went one step further: they still retrict the transistor ordering to permutations induced by the series-parallel structure, but first they solve the problem of finding the smallest number of paths that cover the graph allowing permutations (combining items (1) and (3)), still in linear time, and secondly they find one path under the permutation model by adding dummy devices to it. In a sense, these are the ultimate results of this approach.

All in all, these solutions can be characterized by finding a layout that adheres to the circuit structure as suggested by the circuit diagram; therefore they can handle only series-parallel circuits. Since in all cases this can be done efficiently, the choice between the alternatives depends on whether the order of the transistors as given needs to be respected due to circuit performance considerations. Since sizes and ordering of transistors may be the result of static timing analysis, permuting the devices (during the layout process) may be considered detrimental and thus excluded.

B. Algorithms for General Circuits

Hill's algorithm [3] can handle arbitrary circuits. It is divided into two[2] stages: transistor placement followed by interconnect (intracell routing). Placement starts by pairing p devices with the n devices, forming columns which are then ordered in a row. Pairing is done heuristically, requiring that the two transistors in a pair have the same gate; i.e., they must belong to the same "logical unit" (in Hill's terminology). P-type and n-type transistors that have a common diffusion port are preferred. The transistor pairs are then ordered in a row, using a min-cut procedure repetitively, allowing pairs of different logic gates to be placed together if they are topologically strongly connected. Then, the transistor pairs are flipped[3] (horizontally) to maximize the number of adjacent source and drain ports belonging to the same net, thus reducing the diffusion runners and the number of contacts. Both the pairing stage and the transistor ordering stage attempt to reduce the intracell routing which follows.

The routing step aims to reduce the total wire length while the performance of the cell serves as a constraint. Several routing methods are employed, starting with abutting transistors, going to more expensive methods such as routing over the transistors, and finishing with channel routing which is the most expensive method.

None of the steps guarantees an optimal solution, even to the problem that is set up for that stage alone—starting from min-cut all the way to routing.

A similar approach to automatic CMOS cell generation was proposed by Miyashita, *et al.* [6], consisting of transistor pairing, followed by placement of transistor pairs, and finally a routing phase.

The algorithmic approach for the placement phase is based on iterative improvement. Two objective functions control heuristic: the total sum of the weighted wire lengths, and the weighted maximum channel density. For the routing phase, a modified maze router is used in order to accommodate multilayer wiring, supporting diagonal wiring.

Both methods described in this section, as well as the method presented in this paper, implement the circuit's connectivity as it is specified in the input. The large variety of solutions that is explored in each case is due to geometric combinations that are considered directly in terms of the layout.

IV. The Layout Algorithm

The algorithm is a six-stage procedure. We first describe the top level, and then elaborate on each step separately.

1) Generate transistor pairs as candidates for sharing a column.
2) Form chains of transistor pairs such that adjacent transistors have common diffusion ports on both the p side and the n side; each such chain is assigned a cost reflecting its overall merit.
3) Select a realization of the circuit using the chains that were formed in the previous step so that the total cost of the realization (i.e., the sum of the costs of its constituents) is minimized.

[2]A preprocessing step splits wide transistors into smaller ones connected in parallel.

[3]Given a linear order of transistor pairs, we have observed that the optimal flip can be found in linear time using dynamic programming; it is not clear from the paper what technique was used by Hill.

4) Decide in which order to put the chains that were selected in the previous stage along the cell.
5) Additional interconnections are now required to complete the realization of the circuit. This is done by the routing stage.
6) Finally, compact the stick diagram that was generated.

Fig. 2(b) shows the layout obtained for the circuit described in Fig. 1 (after compaction). The transistor sizes were supplied as parameters by the user.

A. Pairing

The pairing criterion for a p-type and an n-type transistor is either that they both have the same gate or that they have a common diffusion port (source or drain).

As an example for pairing, take a CMOS complementary logic circuit in which each p transistor has a complementary n transistor and vice versa, having the same signal for their gates. Then, if some signal is connected to m gates of p devices, and hence also to the gates of m n devices, m^2 pairs result. When two devices constitute a pair, routing of the gate signal is very easy since the gates are vertically aligned and they can be connected by one piece of polysilicon.

Another example is a dynamic CMOS logic circuit in which transmission gates are widely used. Here the complementary transistors are connected source to source and drain to drain and they are gated by complementary signals. Although the gates cannot be connected as in complementary logic, pairing these transistors eases the routing. Since their sources and drains are vertically aligned, they can be connected without occupying any horizontal track in the layout.

B. Chain Formation

The objective of this step is to form diffusion chains such that each such chain can be placed as a contiguous run of diffusion; i.e., when all the transistors in a chain are arranged in a line, adjacent diffusion ports belong to the same net (both in the p channel and the n channel).

An optimal inplementation comprises several chains which are not necessarily maximal; i.e, some of them may be contained in longer ones. Hence chain formation is done gradually, step by step, where in the ith step we attempt to create all the chains having i pairs. This is accomplished by taking chains of length $i - 1$ and then adding all those pairs whose end terminals abut with the end terminals of the chains (the pair may be flipped).

We assign a cost (merit) to each chain thus formed. This cost may reflect properties of the chain, such as its length, its internal routing density, and electric performance characteristics.

Notice that the chaining process may generate chains whose participating transistor sets are identical, but their pattern (ordering) within the chains is different. In such a case, the chains are treated as equivalent and only the chain with the best score is kept as the representative of each equivalence class. This may result in a suboptimal layout in respects other than diffusion gap minimization, such as overall routing density, but it reduces the time and memory requirements to reach a solution.

C. Circuit Covering

To realize the circuit, we select a subset of the chain set (formed in the previous stage) such that every transistor appears in at least one chain. Obviously, in a legal covering of the circuit each transistor should not appear more than once. Hence, we must solve a *set partitioning* (SP) problem, where the variables (i.e., the sets) are the chains created in step 2, and the constraints (i.e., the elements) are the transistors of the circuit which must be realized.

Every SP variable bears the cost of the corresponding chain, and the SP problem attempts to find a solution in which the cost is minimal. If a unit cost is assigned to each chain, an optimal solution of the SP problem guarantees a cell utilizing minimum diffusion. When each chain's cost reflects the total wire length of the intrachain routing, the cost of an SP solution stands for the expected total wire length in the cell.

The SP problem is known to be NP-complete, but we use a very effective enumerative algorithm, which is outlined here. We first convert the SP covering matrix into a graph as follows.

1) Every vertex corresponds to some SP variable.
2) Two vertices are connected if the sets of SP constraints satisfied by the corresponding variables are disjoint.
3) Every vertex is assigned a rank equal to the number of SP constraints by the corresponding SP variable.

Our basic observation is that every feasible solution to the SP problem corresponds to a clique whose total sum of vertex ranks equals to the number of SP constraints. Thus the SP problem can be solved efficiently by running the clique finding algorithm proposed by Bron and Kerbosch [1], which has the following attractive properties.

1) It is well suited for sparse graphs that result from our dense covering matrices.
2) Backtracking takes place early.
3) Its expected running time is independent of the number of SP constraints (because they are coded as ranks).

Notice that if the number of p devices is different from the number of n devices, then no feasible solution exists. We overcome this problem by introducing dummy devices, whose number and type resolve the difference.

D. Chain Placement

The chains are ordered so as to minimize the interconnect density of the resulting cell (relative to the chains that have been formed). With n chains in the realization (as obtained by solving the SP problem), there are $2^n n!$ different arrangements of chains in the cell, where $n!$ accounts for all permutations of chains along the cell, and 2^n accounts for all the possible orientations per a given permutation (each chain can be either reflected or not). If the number of chains is small (less than 5), which is usu-

ally the case, an exhaustive enumeration is performed (for $n = 4$ there are only 384 cases to check).

For a cell composed of more than 4 chains, a random sample from the space of all possible arrangements is drawn, and the arrangement with lowest channel density is selected. Our experience shows that the "good" arrangements are evenly distributed in the solution space. Thus, if a large enough sample is drawn, a "good" arrangement will be found with high probability. Another alternative would be to employ a branch-and-bound procedure to obtain the optimal chain ordering.

E. Routing

In this phase, we deal with both the intra- and interchain interconnect. There are two consequences of the chaining strategy that affect the routing considerations.

1) If two paired transistors have the same gate signal, or if (either of) their diffusion ports belong to the same net, then the appropriate aligned signal should be routed straight through the column.
2) Since diffusion ports are made adjacent as much as possible, it is desired to complete the routing without insertion of extra columns between adjacent ports.

Also, power supplies run in parallel to transistor chains on metal (we assume that only one level of metal is available for cell layout).

By and large, this is a channel routing situation. We need to guarantee completion, on the one hand, but we do not want to add columns between transistors, on the other hand. The "greedy" channel routing algorithm of Rivest and Fiduccia [9] best suits our needs: it may require additional columns, but they are placed at the end of the channel, i.e., where the transistor chains end and no additional diffusion runs are incurred. We made appropriate changes to the "greedy" channel routing algorithm so as to obey our constraints and in order to accommodate the highly populated, desirably narrow channels that we create, as follows.

- Top priority is given to the case $T_b = T_u \neq 0$; i.e., both top and bottom terminal positions are occupied by the same net, even when the channel is not yet saturated. Then the terminals are connected immediately by a vertical poly strip; i.e., we do not leave this to the collapsing stage.
- Collapsing is allowed in metal over vertical poly wires (this is common in gate columns), thus making better use of the column.
- The decision which track to continue a new net on is integrated with the step of bringing the new nets in; this change tends to help in trying to maintain narrow channels.

As high a quality as they may be, the results of a channel router are usually unacceptable verbatim in the context of functional cell layout, even after the aforementioned adjustments that gear its decisions in a desirable way. In order to improve the acceptability of the layout, a number of measures are taken so that the resulting layout is competitive with the quality of hand designed cells. These improvements are effected both as preprocessing steps that deal with special cases that are better not left to the router itself and as cleanup steps that are performed after the router is done and retouch certain parts of the layout.

There are three preprocessing steps aimed both at reducing the channel's density and improving the performance of the circuit being laid out.

1) Nets that connect ports placed only on one transistor chain (the p chain or the n chain, but not both) are routed over the transistors in metal.
2) Nets connecting several transistor gates (but not other signals, i.e., they can be routed purely in poly) are routed outside the channel area in a fork-like structure (as long as they do not cross other nets in this class, as explained below).
3) If the diffusion ports across a gap belong to the same net (this can happen only on one side since otherwise a chain would have been formed), they are connected in diffusion.

The first two steps are implemented by the same labeling algorithm, which finds an optimal assignment of nets to the appropriate classes in linear time. We shall describe here the algorithm for selecting the nets that are to be implemented as poly forks (step 2), which is more complicated; step 1 can be performed in a similar way.

We model a prospective fork (i.e., a candidate poly net) by its two extreme points (leftmost and rightmost); thus it is represented by an interval. In this way, no attention is paid to the positions of the internal prongs, since indeed these positions are not relevant to any of the constraints and optimization criteria that are involved. The potential forks of the transistor arrangement of Fig. 3(a) are given in Fig. 3(b), where each gate is marked by the name of its net and the corresponding interval is marked by the same name. Each interval is assigned a weight reflecting the desirability of its implementation as a fork; this weight could be, for example, its length, the number of contacts saved by routing it this way, or the maximal density of the channel along its span.

The problem of finding a set of nonoverlapping intervals that maximizes the sum of these weights can be formalized and optimally solved as follows. Define $G = (V, E, w)$ to be a transitively reduced graph where V is the set of intervals, $(u, v) \in E$ if interval u is to the left of interval v and they do not overlap, and for every vertex $v \in V$ assign a real weight $w(v)$. Fig. 3(c) shows the graph corresponding to the arrangement of Fig. 3(a). The following *labeling algorithm* is then applied.

1) For each source s, assign $\lambda(s) = w(s)$.
2) For each unlabeled vertex v all of whose predecessors have already been labeled, assign $\lambda(v) = w(v) + \max\{\lambda(u)\}$, where u ranges over all predecessors of v.
3) A pointer is kept to keep track of the maximizing predecessor of each node. In this way, the solution can be traced in the end.

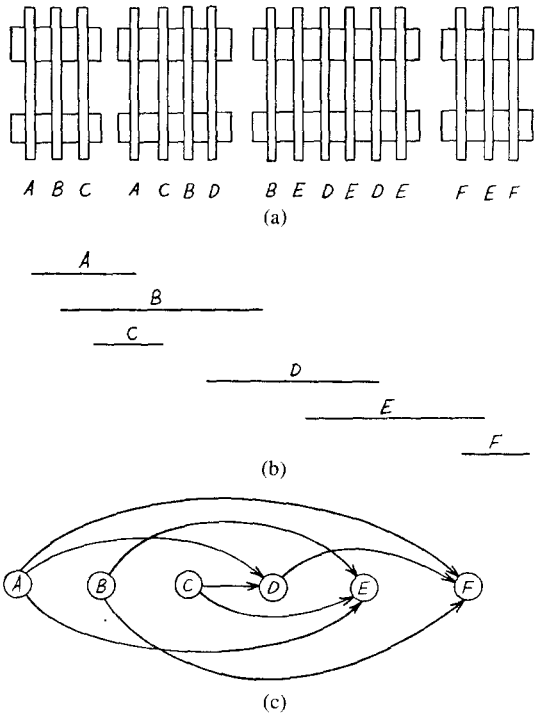


Fig. 3. Modeling polysilicon forks as intervals. (a) Transistor arrangement. (b)Fork intervals. (c) Graph to be labeled.

This algorithm runs in linear time. In order to maximize the chances of successfully routing the cell, this algorithm needs to be run twice, since forks can be implemented by having their horizontal segment pushed under the Gnd line or under the V_{dd} line. We do not know how to find two sets of (each) nonoverlapping intervals that maximize the total sum of weights, and we believe this problem is intractable. Therefore, we simply solve the problem twice in succession, where the intervals participating in the solution for the first pass are removed before the second pass is invoked.

After the "greedy" routing algorithm is done, most vertical wires run on poly even if they do not connect two aligned gates directly. At this point, it is desirable to reduce the number of contacts that were incurred by this methodology and increase the usage of metal where possible. This is done by two clean-up measures, as follows.

1) Vertical wires that connect diffusion ports without crossing assigned tracks are wired on metal.
2) Vertical wires that cross assigned tracks are changed to metal as much as possible in order to make minimal usage of the poly layer and reduce the number of poly–metal contacts.

F. Compaction

All the layout work done so far was performed on a virtual grid. It is desirable, of course, to get rid of any slack area—after the circuit elements are fleshed out—by compaction. This can be done with any circuit compactor one desires, ideally one that provides truly two-dimensional compaction.

We use the compactor of the PSI system [2], which is based on the one-dimensional algorithm from [4]. The circuit is compacted horizontally and vertically in separate stages that can be applied repeatedly and in any order. This compaction regime suits most needs of a cell generator like ours, where the devices are arranged in a matrixlike structure.

Fig. 2(b) shows the results of fleshing out the stick diagram of Fig. 2(a) and then compacting it in both directions.

V. Row Generation

In this section, we demonstrate how several cells that were generated with our algorithm can be composed into a row, constituting a functional unit. Due to its incremental structure, the various stages of the algorithm can be applied at different levels of granularity, thereby affording a nice tradeoff between computation time and efficient layout. In this section, we first described one way in which the combination can be implemented, and then discussed the alternatives and their relative merits.

When several cells are pieced together, a number of signals could become common among them. For example, the carry-out of a one-bit full-adder is the carry-in to its neighbour in an n-bit adder (this kind of global interconnect information is furnished in addition to the cell descriptions, of course). The net adjacency information should obviously affect the ordering of cells along the row, but with our cell generation methodology it can also be used to affect the way in which each individual cell is generated.

We introduce the concept of instantiating a cell in a *composition context*: if the signals that are connected to the right or left of a cell that participates in a functional slice are known in advance, we can take advantage of this information and affect cell generation in such a way that the cell produced is modeled to better fit the intended context than if it were generated stand-alone.

In practice, this concept affects both the placement and the routing stages of the cell generation algorithm. During placement, the left and right connections are used as follows.

1) During chain formation, the density computations that are used in deciding which chain is selected to represent a given set of transistors (in case the same set appears more than once) takes these connections into account.
2) During chain ordering, similar consideration is given to nets that emanate from chains.

After placement is completed, the whole row is routed in one pass. The channel router completes the interconnections in one scan; its speed (practically linear in the size of the circuit) allows us to use it verbatim for the composition phase, even for rows with more than one thousand transistors. Notice that the third preprocessing step mentioned in Section IV-E may result here in large savings.

In the methodology that was described so far, each cell was placed individually, thereby breaking the complexity of the placement problem into two hierarchical stages. Using this methodology, we took advantage of the fact that cells were repeated in identical form along the cell. One could, at the expense of computing time, start using

the cell generation algorithm for the whole assembly at an earlier stage, namely chain ordering. In other words, all the chains selected for the constituent cells can be given as input to the chain ordering algorithm and placed in one step. In a sense, Hill [3] took this approach to the extreme by making one pair ordering stage for a whole composite.

On the other hand, one could finish the routing before row assembly. Here, the signals going to the left and the right of each cell can be given to the Rivest–Fiduccia algorithm (which can accommodate such a specification) in advance, and then the ready-made cells can be glued together. Then, of course, care should be taken as to what track the signals are assigned to in neighbouring cells so that they can be abutted safely.

All in all, our composition method is flexible and well suited for a wide range of applications. Our cell generation algorithm is robust and can be used in a number of variations in this context.

VI. Conclusions

We have described a new algorithmic method for laying out circuits from their schematics which is superior to previous published methods of this class in a number of ways, as follows.

- It accepts a broader class of circuit diagrams (there are no structural limitations on the circuit graph).
- It can accommodate multiple optimization criteria and at the same time honor constraints as specified by the designer.
- More general types of solutions are considered, resulting in smaller and better layouts.
- The algorithmic steps are clear and provide the basis for a flexible framework for extensions.
- It is efficient and practical.

Our algorithms have been programmed into approximately 5000 lines of Pascal (not including the front and the back end, but counting some necessary data base conversions). They have been tested on circuits describing cells with up to 50 transistors, and the running times range from 1 to 10 CPU seconds on an IBM 4381 model 2, depending on the structure of the circuit.

The algorithms have a clear formulation and at the same time are practical and flexible. We have demonstrated their extensibility in the context of a slice generation tool, with which we can put any number of cells in a row. We have implemented functional slices with more than one thousand transistors within seconds, e.g., a 32-bit ripple-carry adder took 15 seconds (on a 4381) to lay out. Our framework can be futher extended to accommodate additional optimization criteria as well as constraints that arise in certain design environments.

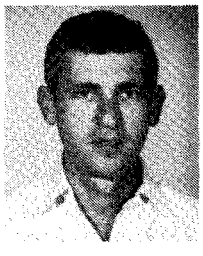
Acknowledgement

The authors would like to thank I. Berger for many helpful discussions and his encouragement. They would also like to thank E. Adams and J.-F. Lee of the IBM Watson Research Center at Yorktown Heights for making PSI [2] available to us.

References

[1] C. Bron and J. Kerbosch, "Algorithm 457—Finding all cliques of an undirected graph," *Commun. Ass. Comput. Mach.*, vol. 16, 1973.
[2] R.-D. Fiebrich, Y.-Z. Liao, G. Koppelman, and E. N. Adams, "PSI—A symbolic layout system," *IBM J. Res. Develop.*, vol. 28, no. 5, pp. 572–580, Sept. 1984.
[3] D. Hill, "Sc2—A hybrid automatic layout system in *Proc. ICCAD*, Nov. 1985, pp. 172–174.
[4] Y. Z. Liao and C. K. Wong, "An algorithm to compact a VLSI symbolic layout with mixed constraints," in *Proc. 20th Design Automat. Conf.*, June 1983, pp. 107–112.
[5] J. Mavor, M. A. Jack, and P. B. Denyer, *Introduction to MOS LSI Design*. Reading, MA: Addison-Wesley, 1983.
[6] H. Miyashita, T. Adachi, and K. Ueda, "An automnatic cell pattern generation system for CMOS transistor-pair array LSI," *Integration*, vol. 4, pp. 115–133, 1986.
[7] R. Müller and T. Lengauer, "Linear algorithms for two CMOS layout problems," in *Proc. Aegean Workshop Comput.*, July 1986.
[8] R. Nair, A Bruss, and J. Reif, "Linear time algorithms for optimal CMOS layout," in *VLSI: Algorithms and Architectures*, P. Bertolazzi and F. Luccio, Eds. New York: Elsevier North-Holland, 1985, pp. 327–338.
[9] R. L. Rivest and C. M. Fiduccia, "A 'Greedy' channel router," in *Proc. 19th Design Automat. Conf.*, June 1982, pp. 418–424.
[10] T. Uehara and W. M. vanCleemput, "Optimal layout of CMOS functional arrays," *IEEE Trans. Comput.*, vol. C-30, 5, pp. 305–312, May 1981.

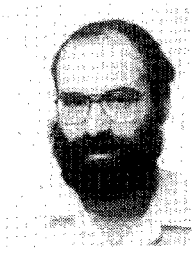
*



Shumel Wimer received the B.Sc. and the M.Sc. degrees in mathematics from Tel Aviv University in 1977 and 1980, respectively.

From 1977 to 1981, he worked at IAI (Israel Aircraft Industry). From 1981 to 1985, he was employed by National Semiconductor Tel Aviv, where he worked on VLSI CAD tools. He is currently a Research Staff Member at the IBM Israel Scientific Center, where he works on algorithms for the layout of VLSI circuits.

*



Ron Y. Pinter (M'83) received the B.Sc. in computer science from the Technion—Israel Institute of Technology in 1975, and the S.M. and Ph.D. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology in 1980 and 1982, respectively.

During 1982–1983, he was a Member of the Technical Staff in the Computing Sciences Research Center, AT&T Bell Laboratories, Murray Hill, NJ. In December 1983, he joined the IBM Israel Scientific Center, where he is currently the manager of the Programming Languages Group. He is also an adjunct lecturer in the Electrical Engineering Department at the Technion—Israel Institute of Technology. His research interests include layout algorithms for integrated circuits, computational geometry, programming language design, and code generation algorithms.

Dr. Pinter is a member of the Association for Computing Machinery and the IEEE Computer Society.

*

Jack A. Feldman, was born in Bucharest, Romania, in 1958. In 1983, he received the B.Sc. and M.Sc. degrees in computer engineering from the Polytechnic Institute of Bucharest. He is currently studying toward the M.Sc. degree in electrical engineering at the Technion—Israel Institute of Technology.

In 1985, he joined the IBM Israel Scientific Center as a Research Fellow, where he is working on problems related to the automated design of VLSI circuits. He is interested in algorithms and systems for design automation.