Depth-First-Search and Dynamic Programming Algorithms for Efficient CMOS Cell Generation

REUVEN BAR-YEHUDA, JACK A. FELDMAN, RON Y. PINTER, AND SHMUEL WIMER

Abstract—We describe a new algorithmic framework for mapping CMOS circuit diagrams into area-efficient, high-performance layouts in the style of one-dimensional transistor arrays. Using efficient search techniques and accurate evaluation methods, the huge solution space that is typical to such problems is traversed extremely fast, yielding designs of hand-layout quality. In addition to generating circuits that meet prespecified layout constraints in the context of a fixed target image, on-the-fly optimizations are performed to meet secondary optimization criteria. A practical dynamic programming routing algorithm is employed to accommodate the special conditions that arise in this context. This algorithm has been implemented and is currently used at IBM for cell library generation.

I. Introduction

THE AUTOMATIC generation of high-performance integrated circuits in the layout style of a one-dimensional transistor array, as suggested by Uehara and vanCleemput [10], has been studied recently from a number of different angles. In some cases [6], [5], [11], the primary goal was to minimize the amount of diffusion required in the artwork, while other considerations, such as reducing internal wiring and accommodating performance constraints, were handled (if at all) as secondary issues. In other cases [4], [1], the order of importance was reversed: another criterion, such as low routing density or minimal wire length, was the driving goal, and then a considerable amount of time was spent minimizing diffusion gaps. In both cases, the primary goal was obtained reasonably efficiently, but accommodating other concerns came at a significant running-time cost (as the sizes of the circuits grew larger).

In this paper, we propose an algorithmic method in which the generation of the layout is driven by a set of optimization criteria and composition constraints making it possible to control various aspects of the layout such as diffusion breaks, metal utilization, and wire length. The solution space is expanded by a **depth-first-search** (DFS) procedure using the constraints to effectively reduce the branching factors, and applying the optimization criteria to sharply bound and eliminate unnecessary expansions. Particular care has been taken in choosing the implementation method so as to provide truly optimal solutions when possible in linear time: several incremental dynamic programming calculations are conducted on the fly, and the supporting data structures are maintained efficiently.

The results of our technique are layouts that fit tight intracell routing requirements, utilizing diffusion adjacencies wherever possible to save layout area and meeting user-specified performance constraints. In addition, the algorithms proposed here can handle *arbitrary* circuit graphs, as opposed to several restricted algorithms that can handle only series–parallel circuits [6], [5] or only circuits having an equal number of P-type and N-type transistors. For example, the CMOS latch shown in Fig. 1 was laid out as shown in Fig. 2, using only four routing tracks, two over the P transistors and two over the N transistors, relatively short metal straps, and as many diffusion adjacencies as possible. The algorithms have been coded in Pascal, and were applied to an entire library of fairly large leaf cells containing several dozen transistors, each at substantial productivity gains.

The rest of this paper is organized as follows. Section II defines the target image of the layout, explaining some of the constraints. Then we describe the algorithms in Section III, and summarize the results in Section IV.

II. The Layout Image

The *image* of the target layout affects both the constraints and the optimization criteria that guide the generation algorithm. Here is one possible setting, which will also be used as the framework for the layout algorithm as described in the next section. This image is illustrated by the layout shown in Fig. 3.

- The transistors are arranged in two horizontal, parallel rows—one for the P-type devices and the other for the N-type transistors.
- The transistors can be assigned only to fixed horizontal locations, in the manner of a grid, at a fixed pitch.

Manuscript received March 28, 1988; revised October 14, 1988 and December 20, 1988. The review of this paper was arranged by Associate Editor A. E. Dunlop.

R. Bar-Yehuda is with the Computer Science Department, Technion, Haifa, Israel, and with the IBM Israel Scientific Center, Technion City, Haifa 32 000, Israel.

J. A. Feldman, R. Y. Pinter, and S. Wimer are with the IBM Israel Scientific Center, Technion City, Haifa 32 000, Israel.

IEEE Log Number 8927530.

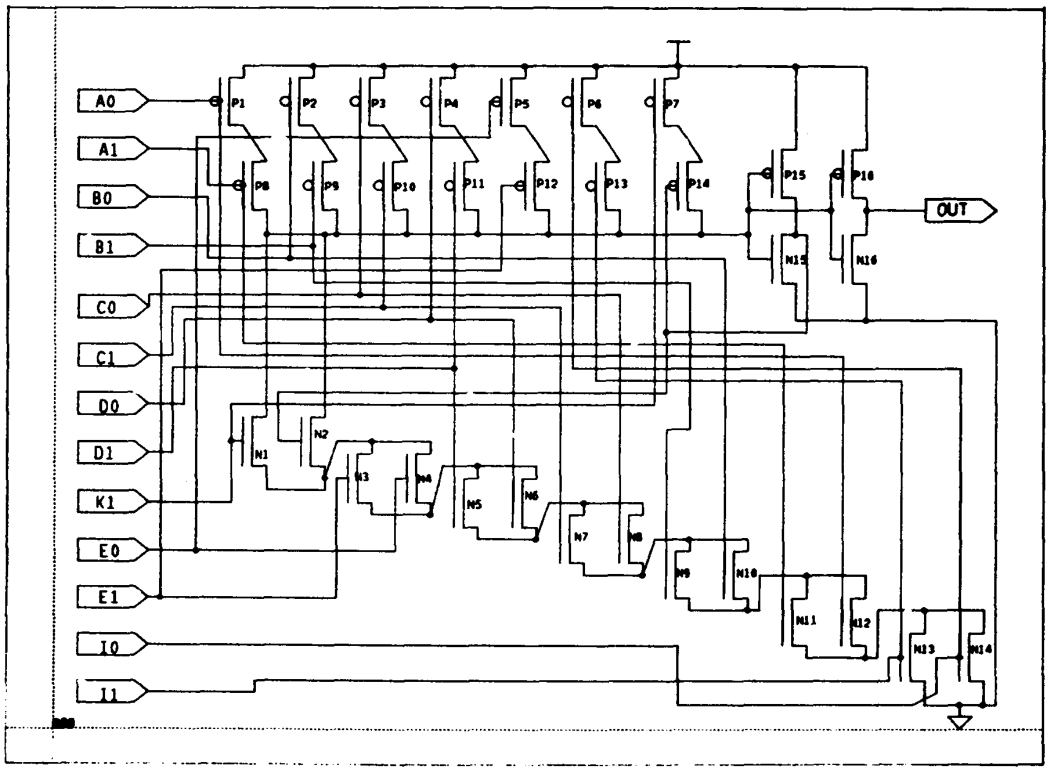


Fig. 1. The circuit diagram for a CMOS latch.

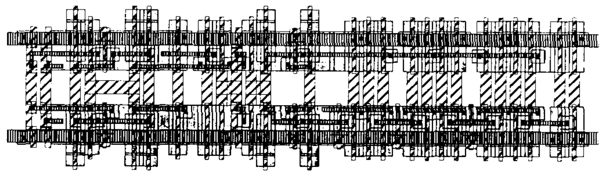


Fig. 2. The layout of the CMOS latch from Fig. 1. This result was obtained by running the algorithm allowing two routing tracks on each side, with minimal stretching of the diffusion runs.

- Diffusion is used to connect adjacent diffusion ports (source or drain) on each row, and vertical polysilicon lines connect aligned gates.
- Diffusion breaks (on either side) may be instantiated as gaps or as isolation devices (transistors whose gate is connected to power, e.g., [4]). Either way, a break incurs a significant increase in width.
- Internal routing is performed in one layer of metal over each row of transistors, using a fixed number of wiring tracks. The image presented in this work allows two wiring tracks on each side.
- Poly-metal contact is allowed on top of the active area of a transistor.
- A net connecting the P and N regions but having no aligned common gates in any P–N transistor pair requires the insertion of an additional vertical poly strip. This strip connects the portions of the net in both the P and the N regions, as illustrated by the left-most poly strip in Fig. 3.

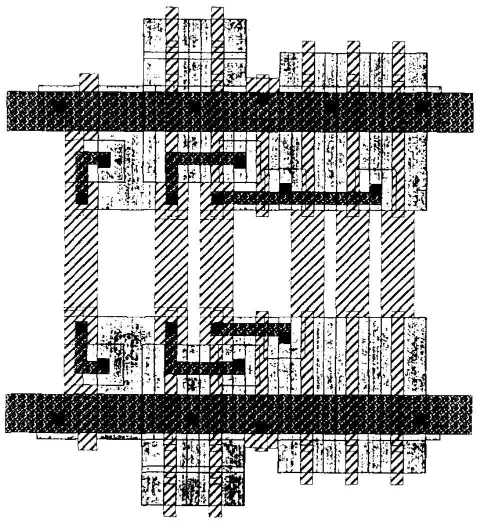


Fig. 3. A small cell, exemplifying the fixed image. Notice the internal routing and the isolation device.

- A reserved area is left between the rows for external routing, but it can be (and is, when possible) used for internal horizontal routing on poly between adjacent gates of the same net to reduce wiring density.

Working with a (relatively) fixed image as the one described here has a number of advantages, as argued in [7].

- *Control:* The algorithms used to generate cells have full control over all parameters of the layout, thus making it possible to predict the final layout. In contrast, the final layout in virtual grid is somewhat unpredictable since compaction is used as a post-process.
- *Complexity:* A fixed grid provides a succinct and clear level of abstraction of the physical layout, thereby making the layout algorithms easier to develop and maintain.
- *Composition:* When library cells are combined into a macro, much simpler layout tools than complex placement and routing techniques can be used in such a structured environment. The composition of cells into larger designs is smoother and, due to the reduction in interconnect, most of the time is also more space efficient.

In addition, the utility of such an image is enhanced when having to formulate constraints and objectives other than just minimizing diffusion breaks or cell density, namely, more accurate physical design measures such as the number and type of contacts, routing jogs, and wire lengths.

III. The Layout Algorithm

The layout algorithm has three main components, as follows:

1) The primary driver is a DFS [3] routine, expanding potential transistor *placements* (or orderings) along the generated array. This driver generates a virtual search tree on which a branch-and-bound procedure is performed. Only placements that can be subsequently routed are generated.
2) During the DFS expansion, the internal optimal orientation of the transistors in each potential placement is determined on the fly. The computation of these orientations is done in constant time for each new node that is expanded.
3) The internal, detailed *routing* of the best placement that was found by the DFS is then performed.

Each component is now described in greater detail in the following subsections.

A. Depth-First-Search Expansion of Transistors

Let $N_1, \cdots, N_n$ and $P_1, \cdots, P_p$ be the N-type and P-type transistors of the input circuit, respectively (if $n \neq p$ we introduce dummy devices, whose number and type resolve the difference). The set of all possible placements can be generated using an enumeration tree, where each node (except the root) is marked by an ordered pair $\langle P_i, N_j \rangle$. If each transistor appears exactly once on the labeled path from the root to a leaf, then such a path represents a (say) left-to-right assignment of the transistors in the layout. A path from the root to an internal node corresponds to a partial assignment. Note that each label stands for all four possible orientations of its associated P–N pair, each having a (possibly different) cost. We explain in Section III-B how the best of these orientations is picked; the cost of a partial layout depends on a number of layout characteristics that can be computed dynamically, such as

1) number of diffusion breaks,
2) alignment of transistors having the same signal for a gate (to facilitate vertical polysilicon connections),
3) total wire length.

These measures are combined into a lexicographic objective function, according to the above order. The location of the vertical poly strips that are inserted to connect nets residing on both regions (see the layout image) are determined in the placement phase, since their insertion might introduce a break in the diffusion run and affect the wiring density. Therefore, their optimal location is found by considering them as if they were transistor pairs.

A simplified variation of the DFS algorithm is given here (using pseudo-Pascal notation):

```
v: =root;
repeat
   while v has unvisited outgoing, acceptable edges do
      begin
         pick (v,u) as the unvisited outgoing edge of
           least cost;
         v:=u; (* forward step *)
      end;
   if (v is a leaf) and (Cost(v) is acceptable)
        then begin
            record solution;
            update acceptability threshold;
        end;
   v := the father of v in the tree; (* backward step *)
until the root has no unvisited outgoing edges;
output (minimal cost solution);
```

The convergence rate of the DFS procedure is mainly affected by the following factors:

1) The branching strategy: It is desirable to reach a good solution as soon as possible. This is obtained by selecting the order of expanding the outgoing edges as those that yield the smallest increment in the cost of the partial placement.
2) The bounding cost: Whenever the cost of a partial solution exceeds the cost of the best complete solution reached so far, backtrack takes place. Therefore, the immediate low-cost solution obtained in item 1 further prunes the search tree.
3) The bounding constraint: The number of feasible edges that can be extended from each node is strongly bounded by wiring density considerations.

An efficient data structure is used to control the order in which the edges emanating from each node are visited:

1) A number of linked lists are threaded among the devices, and each is organized according to one of the

ingredients contributing to the cost function; hence the least-cost computation is fast.

2) All density recalculations are done incrementally, using a constant number of operations per node.

Every forward step of the DFS generates a new virtual edge. There are two possibilities. In the first one all the edges emanating from a node (transistor pair) are generated and sorted according to their order of preference prior to the search, and then stored in the memory. Since there are $O(n)$ transistor pairs, and each of them can be concatenated to other $O(n)$, thus resulting in $O(n)$ edges per node, this approach demands $O(n^2)$ space. Since each edge is traversed only once in each direction, the time complexity of the algorithm is linear in the number of edges (and hence the number of nodes) in the traversed tree. In the second alternative only the nodes along the path from the root are stored in the memory, thus demanding an $O(n)$ space. This, however, spoils the time complexity since whenever a node is traversed in the forward direction, the selection of the next edge to be traversed necessitates regeneration of all the possible edges and selecting among them the desired one. This adds an $O(n)$ factor to the time complexity. Since run time is the limiting factor and for practical problems an $O(n^2)$ space requirement is easily accommodated, the first approach was implemented.

It is important to note that imposing constraints on the resulting layout, as opposed to minimizing a layout measure, can be gracefully incorporated into the DFS scheme. Moreover, it speeds up the DFS expansion. For example, there is no point in minimizing the wiring density as long as it is less than the allowable maximum.

B. Optimal Orientation of Devices

During placement, several possible orientations of the devices in each growing solution may be feasible at the same time, as long as the wiring density of the P and N sides does not exceed the given bound. We would like, however, to keep only the best solution, i.e, the one yielding the most diffusion abutments.

This "optimal flip" problem was addressed in [4], and a branch-and-bound-placement algorithm, which enumerates all the 2^{2n} possible orientations (for a circuit with $2n$ devices), was suggested. This problem was also addressed implicitly in [11] at the pairing stage, which is performed before placement. In this subsection we present a dynamic programming algorithm that solves this optimization problem in linear time. Moreover, our solution can be integrated into the DFS procedure, requiring constant time per node expansion. For the sake of presentation, we describe the scheme as if it were applied to a given sequence of pairs realizing the whole circuit, but it is easy to see how it can be combined with the DFS.

The algorithm proceeds from left to right. At each step a new pair is processed and concatenated in its optimal orientation to the previously processed pairs. A device is in the "0" orientation if its left terminal is the drain and

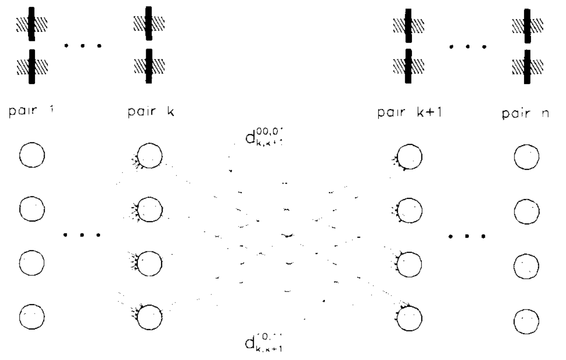


Fig. 4. The structure of the optimal flip dynamic programming algorithm.

the right terminal is the source, or in the "1" orientation if the device is flipped. The orientation of a pair is denoted by the concatenation of the P and the N orientations, and the set $\Psi = \{00,01,10,11\}$ denotes all possible orientations of a pair. Let P_i^σ, where $1 \leq i \leq n$ and $\sigma \in \Psi$, denote that the ith P-N pair from the left is in the σ orientation (the superscript is omitted when the orientation is immaterial). Let $d_{i,i+1}^{\tau,\sigma}$ denote the penalty resulting from concatenating P_{i+1}^σ to P_i^τ. Fig. 4 illustrates all the possible concatenations of kth and $k + 1$th pairs. Every arc is assigned with an appropriate penalty, indicating whether the pairs in the corresponding orientation abut.

Finally, let $s_k^\sigma = (P_1, \cdots, P_{k-1}, P_k^\sigma)$, $1 \leq k \leq n$, $\sigma \in \Psi$, denote the k leftmost pairs whose orientation has been found so as to minimize the cost of their implementation, where the rightmost P-N pair is in orientation σ. Let c_k^σ denote the cost of s_k^σ, resulting from the individual penalties of consecutive pairs.

Since the diffusion ports of a new concatenated pair may interact only with those of its immediate predecessor, it is obvious that given s_k^σ, the optimal orientation of P_{k+1} is independent of the orientations of all the P_i, $1 \leq i \leq k - 1$. Therefore, the cost of s_{k+1}^σ, $\sigma \in \Psi$, is given by

$$c_{k+1}^\sigma = \min_{\tau \in \Psi} \{c_k^\tau + d_{k,k+1}^{\tau,\sigma}\}. \tag{1}$$

The above observation yields a dynamic programming procedure to find the optimal orientation for P_{k+1}. The overall structure of the dynamic programming procedure is illustrated in Fig. 4. After the nth step is done, the optimal orientation is obtained by taking s_n^σ yielding the minimum among $c_n^{00}, \cdots, c_n^{11}$. From this final state we retrieve the optimal orientation by going backward from $k = n$ down to $k = 1$. Note that the calculation in (1) is performed four times for each pair, thus requiring $O(n)$ time altogether for a given pair order. Since s_k^σ is a sequence whose length does not exceed n, only $O(n)$ space is required.

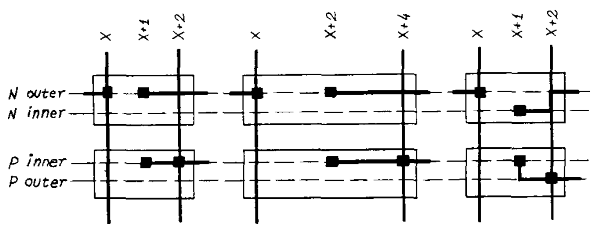
C. Routing

At this stage, some of the internal connections were already taken care of by abutting adjacent diffusion ports, and others are handled trivially using vertical connections on poly between aligned gates. Also, all other connections between nets that reside on both rows (and these are rare) are realized by vertical poly lines whose optimal location is determined in the placement phase, as already described. The routing step deals with the wiring of the remaining connections that must be performed in the given tracks. The feasibility of the routing problem is guaranteed due to the DFS in the placement, which backtracks whenever the wiring density is exceeded. However, some extra area may be encountered due to several spacing rules of the specific image described below. Therefore, the goal of the routing algorithm is to minimize this additional area.

1) Contacts on the same track cannot be placed in consecutive columns, even if they belong to the same net. This situation is illustrated in Fig. 5(a). A straightforward solution is to stretch the diffusion as shown in Fig. 5(b), thus introducing an extra area. A more efficient solution is given in Fig. 5(c) by using jogs. A real situation is shown in Fig. 3 on the leftmost side of the cell.
2) The horizontal distance between a contact and a vertical metal wire that belong to different nets must exceed one unit. To satisfy this rule, the layout in Fig. 6(a) was stretched as shown in Fig. 6(b). A more efficient solution is illustrated in Fig. 6(c). Notice that such a situation occurs in Fig. 3.
3) Contacts on two consecutive gates are not allowed on the outer track in either side. The violation in Fig. 7(a) can be resolved by stretching, as shown in Fig. 7(b), or, more efficiently, by searching for a solution in which the consecutive contacts are either on different tracks or on the inner track, as shown in Fig. 7(c).

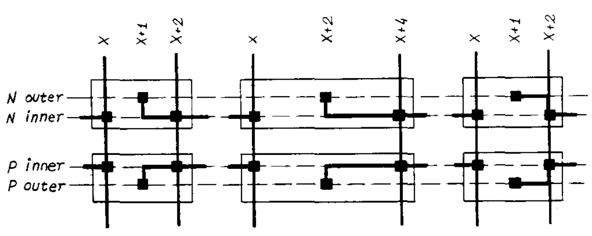
Since routing is allowed over the two transistor rows in a prescribed number of tracks, the situation seems reminiscent of one-dimensional routing, for which the track assignment problem can be solved using interval graph coloring [8]. Such a solution restricts the metal to be instantiated as straight lines. Moreover, interval graph coloring cannot obtain an optimal layout under the spacing rules mentioned above. Also, the above examples indicate that the jog introduction is beneficial. Consequently, an algorithm supporting jogs is in order.

To handle this unique situation, we propose a dynamic programming algorithm which accomplishes the wiring whenever possible with minimal space insertion. A dynamic programming approach has been proposed for the single-row routing problem (e.g., [9], [2]). The functionality of our algorithm is to assign poly-to-metal and diffusion-to-metal contacts, and horizontal metal segments, to the wiring tracks. The vertical metal segments which implement the jogs are determined implicitly by this assignment. To allow the introduction of jogs, we model the contacts as zero-length intervals, and break up wiring segments into unit-length intervals. Initially, each interval is assigned to its original x coordinate as was decided



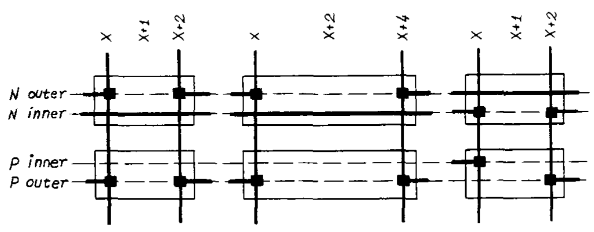
(a) (b) (c)

Fig. 5. Accommodating spacing rule no. 1.



(a) (b) (c)

Fig. 6. Accommodating spacing rule no. 2.



(a) (b) (c)

Fig. 7. Accommodating spacing rule no. 3.

by the placement phase, while its y coordinate, namely, the wiring track to which it will be assigned, is not determined yet. Then, the algorithm assigns the intervals to the wiring tracks and specifies the horizontal spacing. This way, zero-length and unit-length intervals originating from the same net may be assigned to different wiring tracks, thus creating jogs everywhere along the span of the net (internal points as well as end points).

The dynamic programming algorithm proceeds as follows. First, the zero and unit intervals are sorted according to their left-end coordinate, where ties are broken arbitrarily. Then, they are processed in their order one at a time. Let t denote the current step of the algorithm I_t the currently processed interval, and x_t the left-end x coordinate of I_t. Since two items distant at more than two units are not affected by the spacing rules, only those intervals preceding I_t whose right-end x coordinate is greater than or equal to $x_t - 2$ may affect the decision to which track I_t or later intervals will be assigned. This set of intervals, called the *affecting intervals* at time t, is denoted by A_r, and is calculated as a preprocess. By definition, once an interval leaves A_q, it cannot reenter to any A_r, $r > q$.

Evidently, the only information relevant to the assignment of I_t and later intervals to the wiring tracks is how the intervals of A_t are distributed (permuted) among the wiring tracks. Let us identify a permutation of A_t's intervals with a state of the dynamic programming procedure at time t. To every state we associate two costs: a primary cost, which measures the minimal total horizontal spacing required so far to reach that state (due to the spacing rules discussed formerly), and a secondary cost, which measures the total length of vertical wires that have been introduced so far, due to jogs. Also, for every state we associate a vector whose ith element describes the total space inserted up to and including the original i x-coordinate, $1 \leq i \leq x_t$.

Let S_t denote the set of all the states after I_t has been processed. Then, S_{t+1} is generated as follows. For every state in S_t the algorithm attempts to generate every feasible continuation by assigning I_{t+1} to every wiring track. Every legal assignment defines a state of S_{t+1}. In the continuation of a state of S_t to a state of S_{t+1} the spacing vector allows us to know whether or not a space must be inserted due to the current continuation. The spacing vector of a state in S_{t+1} is defined by the spacing vector of its predecessor state in S_t and the space insertion (if any) resulting by the assignment of I_{t+1}.

Generally, several different states in S_t may lead to the same permutation of A_{t+1}, thus defining identical states in S_{t+1}. Since we are concerned only with optimal solutions, the one yielding the lowest cost is saved, while the previous one is discarded. A tie in the primary cost is broken by the secondary cost. A tie in both of them is broken arbitrarily. After the last interval has been processed, the algorithm selects that state with the minimal cost, and the optimal solution is retrieved by going backwards in time.

The number of states at each time step is defined by the different permutations of the affecting intervals, whose number is proportional to the number of wiring tracks. This follows from the fact that k wiring tracks may occupy at most k intervals, and when I_{t+1} is assigned, only x_{t+1}, x_t, and x_{t-1} affect the assignment. Since the intervals (contacts and unit-length wires) are processed one at a time, the overall time complexity of the algorithm is exponential in the number of wiring tracks and linear in the number of intervals. Practically, there are few wiring tracks, so the number of states at every time never exceeds several hundred, which makes the dynamic programming approach attractive.

In [9] the states were coded in a data structure called *trie*, which adds a factor of $O(k \log k)$ to the time complexity, where k is the number of wiring tracks. We propose to use a hashing function for the state coding, thus reducing the above factor to $O(k)$.

IV. Discussion

We have described a new algorithmic method for laying out circuits from their schematics which has several advantages over other published algorithms, especially in terms of running times and the quality of the layouts produced. The algorithm has a clear formulation and at the same time is practical in the context of a cell generation tool. Our framework can accommodate additional optimization criteria as well as constraints that arise in certain design environments.

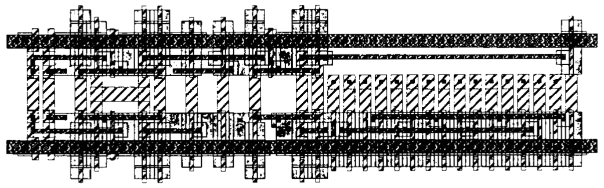


Fig. 8. A large cell, with more N-type than P-type transistors.

The algorithms presented in this paper have been programmed in Pascal. The typical run time for a 60-transistor circuit is a few seconds on an IBM 3090 machine, compared to several hours (or even days) needed by hand layout. To test the programs, an entire library (containing more than 300 cells) was generated automatically and the results were compared to an existing version that was done by hand. The whole process took one day (with one designer), compared to 3 months manually. In terms of quality (cell width and performance), the automatically generated cells were never worse—and were often better—than those created manually. The image of two wiring tracks on each side has been proved a practical one to comprise an extended set of leaf cells. Figs. 3 and 8 exemplify two typical cells, displaying features that were traditionally considered out of the scope for automatic tools, such as routing jogs, dealing with an unequal number of P-type and N-type transistors, and polysilicon "bridges" between adjacent gates.

The recent success of leaf-cell generators increases the demands for more powerful tools to handle much larger cells. Our experience indicates that for the transistor placement problem the DFS procedure reached its limit. Therefore, new directions such as partitioning, clustering, and grouping must be employed to break the problem into smaller pieces which the DFS procedure can handle efficiently. As for routing, since the complexity of the algorithm is dominated by the number of wiring tracks, it is equally applicable for much larger cells.

Acknowledgment

The authors would like to thank I. Berger for his support and encouragement, E. Gofman for his contribution to the ideas that led to the algorithm of Section III-B, and to R. Nair of the IBM Watson Research Center at Yorktown Heights, NY, for making a layout back-end system available. They would also like to thank all the anonymous reviewers for their helpful comments.

References

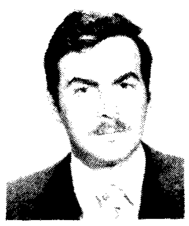
[1] J. Bhasker and S. Sahni, "Optimal linear arrangement of circuit components," *J. VLSI Comput. Syst.*, pp. 87-109, 1987.
[2] D. Du, O. Ibarra and F. Nevada, "Single-row routing with crossover bound," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 190-201, 1987.
[3] S. Even, *Graph Algorithms*. Rockville, MD: Computer Science Press, 1979.

[4] D. D. Hill, "Sc2—A hybrid automatic layout system," in *Proc. ICCAD*, 1985, pp. 172-174.
[5] R. Müller and T. Lengauer, "Linear algorithms for two CMOS layout problems," in *Proc. Aegean Workshop Computing*, July 1986.
[6] 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.
[7] R. Nair, "MLG—A case for virtual grid symbolic layout without compaction," in *Proc. ICCAD*, 1987, pp. 180-183.
[8] T. Ohtsuki *et al.*, "One-dimensional logic gate assignment and interval graphs," *IEEE Trans. Circuits Syst.*, vol. CAS-26, pp. 675-684, 1979.
[9] R. Raghavan and S. Sahani, "Optimal single row routing," in *Proc. ACM/IEEE Design Automat. Conf.*, 1982, pp. 38-45.
[10] T. Uehara and W. M. vanCleemput, "Optimal layout of CMOS functional arrays," *IEEE Trans. Comput.*, vol. C-30, pp. 305-312, 1981.
[11] S. Wimer, R. Y. Pinter, and J. A. Feldman, "Optimal chaining of CMOS transistors in a functional cell," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 795-801, 1987.

*

Reuven Bar-Yehuda received the B.Sc., M.Sc., and Ph.D. degrees in computer science from the Technion—Israel Institute of Technology, Haifa, Israel, in 1978, 1980, and 1983, respectively.

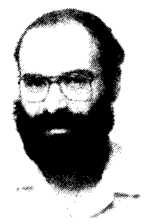
Since 1983 he has been a Lecturer in the Computer Science Department at the Technion. During the years 1984-1986, he was a visiting Assistant Professor at Duke University, Durham, NC. Since 1986 he has been a faculty member at the IBM Israel Scientific Center, Haifa, Israel. His research interests include combinational optimization algorithms, distributed algorithms, radio communication, layout of VLSI circuits, and computational geometry.



Jack A. Feldman was born in Bucharest, Romania. He received the B.Sc. and M.Sc. degrees in computer engineering from the Polytechnic Institute of Bucharest in 1983 and the M.Sc. degree in electrical engineering from the Technion—Israel Institute of Technology, Haifa, Israel, in 1988.

In 1985, he joined the IBM Israel Scientific Center as a Research Fellow. Since 1988 he has been a Research Staff Member at the IBM Scientific Center, Haifa, Israel. His current interest is in algorithms for layout of VLSI circuits and systems.

*



Ron Y. Pinter received the B.Sc. degree in computer science from the Techion—Israel Institute of Technology, Haifa, Israel, in 1975 and the S.M. and Ph.D. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, in 1980 and 1982, respectively.

During the years 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, Haifa, Israel, where he was the manager of the Programming Languages Group. He is also an Adjunct Senior Lecturer in the Electrical Engineering Department at the Technion, and has taught at the Hebrew University, Jerusalem. For the academic year 1988/89, Dr. Pinter is visiting the Department of Computer Science, Yale University, New Haven, CT. His research interests include parallel programming techniques, code generation algorithms, layout for integrated circuits, and computational geometry.

Dr. Pinter is a member of the Association for Computing Machinery and ACM SIGPLAN.

*

Shmuel Wimer, for a photograph and a biography, please see page 145 of the February 1989 issue of this TRANSACTIONS.