GM_Plan: A Gate Matrix Layout Algorithm Based on Artificial Intelligence Planning Techniques

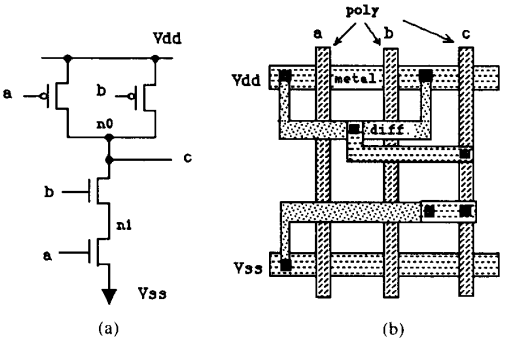
YU HEN HU, SENIOR MEMBER, IEEE, AND SAO-JIE CHEN, MEMBER, IEEE

Abstract—**In this paper, the CMOS gate matrix layout problem is formulated and solved as an artificial intelligence planning problem in which a "plan" (the solution algorithm) is to be generated to achieve a "goal" (the gate matrix layout). The overall goal consists of many subgoals, each of which corresponds to the placement of a gate to a slot, and to the routing of associated nets connecting to that gate. As different nets compete for track (resource) usage, these subgoals interact (interfere) with each other, rendering suboptimal solutions. In this paper, such interaction among subgoals is managed with two artificial intelligence planning techniques: hierarchical subgoal organization and domain independent search control policies. The subgoal hierarchy facilitates an objective classification of the subgoals into priority classes according to a proposed distance measure of connectivity. Two search control policies (general problem solving heuristics)—*most-constraint* (MC) and *least impact* (LI)—are used to guide the search process. The MC policy states that the subgoal whose solution has most constraints should be attempted first. The LI policy states that among many alternate solutions, the one that consumes the least amount of resources, and hence, preserves the most flexibility should be chosen. Using these techniques, we developed a planning-based gate matrix layout algorithm, called GM_Plan, which combines the gate placement and net routing into a single, incremental problem solving loop. Encouraging results have been observed in a number of test examples.**

I. INTRODUCTION

GATE matrix layout is a systematic CMOS layout methodology developed at Bell Laboratories [1]. Owing to the regular structure and relatively high gate density, the gate matrix layout has become an increasingly popular layout design style and has been adopted by a number of automatic layout and leaf-cell module generation systems [2]–[5].

In a gate matrix style layout, a *gate* refers to a circuit node which connects to both a pMOS transistor and an nMOS transistor. In the complementary CMOS layout style, these nodes usually correspond to the common input gate terminals[1] of the pMOS and nMOS transistors. This may be the reason they bear the name *gates*. Sometimes a gate may connect to other transistor terminals of the pMOS and nMOS transistors. For example, the output node c in Fig. 1(a) will be regarded as a gate. Each gate in the gate matrix layout is realized by a vertical polysilicon wire which are placed in parallel forming a linear array as shown in Fig. 1(b).



(a) (b)

Fig. 1. (a) A two-input NAND circuit. (b) Gate matrix layout of the NAND circuit.

Transistors are realized by laying out horizontally diffusion wires across corresponding polysilicon gate segments. Interconnections are made through metal wire segments. An example of a gate matrix layout of the circuit shown in Fig. 1(a) is depicted in Fig. 1(b). Both the diffusion wires and the metal wires may overlap each other occupying the same horizontal track. Moreover, adjacent wire segments on the same track can be lumped together to form a single *net*. On the circuit schematic of Fig. 1(a), a net $n1$ may be identified as the path from V_{ss} to the output node c, trespassing gate a and b. Note that the vertical gates and the horizontal nets intersect each other, forming a grid *matrix* which may inspire the name of *gate matrix layout*.

The realization of a gate matrix layout requires the solution of two major problems: placing gates in a linear array of slots and routing nets on horizontal tracks. Since the number of columns (slots) is fixed, the design objective thus is to select an appropriate ordering of gates in order to minimize the total number of tracks taken by the nets. It has been shown that this is a difficult (*NP*-complete) problem [6], [24]. Hence, heuristics are used to derive practical and efficient gate matrix layout algorithms.

In the past, there have been two major approaches for solving the gate matrix layout problem. The first approach

Manuscript received January 30, 1989; revised June 21, 1989. This work was supported in part by DARPA under Contract MDA 903-86-C-0182 and by the National Science Foundation under Contract MIP-8896111. This paper was recommended by Associate Editor A. E. Dunlop.

Y. H. Hu is with the Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison, WI 53706.

S.-J. Chen is with the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, 10764 China.

IEEE Log Number 9035472.

[1]In gate matrix layout literatures, the *gate* refers to the polysilicon gate terminal of a MOS transistor (the other two terminals being the source and the drain), rather than a logic gate such as AND gate and OR gate.

aims at finding a good "gate ordering" in order to minimize the total number of tracks. After the gate ordering is determined, a left-edge algorithm [7] then will be applied to perform the task of track assignment (net routing). To solve the one-dimensional gate assignment problem, Ohtsuki *et al.* [8] used a graph-theoretic approach and formulated the problem into an interval graph problem. Wing [9], [10] proposed a solution by finding a minimal augmentation to transform a connection graph, which is derived from the topology of the given circuit into an interval graph. Another similar heuristic approach was presented by Li [11] which finds the minimal augmentation on a "vertex-versus-dominant-gates" matrix. In addition to these interval graph-based approaches, Cheng [12] proposed a min-cut algorithm and Hwang *et al.* [13] used a "modified min-net-cut" method to solve this linear placement problem.

The second approach, proposed by Asano and Tanaka [14], attacked the gate matrix layout problem by first "assigning nets" to tracks. The gate ordering was subsequently determined. An "exact algorithm" using a permutation tree representation and exhaustive search was presented to ensure an optimal solution [15]. Since this algorithm is impractical for large-size circuits, a suboptimal "approximate algorithm" based on a greedy heuristic has also been proposed by Asano [15]. Tested with benchmark circuits, Huang and Wing [16] recently reported that the Asano's approximation algorithm always takes less time to compute and results in using fewer tracks compared with the interval graph-based method.

Although the gate matrix layout style has the potential to "combine the placement and routing into one process" [17, p. 14], these existing methods often ignore this fact and regard them as two separate subproblems to be solved one after the other. Since the tasks of gate placement and net routing are tightly coupled, solving them one at a time may lead to inferior solutions for lacking the feedback from the other half of the problem. In this paper, we propose a novel gate matrix layout algorithm, called GM_Plan, in which the processes of gate placement and net routing are combined into a single, incremental problem solving loop.

In GM_Plan, the task of gate matrix layout is formulated as an artificial intelligence planning problem [18]–[20] which consists of two interwoven subgoals: gate placement and net routing. The basic planning approach used in GM_Plan is to partition the problem on hand into subproblems (subgoals) of different priority classes. Subgoals with a higher priority will be attempted first. Then, the partial plan (partial solution) will be passed down as constraints to subproblems of lower priority classes. In this paper, a new clustering distance measure is proposed to facilitate this partitioning process. During the problem solving process, subplans, which are steps to achieve the subgoals of placing a gate and routing corresponding nets, are generated under the control of two domain independent search control policies: the *most-constraint* (MC) policy and the *least impact* (LI) policy. These policies provide guidelines for deriving domain specific problem solving strategies (heuristics) to manage the interaction among subgoals.

In Section II, the gate matrix layout problem will be formulated with some of its characteristics reviewed. In Section III, planning techniques and domain specific heuristics for solving the gate matrix layout problem will be presented. This leads to the development of the design process model and the GM_Plan algorithm in Section IV. Finally, the complexity analysis, simulation examples, and some implementation issues will be discussed in Section V.

II. The Gate Matrix Layout Problem

To produce a gate matrix layout, a CMOS circuit will need to be described in terms of a set of gates $\{g(i)\}$, and a set of nets $\{n(i)\}$. The set of nets connecting to the same gate will be called a gate-net (GN) set. For example, in Fig. 1(a), the GN set corresponding to gate b is $N(b) = \{n0, n1\}$. Similarly, all gates connecting to the same net will be called a net-gate (NG) set. In Fig. 1(a), the set of gates connecting to the net $n1$ form the NG set $G(n1) = \{a, b, c\}$. The topology of the circuit thus can be uniquely described by a GN table listing all the gates and associated GN sets; or equivalently, by a NG table listing all the nets and associated NG sets.

In gate matrix layout literatures, a design is often depicted in a symbolic format as shown in Fig. 2(a). The rectangle region represents a MOS transistor channel area. Vertical wires are polysilicon gates. Horizontal wires represent interconnections in metal layers. The upward/downward vertical arrows indicate connections to V_{ss} or V_{dd}. For convenience, in the rest of this paper, we shall use an even more abstract notation as shown in Fig. 2(b) where only the gate ordering and corresponding track assignments of nets are given. The actual wiring can be incorporated into the final layout easily once the gate placement and net assignment is accomplished.

Using the above abstract representation, a gate matrix layout problem can be formally defined as follows.

Given an NG table or a GN table, a set of linearly aligned array of slots $\{s(i)\}$ and a set of empty tracks $\{t(i)\}$, find a one-to-one mapping from the set of gates $\{g(i)\}$ to the set of slots $\{s(i)\}$:

$$f1: g(i) \rightarrow s(i)$$

and a second mapping from the set of nets $\{n(i)\}$ onto the sets of tracks $\{t(j)\}$:

$$f2: n(i) \rightarrow t(j)$$

such that the total number of tracks required is minimum.

2.1 Lower Bound of the Solution

In a gate matrix layout, the number of tracks required must be at least equal to the largest number of nets connecting to a single gate in the given circuit. Clearly, this number serves as a lower bound for the number of tracks

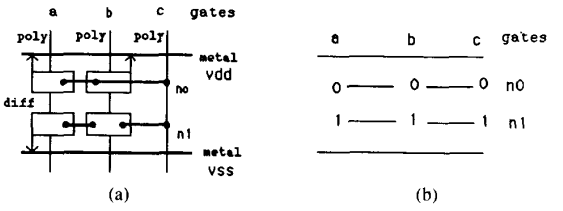


Fig. 2. (a) The symbolic layout of the NAND circuit. (b) The symbolic representation of the NAND circuit.

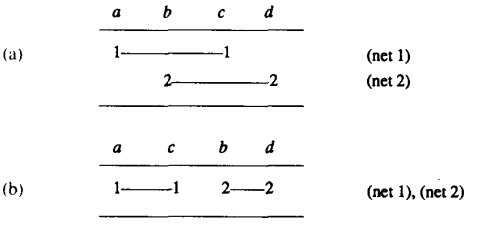


Fig. 3. Interactions among subgoals.

required by that gate matrix layout problem. If the resulting gate matrix layout uses no more tracks than this lower bound, the solution is optimal. However, this lower bound will not be optimal with the presence of *cycles* in a given circuit topology. For example, in the layout below, the path *a-b-c-a* forms a *direct cycle*, and hence, three tracks are needed even if each gate connects to at most two nets:

a	*b*	*c*	
1	—	1	(net 1)
	2	— 2	(net 2)
3	——	—— 3	(net 3).

There are also *indirect cycles*, or sometimes called *hypercycle* in [21, p. 55]. Below is an example of a gate matrix layout where the path *a-b-c-d-a* forms an indirect cycle:

a	*b*	*c*	*d*	
1	1	4	— 4	(net 1)
		2	— 2	(net 2)
3	———		— 3	(net 3).

Yu proved that the size of a "bottleneck hypercycle" is the optimal lower bound of track numbers required in a gate matrix layout [21, p. 56]. Unfortunately, finding this theoretical optimal lower bound is a very difficult (an *NP*-complete) problem. In practice, it does not provide any guidance for finding an optimal solution.

III. Gate Matrix Layout: A Planning Approach

Planning is a general problem solving technique developed in artificial intelligence researches. A *plan* is a course of actions (*subplans*) to achieve a certain *goal* which usually consists of a collection of *subgoals*. Subgoals often interfere (interact) with each other rendering suboptimal or infeasible solutions. The objective of planning, thus is to derive a partial ordering of the subplans such that the overhead due to interaction among subgoals can be reduced. In the gate matrix layout problem, a subgoal is identified as the collection of the following two subsequent tasks:

1) to place a gate in a particular slot;
2) to route corresponding nets on available tracks.

If the subgoal of each gate in the given GN table is accomplished, the gate matrix layout (the overall goal) will be achieved. Since a net must be connected to two or more gates, these subgoals interact with each other competing for empty tracks to route their corresponding nets. For example, in Fig. 3(a), net 1 and net 2 both demand the track segment between gate *b* and gate *c*. As a result, two tracks are used.

If, on the other hand, the order of these two gates are interchanged, then only one track will be sufficient to route both nets as shown in Fig. 3(b). Hence, formulated as a planning problem, the objective of a gate matrix layout is to find a proper ordering of subplans (gate assignment and net routing) so as to minimize the overhead due to subgoal interaction (excessive usage of tracks). In this paper, two planning techniques, hierarchical subgoal organization and domain independent search control, will be applied to develop the algorithm—GM_Plan.

3.1 Hierarchical Subgoal Organization

Hierarchical subgoal organization by partitioning a problem into subproblems of different priority classes is a popular "divide-and-conquer" technique used by many planning-based problem solvers. The main objective of this partitioning step is to improve the search efficiency to overcome the combinatorial explosion of the solution space [30]. If only the highest priority subproblems on the top level are considered, a partial plan achieving these subgoals can be accomplished rather quickly because there are fewer subgoals to be considered at the same time. The underlying assumption of this technique is that subproblems can be arranged according to the degree of difficulty (criticality or relevance) in achieving their corresponding subgoals. This calls for an objective measure to evaluate the degree of criticality or relevance among subgoals. In GM_Plan, a novel *distance measure* and a notion of *nearest-neighbor group* are proposed to facilitate this hierarchical classification of subgoals.

3.1.1 Connectivity-Based Distance Measure: For serially connected transistors, permutation of their order does not affect the correctness of the specification. A similar argument can also be applied to parallel connection transistors.[2] Hence, it would be meaningless to measure

[2]There are special cases where this permutation is prohibited: for example, interchanging two transistors may cause a charge sharing problem, or the driving capability of the transistors will be affected due to this permutation of orders.

the "distance" between two gates in terms of the difference between their respective physical locations (i.e., slot numbers). Since the problem requirement is to connect all gates within a NG set, it seems more appropriate to consider two gates to be *adjacent* if they are linked *directly* by at least one net. Therefore, we propose a new definition of *distance* between two gates as follows.

Definition 1. Distance Between Two Gates:

Let $d(g1, g2)$ denote the *distance* between two different gates $g1$ and $g2$; then

$d(g1, g2) = 1$ if $\exists$ a net connecting both $g1$ and $g2$, and

$d(g1, g2) = n (n \geq 2)$ if

$d(g1, g2) > n - 1$ and

$\exists$ another gate, say, $g3$ such that

$d(g1, g3) = n - 1$, and $d(g2, g3) = 1$; or

$d(g1, g3) = 1$, and $d(g2, g3) = n - 1$.

This definition can be generalized to define the distance between two sets of gates.

Definition 2. Distance Between Two Sets of Gates: Let $G(1)$ and $G(2)$ be two disjoint, nonempty sets of gates, and $d(G(1), G(2))$ denote the *distance* between $G(1)$ and $G(2)$; then

$$d(G(1), G(2)) = n \text{ if } \exists\ g1 \in G(1) \text{ and } g2 \in G(2), \text{ such that } \mathrm{Min}[d(g1, g2)] = n.$$

For illustration purposes, in Fig. 4, $d(g1, gi) = 1$, and $d(g2, gi) = 1$, but $d(g1, g2) = 2$. Also, the distance between the gate set $G(1)$ and $G(2)$ is two. That is, $d(G(1), G(2)) = 2$. Please note that this distance measure is based on the *connectivity* between gates, or sets of gates, and hence, has nothing to do with their physical locations.

3.1.2. Nearest-Neighbor Group: In GM_Plan, the set of gates that have been placed is called *PL*aced *G*ate *S*et (PLGS). Using the proposed distance measure, the remaining unplaced gates can be classified into different priority classes, each of which contains gates having the same distance from the PLGS. On top of this hierarchy is the *nearest-neighbor group* (NNG) that consists of the most relevant gates (most critical subgoals). Each member in the NNG has direct connection with the PLGS. That is, the distance from each gate in the NNG to the PLGS is equal to one. For example, in Fig. 5, if PLGS = $\{sg\}$, then NNG = $\{g2, g3, g4\}$.

It should be noted that the contents in the NNG, and the remaining levels in the hierarchy can be incrementally updated during the design process. Whenever a gate is placed, it *pulls* its distance-1 neighbors from a lower priority class into the current NNG. This change, in turn, ripples to the rest of levels in the hierarchy by updating their respective contents. In GM_Plan, however, only the NNG is scanned in order to find the next gate to be placed. Hence, only the top of this subgoal hierarchy, namely NNG, is maintained and incrementally updated.

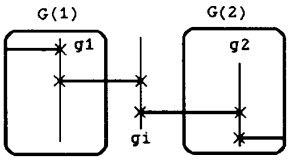


Fig. 4. Distances between gates, and sets of gates.

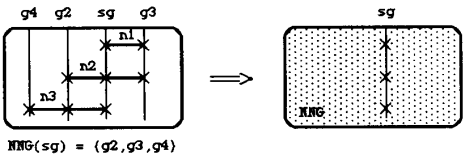


Fig. 5. The concept of the NNG.

3.2. Domain Independent Search Control Policies

Domain independent search control is a technique concerned with the control of planning decisions—knowing when to and when not to make commitments of resources in a subplan to achieve a subgoal. To avoid premature commitments,[3] a control policy, called "least-commitment," was used in [20] which avoids making a decision (committing certain resources) until compelling evidence appears. However, when applied to the gate matrix layout problem, this policy leads to too many deferred commitments and renders the later decision-making process more difficult if not infeasible. In the past, we have devised two domain independent search control policies [22], [23] which may also be applicable to the current gate matrix layout problem.

The first one, MC, is the policy of selecting a subgoal that has most constraints, i.e., one that has the smallest solution space, and achieving it before other subgoals. The idea is that since all subgoals must be accomplished to achieve the overall goal, the one with most constraints must be addressed first, even at the expense of consuming some resources of potential use by other subgoals. In GM_Plan, this MC policy will be applied to the gate selection and net selection phases.

The second one, LI, is the policy of choosing a solution among many feasible ones to the subgoal so as to have the last impact on the remaining subgoals. The rationale is to preserve as much flexibility or as many resources as possible for achieving future subgoals. In GM_Plan, the LI policy is applied during the gate assignment and net routing phases.

IV. GM_Plan: The Design Process Model

Based on the above two planning techniques, the design process model of GM_Plan can now be presented. In GM_Plan, the two subproblems (gate placement and net routing), which constitute a subgoal, are solved for each gate and corresponding nets iteratively. At the beginning

[3]Premature commitment of scarce resources to certain subgoals should be avoided because it may cause other subgoals to take unnecessary detours and render the overall solution suboptimal.

of each iteration, a gate is selected from the set of unplaced gates and placed according to the MC and LI policies. After a gate is placed, these two policies will be applied again to choose a subset of nets connecting the current gate and to route them on available tracks. This concludes a single iteration. The routing of the remaining nets, which connect to the current gate, but are not routed in the current iteration, will be deferred until a later *deferred routing* phase or the final *wrap-up* routing phase.

A distinctive feature of GM_Plan is to route some nets to tracks before all the gates are placed. A main advantage of this approach is that the net routing information can be fed back as constraints to later iterations so that more sensible decisions (gate selection and slot assignment) can be made. Moreover, in many occasions, there is only one unique solution available (e.g., only one track available to route a net). Hence, routing that net immediately after the current gate is placed does not constitute a *premature commitment* of resources (tracks), and should not affect the optimality of the overall solution. For example, in GM_Plan, the first gate to be placed, called the *seed gate*, is the gate having the maximum number of nets in its GN set. After this *seed gate* is placed, all the nets connecting to it will be routed immediately to tracks as this action will not affect the optimality of the solution.

Conventional gate matrix layout algorithms defer the routing of nets until all gates are placed. Owing to the lack of routing information, the gate placement decisions are made largely independent of the requirements of net routing. On the other end, if we route *every* net connecting to the current gate right after it is placed, we may end up with suboptimal solutions due to premature commitment.[4] In GM_Plan, compromises are sought by i) routing only a portion of nets immediately after the gate is placed, and ii) deferring further actions on the remaining nets until two later stages (the deferred routing phase, and the wrap-up routing phase), where more information will be available to facilitate a better decision.

In the rest of this section, the implementation details of the gate placement process, and the net routing process will be discussed.

4.1. The Gate Placement Process

Gate placement in GM_Plan is implemented in two steps: gate selection (selecting a gate), a slot assignment (choosing a slot). In each interation, the following steps will be applied.

4.1.1 Gate Selection: The task here is to decide which, among all the yet-to-be placed gates, should be selected and placed next. The MC policy is used to choose a gate presently with the most constraints to be placed. In GM_Plan, the MC gate is interpreted as the gate that has the most relevance (connections) to the PLGS. The degree of relevance (criticality) is expressed with the connectivity-based distance measure defined earlier in Section III-3.1. Whenever a tie is encountered, more domain specific heuristic rules, based on the same MC policy, will be used to break the tie. These gate selection rules, arranged in the order of their priorities, are listed below.

Gate selection procedure:

a) Select the gate g such that the intersection between its GN set $N(g)$ and the set of nets corresponding to the PLGS set, $N(\text{PLGS})$, has the maximum number of nets.

b) If there is more than one such gate or no such gate, select the gate that is of distance one to the seed gate.

c) If there is more than one such gate or no such gate, select the gate that has a maximum number of nets per gate. This is also the rule used for selecting the seed gate at beginning as the first gate to be placed.

d) If there is more than one such gate, arbitrarily select one.

4.1.2. Slot Assignment: Once a gate is selected, it will be placed into a slot. In GM_Plan, slots are dynamically defined as the interval between each pair of adjacent placed gates, including the two ends of the array of placed gates. The total number of possible slots at present iteration, is the total number of the currently placed-gates plus one.

The LI domain independent search control policy, which governs the slot assignment process, is accomplished using heuristic research. To avoid an exhaustive search for every slot, those slots whose neighboring gates having no direct connection to the current gate will be excluded from the candidate list.

The heuristic search is accomplished by evaluating a cost function f for each of the remaining slots. This cost function consists of a linear combination of four terms:

$$f = h0\,(\text{connection cost of fixed nets}) + h1\,(\text{expansion cost of fixed nets}) + h2\,(\text{connection cost of floating nets}) + h3\,(\text{expansion cost of floating nets}).$$

Here the *fixed nets* are nets which have already been routed to a track; and the *floating nets* are those yet to be routed. The *connecting cost* is an estimate of the increase in track area usage due to the connection of nets to the current gate should it be placed into that slot. The *expansion cost* is an estimate of the increase in track area usage due to the expansion (stretching) of nets trespassing that slot.

Four possible scenarios are illustrated in Fig. 6 where g is the current gate to be placed in a slot between gl and gr. Nets $n1$ and $n2$ are both fixed nets connecting to the current gate and contribute to the cost of $h0$. In the case of $n1$, as it extends to both sides of the current gate, only 1 unit of cost is assigned. In the case of $n2$, as it extends to only one side of the current gate, the physical length (number of slots it covers) will be used as an estimate of its cost. Nets $n3$ and $n4$ are examples of $h1$ type cost, $n5$, $n6$ are examples of $h2$ type cost, and $n7$, $n8$ are examples of $h3$ type cost. In GM_Plan, each net of $h1$ type is as-

[4] In fact, this corresponds to the exercise of the so called *greedy heuristic* [24].

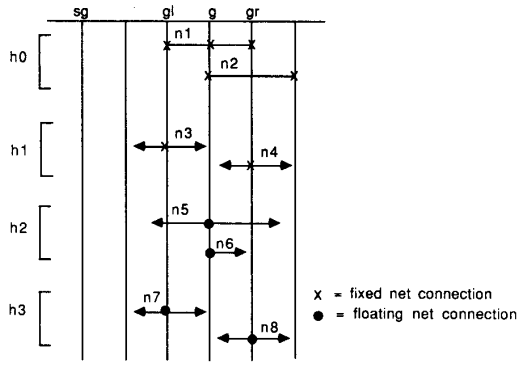


Fig. 6. The Heuristic cost functions.

signed with 1 unit of cost. As for floating nets of both the $h2$ and $h3$ types, only $1/2$ unit of cost will be assigned to each of them. For example, in Fig. 6, $h0 = 1 + 2 = 3$, $h1 = 2$, $h2 = 2(0.5) = 1$, and $h3 = 2(0.5) = 1$. Hence, the total cost is

$$f = 3 + 2 + 1 + 1 = 7.$$

Since the value of this cost function f is a rough estimate of the increase in track area usage when the current gate is placed to the particular slot. According to the LI policy, one should choose the slot which minimizes the value of f. The detailed slot assignment procedures are now summarized below.

Slot assignment procedure:

a) Choose the slot that, when the selected gate is placed, no additional tracks will be added immediately.

b) If more than one slot satisfies *a*), go to *c*); otherwise if no such slot exists, (i.e., some tracks have to be added no matter what slot is assigned), choose the slot that minimizes the number of tracks to be added.

c) If more than one slot satisfies a) or b), choose the slot that minimizes the heuristic cost function f.

d) If more than one slot satisfies c), choose the one that is the closest to one of the two ends of the gate matrix layout.

e) If there is still more than one such slot, arbitrarily choose one.

4.2. The Net Routing Process

After a gate is placed, the next step is to route the nets connecting to that gate by assigning them to unoccupied tracks. As mentioned above, in GM_Plan, nets are distinguished into "fixed nets" and "floating nets." First, the fixed nets, having been assigned to specific tracks in earlier iterations, present constraints which must be satisfied. Hence, tracks on the currently selected slot must be assigned to fixed nets that are either connected to the current gate, or simply trespassing it. Moreover, a track may be *reserved* by a neighboring *hanging* fixed net which still has more gates, not yet in the PLGS, to be connected. These constraints could significantly reduce the number of tracks needed to be examined, and therefore, enhance the search efficiency of the algorithm.

As for the floating nets, they will be converted into fixed nets and routed to tracks immediately if the following unique_route criteria, mentioned earlier, is met.

Convert a floating net to a fixed net if i) it connects to two or more gates in the PLGS, and ii) there is only one unique track (among the existing vacant tracks) which can route it.

In GM_Plan, the routing of the remaining floating nets will be accomplished later in the *deferred routing* phase, or the final *wrap-up* routing phase. Deferred routing is performed after all the distance-1 neighboring gates to the seed gates are placed. Since all the NNG gates of the seed gate are usually tightly coupled, postponing routing the remaining floating nets after all these gates are placed will enable us to pay more attention to the interactions among these subgoals. The net selection and routing procedures during the deferred routing phase are also implemented with domain specific heuristics based on the MC and LI policies.

4.2.1. Net Selection: Since not every net will be routed during the deferred routing phase, the MC policy is applied to first identify the subset of the floating nets to be converted into fixed nets, and next help ordering the nets in this subset. In order to identify this subset of nets, we have the following two heuristic rules.

1) Net selection procedure

a) Select a floating net into the to-be-converted subset of nets if it only connects to two gates.

b) Select a floating net into the to-be-converted subset of nets if it connects to two or more gates in the placed gate set PLGS, and it can be routed to an unreserved tracks without adding new tracks.

After these subsets of nets are identified, its individual members will be routed to tracks one by one according to a prioritized order determined by the heuristics below.

2) Net selection procedure

a) Select the net that has the largest number of gates in its NG set.

b) If more than one net satisfies a), select the net that has the largest number of gates in the intersection between its NG set and the PLGS.

c) If there is more than one such net, arbitrarily select one.

4.2.2. Track Assignment: To route the selected subset of nets to tracks, first, constraints are used to eliminate tracks which have already taken by fixed nets from the candidate list. Tracks which are "*reserved*" by a *hanging* fixed net shall also be avoided when possible. New tracks will be added only if the number of available tracks is less than that of the unrouted nets. The heuristic rules for this purpose are the following.

Track assignment procedure:

a) Choose the track whose vacant track area best fits the net to be routed.

b) If more than one such track exists, choose the track on which fewer nets may be routed.

c) If more than one such track exits, arbitrarily choose one.

4.3. The Deferred Routing and the Wrap-Up Routing Phases

As explained earlier, the purpose of deferred routing and wrap-up routing is to route those floating nets which have not been converted into fixed nets during previous iterations. The difference between these two phases is that in the deferred routing phase, only high priority subgoals, selected using the net selection procedure (1) will be routed. While in the wrap-up routing phase, which is carried out after all the gates are placed, *every* unrouted net has to be routed, with the procedure outlined in net selection (2) and track assignment above, in order to complete the gate matrix layout.

The timing for deferred routing is a delicate issue. On one hand, we do not want to put off making any routing decisions until the wrap-up routing phase since the gate assignment process will not be benefit from the availability of the net routing information. On the other hand, we want to withhold making premature decisions. The compromise made in GM_Plan—to perform deferred routing once after all NNG gates of the seed gate are placed—seems to be working satisfactorily for the set of test circuits. Clearly, for problems with many tightly couple gate clusters, more than one deferred routing may be needed. The criteria for determining the frequency and timing to perform deferred routing is still an open research question.

We are now ready to present the complete GM_Plan algorithm.

The GM_Plan Algorithm

```
Algorithm GM_Plan ():
0.  Prepare_Table (NG, GN);                    /* Prepare NG, GN tables.*/
1.  Unique_pl&route ()
2.    { select_seed (sg);                      /*Select a seed gate from the given GN table; and assign
3.      unique_place (sg);                       it to a slot. Route each net of the seed gate to a
4.      unique_route (N(sg));                    track.*/
5.      PLGS = {sg};                           /*Identify placed Gate-Set */
6.    }
7.  Form_NNG ()                                /*Select the nearest-neighbor group gates of the seed
                                                 gate from the set of unplaced gates */
8.    { NNG = G(N(sg))\{≈sg}; }                /*Nearest-Neighbor-Group. The notation "\" is the set
                                                 difference operator */
9.  CRGS = NNG;                                /*Current Gate-Set * /
10. Layout ()
11.   { While (CRGS!=NULL) repeat
12.     {Place_gate ()                         /*The gate selection procedure will be applied to select
13.       {select( g, CRGS) with MC policies;    a gate to be placed next. Then, the slot assignment
14.         place ( g, slot) with LI policies;   procedure will be applied to place this selected gate
          }                                     into an available slot. */
15.
16.      Route_net (N( g))                     /*The fixed nets will be routed first. The net selection
17.       {select (n, N( g)) with MC policies;   procedures and the net routing procedures will be
18.        if (routable) route (n, track) with LI applied also to route part of the floating nets */
           policies;
19.       }
20.      if (end of NNG) deferred_routing (N(sg));
21.      PLGS = PLGS ∪ { g };                  /* Adding g to the placed gate set */
22.      CRGS = {CRGS ∪ G(N( g)}\PLGS;         /* Update current gate set CRGS */
23.    }                                       /* End of While loop */
24.  }
25. Wrap_up_Route (all-nets).
```

4.4. An Example

Now, an example ($w1$) from [9] will be given to demonstrate the gate matrix layout procedures of GM_Plan.

The original input NG table contains 21 gates, and 18 nets:

Net | Gates	Net | Gates
n1 | 1 2 3	n10 | 3 11 12
n2 | 3 4	n11 | 11 14
n3 | 5 6	n12 | 11 16
n4 | 5 6 7	n13 | 14 15
n5 | 7 8	n14 | 16 17
n6 | 8 9	n15 | 13 19
n7 | 7 8 9	n16 | 13 18
n8 | 4 8 10 12	n17 | 19 20
n9 | 3 11 13	n18 | 18 21

It can be found that the seed gate is gate #3 ($sg = 3$). The associated gate set is $N(3) = \{n1, n2, n9, n10\}$. Hence, the NNG gates are derived as:

$$\begin{aligned}\text{NNG} &= G(N(3))\backslash\{3\} = \{G(n1) \cup G(n2) \\ &\quad \cup G(n9) \cup G(n10)\} \setminus \{3\} \\ &= \{\{1, 2, 3,\} \cup \{3, 4\} \cup \{3, 11, 13\} \\ &\quad \cup \{3, 11, 12\}\} \setminus \{3\} \\ &= \{1, 2, 4, 11, 12, 13\}.\end{aligned}$$

Some snapshots taken during the gate matrix layout process are presented in Fig. 7. In Fig. 7(a), the placement and routing of the first seed gate is shown. In Fig. 7(b), a second gate #11 is brought into the picture, and nets $n9$ and $n10$ are assigned to tracks and connected. However, nets $n11$ and $n12$ are not placed now due to lack of sufficient information. In Fig. 7(c), all the NNG gates of the seed gate are placed and the deferred routing performed. Note that as a result of deferred routing, nets $n11$, $n12$, of gate 11, and $n15$ of gate 13 are now routed to tracks becoming fixed nets. They are assigned to different tracks to avoid potential conflict. In Fig. 7(d) and (e), the snapshots before and after the final wrap-up routing are depicted. Since the final number of track used ($=4$) is equal to the lower bound (maximum number of nets connecting to the seed gate #3), it is an optimal solution.

The snapshots *before wrap-up routing* and *after wrap-up routing* illustrate the distinctive feature of GM_Plan: higher priorities are set for more critical tasks, such as gates with more net connections, and nets with more gate connections. Less demanding tasks then can be completed later without affecting the overall quality of the gate matrix layout.

V. Complexity Analysis, Simulation Results, and Implementation Issues

Two criteria are used in this paper to compare GM_Plan with other gate matrix layout algorithms. First, a theoretical complexity analysis is conducted to show that GM_Plan is a polynomial time algorithm. Next, the performance of GM_Plan is compared with the best known results by using a set of benchmark problems.

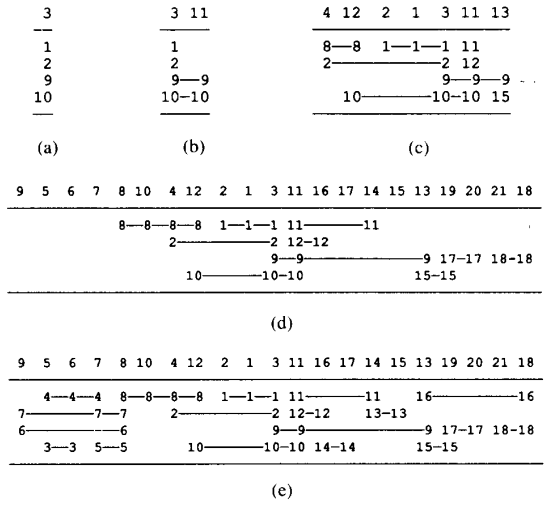


Fig. 7. Snapshots of the GM_Plan design example.

5.1. Complexity Analysis

Let us adopt the following set of notations: n = the number of gates, m = the number of nets, α = the max number of gates per net, β = the max number of nets per gate, and t_n = the number of tracks used for the gate matrix layout. Clearly, it must be $n \geq \alpha$ and $m \geq t_n \geq \beta$. Now, by referring to the GM_Plan algorithm, one has the following operation counts.

The procedure "select_seed," (line 2), which selects a seed-gate from the GN table, takes $O(n)$ time. The procedures, "unique_place" (line 3) and "unique_route" (line 4) both take only constant time. The NNG (line 8) can be formed in $O(\alpha^*\beta)$ time. The "Gate Selection" procedure (line 13) requires $O(n^*m)$ time units. This is because the intersection of $N(g)$ and $N(\text{PLGS})$ will need $O(m)$ time units, while there are at most n gates to be evaluated. The "Slot Assignment" procedure (line 14) also needs $O(n^*t)$ time, where each gate may connect to $O(t_n)$ nets, and at most $O(n)$ slots to be evaluated. The "Net Selection" procedure (line 17) needs at most $O(\beta)$ time. Also, the "Track Assignment" procedure (line 18) searches t_n tracks, and hence, needs $O(t_n)$ time to route a current net. There is also a hidden cost of $O(n^*t_n)$ in order to check the range of a routed net. The time spent on updating existing floating nets and perhaps converting some of them into fixed net is also $O(\beta)$. The "wrap_up_route" procedure (line 25) needs $O(n^*t_n)$ time. The same is also needed for the procedure "deferred_routing" in line 20.

To summarize, the maximum time required is $O(n^*m)$ *and the iteration count is* $O(n)$. Thus the overall time complexity of GM_Plan is $O(n^2m)$. As to the space complexity, we use two double-linked lists (taking $O(n)$ memory space) to store the placed gates and available slots, respectively, and a single-linked list (taking $O(t_n)$ space) to store the routing track information. In total, the memory space taken is of $O(n^*t_n)$.

The above numbers may be compared with the computation complexities of several existing gate matrix layout algorithms. The results are conveniently summarized in Table I. For the purpose of comparison, various notations are converted (with our best efforts) to equivalent notations used in this paper. However, a conclusion in terms of efficiencies of various algorithms cannot be drawn from this table due to different problem formulations and the lack of implementation details.

TABLE I
COMPARISON OF COMPUTATION COMPLEXITIES OF GATE MATRIX LAYOUT ALGORITHMS

Authors	*Ref.*	*Time complexity*	*Our Notation*	*Note*
Ohtsuki et al.	[8]	O(|V|*(|E|+|F|))	O(m(n + C))[a]	|V|: # of vertex, |E|: # of edges |F|: # of augmentation
Asano	[15]	$O(m^2*v)$	$O(m^3)$ to $O(m^2*m!)$	Exact method
		O(m log2m)	O(m log2m)	Approximate method
Li	[11]	$O(p^2 + q)$	$O(\alpha^2 + n)$	
Huang & Wing	[16]	$O(m^2)$	$O(m^2)$	
Hwang	[13]	O(N log N)	O(n log n)	
Hu & Chen			$O(n^2m)$	

[a] There is no equivalent notation for |F| in our notations. Here C is some number having the same order of magnitude as that of n.

TABLE II
GATE MATRIX LAYOUT WITH GM_PLAN

Name	*Ref.*	*# of Nets* (m)	*# of Gates* (n)	*Lower Bound* (β)	*Known Solution* (# tracks)	*GM_Plan Solution* (# tracks)	*CPU time Vax 750* (sec)
v4000	[25]	10	17	5		7	2.2
v4050	[25]	13	16	5		6	1.6
v4090	[25]	23	27	9		11	4.8
v4470	[25]	37	47	5		14	9.3
vc1	[2]	15	25	9	9	10	3.9
vl	[26]	6	8	3	3	3*	0.5
vw1	[9]	5	7	4	4	4*	0.5
vw2	[10]	8	8	3	5	5*	0.7
w1	[9]	18	21	4	4	4*	1.9
w2	[27]	48	33	14	14	14*	8.8
w3	[27]	84	70	11	23	21	25.8
w4	[27]	202	141	18	36	46	105.7
wan	[27]	8	7	6	6	6*	0.6
wli	[11]	11	10	4	4	6	0.6
wsn	[28]	17	25	4	8	8*	3.2
x0	[29]	40	48	6	15	15	11.3

5.2. *Benchmark Examples*

The GM_Plan algorithm has been implemented in C programming language on a VAX 11/750 computer running UNIX (version 4.3 BSD). It accepts an NG table as input and generates a character-based symbolic gate matrix layout. Sixteen benchmark circuits were tested, the results are listed in Table II. The first four examples are from Heinbuch's book [25]: "v4000" is a 4 × 1 mux, "v4050" a 2 × 4 decoder, "v4090" a 3 × 8 decoder, and "v4470" a 4-b comparator. There are another four examples from Wing and Huang [27]: "*w2*" is ITT1, "*w3*" is a 4-b ALU, "*w4*" is ITT2, and "wan" a full-adder. The remaining examples are collected from the published literatures and their sources are given in column 2. The third, fourth, and the fifth column give the sizes of the problem. The column titled "known solutions" contains the track numbers of each example reported in the corresponding references. The solutions obtained using GM_Plan are listed in the column under the title "GM_Plan Solution." Those entries in this column followed by a "*" indicate that they are the minimum track numbers (optimal solutions). The CPU time in the last column is the time GM_Plan takes to run in the VAX 750 for corresponding examples.

Note that both the optimal track numbers of "*vw2*" and "*wsn*" in Table II use more tracks than their corresponding lower bounds. This is due to the existence of direct or indirect cycles in these circuits. The optimality of the 5-track solution of "*vw2*" is verified by hand, and the 8-track solution of "*wsn*" is proved to be optimal by Yu [21, p. 57] as it is the size of the "bottleneck hypercycle" in that circuit.

Of the last 12 examples in Table II, optimal solutions are reached in seven of them. The GM_Plan solution of "*w3*" is 21-tracks, better than the known 23-track solution obtained from Wing [27]. The solutions of the other six examples ("*v1*," "*vw1*," "*vw2*," "*w1*," "*wan*," and "*wsn*") are equal to the known ones. Three examples ("*w4*," "*wli*," and "*vc1*") produce slightly inferior solutions (larger-track numbers).

5.3. *Implementation Issues*

The planning methods employed in GM_Plan is very general in that various design constraints may be added without the need to reformulate the entire algorithm. For example, in certain gate matrix layout problems, some gates must be located at fixed locations (boundary constraint). In GM_Plan, this constraint can easily be incorporated in the slot assignment procedure by fixing those gates at the beginning. As another example, if transistor sizing is considered, a large transistor may be regarded as a parallel connection of several standard transistors which have to be laid out on adjacent tracks. These implementation details can be incorporated in the GM_Plan without difficulty.

VI. CONCLUSION

In this paper, we have presented a polynomial time algorithm GM_Plan for the gate matrix layout of CMOS digital circuits. Using both hierarchical subgoal organization and two domain independent search control policies, this new method is designed to be more sensitive to the interaction among subgoals. Testing results from benchmark examples have been very encouraging. Further research is underway aiming at the following directions. 1) The implementation of an efficient backtracking facility to further reduce the chances of making premature commitment during the problem solving process. 2) The use of iterative learning techniques to further improve the quality of the output.

ACKNOWLEDGMENT

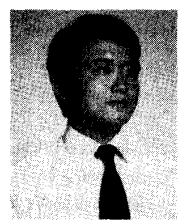
The authors wish to thank Prof. D. Y. Y. Yun of the University of Hawaii, Prof. W. Ho of the University of Southern California, and Dr. N. P. Keng of Intel Co., all formerly associated with the Southern Methodist Univer-

sity, for stimulating discussions on the applications of planning techniques to VLSI CAD. Their thanks are also extended to Dr. O. Wing and Dr. S. Huang of Columbia University for providing examples, and many helpful discussions, and to Dr. C. K. Cheng of U. C. San Diego, Dr. J. F. Beetem and Dr. B. N. Mayo of U. W. Madison, and Dr. J. T. Li of Advanced Micro Devices for many useful comments on this manuscript. The anonymous reviewers' constructive suggestions are also appreciated.

REFERENCES

[1] A. D. Lopez and H.-F. S. Law, "A dense gate matrix layout method for MOS VLSI," *IEEE Trans. Electron Devices*, vol. ED-27, pp. 1671-1675, Aug. 1980.
[2] Y. C. Chang, S. C. Chang, and L. H. Hsu, "Automated layout generation using gate matrix approach," in *Proc. 24th Design Automation Conf.*, 1987, pp. 552-558.
[3] J. S.-J. Chen and D. Y. Chen, "A design rule independent cell compiler," in *Proc. 24th Design Automation Conf.*, 1987, pp. 466-471.
[4] Y.-L. S. Lin and D. D. Gajski, "LES: A layout expert system," in *Proc. 24th Design Automation Conf.*, 1987, pp. 672-678.
[5] M. L. Yu and W. J. Kubitz, "A VLSI cell synthesizer with structural constraint considerations," in *Proc. ICCAD*, 1985, pp. 58-60.
[6] T. Kashiwabara and T. Fujisawa, "A NP-complete problem on interval graph," in *Proc. ISCAS*, 1979, pp. 82-83.
[7] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," in *Proc. 8th Design Automation Workshop*, 1971, pp. 155-169.
[8] T. Ohtsuki, H. Mori, E. S. Kuh, and T. Fujisawa, "One-dimensional logic gate assignment and interval graph," *IEEE Trans. Circuits Syst.*, vol. CAS-26, pp. 675-684, Sept. 1979.
[9] O. Wing, "Automated gate matrix layout," in *Proc. Int. Symp. Circuits and Systems*, 1982, pp. 681-685.
[10] O. Wing, S. Huang and R. Wang, "Gate matrix layout," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, pp. 220-231, July 1985.
[11] J. T. Li, "Algorithms for gate matrix layout," in *Proc. ISCAS*, 1983, pp. 1013-1016.
[12] C. K. Cheng, "Decomposition algorithm for linear placement and application to VLSI design," in *Proc. ISCAS*, 1985, pp. 1047-1050.
[13] D. K. Hwang, W. K. Fuchs, and S. M. Kang, "An efficient approach to gate matrix layout," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 802-809, Sept. 1987.
[14] T. Asano and K. Tanaka, "A gate placement algorithm for one-dimensional arrays," *J. Inform. Proc.*, vol. 1, no. 1, pp. 47-52, 1978.
[15] T. Asano, "An optimum gate placement algorithm for MOS one-dimensional arrays," *J. Digital Syst.*, vol. VI, no. 1, pp. 1-27, 1982.
[16] S. Huang and O. Wing, "Improved gate matrix layout," in *Proc. ICCAD*, 1986, pp. 320-323.
[17] T. C. Hu and E. S. Kuh, "Theory and concepts of circuit layout," in *VLSI Circuit Layout: Theory and Design*, eds. T. C. Hu and E. S. Kuh. New York: IEEE Press, 1985.
[18] E. D. Sacerdoti, "Planning in a hierarchy of abstraction spaces," *Artif. Intell.*, vol. 5, pp. 115-135, 1974.
[19] M. Stefik, "Planning with constraints, (MOLGEN: Part I)," *Artif. Intell.*, vol. 16, pp. 111-139, 1981.
[20] ——, "Planning with domain independent search control, (MOLGEN: Part II)," *Artif. Intell.*, vol. 16, pp. 141-169, 1981.
[21] M. L. Yu, "Automatic random logic layout synthesis—A module generation approach," Ph.D. dissertation, Univ. Illinois at Urbana-Champaign, 1986.
[22] W. P. C. Ho, D. Y. Y. Yun, and Y. H. Hu, "Planning strategies for switchbox routing," in *Proc. Int. Conf. on Computer Design*, 1985, pp. 463-467.
[23] D. Y. Y. Yun and N. P. Keng, "A planning/scheduling methodology for the constrained resource problem," in *Proc. Int. Joint Conf. Artificial Intelligence*, 1989, pp. 998-1003.
[24] N. Deo, M. S. Krishnamoorthy, and M. A. Langston, "Exact and approximate solutions for the gate matrix layout problem," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 79-84, Jan. 1987.
[25] D. V. Heinbuch, *CMOS3 Cell Library*. Reading, MA, Addison-Wesley, 1988.
[26] H. W. Leong, "A new algorithm for gate matrix layout," in *Proc. ICCAD*, 1986, pp. 316-319.
[27] O. Wing and S. Huang, Private correspondence, Jan.-Aug. 1988.
[28] O. Wing, "Interval-graph-based circuit layout," in *Proc. ICCAD*, 1983, pp. 84-85.
[29] K. Nakatani, T. Fujii, T. Kikuno, and N. Yoshida, "A heuristic algorithm for gate matrix layout," in *Proc. ICCAD*, 1986, pp. 324-327.
[30] R. E. Korf, "Planning as search: A quantitative approach," *Artif. Intell.*, vol. 33, pp. 65-88, 1987.

*

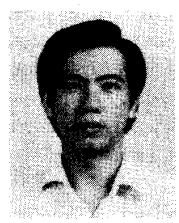


Yu Hen Hu (S'79-M'83-SM'87) received the B.S.E.E. degree from National Taiwan University, Taipei, Taiwan, in 1976, and the M.S.E.E. and Ph.D. degrees in electrical engineering from University of Southern California, Los Angeles, California in 1980 and 1982, respectively.

He is currently an assistant professor of the Department of Electrical and Computer Engineering, University of Wisconsin, Madison, Wisconsin. From 1983 to 1987, he was an assistant professor of the Electrical Engineering Department of Southern Methodist University, Dallas, Texas. His research interests include VLSI signal processing, spectrum estimation, fast algorithms and parallel computer architectures, and computer-aided design tools for VLSI using artificial intelligence. He is currently an associate editor for the IEEE TRANSACTIONS OF ACOUSTIC, SPEECH, AND SIGNAL PROCESSING in the area of system identification and fast algorithms.

Dr. Hu is a member of SIAM and Eta Kappa Nu.

*



Sao-Jie Chen (S'85-M'88) received the B.S. and M.S. degrees in electrical engineering from the National Taiwan University, Taipei, Taiwan, in 1977 and 1982, and the Ph.D. degree in electrical engineering from the Southern Methodist University, Dallas, in 1988.

Since 1982, he has been a member of the faculty in the Department of Electrical Engineering, National Taiwan University, where he is currently an Associate Professor. During the fall of 1987, he held a visiting appointment at the Department of Electrical and Computer Engineering, University of Wisconsin, Madison. His current research interests include VLSI physical design automation, fault-tolerant computing, and supercomputer architecture.

Dr. Chen is a member of the Association for Computing Machinery.