On Design Rule Correct Maze Routing

Ed. P. Huijbregts[1], Jos T.J. van Eijndhoven and Jochen A.G. Jess
Eindhoven University of Technology, Department of Electrical Engineering
Design Automation Section, The Netherlands

Abstract

This paper addresses the problem of design rule correct routing, i.e. the avoidance of illegal wiring patterns during routing. These illegal wiring patterns are due to the set of design rules accompanying each specific technology. To avoid software tuning for different technologies, the routing space is modelled as a grid graph, and all design rules are described in terms of the grid graph, including rules that describe illegal wiring patterns. The problem of finding valid, (i.e. containing no illegal wiring patterns) minimum cost connections is shown to be NP–complete, even for single nets. Although this restriction occurs in most technologies, literature does not mention any routing algorithm capable of handling these situations correctly. Two heuristics are presented to solve the routing problem, both ensuring all paths found to be valid.

1 Introduction

We are developing a generic layout system [3][4][7], in the sense that apart from the network to be laid out, also information about the particular technology/process, prefabricated geometrical structures and the chosen design strategy is considered as input. Output of the layout system is a set of geometric masks that guides the fabrication process of the circuit. Each mask describes the structural pattern of some material deposited on the silicon during the fabrication of the circuit.

To produce a technically functioning circuit, we must ensure that the images on each mask meet certain requirements, and that the different masks are consistent with respect to one another. The rules that determine what is allowed here are called *design rules* and they are particular to the given fabrication process. [6] distinguishes three types of design rules:

- **Spacing rules**: features on possibly different masks belonging to different circuit elements must have a minimum separation distance depending on the masks involved, to prevent the inadvertent fusion of regions on the silicon.
- **Minimum size rules**: features must have a minimum size, depending on the mask and the feature, ensuring that no region on the chip surface that should be connected is broken during the fabrication process.
- **Shape rules**: features belonging to the same circuit element must have a prescribed shape and a location in a prescribed position with respect to one another, to ensure that the geometry of a circuit element is transferred correctly from the masks onto the silicon.

In this paper, we distinguish two kinds of shape rules:

- **Functional (shape) rules**: features belonging to the same circuit element must always have a prescribed shape and location with respect to one another to ensure correct functional behavior.
- **Critical (shape) rules**: features may never have a prescribed shape and location with respect to one another to ensure correct electrical behavior.

Practically in all technologies, critical rules of some kind occur. For instance, the use of stacked vias is often prohibited. We will show the introduction of such rules results in NP–hard routing problems. However, algorithms presented in literature sofar, that claim to be able to handle complicated design rules. e.g. [5][8], pay no attention to the effect they have on the increasing complexity of the routing problem.

Apart from strict geometrical design rules that must be complied with, there are also some patterns that are not forbidden but the use of which should be limited. Usually the electrical behavior of the circuit is influenced negatively, and/or the reliability of the circuit is affected. For instance, the resistance of polysilicon wires is much larger than that of metal wires, thus long polysilicon wires should be avoided. Also the extensive use of vias will influence the reliability of the design. This leads to a last type of design rules:

- **Preference rules**: different features have different electrical behavior and the use of these features should therefore be limited.

The paper is organized as follows. Section 2 describes our routing space model. The resulting routing problem is proven NP–complete in section 3. Section 4 presents maze routing principles, in order to explain the heursitics described in section 5. Results and conclusions are given in section 6 and 7.

2 Layout modelling

To be able to cope with the diversity in technologies, processes and the different sets of design rules that go along with them, we introduce a general routing space model that allows us to describe all possible design rules. We assume that the routing space is defined by a rectangle in which the application specific interconnections must be designed in a prescribed number of wiring layers. The routing space can be formulated in terms of a grid graph if [4][7]:

- Wire widths and via sizes are not subject of design and are uniform for every single wiring track.
- Wiring patterns generated by the router are restricted to be aligned with the Cartesian coordinate axes.

1. This work is supported by FOM under proj. nr. EEL88.1428

The routing space is modelled as a 3–dimensional undirected *grid graph* $G(V, E)$, a 3–dimensional $X \times Y \times Z$ array of vertices. A vertex (or grid point) v may be denoted by its coordinates (x, y, z), where $0 \leq x \leq X$, $0 \leq y \leq Y$ and $0 \leq z \leq Z$. Points having the same z–coordinate constitute a plane or *layer.* Usually the number of layers Z is small compared to X and Y, giving the routing space a quasi–planar flavor. The set $L_h(a, b) = \{(x, y, z) | y = a \wedge z = b\}$ forms a horizontal grid line. In an analog way a vertical grid line is given by $L_v(a, b) = \{(x, y, z) | x = a \wedge z = b\}$. Two vertices (x_1, y_1, z_1) and (x_2, y_2, z_2) are called *neighbors* if $|x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2| = 1$. The set of neighboring vertices of a vertex v is denoted as $N(v)$. Between each pair of neighbors an edge exists. Thus each vertex may have up to six edges called *north, east, south ,west, up* and *down* as depicted in figure 1. The up and down edges represent vias, the contacts between wiring patterns on adjacent layers. An edge between neighboring vertices v and w is represented as (v, w).

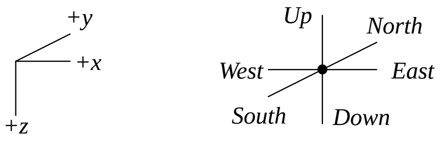

Figure 1 Orientation within grid.

Modelling spacing and size rules

A 4–tupple $D_i(a, b) = \{c, w_w, w_o, w_h\}$ is associated with every grid line $L_i(a, b)$, where i is either h or v. The absolute layout coordinate c denotes the y–coordinate for a horizontal grid line or the x–coordinate for a vertical grid line. The width of wires running along the grid line, the width of via overlaps and the width of contact holes that may be placed at grid points along the grid line are denoted by respectively w_w, w_o and w_h (see figure 2). To reduce the complexity we state that the 4–tupple does not depend on the x–coordinate of the segment of a horizontal grid line, or the y–coordinate of the segment of a vertical grid line. Furthermore we assume that the grid definition is identical for the different wiring layers, which implies that the layer with the highest grid resolution dictates the grid resolution in the other wiring layers.

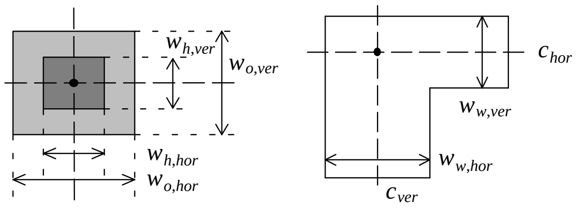

Figure 2 Via and wire segment with grid line definitions.

Note that by associating a layout coordinate c with every grid line $L_i(a, b)$, the grid lines need not be equidistant, but are determined by the condition that every possible via and terminal position must be covered by a grid point. This condition defines a lower limit on the distance between two adjacent grid lines. Higher grid resolutions are allowed at the cost of introducing critical rules between grid lines. These critical rules are necessary to avoid the situation that designed interconnections on adjacent grid lines, which are regarded as unrelated, overlap if the actual grid line distances and interconnect widths are taken into account. Higher grid resolutions also imply that not every grid line is capable of representing a possible wiring track in every layer and not every grid point may represent a possible via or terminal position.

Modelling critical rules

As explained in the previous section critical rules are used to describe illegal wiring patterns that should be avoided during routing. To be able to explain our modelling technique we need the following definitions.

An edge $e \in E$ is said to be *active* if it is part of a wiring pattern. Let $A \subseteq E$ denote the set of active edges. Edges may be *activated* during routing. On the grid graph a set of *critical rules* $CR \subseteq 2^E$ is given. A layout is *valid* if and only if

$$\forall_{r \in CR} \; |r \setminus A| > 0 \qquad (1)$$

i.e. if for all critical rules at least one of its edges is not active. Notice, a critical rules consisting of exactly one edge prohibits this edge from being used by the router. The *critical rule set* of a specific edge $e \in E$ is given by $CR(e) = \{r \in CR \mid e \in r\}$. An edge $e' \in r$, where $r \in CR(e)$ is called a *critical edge* for edge e. According to equation (1), e may only be activated during routing if

$$\forall_{r \in CR(e)} \; |r \setminus A| > 1 \qquad (2)$$

An edge that may be activated is called a *valid edge*. A path $p(V', E')$ is valid if all edges $e \in E'$ are valid.

Modelling preference rules

To limit the usage of non–preferred wiring patterns, the user may assign a cost to each edge of the grid graph using the edge labelling function $c : E \rightarrow \{1, 2, ..., c_{\max}\}$. The routing problem is thus assumed to be a minimum cost path problem. The cost $c(p)$ of a path $p(V', E')$ is defined as the sum of the costs of all edges $e \in E'$.

Implementation aspects

Edges are not explicitly described but located at the vertices. Each vertex contains its north, east and down edge. Thus if we want to inspect the north edge of vertex (x, y, z) we simply look at the north field of this vertex, whereas if we want to inspect its south edge we look at the north field of vertex $(x, y-1, z)$. Since three edges are located at a vertex, it is necessary to add three edge cost labels per vertex. In practice however, the diversity of the occurring triples of cost labels is not very large. Therefore the triples are stored in a table and the index of the triple is stored at the vertex. The gain is twofold. Firstly, only one label per vertex is needed instead of three. Secondly, triples describing the same edge cost labels for the three edges are only stored once in the table.

The set of critical rules defined for an edge is stored as a list of critical rules. For each critical rule, the critical edge set is stored as a list of edges. To store critical rules efficiently, it is important to know the same critical rule is usually defined for a large number of edges of the grid graph. Thus the number of different critical rules will not be large. Since edges are fully specified by a 4–tupple (x, y, z, d), where (x, y, z) are vertex coordinates and d denotes the direction of the edge (one of north, east or down), a critical edge set may be stored as a list of 4–tupple. This allows to store these 4–tupple as offsets between the vertices rather then using their absolute coordinates. Furthermore, by combining the sets of critical rules defined for the three edges of each vertex into a 3–tupple and storing this 3–tupple in a table, only the index of the table has to be stored at each vertex.

3 Routing problems

In this section two routing problems are defined. The first problem (VP) is concerned with finding paths that are valid, meaning that none of the critical rules is violated. The second problem (MCVP) also takes into account the preference rules and concerns itself with finding valid minimum cost paths:

Problem: Valid Path (VP)
Given a graph $G(V, E)$ and two vertices v and w. Does there exist a valid path $P(v, w)$?

Problem: Minimum Cost Valid Path (MCVP)
Given a graph $G(V, E, c)$, two vertices v and w and a $K \in \mathbb{N}$. Does there exists a valid path $p(v, w)$ with cost $c(p(v, w)) \le K$?

Theorem 1: Both VP and MCVP are NP–complete.

Proof: The theorem is proven by polynomial time transformation from SAT [1]. Proof ommited for lack of space.

As a result, finding valid minimum cost paths is NP–hard.

4 Maze routing

Our layout system uses the k–directional maze router of [2] [3]. It will simultaneously grow search waves around each terminal of a net. In this section only the principle of growing search waves is explained.

To catch the wiring actions that occur in the grid, both vertices and edges are assigned status labels. The status of an edge, denoted by S_e, may be one of *{INITIAL, INHIBIT, IMAGE, ROUTER}*. *INITIAL* indicates that the edge is not used in any wiring pattern and thus is free for routing. *INHIBIT* indicates that the edge may never be used for routing purposes. An edge has status *IMAGE* if it is part of a predefined wiring pattern, and edges belonging to wiring patterns generated during routing are assigned edge status *ROUTER*. By definition of active edges, A is thus given by

$$A = \{e \in E \mid S_e(e) = IMAGE \vee ROUTER\} \qquad (3)$$

The status of a vertex, denoted by S_v, may be one of *{FREE, BLOCK, START, N, E, S, W, U, D}*. A vertex is labelled *FREE* if all of its edges are labelled either *INITIAL* or *INHIBIT*. Otherwise it is labelled *BLOCK*. A vertex is labelled *START* if it denotes a vertex belonging to a terminal of a net routing problem and it is labelled *N* (*E, S, W, U* or *D*) if it is reached during routing from its north (east, south, west, up or down) neighbor.

A vertex $w \in N(v)$ is an *extension* of v if $S_v(w) = FREE$ and $S_e(v, w) = INITIAL$. The cost $c(w)$ of an extension is defined as $c(w) = c(v) + c(v, w)$. Extensions are stored in a priority queue keyed by their cost. A vertex is *expanded* if its extensions are queued.

Suppose that we want to determine minimum cost paths between a *root* vertex r and all other vertices. The router queues this vertex with cost $c(r) = 0$. By repeatedly extracting from the priority queue the cheapest extension and expanding it, new vertices will be reached. The set of vertices thus reached is called a *search wave*. The set of edges through which a new vertex w is reached constitutes a minimum cost path $p(r, w)$. To be able to trace such a path efficiently, w changes its status label $S_v(w)$ from *FREE* to one of *N, E, S, W, U, D*, representing the direction from which w is reached. Notice that changing the status label of a vertex prevents it from being expanded more than once and hence no selfloops are generated during search wave expansion. See figure 3.

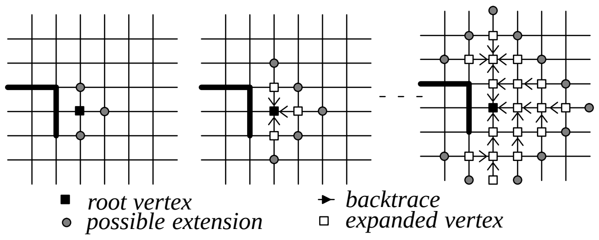

Figure 3 Search wave expansion around one root vertex.

5 An efficient heuristic

Suppose that a search wave is started around root vertex r. The search wave will grow by extracting the cheapest extension, say $e(v, w)$, from the priority queue and adding it to the search wave. Let $r \in CR(e)$ be a critical rule defined on e, then any critical edge $e_c \in r$ may be one of 4 types:

1 $e_c \notin E_{sw}$ and $S_e(e_c) = INITIAL \vee INHIBIT$.
2 $e_c \notin E_{sw}$ and $S_e(e_c) = IMAGE \vee ROUTER$.
3 $e_c \in E_{sw}$ and $e_c \in p(r, v)$.
4 $e_c \in E_{sw}$ and $e_c \notin p(r, v)$.

Here, $p(r, v)$ denotes the path by which the vertex v is reached from the root of the search wave according to the trace back information and E_{sw} denotes the set of edges that is covered by the search wave (see figure 4).

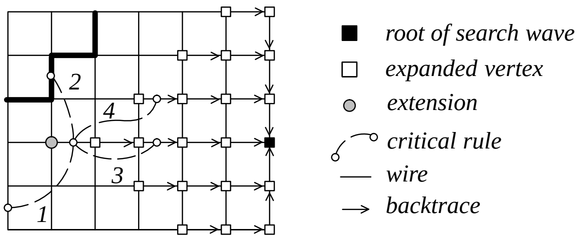

Figure 4 Edge types occuring during search wave expansion.

Edges of type 2 and 3 form the set of edges that may cause the extension to be invalid. Notice that type 2 edges are constituted by already existing wire patterns. The effect they have on the validity of an extension is called the *external–blocking effect*. Edges of type 4 are created during search wave expansion. In a way, the search wave blocks it own expansion. This effect is called the *self–blocking effect*. These edges do not affect the validity of the current extension, since by definition none of them belongs to the path containing the extension.

We will present two heuristic extensions on the basic maze routing algorithm. Both of the heuristics check the validity of an extension extracted from the priority queue, and based on the outcome of this check, the extension may or may not be added to the search wave. Thus the validity of the minimum cost paths found is ensured. The heuristics differ in run time complexities and in the fact that only one of them is able to distinguish between type 3 and 4 edges.

```
procedure heuristic1 (extension e(v,w) )
    for all r ∈ CR(e) do
        if |r \ A| ≤ 1
            return invalid fi od;
    return valid;
```

Heuristic 1 assumes that during search wave propagation each newly added valid extension changes its edge status from *INITIAL* to *ROUTER*. After a valid path is found the status of all edges of the search wave not along the path are reset to *INITIAL*. This way, all edges belonging the search wave are active and hence no distinction can be made between edges of type 3 and 4. Every time a possible extension is extracted from the priority queue, its validity is checked against all active edges in the grid graph, thus also against all edges $e \in E_{sw}$.

```
procedure heuristic2 (extension (v,w) )
    u := v;
    while S_v(u) ≠ START do
        let u′ be the vertex pointed to by S_v(u);
        S_e(e(u, u′)) := ROUTER; u := u′ od;
    heuristic1 ( extension (v,w) );
    u := v;
    while S_v(u) ≠ START do
        let u′ be the vertex pointed to by S_v(u);
        S_e e((u, u′)) := INITIAL; u := u′ od;
```

To distinguish between type 3 and type 4 edges, heuristic 2 assumes that the edge status of newly added extensions remains *INITIAL* until a valid path is found. After such a path is found only the edges along the path will change status from *INITIAL* to *ROUTER*. Whenever a possible extension (v, w) is extracted from the priority queue, the status of all edges along the path $p(r, v)$ is set to *ROUTER*. Next the validity of the extension is checked and depending on the result of the check the extension is added to the search wave. After checking the validity of the extension the status of all edges along the path is reset to *INITIAL*.

Neither of the presented heuristics guarantees to find a minimum cost path. This is easily seen by the example in figure 5. When a search wave is grown about vertex v, vertex u is reached via p_1 with cost 2, search wave expansion is stopped in this direction because paths p_1 and p_3 are mutual exclusive, and p_4 is found as a solution. If however, u was reached through p_2 with cost 3, the minimum cost path $p(v, w) = p_2 p_3$ with cost 4 was found. Furthermore neither of the heuristics guarantees to find a path. For example no path was found if p_4 did not exist.

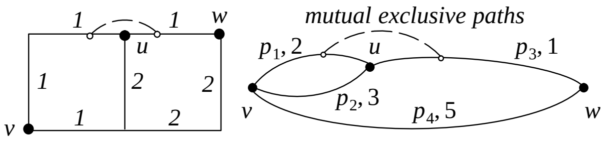

Figure 5 A valid path p(v,w) through u of cost 4 exists. However both heuristics find a valid path of cost 5.

Run time complexity

Suppose that a search wave is grown around a vertex r of a grid graph extending into infinity in all directions, and that the cost is 1 for all edges of the graph. The search wave is grown until all vertices are reached that have a distance n to r. It is easy to see that the number of extensions is then

$$\#extensions = \sum_{i=0}^{n} 6i = 2n(n+1). \tag{4}$$

Assume that the maze routing algorithm only takes constant time to handle each extension (this is a valid assumption [2]) and that the time to check the validity of an edge is $O(c)$, where c is the number of critical edges to be checked for each extension. Then the run time complexity of heuristic 1 is $O(cn^2)$. To compute the run time complexity of heuristic 2 one should realize that for each extension (v, w) at distance i from r, the path $p(r, v)$ is traced back twice, once to activate the edges and once to de–activate them, passing $2i$ extensions, leading to

$$\#extensions = \sum_{i=0}^{n} 6i.2i = 2n(n+1)(2n+1). \tag{5}$$

Since the number of edges checked remains $2n(n+1)$, heuristic 2 has run time complexity $O(n^3 + cn^2)$.

Speed up

The following observations lead to a speed up of the heuristics.The external–blocking effect is minimized by changing the edge status to *INHIBIT* for all edges $e \in E$ with which $|r \setminus A| = 1,\ r \in CR(e)$, since these edges are no longer valid. Furthermore, the number of edges traced back for each extension by heuristic 2 may be bounded. Let c denote the cost with which an edge is reached from the root of the search wave. Then backtracing may be stopped if an edge is reached for which c is smaller than the minimum c of all critical edges defined for the extension. (We did not implement the above method because an extra label for each edge is needed to store c, resulting in a large increase of memory).

6 Results

A number of circuits has been designed. All designs were made using process ECPD15/1, provided by EuroChip. This technology offers 3 layers for routing, one polysilicon layer *ps* and two metal layers, respectively *in* and *ins*. Design rules prohibit the creation of a via from *ins* to *in* above either a via from *in* to *ps* or any wiring pattern in *ps*. The prohibited wiring patterns and their resulting critical rules are shown in figure 6.

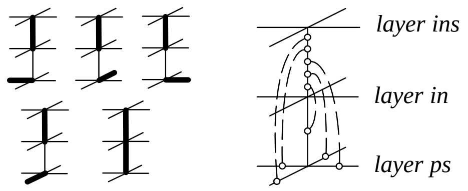

Figure 6 Illegal wire patterns for process ES2, and corresponding critical rules.

Suppose a routing problem consists of connecting two terminals, one positioned in the *ps* layer, the other in the *ins* layer as shown in figure 7. Assuming all edges to have the same cost, heuristic 1 is unable to solve this problem because of the self–blocking effect. To overcome this problem, we can tune the edge cost to minimize the self–blocking effect, thereby prohibiting the use of the edge costs to model the preference rules. Heuristic 2, however, is able to solve this problem regardless of the edge cost. Therefore, no edge cost tuning is necessary and edge costs can be used to truly reflect the preference rules.

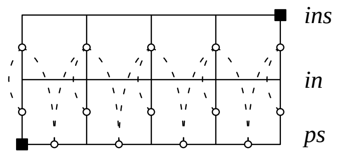

Figure 7 Heuristic 1 fails if all edge costs are equal. However, heuristic 2 will always find a connection.

All circuits were routed by both heuristics. Both heuristics used the same placement and global routing for each circuit, however, edge cost tuning is used for heuristic 1. Results are shown in table 1 and 2. Both tables list the name of the circuit in column 1, followed by the number of nets and the number of nets that failed to be routed in column 2. Furthermore the average net length is given (only nets that were completed by both heuristics were taken into account) for each circuit and the total number of extensions that were checked on validity during routing in respectively column 3 and 4. For heuristic 2, also the number of traced back edges and its ratio with the number of extensions is given in column 5 and 6. Finally the last column shows run times in seconds.

As can be seen from the results heuristic 2 is able to complete more nets as expected and the average net length is slightly smaller when compared to heuristic 1. This can be explained due to the fact that heuristic 1 is more restrictive than heuristic 2, in the sense that no distinction is made between edges of type 3 or 4. Thus, extensions seen to be valid by heuristic 2 may be seen invalid by heuristic 1. Heuristic 2 is seen to be slower than heuristic 1, due to the backtracing of the edges along the path of an extension. Experiments on larger circuits show a linear dependence between the number of extensions and the number of traced back edges.

Table 1: Results of heuristic 1.

circuit	#nets (failed)	av. length	#extensions (x10^3)	time (s)
mult4	55 (2)	47,89	391	10
rd53	71 (0)	35,73	267	8
prim8	159 (1)	104,93	2.960	87
prim9	473 (3)	148,12	8.401	241

Table 2: Results of heuristic 2.

circuit	#nets (failed)	av. length	#extensions (x$10^{3)}$	#backtraces (x10^3)	ratio	time (s)
mult4	55 (1)	46,25	305	1.644	5,38	14
rd53	71 (0)	35,59	245	281	1,15	9
prim8	159 (0)	105,2	2.740	4.231	1,54	101
prim9	473 (1)	146,54	7.931	25.808	3,25	319

7 Conclusions

We have shown that it is possible to model the routing space as a grid graph and describe design rules in terms of this grid. The notion of critical rules is introduced to describe wiring patterns that should be avoided during routing. The resulting routing problem, that of finding minimum cost connections containing no illegal wiring patterns, is shown to be NP–complete, even for single nets. Two heuristics are given to solve the routing problem. One heuristic has lower run time complexity but is more restrictive than the other, in the sense that the latter will be able to solve more of the routing problems.

References

[1] Garey, M.R. and D.S. Johnson, ”*Computers and Intractability: A Guide to the Theory of NP–Completeness*,” Freeman, San Fransisco, CA, 1979.

[2] Huijbregts, E.P. and J.A.G. Jess, ”A Multiple Terminal Net Routing Algorithm Using Failure Prediction,” *Proc. Int. Conf. on VLSI Design*, Bombay, India, pp. 84–89, 1993.

[3] Huijbregts, E.P. and J.A.G. Jess, ”General Gate Array Routing using a k–Terminal Net Routing Algorithm with Failure Prediction,” in *IEEE Trans. on VLSI System*s, vol. 1, nr. 4, december 1993.

[4] Jess, J.A.G. and A.G.J. Slenter, ”The prototype of an open design system for gate arrays”, *ESPRIT ’86, Results and Achievements*, pp. 541–550, 1986.

[5] Lau, K.M., C. Wiley and S.A. Szygenda, “M3DII: a configurable multilayer router for compact custom cell design,” in *Proc. Int. Symp. on Circuits and Systems*, vol. 4 of 5, pp. 1928–1931, Singapore, june 11–14, 1991.

[6] Lengauer, T., ”*Combinatorial Algorithms for Intergrated Circuit Layout*,” John Wiley & Sons, Inc., 1990.

[7] Slenter, A.G.J., ”A Generalised Approach to Gate Array Layout Design Automation”, *Ph.D. dissertation*, Eindhoven University of Technology, The Netherlands, 1990.

[8] Yamauchi, T., T. Nakata, N. Koike, A. Ishizuka and N. Nishiguchi, “PROTON: a parallel detailed router on an MIMD parallel machine,” in *Proc. Int. Conf. on Computer Aided Design*, pp. 340–343, november 11–14, 1991.