An Incremental Alteration Placement Algorithm for Macrocell Array Design

By

Tsz Shing Cheung

A Master Thesis

Submitted in partial fulfilment of the requirements

for the award of

Master of Philosophy

of the Chinese University of Hong Kong

ACKNOWLEDGMENT

I would like to thank my supervisor, Dr. C. S. Choy for his invaluable advice and assistance throughout the course of the research program and his patient reading and constructive comments in the preparation of this thesis. Dr. Choy is my supervisor since I was in the final year of the undergraduate course in 1987. I would like to express my gratitude for his hearty support and encouragement during this three years.

Special thanks are extended to my enlightening teacher, Dr. Y. C. Chan for his words of encouragement and helpful advice since I was a freshmen in the university.

Special thanks also to Mr. W. K. Lam, my colleagues in the VLSI/ASIC Research Laboratory of the Chinese University of Hong Kong, my girlfriend, Maggie, and Mr. C. K. Chan, my roommate in the student dormitory for their patient listening to my ideas of carrying out research and substantial suggestions on my work throughout my past two years research.

Finally, I wish to express my deepest gratitude to my parents for their moral and financial support throughout the course of my whole education.

TABLE OF CONTENTS

TABLE OF CONTENTS [continued]

TABLE OF CONTENTS [continued]

TABLE OF CONTENTS [continued]

LIST OF FIGURES

LIST OF FIGURES [continued]

LIST OF FIGURES [continued]

Title: An Incremental Alteration Placement Algorithm for Macrocell Array Design

Abstract

Most algorithms on Placement such as Min-cut algorithm and Simulated Annealing algorithm tend to concentrate on completing the placement from scratch. Little concern is put on minor placement adjustment due to small alteration made on the circuit/schematic. However, there are many cases that the schematics are subject to change throughout the design process owing to change of requirements on the performance of the circuit or system. As a result, the placement phase is completely re-done or manual adjustment is required to deal with these changes. In this thesis, a placement algorithm for incremental layout alteration is proposed for macrocell array designs algorithm. The algorithm composes of two phases: the first phase extracts information from the schematic so as to facilitate analysis of the interconnections, whereas the second phase makes the incremental alteration to the layout concerned. The program takes reasonable time to run and it saves the designer from re-executing the whole placement process or manually correcting the placement.

1. Introduction

The layout process of integrated circuits involves placing devices (cells) in a two dimensional finite space and interconnecting pins of these devices according to the schematic of the circuit to be implemented [Sangiovanni-Vincentelli87]. The goal of this process is to complete the placement and interconnection of a design in the smallest possible area satisfying a set of design constraints (such as those based on the positions and sizes of the cells to be placed and routed), a set of technological constraints (such as those based on design rules and the number of layers that can be used to route the cells), and a set of performance constraints (such as those based on the timing of the logic to be implemented). In the case of macrocell arrays which are the concern of this thesis, the major constraints are limited to design constraints (e.g. positions and interconnections) and performance constraints (e.g. critical path delay and total wire length).

Although IC layout process is a complex combinatorial optimization problem (NP-completeness) [Garey79], automation is still possible. An automation of layout of Integrated Circuit is usually divided into a series of tasks [Milford88], as follows:
1. Partitioning of a very large system into smaller sub-units which is group (or cluster) of cells and single components.
2. Placement of the sub-units into absolute or relative locations on a chip to minimize the overall area and to ensure that the final stage of finding the interconnections is possible.
3. Routing of the interconnections.

In this project, the incremental alteration placement algorithm is divided into two phases, namely
(1) Affinity Clustering (or Grouping) Phase, which is a phase similar to the first task mentioned above, and
(2) Alteration (Component Addition) Phase, which is the alteration of an existing placement according to minor change(s) to the design so that re-execution of the second task is not necessary.

1.1 The Affinity Clustering Phase

In the Clustering phase, the connections of the whole circuit are extracted into a linked list from the netlist of the circuit. Then, groups will be classified through a progressive affinity clustering approach. The feature of this approach is that all operations are performed on a dynamic data structure (i.e. a linked list). With this data structure, elements can be inserted or deleted by random access method and no sorting or re-construction of the data list is needed. Another advantage is that the alteration algorithm can be implemented more conveniently on this data structure.

Criterion of the clustering is to ensure that cells within a group are well related and cells belonging to different groups are well disconnected. Emphasis is on the density of connection/relation between elements. In addition, a simplified model of the whole schematic is built in which the primary and secondary adhesions can be easily accessed. This model is a substantial representation of the graph of the schematic.

1.2 The Alteration Phase

If a component is to be added to the circuit, it would be assigned to a group to which the component has most relation (i.e. connections to the group). Then, the component would be added through the following steps:

1. An empty space (if any) around the group to accommodate the added component should be identified for alteration. The empty space should be large enough for the insertion of the added component.
2. If only one appropriate empty space can be identified, the component would be placed to that space without further processing. If more than one empty space are identified, the one with the minimum wiring length would be chosen.
3. If an adjacent empty space cannot be allocated, the nearest empty space to the group would be identified by a method similar to the Lee's algorithm [Lee61]. Then, the path with minimum perturbation to the original placement would be determined.
4. Finally, the group concerned is expanded towards the empty space along the pre-determined path in step 3. Each expansion on the path would be based on a minimum perturbation criteria.

1.3 Floorplan of Macrocell Array

In figure 1-1 (a), a typical floorplan of a macrocell array is shown. The macrocells are concentrated around the centre on the chip, while the I/O cells are around the boundary. Between every two rows of macrocells are routing channels. Spaces between the I/O cells and macrocells are also routing spaces for interconnections.

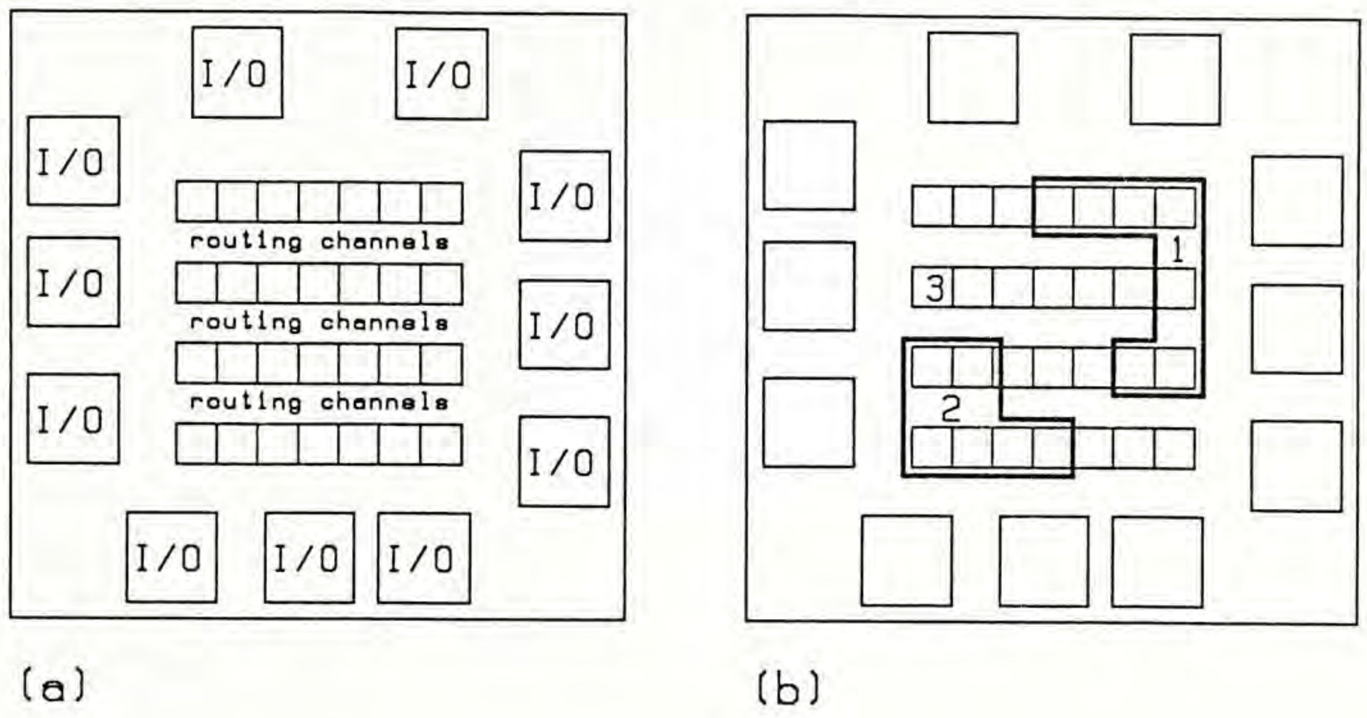

Figure 1-1 A Typical Layout of Macrocell Array and An Example of Clustering on it

Usually, cells belong to the same functional block (group) will be placed close together on the chip. An example of clustering of cells are shown on figure 1-1 (b). Group 1 consists of 7 elements and group 2 consists of 6. Group 3 consists of one element only and is named a "single element group". These elements are usually random logic components with few connections to other cells.

1.4 Chip Model

The chip model of this thesis is a nXn array of integers. There are two areas of concern in the choice of the model: (1) location representation and (2) interconnection length estimation.

1.4.1 Location Representation

Since the placement of cell is on a 2-dimensional plane, an appropriate representation of the locations can facilitate the computation of cost functions (e.g. interconnection length). The floorplan of a macrocell array is as follows:

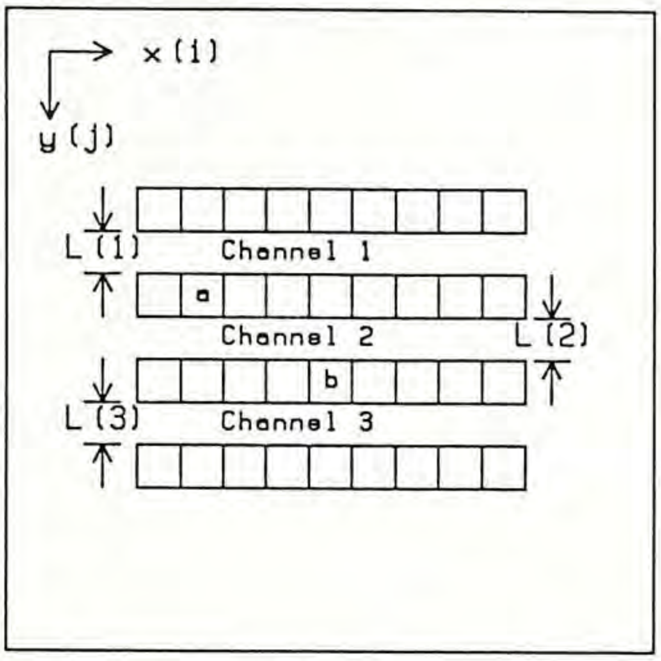

Figure 1-2 Floorplan of a Macrocell Array

The location of cells on the macrocell array is represented by the x-y coordinates. Channel widths are different in values and are denoted by L(i), where i is the number of channel. A matrix of real number is necessary to represent the location of cells and an 1-D array of real number is necessary to represent the channel width.

For a simplified model of the array, we use the nXn plane:

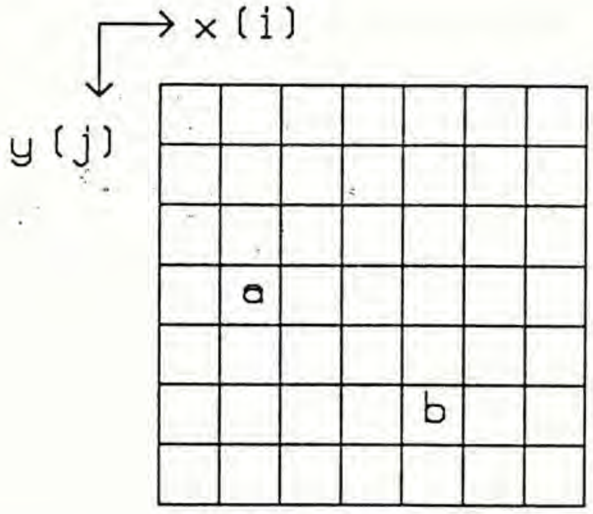

Figure 1-3 The Simplified Model of the Macrocell Array

In the simplified model, the location can be represented by a 2-D array of integers. It saves both the computation time and memory.

1.4.2 Interconnection Length Estimation

Since the actual wiring path is not known until routing, the interconnection length can only be approximated. Two assumptions are made on the programs:
1. interconnection from a module is assumed to start from the middle of the cell.
2. interconnection length between two cells measured in Manhattan Distance.

For a macrocell array, the length between two cells (x_a,y_a) and (x_b,y_b) is

$L_{MacAry} = |x_a-x_b| + |y_a-y_b|$

$= | i_a-i_b|*w + |j_a-j_b|*h$ + sum of channel widths between the two cells

where x and y are real numbers,

w is the width of macrocell, h is the height.

To calculate the length between two cells in this array, we have to take into account the channel widths, and the width and height of macrocell.

For the simplified array, the length between a and b is

$L_{Smp} = | i_a-i_b| + |j_a-j_b|$

where i and j are integers.

Thus, the computation time on interconnection length is dramatically reduced by using the simplified model.

1.5 Cost Function Evaluation

There are many methods to evaluate the quality of a layout. The most common one is the measure of net length. Some of the techniques of net-length calculation are described as follows:

1.5.1 Net-length Calculation

Net length estimation is difficult especially for multi-terminal nets, i.e. for nets that have more than two pins. [Sangiovanni-Vincentelli87] Many different, but electrically equivalent, ways of interconnecting a set of pins are possible. In Figure 1-2, three equivalent implementations for a four-pin net are shown. In most VLSI chips, interconnections are implemented exclusively with Manhattan geometries.

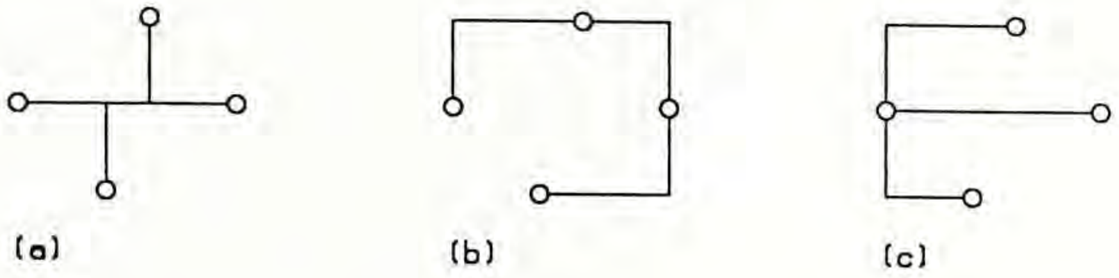

Figure 1-4 Equivalent implementations for a four pin net

Normally, the shortest wiring length is preferred if other considerations are ignored. In fact, in general, shorter interconnections imply better electrical performance. The length of an interconnection is measured according to the Manhattan distance (L_{MHT}): with this metric, two points a and b with coordinates (x_a,y_a) and (x_b,y_b) have distance

$$d(a,b) = |x_a-x_b| + |y_a-y_b|. \qquad \text{........................ Eq. 1-1.}$$

Finding a rectilinear interconnection path of minimum length for a single n-pin net at a first glance seems quite simple, but it is in fact NP-hard [Garey79, Goto86]. It is referred to as the rectilinear Steiner tree problem.

1.5.2 Net-length Estimated by Half of the Perimeter of Bounding Box

The bounding box is defined as the smallest rectangle which encloses the signal set between modules [Schwei76, Goto78]. In the works of Sechen [Sechen87], Elder, Zenewicz and Alvarodiaz [Elder84], and Goto [Goto81], the half perimeter length of the bounding box is used as the estimation of the connection length. An example of the half perimeters of two nets is shown in the Figure 1-5.

According to the simulation result on random and optimized placements by Sechen [Sechen87], the estimation is within 5% of the actual value in the final layout.

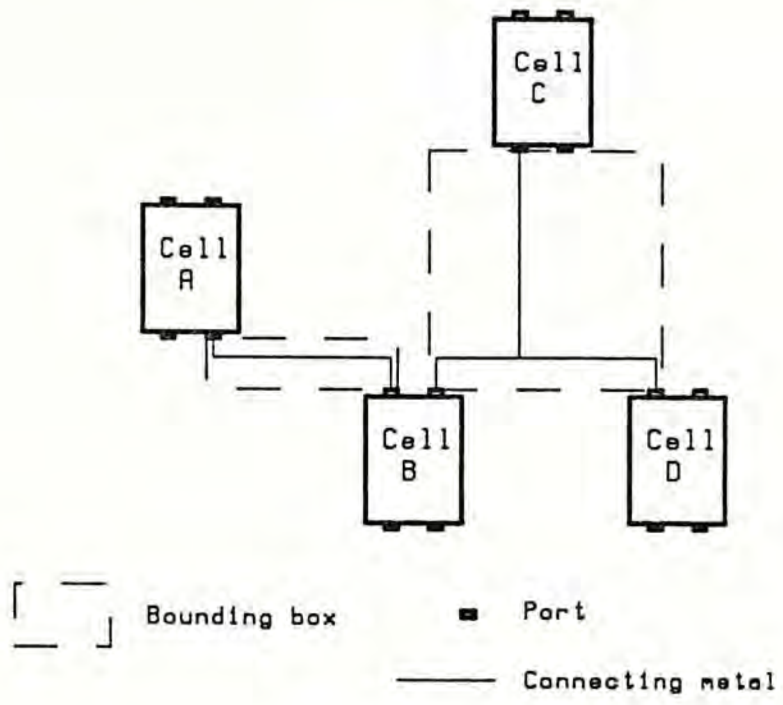

Figure 1-5 Half Perimeter of Bounding Box is the Minimum Length to a Net

Since the connecting wires in a Macrocell Array are in a rectilinear grid system, Manhattan distance is the most appropriate measure of net length. Hence, the Manhattan geometries approach is used in the estimation of interconnection of circuits in this project.

1.6 Thesis Layout

This thesis is organized as follows: in Section 2, the most common methodologies of partitioning and placement will be summarized. In Section 3, the Incremental Alteration Placement Algorithm will be described while in Section 4, its implementation. In Section 5, results of the algorithm will be examined, while in Section 6, the discussion. Then follows the conclusion in Section 7, list of references in Section 8, and appendices in Section 9.

2. *Reviews of Partitioning and Placement Methods*

There are many developed methods on partitioning and placement of VLSI. Some of their principles and idea are summarized as follows:

2.1 Partitioning Methods

The problem of partitioning is to divide a graph with n nodes so that each partition cut the least number of edges which joins the nodes [Barnes85].

Let G be an undirected graph having nodes N= {1, ... , n} and edges set E. It is often of interest to partition the nodes of G into a given number, say k, of disjoint subsets $S_1, \ldots, S_k$, of specified size $|S_1| = m_1 \geq \ldots \geq |S_k| = m_k$, in such a way to minimize the number of edges joining nodes in distinct subsets of the partition.

For example, consider the problem of partitioning the nodes of the following graph into two sets containing 5 nodes each. The partition N= {1,2,3,4,5} U {6,7,8,9,10} cuts 3 edges and appears to be optimal.

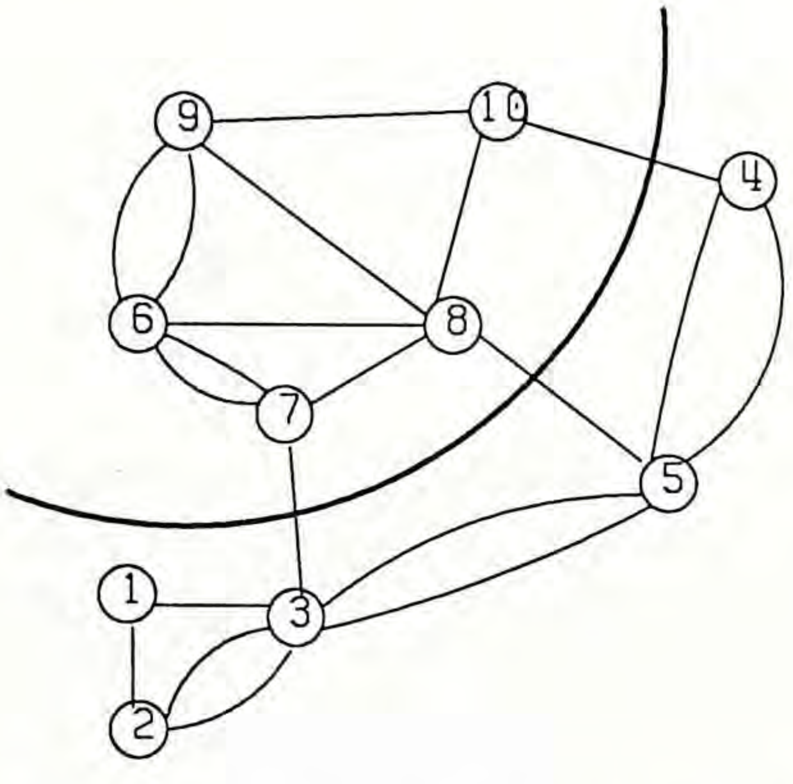

Figure 2-1 Partitioning of a Graph with 10 Nodes

Several heuristic algorithms have been proposed for solving graph partitioning problems. We would like to mention here the works of Barnes [Barnes82, Barnes84], Kernighan and Lin [Kernighan-Lin70].

There are four major partitioning algorithms [Preas88]:

2.1.1 Direct Method

The direct method starts with a seed node of (or set of seed nodes for) each cluster and assigns a node at a time to one partition, using preferences to satisfy the constraints. Various embellishments have been made to this method. For example, after the initial partition is complete, the algorithm can be restarted by selecting new seeds for the clusters; this may produce a better partition [Kodres72, Mennone71].

In this project, the partitioning method used belongs to the direct method.

2.1.2 Group Migration Method

The group migration method, also known as the Kernighan-Lin algorithm, starts with some partition, usually generated randomly, and then moves components between partitions to improve the partitioning.

This algorithm not only is rather fast but also it often produces very good results. [Sangiovanni-Vincentelli87] The basic idea of the algorithm is again to interchange module among the two elements of the partition to obtain a better solution. A scoring function is used to evaluate the interchanges. This scoring function measures the difference in cost between the solution before the interchange and the one after the interchange.

2.1.3 Metric Allocation Methods

A family of metric allocation methods attempts to find a metric other than the structure of the interconnection graph which in some way reflects the direct and indirect connectedness of the nets. In these approaches nodes are put together on the basis of the metric, not on the basis of their connectedness. Therefore, the number of connections is only indirectly minimized. Several approaches for doing the actual partitioning exist [Charney68, Donath72].

1. Horizontal Peak Congestion:

Horizontal peak congestion is defined as the maximum number of nets crossing a single one of the equidistant vertical cutlines across the chip. This is a lower bound on the number of horizontal routing tracks needed.

2. Vertical Peak Congestion:

Vertical peak congestion is defined as the maximum number of nets crossing a single one of the equidistant horizontal cutlines across the chip. This is a lower bound on the number of vertical routing tracks needed.

3. Two-dimensional Peak Congestion:

Two-dimensional peak congestion is defined as the maximum number of net bounding boxes overlapping a single one of the rectangles defined by equidistant vertical and horizontal cutlines across the chip. This metric provides a measure of local congestion.

4. Estimated Wire Length:

Estimated wire length is defined as the sum of all net bounding box half perimeters.

5. Actual Routing Completion and Wire Length:

The design is first routed with a router that uses a modified line search technique and has performance that is linear in the number of connections. Any disconnects were then attempted with a maze router.

2.1.4 Simulated Annealing

Simulated annealing is a process analogous the heat annealing of doped semiconductor wafer or crystals. Since the natural formation of bondings among molecules in these substances is always looking for a minimum potential energy, the process can be imitated to find the global minimum in a multi-objective problem. The algorithm of simulated annealing is as follows:

```
Start with some state, So;
T = To;
repeat
  while (not at equilibrium) do
  begin
    Perturb S to get a new state Sn;
    E := E(Sn) - E(S);
```

```
      if E<0 then
        replace S with Sn
      else
        with probability exp(-E/kT) replace S with Sn
    end;
    T := c*T;   { 0<c<1 }
  until (frozen);
```

k : kelvin constant,
T : temperature,
E(S) : Energy of state S,
c : proportionality constant (decrease rate of T).

Table 2-1. Algorithm of Simulated Annealing

The simulated annealing method is a non-convex optimization algorithm. The partitioning problem is cast in two parts: a cost function, i.e. E(S), which classifies any feasible solution, and a set of moves (i.e. Perturbation), which allow movement from solution (S) to solution (Sn). The algorithm starts at a random solution (i.e. So) and makes stochastically chosen moves to modify that solution. Initially the moves which are accepted include a high proportion of moves which increase the solution's cost. As the algorithm progresses, the proportion of such moves is decreased until finally almost no moves that increase the cost are accepted.

2.2 Placement Methods

The problem of placement of Integrated Circuit is to place components with certain shape and size on a plane such that the following purposes can be achieved:
(1) Total interconnection length be optimized.
(2) Delay on critical path should be small for proper performance of the circuit.
(3) The skews in propagation delays in the input-to-output paths should be as small as possible so that problems like race and harzard would not occur.
(4) Power consumption be optimized and power distribution be even.

Constrained by these factors, placement becomes a very difficult problem (NP-hard). In general, no algorithm is able to cope with the full complexity of placement [Sangiovanni-Vincentelli87]. Approximations are used to reduce the computation of the cost functions (e.g. total interconnection length and delay on critical path) and of the constraints (e.g. area and timing constraints) so that the problem can be solved in reasonable computer time.

There are many placement methodologies [Preas88]. Two of the most common ones are introduced here:

2.2.1 Min-cut Methods [Russell85]

Rather than simply placing a number of modules, some methods attempt to partition the network in a rational way, and the Min-cut algorithm [Lauther79] is one. Modules are placed to the left or right of a cut line parallel to the y-axis in such a way as to reduce the number of connections crossing the line to a minimum, with the difference in area between the two halves not exceeding a certain threshold. The partitions are themselves cut parallel to the x-axis and the process repeated recursively. The development is illustrated in Figure 2-2, in which a graphical representation of the design shows the modules as arcs in the graph and the channel between as nodes. Since the partitioning reduces the number of connections between modules, routing problems should be eased.

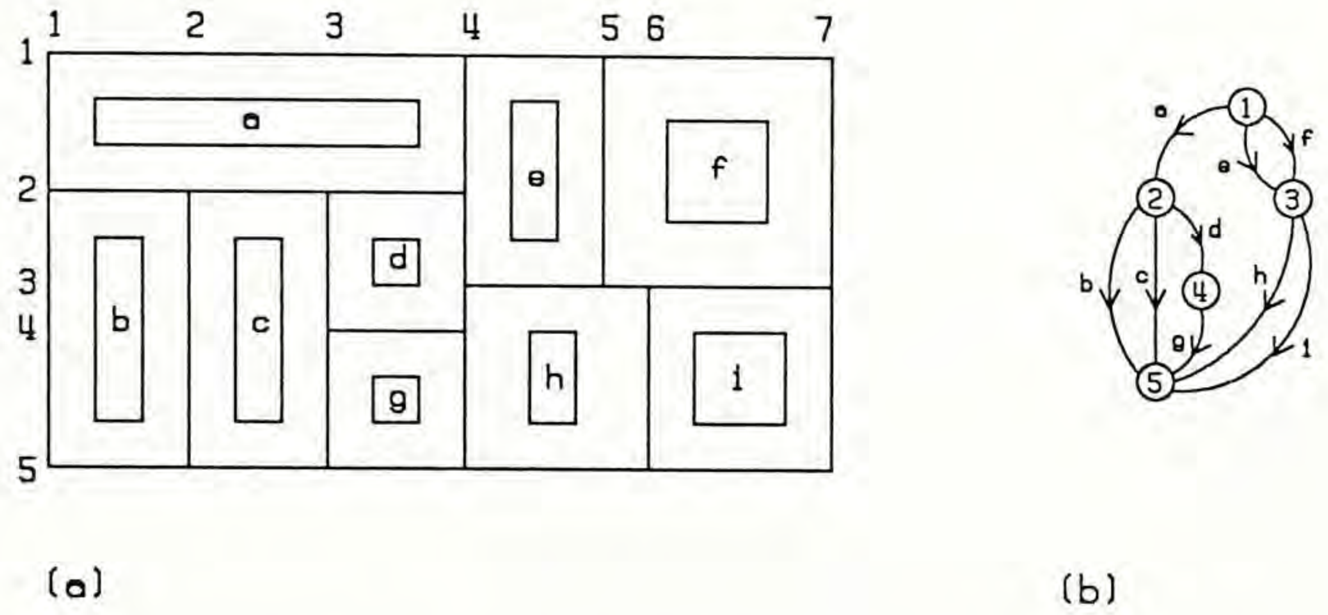

Figure 2-2 A Layout and its Graphical Representation

2.2.2 Affinity Clustering Methods [Elder84]

In the affinity clustering methods, circuits are clustered by the affinity (i.e. the interconnection among each element of the circuits).

During clustering, individual circuits are grouped to form new placement entities. In the first clustering pass, the clusters are individual circuits. As repetitive passes are performed, the clusters may become groups of circuits. The grouping algorithm uses the pairwise Attractive Forces between current clusters, the external pulls on the clusters, and their relative sizes. The strength of the attractive forces is a function of the number of nets connecting the clusters being considered. Multiple passes are performed, possibly producing new clusters and new strengths. The number of passes is a user-controlled parameter. Limits are placed on the resulting cluster size. The clustering procedure ends when no more clusters are formed or the preset number of passes is reached. This process is similar to that of Feuer et al. [Feuer77] and Lallier and Jackson [Lallier79].

The clusters formed are used as objects for the zoning of placement. Clustering tends to alleviate the local optimum problem, since closely coupled circuits are moved together. This approach uses much less computer time because the number of objects is drastically reduced.

Since the methodology used in the project is similar to the affinity clustering methods, more procedure related to this method is described. One of the zoning procedure, which is the next step to the affinity clustering procedure, is summarized as follows:

The zoning step establishes cut lines, imaginary lines dividing the chip into sections called zones. The first cut line is vertical and divides the chip into two equal zones. The second is horizontal and divides these two zones into four, and so on. As each cut line is introduced, zones become smaller. The first two cut lines are illustrated in Figure 2-3 and 2-4. V1 is the first vertical cut line, a chip bisector; H1 is the first horizontal cut line, a chip quadrasector.

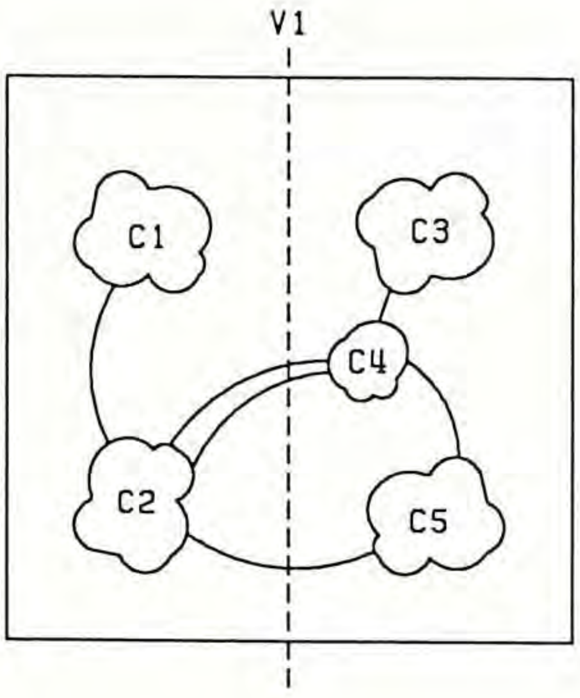

Figure 2-3 First Vertical Cut Line on the Chip

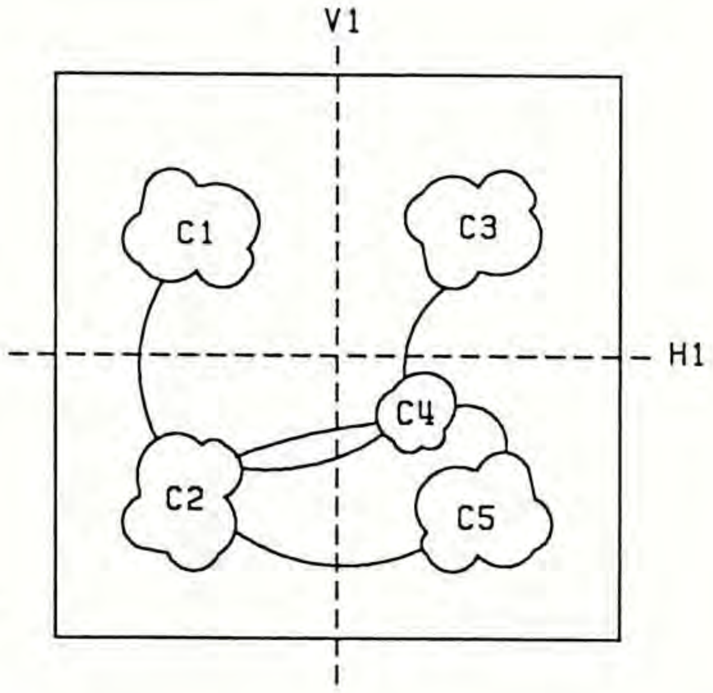

Figure 2-4 First Horizontal Cut Line on the Chip

Clusters move across cut lines attempting to minimize the number of nets crossing the line. Minimizing wiring congestion across a cut line also minimizes total wire length for the design. A cluster assigned to a particular zone must remain in that zone or a zone derived from its original zone.

The zoning process repeats until the designer-specified number of cut lines is made. Each cluster is assigned to a chip area. The size of this area varies according to the cluster size and number of cut lines. For example, when two cut lines are used (quadrasection), the area of each of the four zones would equal one-quarter of the total chip area. The zoning process is similar to that of Corrigan [Corrigan79] and Breuer [Breuer77]; the major difference is that clusters are used, rather that individual circuits. Lallier and Jackson [Lallier79] used clusters, but with a different interchange technique.

2.2.3 Other Placement Methods

Other placement methods like simulated annealing [Appendix I, Durand89], greedy clustering approach [Sudo83], force-directed methods [Goto81, Sudo83, Goto86], eigenvalue approach [Sangiovanni-Vincentelli87], have been hot topics in the current research and development of placement algorithms and softwares. However, due to the limitation of space and time, these methods would not be discussed here.

3. Algorithm

The idea of the Incremental Alteration Placement Algorithm (IAPA) can be illustrated by the example in Figure 3-1. In (a), there is an original placement on the 5x5 macrocell array. The design consists of five groups: A, B, C, D, and E. A new element, I, is to be added to group B. The nearest empty space, X, to group B will be found and element I will be added at an appropriate position along the direction from group B to the empty space X. The final layout is shown in (b). It is noted that modification in layout is limited to group B and group E. In other words, it will cause the least degree of perturbation to the original layout.

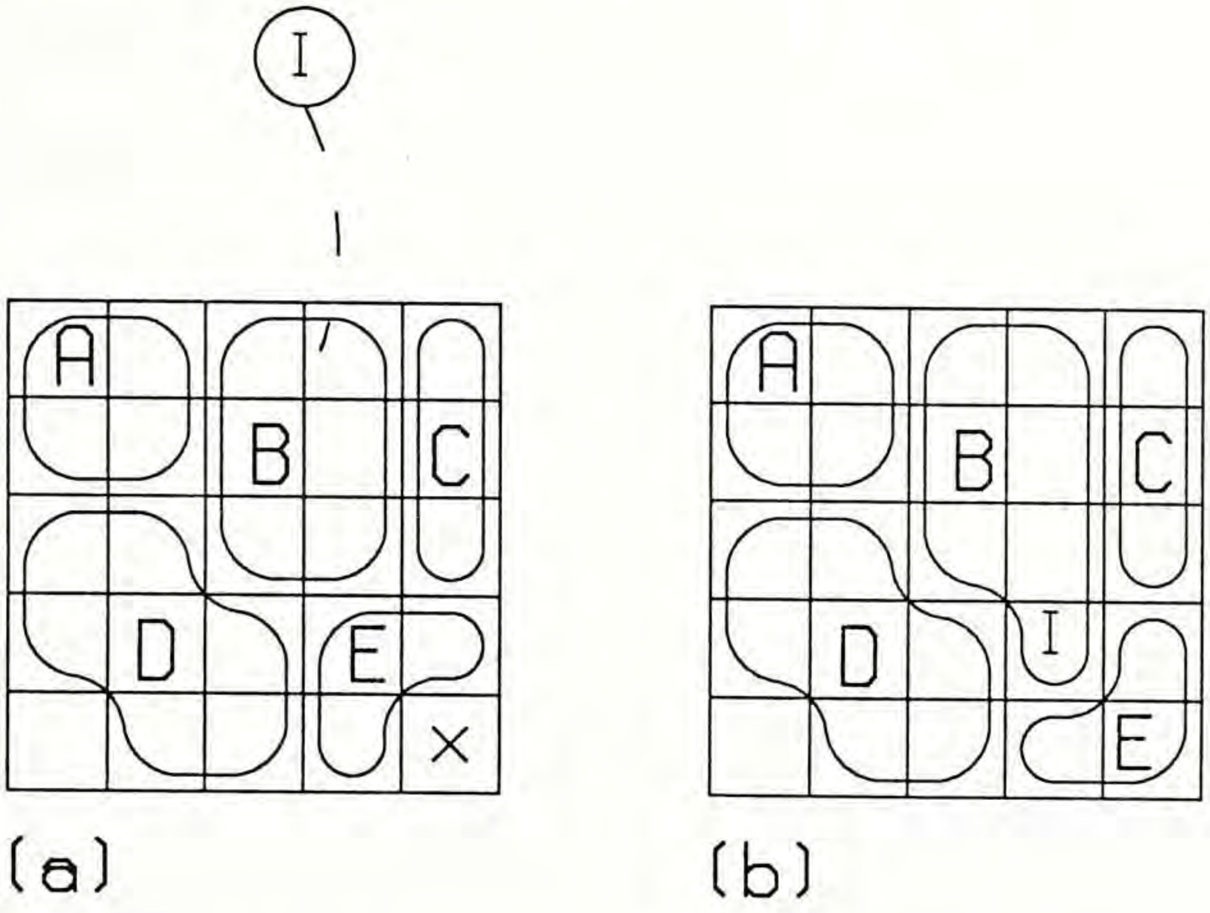

Figure 3-1 Example of Incremental Layout Modification

The Incremental Alteration Placement Algorithm is mainly divided into two phases, namely, the affinity clustering phase, and the alteration phase. The affinity clustering phase carries out a simple partitioning task, while the alteration phase works out the placement when there is minor change to the design. Details are described in the following sections.

3.1 The Affinity Clustering Phase

The affinity clustering phase in this project is similar to a partitioning phase in the automatic layout system of IC design. However, the main concern is to form clusters by interconnections, while a general partitioning phase may also consider critical path delays, fan-out distribution, and power distribution.

Starting from the netlist of a schematic, the interconnections of the whole circuit will be found and stored in a linked list format. Clusters will be formed by counting the affinity between each element. Element pairs with higher affinity will be clustered in the first clustering pass while those with lower affinity in later passes. A minimum number of connections should be satisfied for clustering. The clustering criteria in this phase belong to the direct method discussed in Section 2. Rules of selecting seed nodes and assignment of nodes are mainly based on the affinity (connectivity) among nodes and/or groups.

The affinity clustering phase consists of five parts: (1) Construction of connection list, (2) Primary grouping, (3) Element appendage to existing groups, (4) Loose appendage of ungrouped elements, and (5) Single element groups formation.

3.1.1 Construction of Connection Lists

Construction of connection lists is for the analysis of the interconnections among cells in the circuit. Primarily, four connection lists are obtained from the original connection of the circuit. The original connection should be "directed connection" (with direction from output port to input port), since the netlist of a circuit should usually contain the information on the direction of each net.

The procedure to obtain the connection lists is as follows:

(1) store the direction connections in an array (directed connection array).

(2) convert the directed connections into undirected connections and store the latter ones in another array (undirected connection array).

(3) obtain the linked list (directed connection list) from the direction connection array.

(4) obtain another linked list (undirected connection list) from the undirected connection array.

(5) extract the fan-out connection list from the directed connection list.

(6) merge the undirected connection list and fan-out connection list into the link-related connection list.

Considering the following circuit:

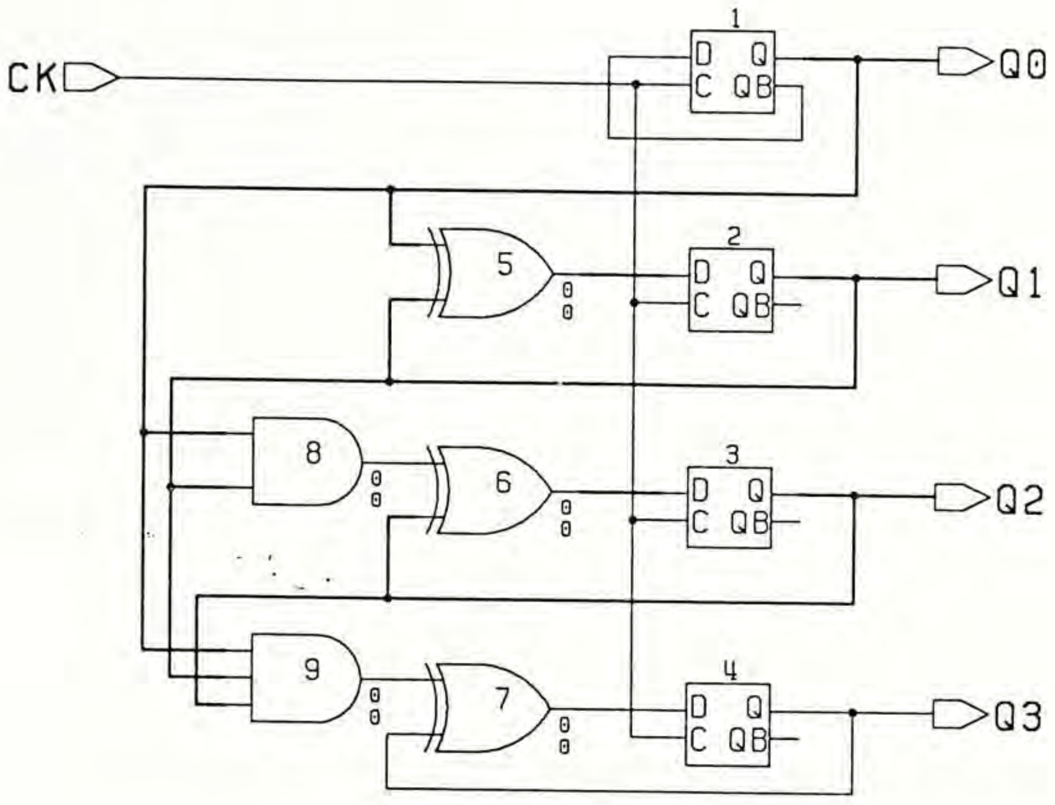

Figure 3-2 A 4-bit synchronous counter

The counter has the following 'directed' connections:

c[1,5]=1, c[1,8]=1, c[1,9]=1, c[2,5]=1, c[2,8]=1,
c[2,9]=1, c[3,6]=1, c[3,9]=1, c[4,7]=1, c[5,2]=1,
c[6,3]=1, c[7,4]=1, c[8,6]=1, c[9,7]=1.

Table 3-1. Connections of the 4-bit synchronous counter

One of the cell 1 connection which is fed back to itself is not counted because it can be viewed as an internal connection, that is, the connection is inside the layout boundary of the cell.

The connections in Table 3-1 are stored as an array in the computer. However, for the convenience of calculation, the connection values are translated to data in a linked list.

Directed connection list	Undirected connection list	Fan-out connection list	Link-related connection list
1,5: 1	1,5: 1	5,8: 2	1,5: 1
1,8: 1	1,8: 1	5,9: 2	1,8: 1
1,9: 1	1,9: 1	6,9: 1	1,9: 1
2,5: 1	2,5: 2	8,9: 2	2,5: 2
2,8: 1	2,8: 1		2,8: 1
2,9: 1	2,9: 1		2,9: 1
3,6: 1	3,6: 2		3,6: 2
3,9: 1	3,9: 1		3,9: 1
4,7: 1	4,7: 2		4,7: 2
5,2: 1	6,8: 1		5,8: 2
6,3: 1	7,9: 1		5,9: 2
7,4: 1			6,8: 1
8,6: 1			6,9: 1
9,7: 1			7,9: 1
			8,9: 2

Table 3-2. Connection Lists on the 4-bit synchronous counter

The undirected connection list is obtained from the directed connection list by sorting the data in ascending numerical order. Hence, 'directed' connections will become 'undirected' connections. Duplicated values on a paired connection would contribute to the scalar sum of the two values. For example, the c[2,5]=1 and c[5,2]=1 connections in the directed connection list becomes the c[2,5]=2 in the undirected connection list.

The fan-out connection list is also obtained from the directed connection list. It is done by the assumption that cell with connection from the same source should be of some affinity. The case is illustrated in the following circuit segment:

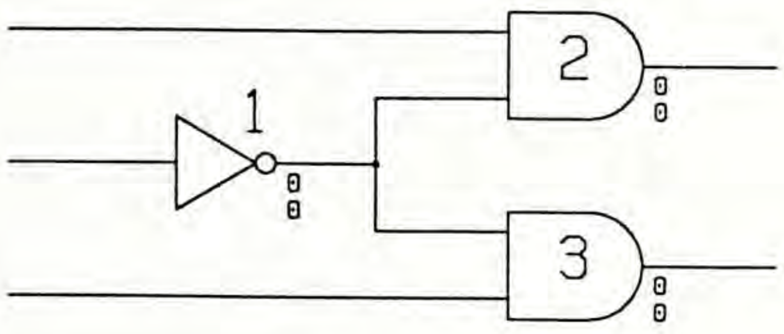

Figure 3-3 A circuit Segment Showing the Affinity Caused by Fan-out of a Logic Gate

The cell 2 and 3, in some sense, should belong to the same group. That is, cell 2 and 3 have a virtual connection.

The link-related connection list is obtained by merging the undirected connection list and the fan-out connection list. On this point, merging of two identical pairs from the two lists may be weighted. The formula would have the form:

```
merged value = w1*(undirected connection)+w2*(fan-out connection)
                                          .............   Eq. 3-1.
```

where w1 and w2 are the weights of the undirected connection and the fan-out connection respectively. They are constants to be determined by the designer.

The weights are mentioned because it is meaningful to distinguish whether the undirected connection or the fan-out connection is of more important.

3.1.2 Primary Grouping

In this phase, the groups are formed by progressively scanning through the link-related connection list.

From the link-related connection list, there are several steps to form groups on the circuit. The steps are as follows:

1. scanning the connection list, determine the maximum connectivity among cells.
2. starting from the maximum connectivity, search connection pairs with the largest connectivity.
3. check if either of the two elements in the connection exists in the group list.
4. if exist then append the other element to the group and update the connectivity attributes* of the two elements; if not exist then shift to the next connection pair.
5. if neither of the elements in the connection exist in the group list, create a new group with these two elements; update the connectivity attributes.
6. if both elements in the connection exist in the group list, update only the connectivity attributes.
7. if not end of connection list, goto step 2.
8. stop.

* Connectivity attributes include the two mostly connected elements and their number of connections to the element.

In more programmable form, the algorithm is as follows:

```
determine the maximum_connectivity on the merged connection list;
for i:= maximum_connectivity downto lower_bound_for_clustering do
while not(end of merged connection list) do
begin
  scan the merged connection list, finding connection with
  connectivity i;
  if the connection, lp, found then
  begin
    if (both of the two elements on the connection exists in
        a group G) then
      update connectivity attributes of the two elements
    else if (only one of the two elements on the connection
                    exists in a group G)
          and (size of G <= upper_limit_of_group_size) then
      begin
        assign another element to G;
        increment group size of G;
        update connectivity attributes of the two elements
      end
    else
      begin
        create a new group, G', on the group list;
        set group size of G' to 2;
        create connectivity attributes of the two elements
      end;
    dispose(lp)
  end;  {if}
end;  {while}
```

Table 3-3. Algorithm of the Progressive Clustering Process from the Connection List.

The algorithm is called progressive clustering process because the groups are formed by appending elements on existing groups. And, those groups are formed by obtaining information from progressively scanning along the connection linked list.

Through the above steps, the 4-bit synchronous counter will be grouped as follows:
The maximum connectivity is 2.
Starting from 2, the elements are grouped as N= {2,5,8,9}U{3,6}U{4,7} in the first clustering pass.
The lower_bound_for_clustering is usually at least 2. Hence, after the first pass, the procedure will stop. Hence, the element '1' is left as ungrouped element.

3.1.3 Element Appendage to Existing Groups

Before introducing the procedure of element appendage to existing groups, we would like to bring out the term "Belong Tendency". Belong Tendency (BT in short) is the number of connection(s) of an element with respect to a group. For example, element 3 has the following connections in the circuit with 7 elements:

c[1,3]= 1, c[3,4]= 1, c[3,5]=1, c[3,6]= 1, c[3,7]= 1.

And, the partition of the circuit is N= {1,2,7} U {4,5,6}.

Then, the BT of element 3 to the group {1,2,7} is 2, and that to {4,5,6} is 3.

Let's denote the Belong Tendency of an element, e, to a group, G, by BT(e,G),

Hence, BT(3,{1,2,7}) = 2, and

BT(3,{4,5,6}) = 3.

Taking the 4-bit synchronous counter as example, the partition after the primary clustering pass is N = {2,5,8,9}U{3,6}U{4,7}. The Belong Tendencies of the ungrouped element '1' are:

BT(1,{2,5,8,9}) = 3,

BT(1,{3,6}) = 0, and

BT(1,{4,7}) = 0.

The steps on the affinity appendage of elements are as follows:

1. initialize the values of the belong tendencies.
2. starting from the first group,
3. count the belong tendency(-ies) of ungrouped element(s) which is connected to the group.
4. determine the maximum value of belong tendencies to the group.
5. starting from the maximum belong tendency.
6. assign the element with the largest belong tendency to the group.
7. increment group size.
8. update connectivity attributes of the element.
9. if there is still related element(s), goto step 6.
10. if not end of group list, goto step 3.
11. stop.

In more programmable form, the algorithm is as follows:

```
start from the head of the group list;
while not(end of the group list) do
begin
  G:= the current group;
```

```
    if (size of G < upper_limit_of_group_size) then
    begin
      initialize values of belong_tendency;
      start from the head of the connection list;
      while not(end of the connection list) do
      begin
        lp:= the current connection;
        if (one of the two elements of lp belongs to G) then
        begin
          increment belong_tendency[another element];
          store connectivity attributes;
          dispose(lp)
        end;
      end;
      determine maximum value of belong_tendency;
      for bound:= max_value_of_belong_tendency downto
                  limit_of_belong do
        for i:= 1 to max_number_of_element do
        begin
          if (belong_tendency[i]>bound)
               and (i not in group list) then
          begin
            assign i to G;
            increment size of G;
            update connectivity attributes of the two elements
          end;  {if}
        end;  {for}
    end;  {if}
  end;  {while}
```

Table 3-4. Algorithm of Affinity Appendage by Measuring Belong Tendencies of Ungrouped Elements.

The appendage of ungrouped elements to the existing group is through the measure of Belong Tendency. These elements, with its loose connection with other elements (connectivity<2), would not be grouped in the first pass of grouping (i.e. the primary grouping pass). However, they may have more connections to a group instead of a single element. Hence, this pass is necessary to cluster these elements.

Since the belong tendency of element 1 to the group {2, 5, 8, 9} is the largest, element 1 belongs to this group and the partition after this pass is N = {1, 2, 5, 8, 9}U{3, 6}U{4, 7}.

Grouping Mathematics:

To limit the size of each group, the appendage of the element is constrained by the following criterion:

size1 + size2 - connectivity $\leq$ threshold ------------ Eq. 3-2(a)

OR

size1 + size2 $\leq$ threshold + connectivity ------------ Eq. 3-2(b)

where size1 and size2 are sizes of the two groups,

connectivity is the number of interconnections between the two groups, and

threshold is a constant (by experience, 3n/4 is a suitable value for a expected group size of n for large groups).

This criterion is only a preliminary one. It means that two groups with sizes "size1" and "size2" will not be grouped unless their sum of sizes minus their number of interconnections is less than a threshold. The formula will prevent the formation of any loosely connected group with extra large size. For example, if the sizes of G1 and G2 are 6 and 7 respectively. Then, the value on the left hand side of Eq. 3-2(b) is 13. If we expect the size of a large group should be 8, we have to choose a "threshold" value for the grouping of G1 and G2 because their sum sizes is much larger than what we expected. By experience*, the three-fourth of this expected value is suitable for the threshold (i.e. 6). Then, the number of interconnections of the two groups should be at least 7 for their grouping to become a group of size 13. However, if G1 is with size 2, a "connectivity" of 3 is enough for the formation of a group of size 9. In other words, the strictness of the constraint is lowered if the sum of sizes of G1 and G2 is not too larger than the expected group size for a large group.

3.1.4 Loose Appendage of Ungrouped Elements

The loose appendage phase is to group elements which have only loose connectivity to other elements (e.g. no. of connection = 1). The steps on this part is:

1. starting from the first connection pair in the connection list,
2. if only one element of the connection pair is in the group list,
   (i) search the group, G, to which the element belongs,
   (ii) check the size of G,
   (iii) append the other element of the pair to G,
   (iv) update the connectivity attributes.

* Author's experience in digital circuit design, referring to circuits in [Cheng89, Cheung88].

3. if neither of the elements exists in the group list,
   (i) create a new group with these two elements,
   (ii) set the connectivity attributes.
4. if not end of connection list, goto step 2.
5. stop.

The algorithm in programmable form is as follows:

```
start from the head of the connection list;
while not(end of the connection list) do
begin
  lp:= the current connection;
  if (only one element of lp is in the group list) then
    begin
      start from the head of the group list;
      while not(end of the group list) do
      begin
        G:= the current group;
        if (size of G < upper_limit_of_group_size) then
        begin
          identify the element in lp which belongs to G;
          locate where the element should be inserted;
          append another element to G;
          update connectivity attributes;
          dispose(lp)
        end;  {if}
      end;  {while}
    end  {then}
  else
    begin
      create a new group, G', on the group list;
      set group size of G' to 2;
      create connectivity attributes of the two elements
    end;  {else}
  dispose(lp)
end;  {while}
```

Table 3-5. Algorithm of Loose Appendage of Elements

3.1.5 Single Element Groups Formation

If there is still any elements which cannot be appended to any existing groups (that means they are elements with small connectivity to other elements), they will be classified as single element groups. The steps in this procedure is:

1. starting from the first connection pair.
2. find element which does not appear in the group list.

3. if such element is found,
   (i) create a new group (single element group) with this element,
   (ii) set the connectivity attributes.
4. if not end of connection list, goto step 2.
5. stop.

The algorithm in programmable form is as follows:

```
start from the head of the connection list;
while not(end of connection list) do
begin
  lp:= the current connection pair;
  if (any element of lp not in the group list) then
  begin
    create a new group, G', in the group list;
    set size of G' to 1;     {since it is single element group}
    add connectivity attributes of the element;
  end;  {if}
  shift to the next connection pair;
end;  {while}
```

Table 3-6. Algorithm of Creating Single Element Group

3.2 The Alteration Phase

The aim of this phase is to find out a solution on an existing placement according to minor change to the design. The change pin-pointed in this thesis is the addition of element(s) to an original design. There are two main reasons:
(1) the addition of element(s) or component(s) to a logic circuit or system is relatively common as compared to the removal of component(s).*
(2) removal of components is relatively straight-forward to carry out by simply omitting the components. Since the original design already have enough space, no alteration is necessary. Even if compaction of components is necessary, the process can be done by traditional shrinking algorithms [Dunlop85, LaPotin86]. However, in the case of element addition, no published method is available.

* Examples are available in the design work of the author in "Serial Data Synchronizers/Desynchronizers implemented on Macrocell Arrays", BSc Thesis, Dept. of Electronic Engineering, The Chinese University of Hong Kong, 1988.

The alteration phase consists of four steps, namely, (1) Element Assignment to a Group, (2) Empty Space Searching, (3) Determination of Direction of Element Allocation, and (4) Element Allocation.

Step 1 is to determine which group the added element belongs to. The measure is by number of connections.

Step 2 is to find the nearest empty space to the assigned group of the added element.

Step 3 is to determine the optimal path joining the group and the nearest empty space which satisfies two criteria (will be stated in section 3.2.3).

Step 4 is the placing of the added element to the array.

However, execution of step 3 and 4 depends on the result of step 2. The case is shown in the Figure 3-4.

In Figure 3-4, ES denotes Empty Space. In the step "Empty Space Allocation", if any ES on the neighbourhood of the group determined in step 1 is identified, direct addition of the new element to the empty space will be executed. However, if such an ES cannot be identified, step 3 and 4 will be executed.

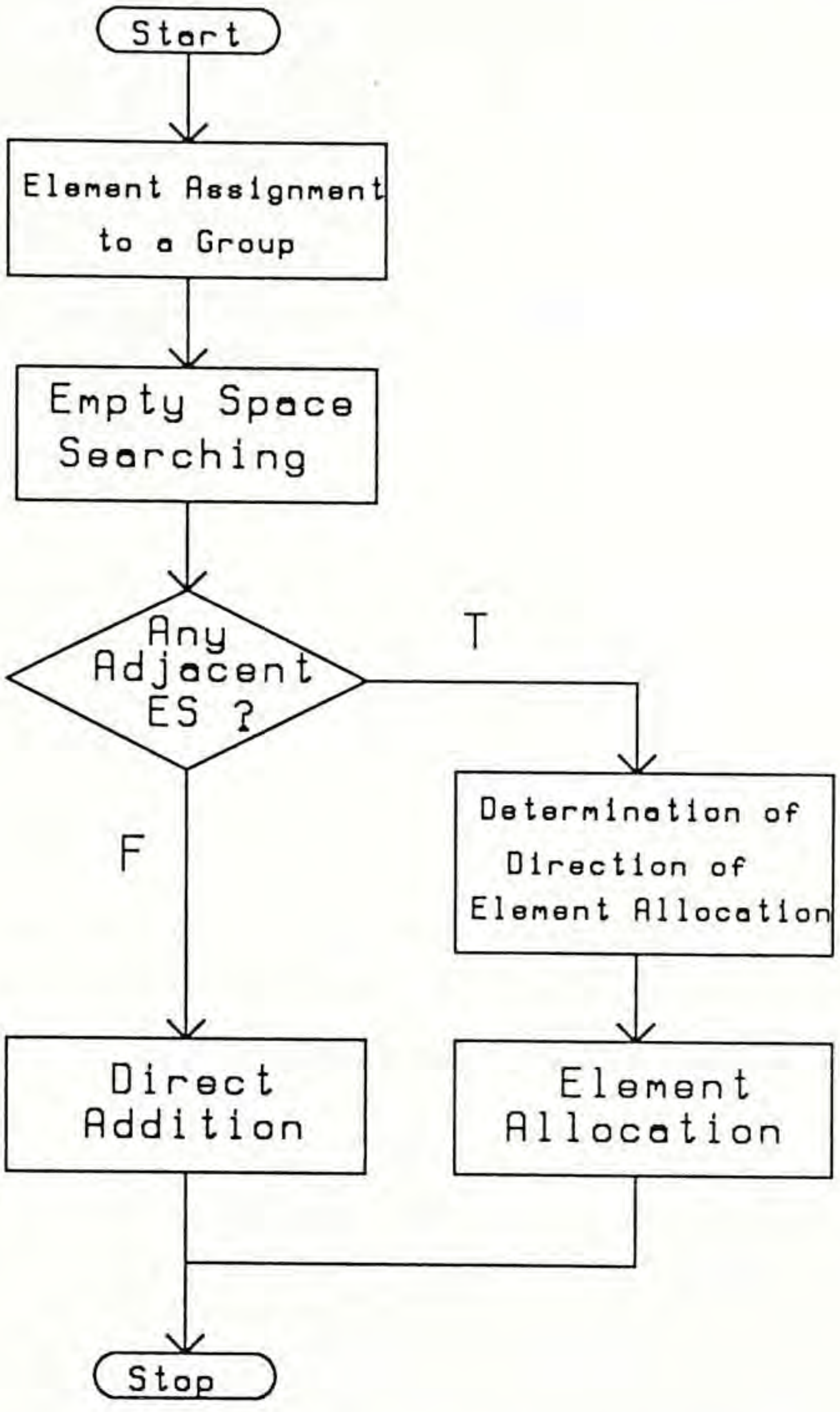

Figure 3-4 Flowchart on the Alteration Phase

3.2.1 Element Assignment to a Group

The assignment of an added element to a group is mainly based on measure of the belong tendency of the element to the groups. That is, an new element is assigned to a group with most connection to the element.

In mathematical form (algorithmic form),

$e \in Gi$ in the case that $BT(e,Gi) \geq BT(e,Gj)$ $\forall$ $Gj \in GS$, $Gi \in GS$.
where e is the added element,
GS is the group set, and
Gi, Gj are groups in GS.
BT(e,G) is the belong tendency of e to G.

The new element will be included in the appropriate group and its connectivity attributes will be updated.

3.2.2 Empty Space Searching

After assignment of the new element to a group, an empty space should be identified for the addition of the element. Firstly, an empty space around the group to accommodate the added component should be identified. This step is termed "Neighbour Search". That means we are attempting to find out an empty space in the neighbourhood of the assigned group.

If only one appropriate empty space can be identified, the component would be placed to that space without further processing. However, if more than one empty space are identified, the one with the minimum cost would be chosen.

If an adjacent empty space cannot be allocated, the nearest empty space is allocated by applying the Lee's Algorithm [Lee61]. The group to be connected is the source and the empty space is the target, ξ . The source cells are marked with an integer. Then, in scanning through the whole matrix, the neighbour cells to source cells are marked with the next integer. Subsequently, more cells next to these marked cells will similarly be marked with increasing integer numbers. The process is repeated until an empty space is found.

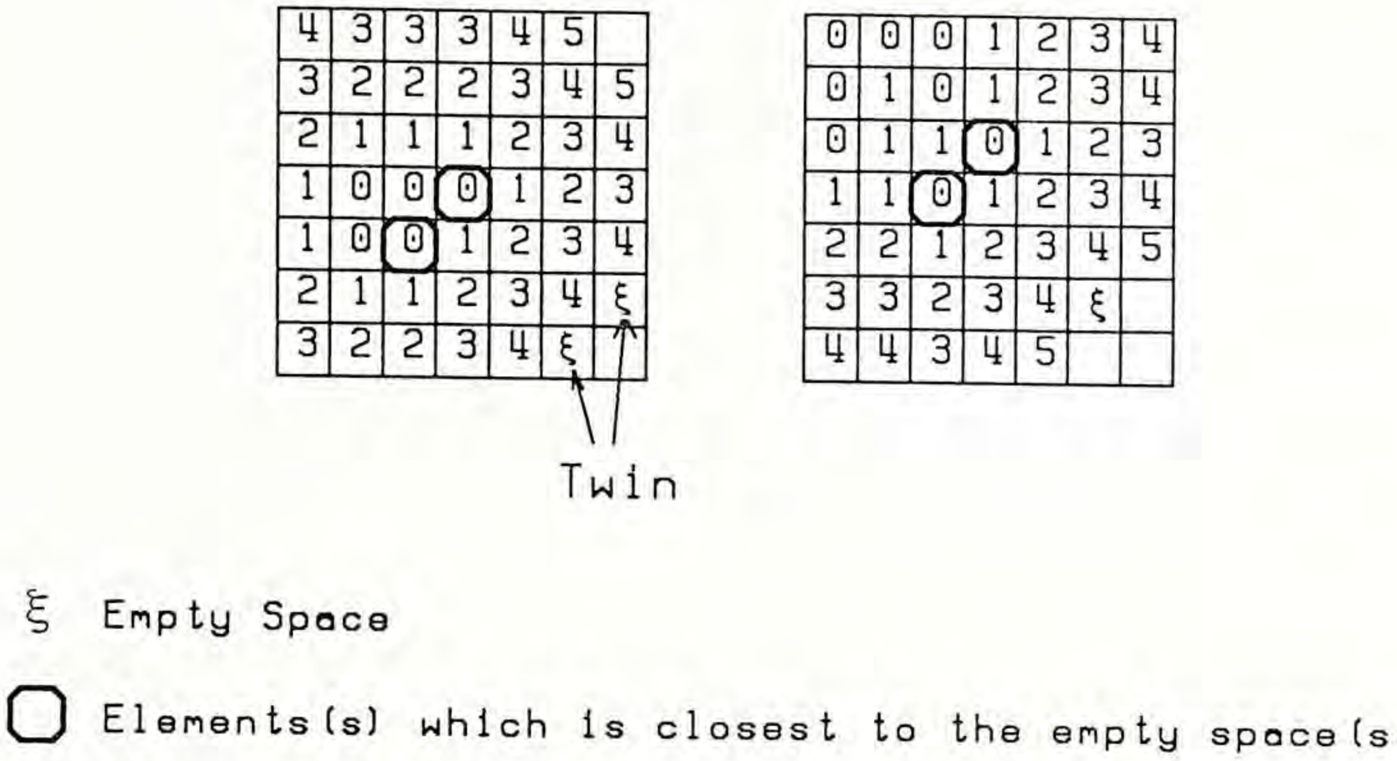

Figure 3-5 Determination of the Nearest Empty Space by Lee's Algorithm

In some occurrence, there will be more than one empty space in the plane. An example of twin empty space is shown the Figure 3-5. In these cases, the empty space with least "potential energy" will be selected in the following procedure. The term "potential energy" will be defined by Eq. 3-4 in section 3.2.3.2.

After identifying the empty space, it is also necessary to determine which element(s) in the assigned group is/are closest to the empty space, ξ . It is because there may be more than one element with the smallest Manhattan Distance to the empty space. As in the examples in Figure 3-5, the elements encircled are with same Manhattan Distance to ξ. These elements are termed "indicator cells" since they indicate the location and direction to place the added element. However, up to this stage, it is still not possible to determine which cell is the most appropriate one to the added element. Hence, all the combinations of these nearest elements will be tried and their cost values as stated by Eq. 3-4 (section 3.2.3.2) will be calculated in the next step. The indicator cell with the lowest cost value will be chosen.

3.2.3 Determination of Direction of Element Allocation

Although the empty space(s) and the indicator cell(s) with smallest distance are located, the path which joins the two points with the optimal cost should also be found. The optimal cost in this sense is judged by two criteria:

1. the number of groups on the path should be optimized. ------- Cri. 1
2. the accumulated size of the "passing groups" should be optimized. ------- Cri. 2

The term "optimization" is used instead of "minimization" since algorithmic calculations usually find out the "optimal" solution instead of the "minimal" one [Sangiovanni-Vincentelli87, Barnes85].

This step is composed of two parts: (1) Cross-cut Direction of Allocation, and (2) Dynamic Determination of Optimal Path by Size Functions. This step will be done on each "indicator cell" to the empty spaces (Figure 3-5) and the one with the least cost is selected.

3.2.3.1 Cross-cut Direction of Allocation

In a 2-dimensional coordinate plane, the line joining (x_1,y_1) and (x_2,y_2) can be represented by the geometric equation:

$$y = (y_2-y_1)/(x_2-x_1) + c \qquad \text{............. Eq. 3-3.}$$

where $c = (x_1y_2-x_2y_1)/(x_1-x_2)$.

However, in a 2-dimensional grid plane, such line cannot be represented by geometric equation. Hence, the approximation named "cross-cut" is introduced.

The cross-cut between two points is defined as the shortest path which terminates at the two points and is closest to the straight line joining them.

The direction of cross-cut is determined by the line joining the indicator cell, c, and the empty space, ξ. And, the classification of direction is shown in Figure 3-6. The direction is determined to be in the increasing i and j direction for the simplicity of calculation. This criteria is determined by the α-axis, which has the equation: $i+j = 0$. As a single value in the range is mapped to several value in the domain, it is necessary to interchange the positions of range and domain. The cases are shown in Figure 3-6 (b) and (c): in (b), a single value of i is mapped to two j values, hence j should be the range and i be the domain, while in (c), vice versa. To classify these two cases, the quadrants are divided again by the β-axis. Hence, for slope magnitude greater than one, j-axis is the abscissa and i-axis is the ordinate, while for slope magnitude less than or equals one, i-axis becomes the abscissa and j-axis the ordinate.

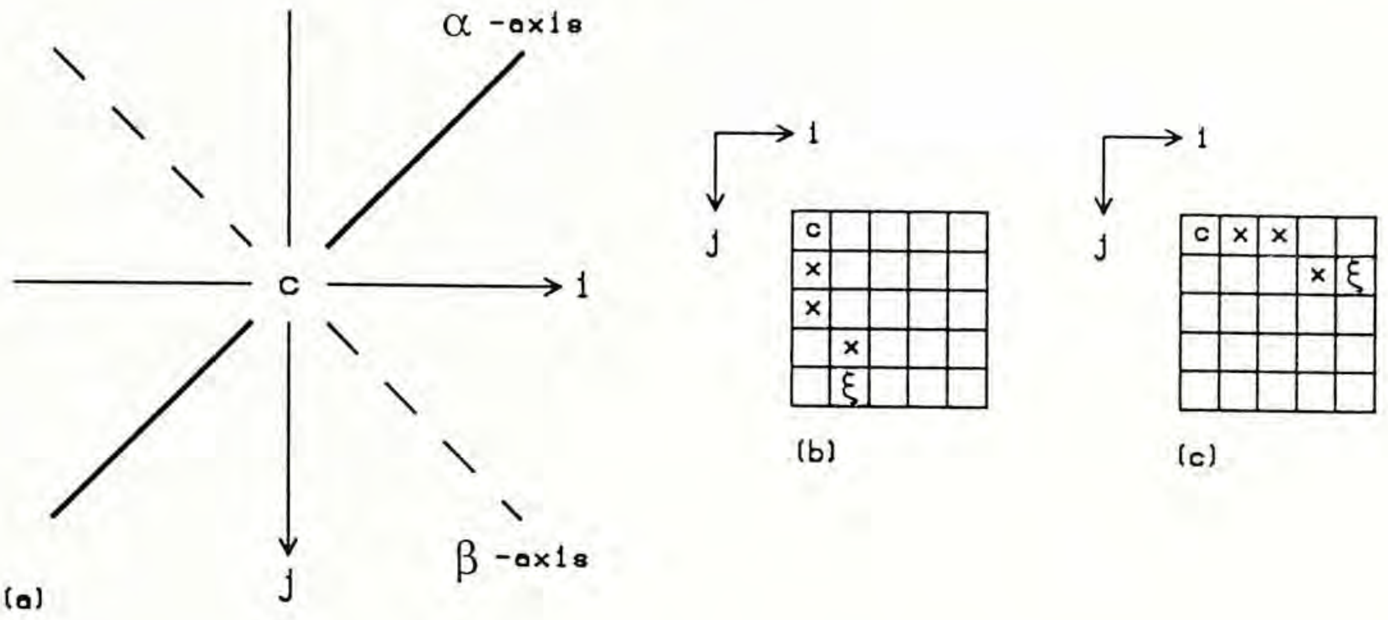

Figure 3-6 Division of direction of cross-cut

As a value lays on α or β-axis, it is necessary to set bounds on these axes such that we can determine to which region the value belongs. Hence, closed bounds and open bounds are set on the α-axis and β-axis, where a closed bound includes values on the bound while an open bound excludes such values. In figure 3-7, the square bracket, [, denotes the closed bound, while the parenthesis, (, denotes the open bound.

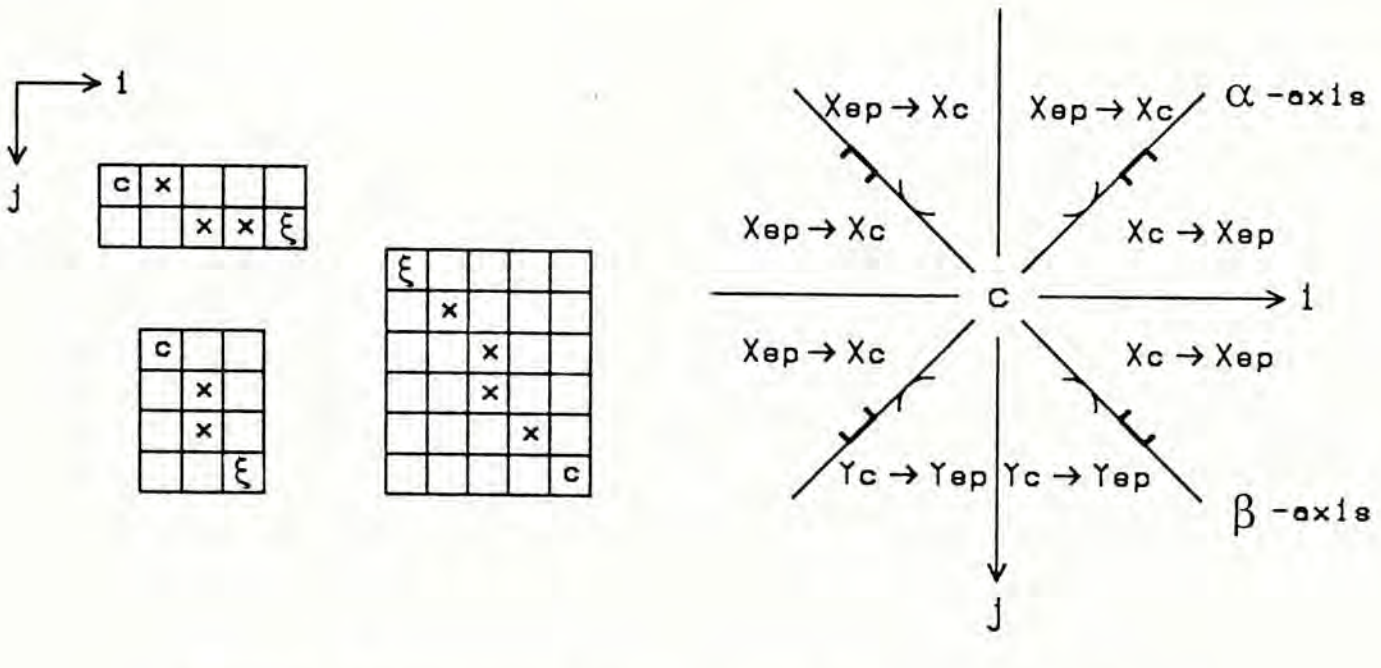

Figure 3-7 Cross-cut Examples and Its Direction Classification

This whole procedure on determining the shortest path joining the indicator cells and the empty spaces are termed "Cross-cut Determination".

3.2.3.2 Dynamic Determination of Path Based on Size Functions

Although the shortest path joining c and ξ is found by the above method, it is necessary to locate the optimal path which satisfies the two criteria stated in section 3.2.3.

In short, the determination of the path of expansion depends on two functions:

F_1 = no. of group passing through

F_2 = Σ size of "passing" groups

and the cost function is defined by

Cost Function = $w1*F_1 + w2*F_2$ .... Eq. 3-4.

Since there are many connections among elements within a group, the total length of interconnections should be increased if there is change to the original placement of any element in a group. Hence, it is more advantageous to change the placement of as little number of groups as possible. On the belief that move of several elements in the same large group will cause less perturbation to the placement than move of elements in several small groups, it is reasonable that w_1 should be greater than w_2. Since F_1 is by nature a smaller number than F_2, the ratio $w_1 : w_2 = 5$ is proposed.

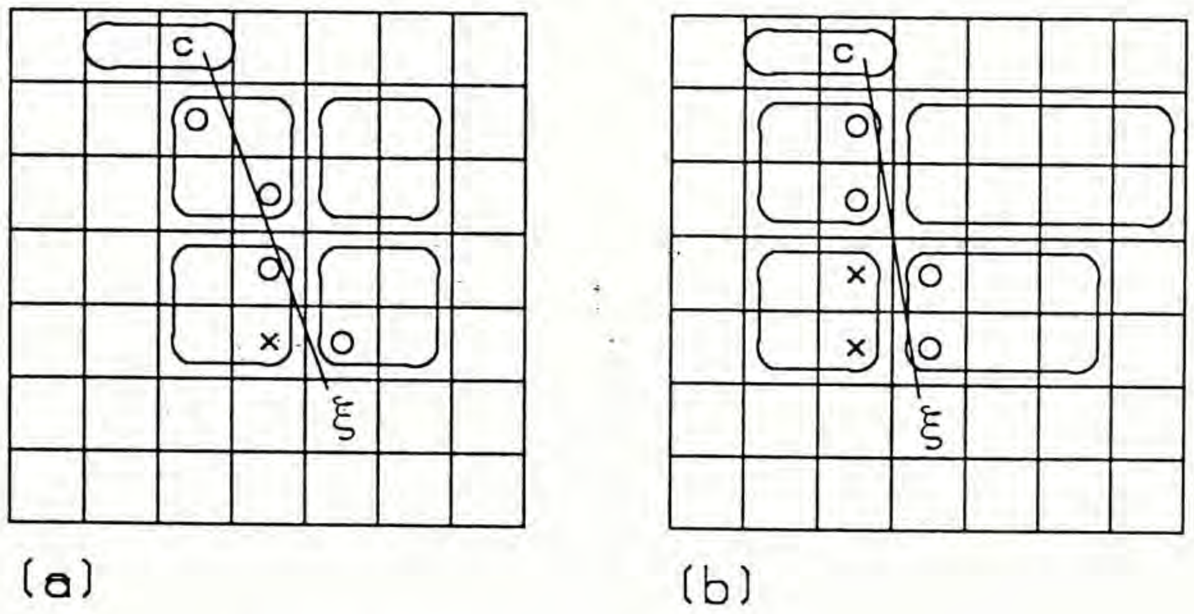

Figure 3-8 Examples on Path Determination

In Figure 3-8 (a), the path with circles is the cross-cut and it passes through three groups. However, if the circle closest to the empty space, ξ, is replaced by the cross, the number of "passing" groups would be reduced to 2. Thus, the latter path is preferred.

However, if the number of groups cannot be reduced, it is better to choose a path which passing through groups with smaller sizes. As in Figure 3-8 (b), if the path with crosses is chosen, the accumulated size of the crossing groups would be minimized.

To determine the optimal path, the following three steps are proposed. They are:
step 1: Segmentation of Cross-cut,
step 2: Partial Optimization of Segments, and
step 3: Dynamic Linking of Path Segments.

3.2.3.2.1 Segmentation of Cross-cut

This step is to divide the cross-cut into segments with Manhattan Lengths (L_{MHT}) less than or equal to 5. A value of 5 is chosen because there will be too much variety on a path segment with L_{MHT} larger than 5. On the other hand, a small value of L_{MHT} would result in exhaustive search on the plane which is not appropriate for an applicable algorithm. Some of the examples are shown in Figure 3-9.

In Figure 3-9 (a), L_{MHT} between c and ξ is 4, and there is one segment only. However, in figure (b) to (d), Manhattan Lengths of the cross-cuts are greater than 5. For simplicity of calculation, they are divided into segments. Optimal path of each segment will be found in step 2 and the linking of segments will be done in step 3.

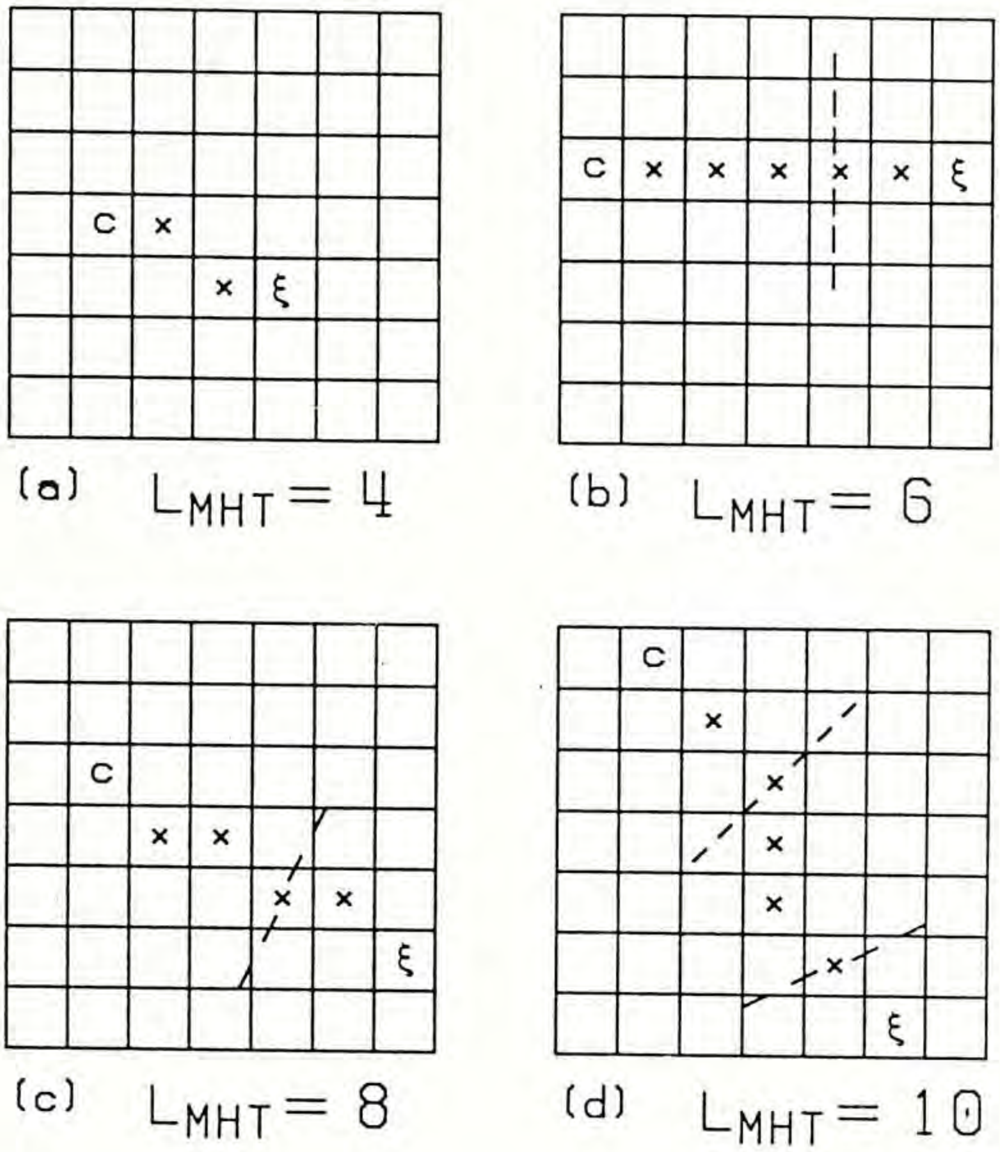

Figure 3-9 Examples on Segmentation of Cross-cuts

3.2.3.2.2 Partial Optimization of Segments

In the previous step, cross-cut is divided into segments. Since the L_{MHT} of these segments is less than or equal to 5, the templates in Figure 3-10 are used to find the optimal paths with L_{MHT} of 4 and 5. The template with appropriate shape will be fitted on a path segment and the cost of each path will be found. Combining these path segments will form the whole path. The combination of path segments with the least cost function (Eq. 3-4) will be chosen in the next section.

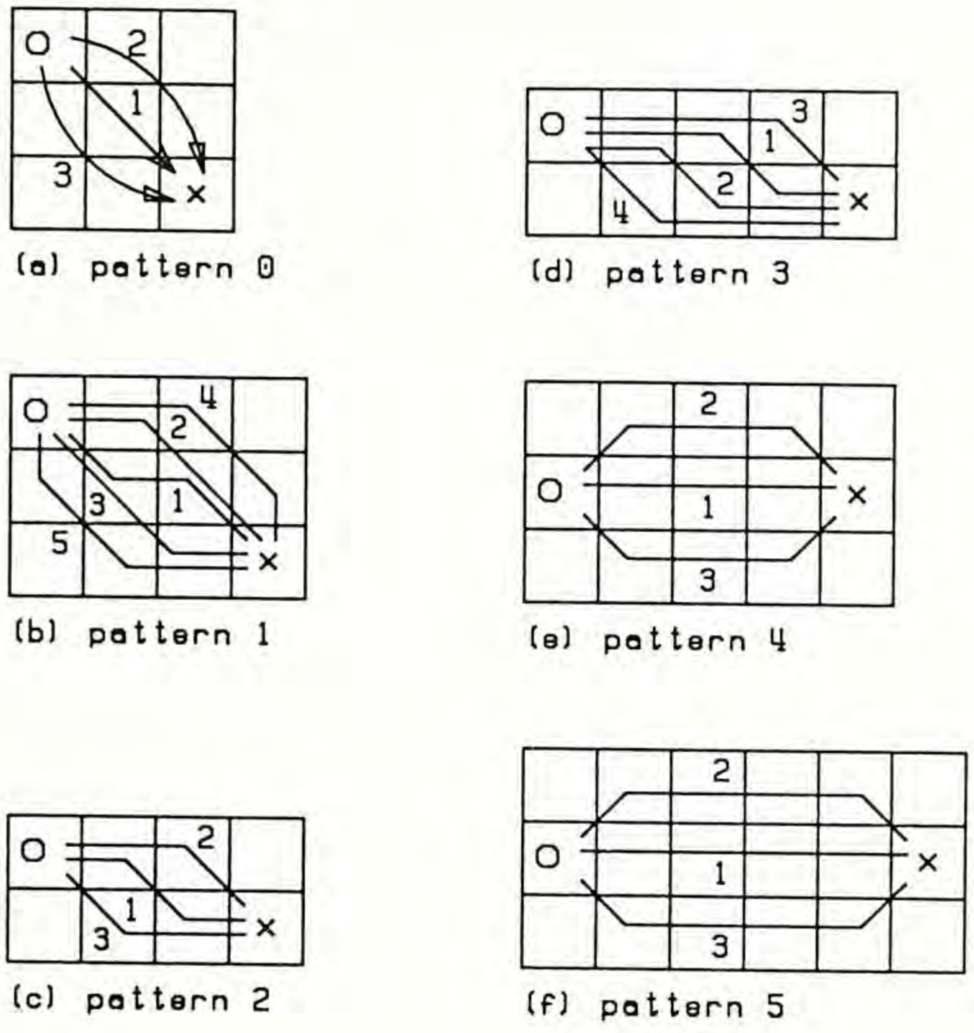

Figure 3-10 Path Templates on Manhattan Lengths of 4 and 5

In the figure, o and x denote the end-points of a segment. Numbering of paths indicates the preference of selection, shorter path with higher priority. The above templates are invariant to axes transformation. That is, the templates can be applied to path segments with x-y coordinates transformed.

In Figure 3-9, it is noted that some segments are of $L_{MHT}=2$ or $L_{MHT}=3$. Templates for these segments are as follows:

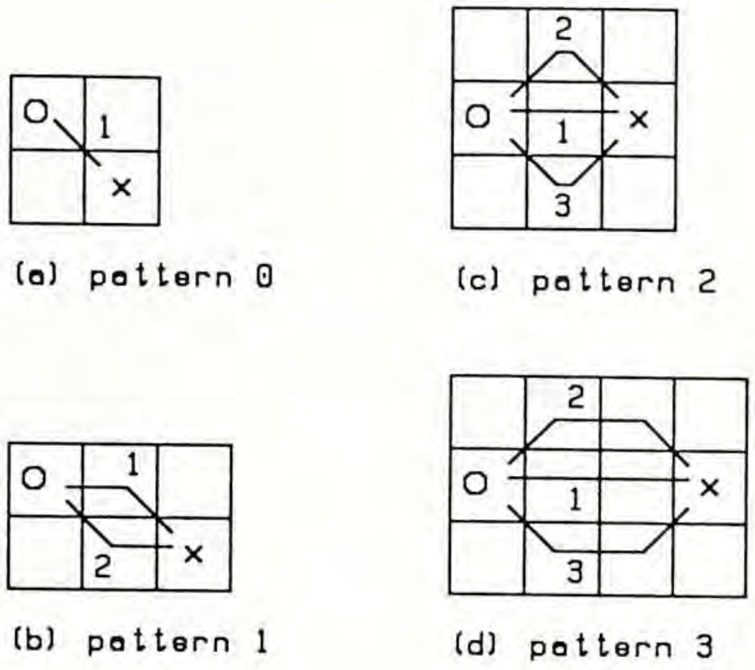

Figure 3-11 Path Templates on Manhattan Lengths of 2 and 3

For segments with L_{MHT} of 1, o and x are adjacent and thus template is not necessary.

3.2.3.2.3 Dynamic Linking of Segments

To link the segments, we aim to find the path with the least cost function. However, there are many combinations of the segments as the distance between c and ξ is large. To facilitate the calculation, it is constructive to introduce the Dynamic Programming (DP) approach [Denardo82, Bertsekas87]. To illustrate the principle of DP, let's analyze the following flow diagram:

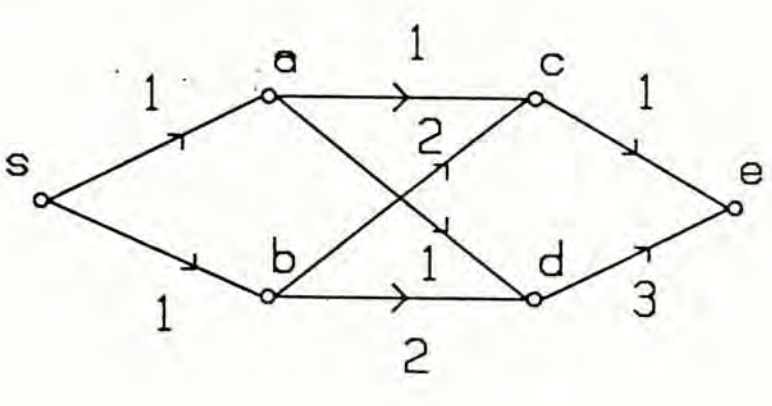

Figure 3-12 Linking of Segments on a Path

Nodes are denoted by alphabets and the costs in the path are indicated by the integer. The calculation process is shown in the following table. Pxy denotes minimum cost between x and y.

```
  Cost          Pxy
 ce :  1  -->  Pce
 de :  3  -->  Pde

 ae: ace : Pce + 1 = 2 --> Pae
     ade : Pde + 1 = 4
 be: bce : Pce + 2 = 3 --> Pbe
     bde : Pde + 2 = 5

 se: sae : Pae + 1 = 3 --> Pse
     sbe : Pbe + 1 = 6
```

Hence, the minimum cost is 3 and the optimal path is (s,a,c,e).

Table 3-7. Calculation of Shortest Path by Dynamic Programming

The algorithm of Dynamic Programming [Denardo82] is:

```
1. Set Vj to infinity for j = 1,2, ... , N.
2. for i:= 1 to N-1 do
    3. for j:= i+1 to N do
        Vj:= min{Vj, Vi + Cij}
```

where Vx is the minimum cost from x to the terminal node, and
Cxy is the cost between x and y.

Table 3-8. Algorithm of Dynamic Programming

3.2.4 Element Allocation

After locating the optimal path of element allocation, there are two steps to place the added element to the array:

1. shift the cells along the optimal path towards the empty space until the empty space is adjacent to the indicator cell, c.
2. place the added element at the evacuated space.

Two examples on the process of element allocation are shown in Figure 3-13.

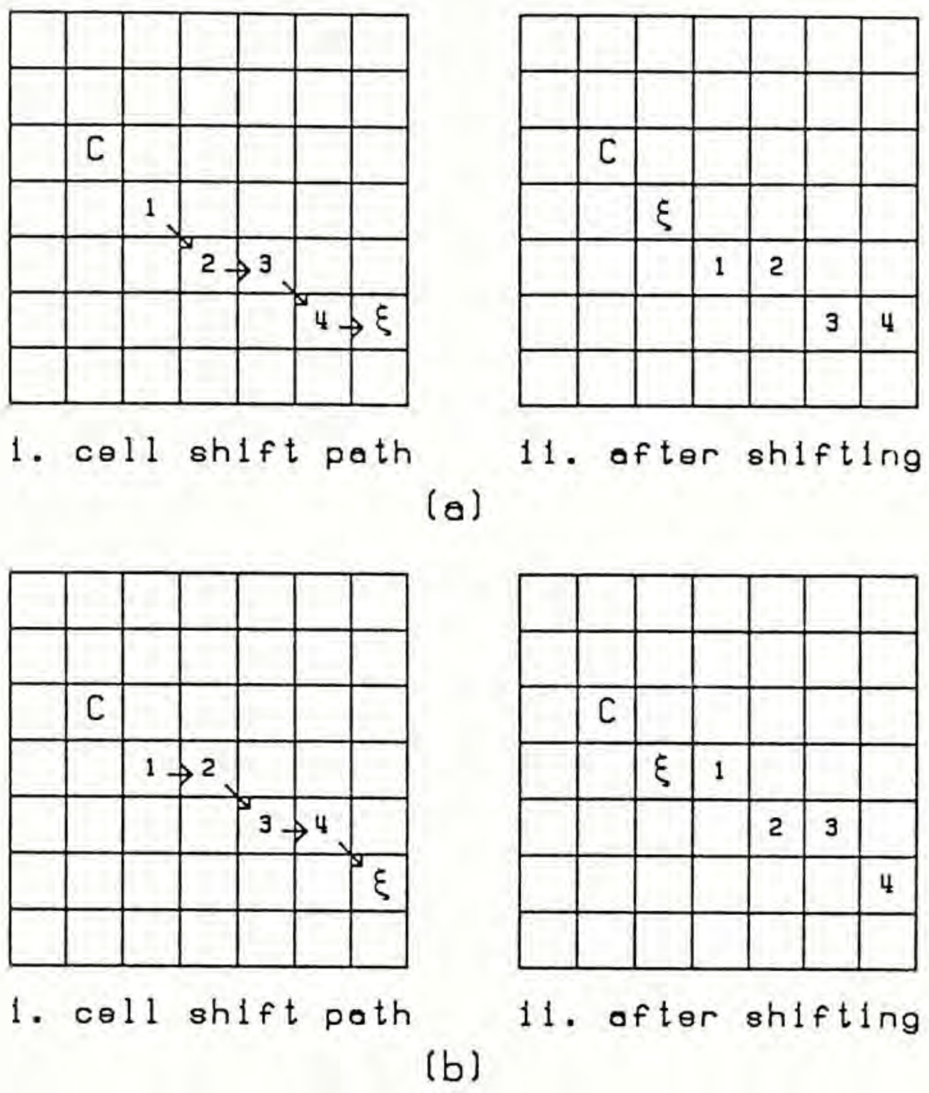

Figure 3-13 Examples on New Element Allocation

Integers on the slots identify the cells to be shifted. Figure (a)ii and (b)ii are the placements after the first step. The new positions of the shifted cells are shown. In these placements, the empty spaces are adjacent to the indicator cell, c. Then, the added element will be placed in these empty spaces.

Finishing the Element Allocation step in the Alteration phase, the addition of an element is completed. To add another element, it is necessary to start at the step "Element Assignment to a Group". After the completion of placement alteration, the connectivity attributes and the placement database will be updated so that further processing is possible. In the next section, the implementation of this algorithm will be described.

4. Implementation

The program is mainly composed of two parts: (1) the affinity clustering phase, (2) the alteration phase. The programs are written in Pascal.

The programming language Pascal was chosen in the implementation of the algorithm because of the following reasons:
Pascal's features: [Holden87]

1. Designed to support predominantly numerical, sequentially executed algorithmically based problem model.
2. Block Structuring.
3. Scope variables, i.e. variables with ranges of validation.
4. Procedure/Function Block Definitions.
5. Facilities for forming complex data types using records and arrays.
6. Pointer and record facility useful for forming linked list and transferring of data.
7. Data structures can be dynamic to some extent. (useful for implementing dynamic functions and procedures)
8. Number and Set manipulation facilities. (useful for numerically based decision making)

However, Pascal do not easily support symbolically represented entities such as strings and coordinates. This can be overcome by using data structures like records and dynamic variables.

4.1 The System Flow

The system flow diagram of the Incremental Alteration Placement Algorithm is shown in figure 4-1. The pre-layout simulation phase is done before the placement phase. If there is change in the design after the placement is completed, the Incremental Alteration Placement Algorithm (IAPA) is executed. Another possible way to obtain the clusters (or groups) is by inheriting the hierarchical division from the original schematic. That means the clusters are classified by the functional blocks of the design. However, in some cases, some components, especially interface elements and random logics gates, may not belong to any functional block. Thus, the affinity clustering phase is also applicable in these cases.

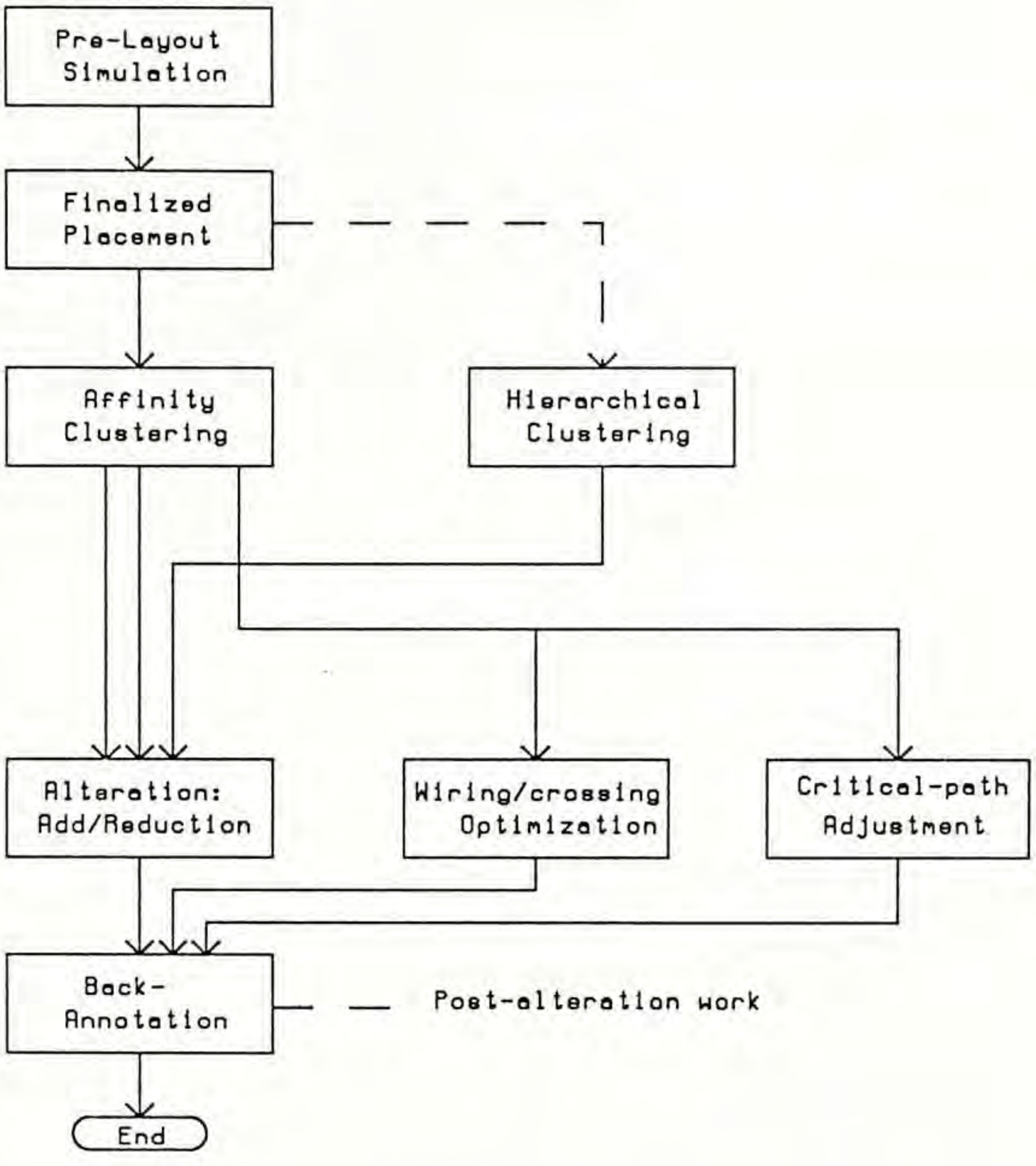

Figure 4-1 The System Flow Diagram

The function of the affinity clustering phase appears in three forms:

1. partitioning for the alteration phase.
2. to minimize wiring congestion across cut lines.
3. to adjust element placement along critical paths.

In this thesis, the main concern is the construction of the affinity clustering phase and the alteration phase. Wiring optimization and critical path adjustment will be discussed in section 6.

For a complete CAD system, the new placement is checked in the Back-annotation phase in which the simulation is done on the actual layout.

4.1.1 The Affinity Clustering Phase

In this phase, clusters are formed by progressively gathering elements with high connectivity. The method is to construct connection lists from the connection array and then to obtain information by scanning through the connection lists. The flow of the program is shown in figure 4-2. At the beginning of the program, the "placement" and "connectivity" files are read. The "placement" file contains result of the design on a simulated annealing placement program (fplace77.pas).

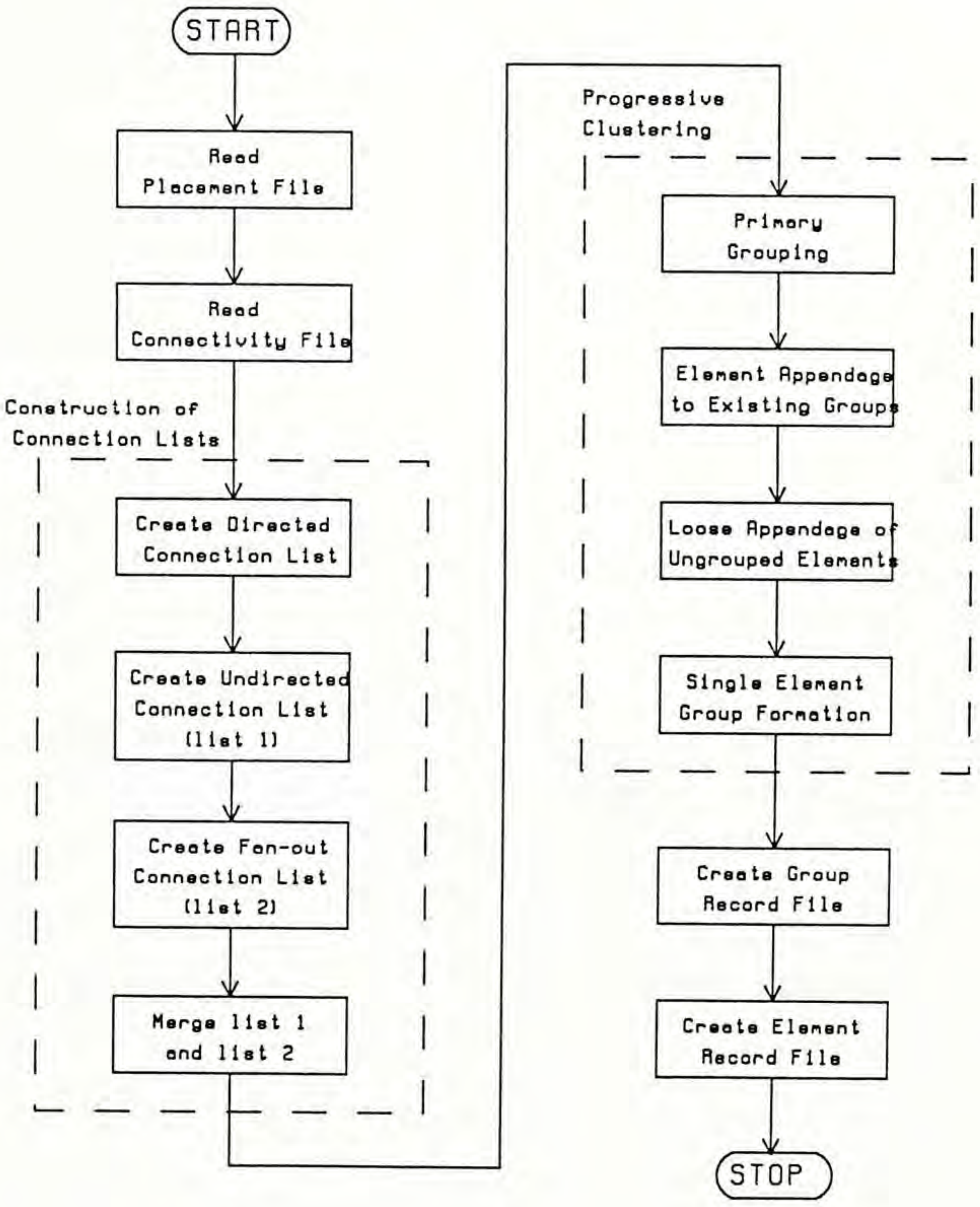
START
Read Placement File
Read Connectivity File
Construction of Connection Lists
Create Directed Connection List
Create Undirected Connection List (list 1)
Create Fan-out Connection List (list 2)
Merge list 1 and list 2
Progressive Clustering
Primary Grouping
Element Appendage to Existing Groups
Loose Appendage of Ungrouped Elements
Single Element Group Formation
Create Group Record File
Create Element Record File
STOP

Figure 4-2 Flowchart of the Affinity Clustering Phase

The construction of connection lists is composed of four parts: (1) create directed connection list, (2) create undirected connection list (list 1), (3) create fan-out connection list (list 2), and (4) merge list 1 and list 2 to form the link-related connection list. All the information on connection among cells are summarized in the link-related connection list.

The progressive clustering is composed of four parts: (1) primary grouping, (2) element appendage to existing groups, (3) loose appendage of ungrouped elements, and (4) single element group formation. From data on the link-related connection list, clusters are built and stored in a group list. Data of the group list is stored in two files: "group record file" and "element record file". Group record file contains information on the groups while the element record file contains information on every elements.

In the group list, each element belonging to a group are linked by pointers. The structure of the connection lists and the group list are represented exclusively with dynamic variables, that is, the size of the lists are flexible. In addition, the variables are in order according to their values. Therefore, no sorting procedure is necessary when the database is updated.

4.1.2 The Alteration Phase

In this phase, a new placement solution is found when there is minor change to the design. The flowchart of this phase is shown in figure 4-3. At the beginning of the program, "group record file" and "element record file" are read from the disk. Then, the clusters are restored from data on the two files. In this program, since the structure of clusters is different from that in the previous program, they are named "tree". The "tree" are constructed by two subroutines: build_tree and link_tree. The fill_in_set subroutine updates a set which includes all the elements with group and location assigned. The list_assign subroutine converts information on new elements to the link-related connection list.

The element_assign subroutine assign a new element to the group with the most number of connections. The group is called the "attached group". Element_append includes the new element to the "tree" and update the connectivity attributes. Database setting prepares the two-dimensional planes for finding nearest empty space to the attached group. Then, we will try to find am empty space in the neighbourhood of the group. If found, the placement is done by simple location assignment. The Detail procedure is by computation of the total interconnection lengths and to find the empty space with the least value. The FCFS procedure means First-Come-First-Serve. That is, the new element is placed to the first empty space found. The user can choose between these two procedures.

If an empty space on the neighbourhood cannot be found, we are trying to locate the nearest one. Then, the shortest path joining the empty space (ES) and the indicator cell (C) is found by the Cross_cut subroutine. Size_determine finds out the cost function of the "passing groups" in each path joining ES and C. Path_determine select the path with the least cost function. Expansion is the placement of the new element in the array. If there is another new element, the step returns to the Element_assign subroutine.

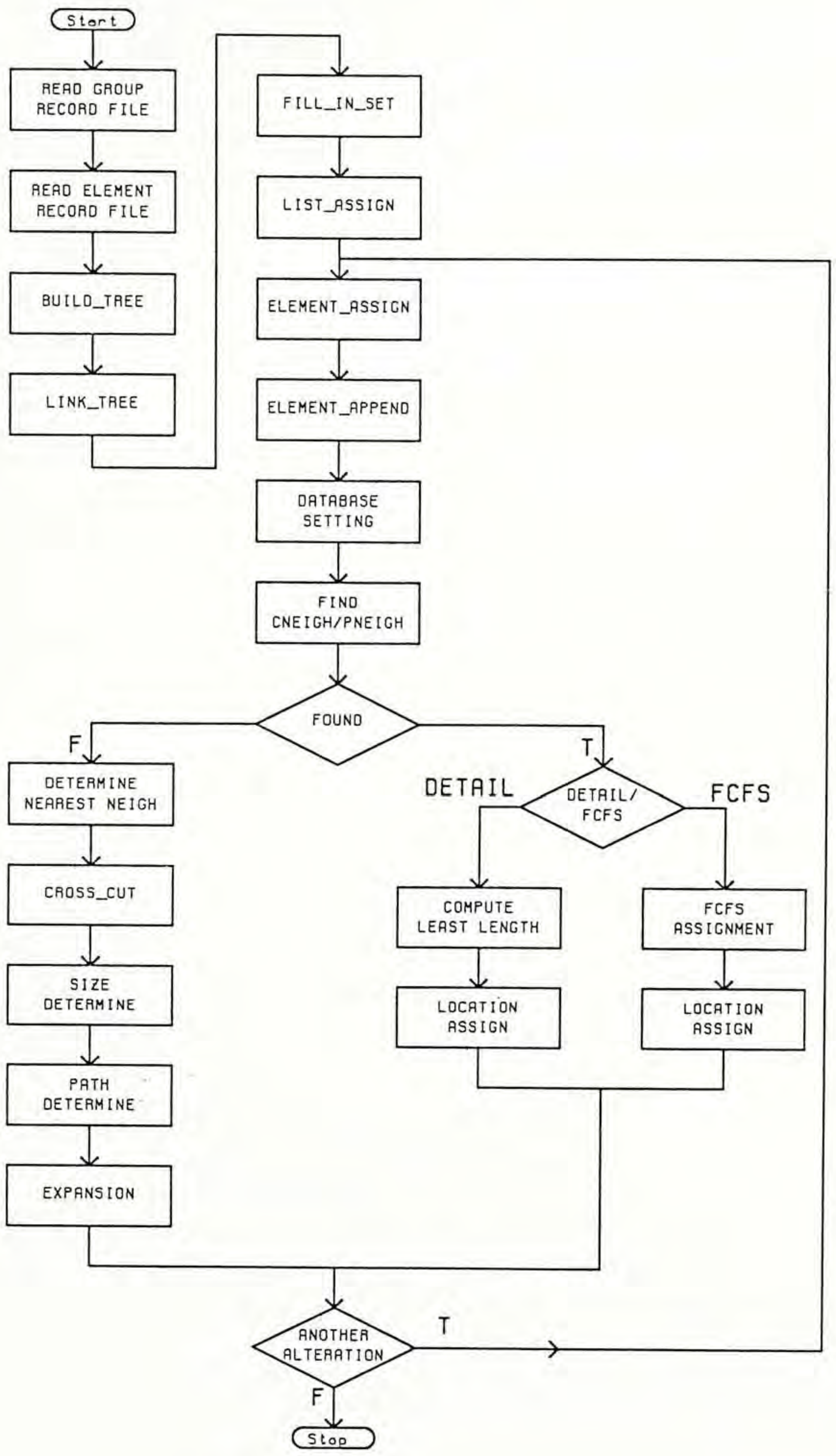

Figure 4-3 Flowchart of the Alteration Phase

4.2 Data Structures

The variables like arrays and records are static variables. This means that all necessary memory is allocated for that variable at the time the program containing the variable declaration is about to begin execution. It remains in existence as long as the program is executing. This approach contrasts sharply with the class of variables we used mainly in the program -- the dynamic variables.

A dynamic variable is created and destroyed dynamically during the execution of a program [Schneider82]. Unlike static variables, dynamic variables are not referenced indirectly by pointers to the newly created variable.

In the first program, namely the affinity clustering phase, there are mainly two variables: connection list and group list. The data structure of the connection lists is represented by the declaration in Table 4-1.

```
lptr = ^listrec;     {list variable}
listrec = record
            next   : lptr;      {pointer to the next variable}
            first  : integer;   {first element}
            second : integer;   {second element}
            con    : integer;   {connectivity}
          end;
```

Table 4-1. Declaration on the connection list

In a connection list variable, "first" and "second" are the elements in a connection, and "con" is the number of connection between "first" and "second" elements. "next" is the pointer to the next variable.

Graphically, the structure is shown in figure 4-4.

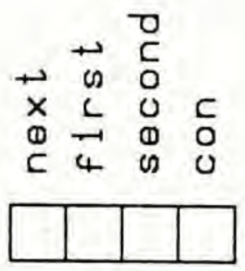

Figure 4-4 Structure of a Connection List Variable

The link-related connection list of the 4-bit synchronous counter example is as follows:

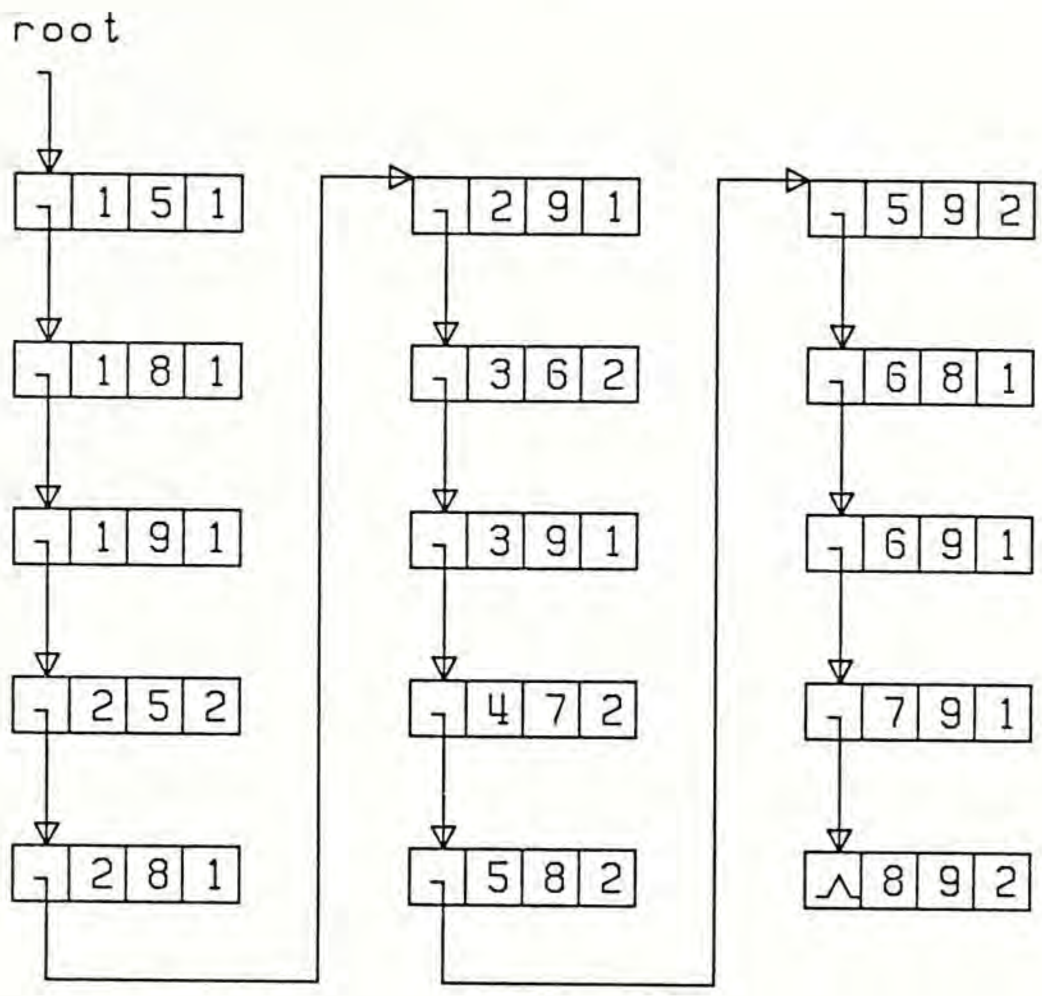

Figure 4-5 The Link-related Connection List of the 4-bit Synchronous Counter

From the above connection list, clusters are formed and stored in the group list. The data structure of the group list are represented by pointers with the following declarations (Table 4-2):

```
  cptr = ^cmpdrec;      {compound variable}
 cmpdrec = record                  {compound record}
            inst  : integer;  {instance name (in number)}
            coox  : integer;  {coordinate in x-axis (i-axis)}
            cooy  : integer;  {coordinate in y-axis (j-axis)}
            pri   : integer;  {primary adhesive element}
            pc    : integer;  {primary adhesive connectivity}
            sec   : integer;  {secondary adhesive element}
            sc    : integer;  {secondary adhesive connectivity}
            ep    : cptr      {next-element pointer}
          end;

  bptr = ^bulkrec;       {bulk variable}
 bulkrec = record
```

```
        size     : integer;  {molecule size}
        molecule : bptr;     {next-molecule pointer}
        atom     : cptr      {pointer to the first
                                  element (atom)}
    end;
```

N.B. coox and cooy are used because of their relative ease of identification in program statements.

Table 4-2. Declaration on the Group List in the Affinity Clustering Phase

In the declaration, the "compound variable" refers to the element's variable, while the "bulk variable" is the group's variable. In the "compound variable", "inst" denotes the instance name of the element; the name being represented by integer (number). "coox" and "cooy" are the x and y coordinates of the element in the macrocell array. "pri" is the element (in integer) with the largest number of connections to the instance, and "pc" is the corresponding number of connections. "sec" is the element with the second largest number of connections to the instance and "sc" is the corresponding number of connections. Only the two most related elements to the instance is recorded because the fixed structure of the pointer variables can save both computation time and memory. "ep" is the pointer to the next element variable.

In the "bulk variable", "size" denotes the number of elements in that group. "molecule" is the pointer to the next group (molecule) and "atom" is the pointer to the first element (atom) in the group. "molecule" and "atom" are used to represent group and element respectively because a group of elements seems to be a molecule containing a few atoms. In addition, there are connections among the elements in a group, just like the attractive forces among atoms in a molecule.

Graphically, the structures are as shown in Figure 4-6:

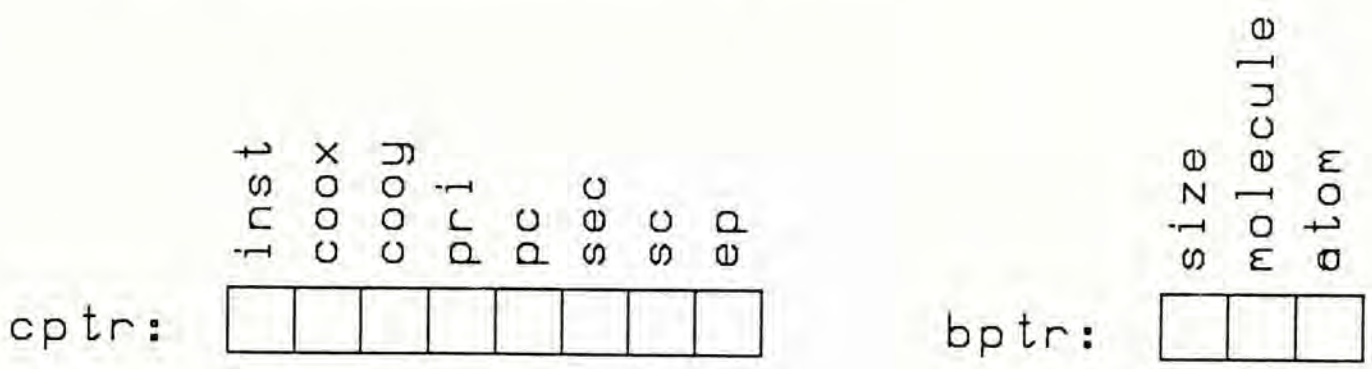

Figure 4-6 Structure of the compound pointer (cptr) and bulk pointer (bptr)

The group list of the 4-bit synchronous counter and the 7-bit ring register (Figure 4-8) after the Primary Grouping step of the Affinity Clustering phase is shown in the following figure:

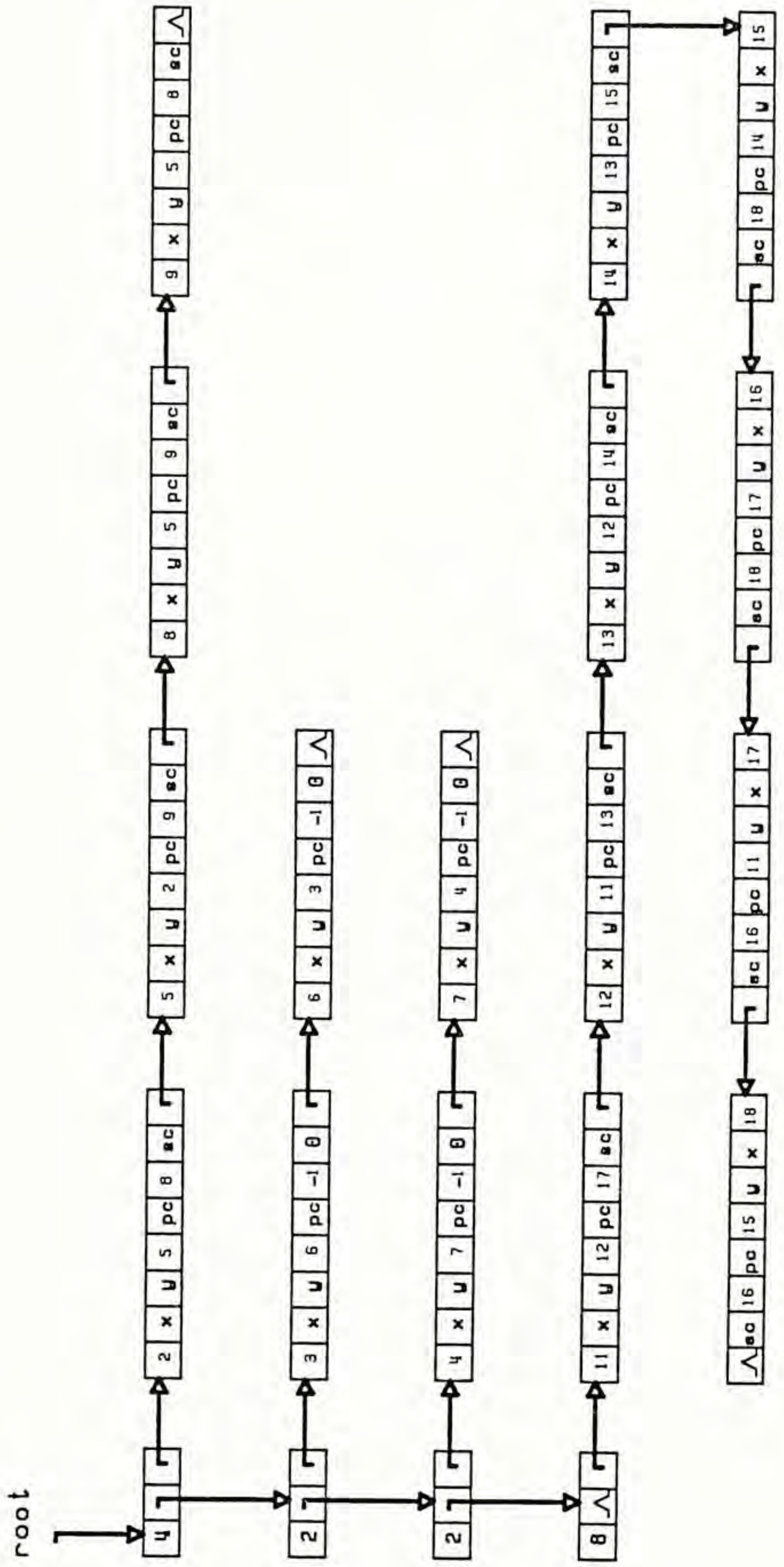

Figure 4-7 Group List of the 4-bit Synchronous Counter and the 7-bit Ring Register

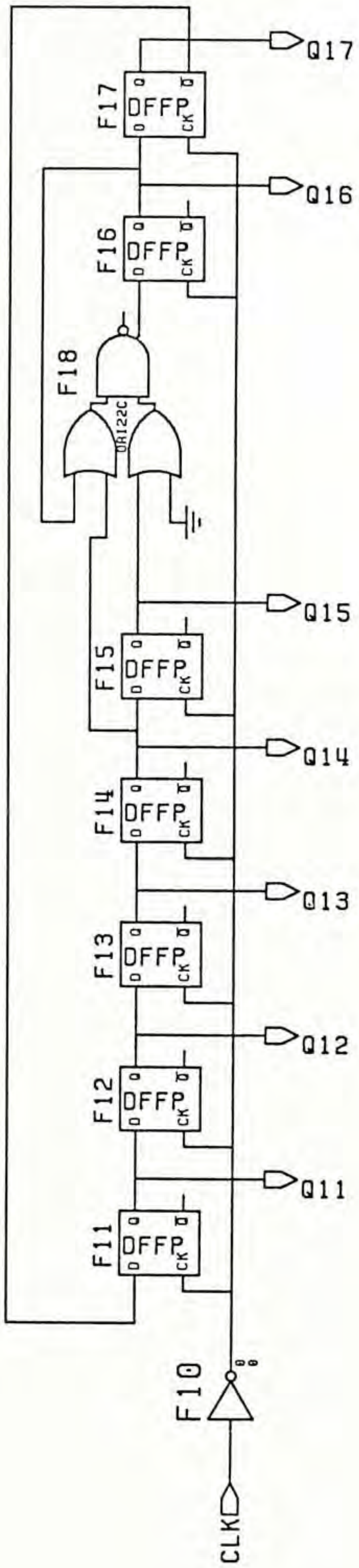

Figure 4-8 A 7-bit Ring Register

The pointer variables in the first column are the "bulk variables". Elements in each row belongs to a group. They are linked by the "ep" pointers. The "molecule" pointers of "bulks" point to the first elements in the groups. The elements are arranged in increasing order of the "inst" values. The "nil" pointer variable is denoted by the symbol _/_ . That means the pointer variable is not pointing to anything.

In the second program, namely the alteration phase, there are also two main kind of variables: cluster tree and map list. The data structure of the cluster tree is similar to that of the group list in the first program. However, instead of recording primary and secondary adhesive elements with integers, they are linked to the instance by the pointer variables "pri" and "sec". Declaration on the Cluster Tree is as follows:

```
  tptr = ^treerec;    {tree element variable}
 treerec = record               {compound record}
             inst   : integer;  {instance name (in number)}
             coox   : integer;  {coordinate in x-axis (i-axis)}
             cooy   : integer;  {coordinate in y-axis (j-axis)}
             pri    : tptr;     {primary adhesive element pointer}
             pc     : integer;  {primary adhesive connectivity}
             sec    : tptr;     {secondary adhesive element
                                  pointer}
             sc     : integer;  {secondary adhesive connectivity}
             ep     : cptr      {next-element pointer}
           end;

  zptr = ^zisprec;     {cluster variable}
 zisprec = record
             size     : integer;  {molecule size}
             molecule : zptr;     {next-molecule pointer}
             atom     : tptr      {pointer to the first
                                       element (atom)}
           end;
```

N.B. 1. "pri" and "sec" are pointer variables instead of integer variables.
2. "zisp" is the name of cluster used in the program; a variable with special name is more easy to be identified in the program.

Table 4-3. Declaration on the Cluster Tree in the Alteration Phase

In the above declaration, most of the attributes are similar to that in the group list. The differences are the two variables "pri" and "sec".

The cluster tree of the 4-bit synchronous counter and the 7-bit ring register after the Primary Grouping step of the Affinity Clustering phase is shown in figure 4-9.

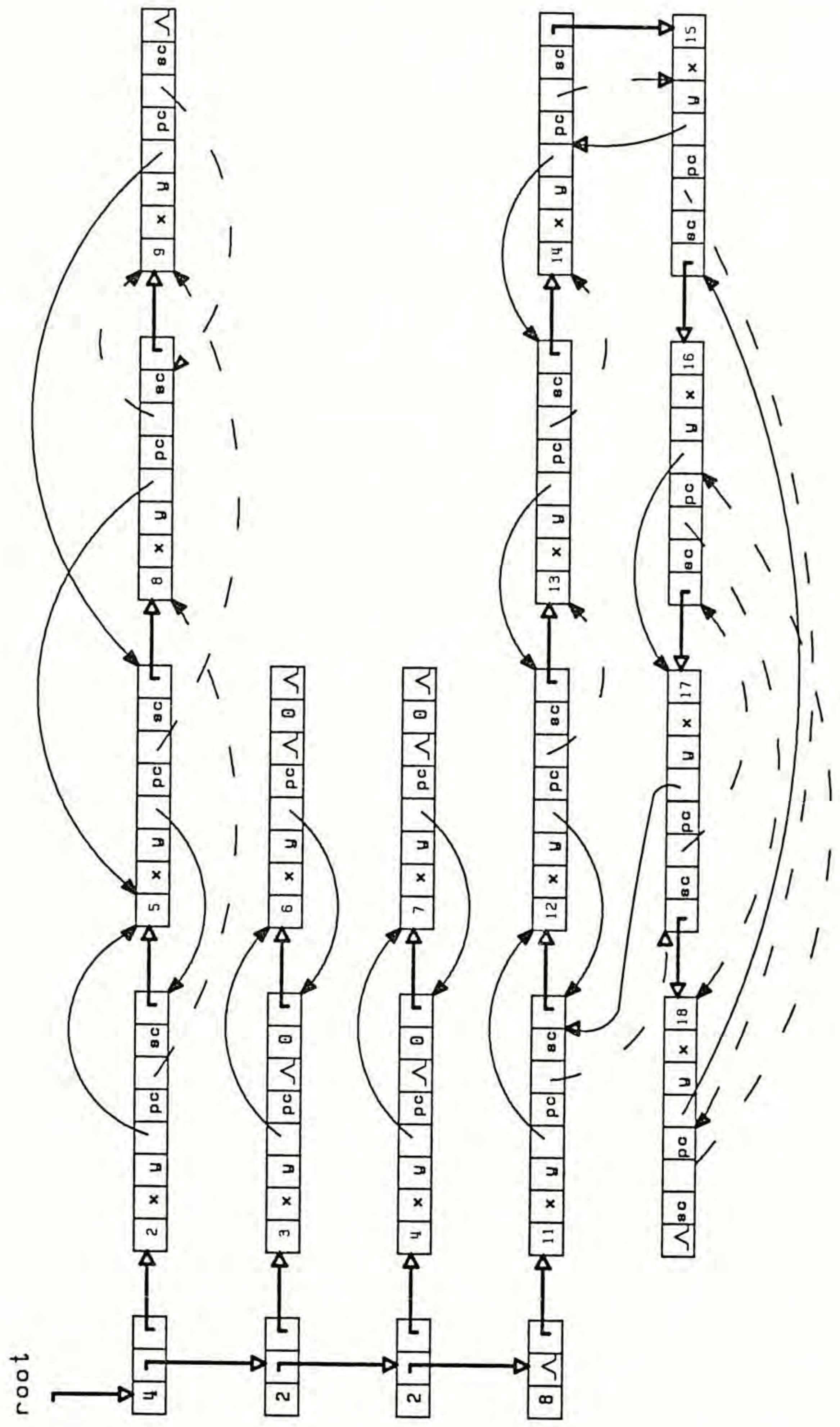

Figure 4-9 Cluster Tree of the 4-bit Synchronous Counter and the 7-bit Ring Register

Pointer variables in the first column are the "cluster variables". Elements in each row belongs to a cluster. They are linked by the "ep" pointers. The elements with "primary" and "secondary" adhesion to the instance are linked by the "pri" and "sec" pointers respectively.

Advantage of using "pri" and "sec" pointers in the cluster tree is that information on the two most related elements can be easily accessed. For example, if "tp" is the record of instance "2",

```
1.   the primary adhesive element is    tp^.pri^.inst = "5"
          with number of connections    tp^.pc        = pc
          and coordinates  (tp^.pri^.coox, tp^.pri^.cooy).
2.   the secondary adhesive element is  tp^.sec^.inst = "8"
          with number of connections    tp^.sc        = sc
          and coordinates  (tp^.sec^.coox, tp^.sec^.cooy).
where pc and sc are integers and pc ≥ sc.
```

Elements of the same group are referenced by the "ep" pointers until the end of the row. It is noticed that elements with a group have the most number of interconnections.

4.2.1 Insertion of Elements to a Linked List

There are mainly three basic operations on a linked list: (1) creation of root, (2) insertion of element, and (3) deletion of element. Since the mostly used one in the programs is the insertion of element, we would like to introduce its operation.

Taking the following linked list element as example:

```
ptr^ = ptr_record;
ptr_record = record
               inst : integer;    {instance}
               next : ptr         {pointer to next element}
             end;
```

Table 4-4. Declaration on the Element of a Linked List.

A proposed procedure [Schneider82] on the insertion of element(s) to the linked list is shown in Table 4-5. Its advantages are that least variable is need in the operation and the elements can be sorted in ascending (or descending) order.

```
procedure insert(var head : ptr; newinst: integer);
{this procedure will insert a new element (newinst) into the
  linked list at the correct place (with ascending order). }
var
  p, q  : ptr; {temporary variables used to search the linked
               list}
  newp  : ptr; {new node inserted in the list}
  found : boolean;
begin
  found := false;
  p:= head; {start searching at the head of the list}
  q:= head;
  while (p<>nil) and (not found) do
  begin
    if (p^.inst<newinst) then
    begin
      q:= p;        {save a pointer to this element}
      p:= p^.next   {move to the next}
    end
    else
      found := true  {we found where it belong}
  end; {of while loop}

{when we arrive here we know where the element properly goes-
 between the node pointed to by q and p.  This is the situation
 depicted in Figure 4-10. }

  new(newp);
  newp^.inst := newinst;

  {now let's insert the new node into the list by adjusting the
   pointers}

  newp^.next := p;
  if q<>p then
    q^.next:= newp
  else
  head := newp
  {this last test was needed for the case where we are
   inserting the head of the list. }
end; {insert}
```

Table 4-5. An Algorithm on Insertion of Element to a Linked List.

Graphically, the operation is shown in figure 4-10.

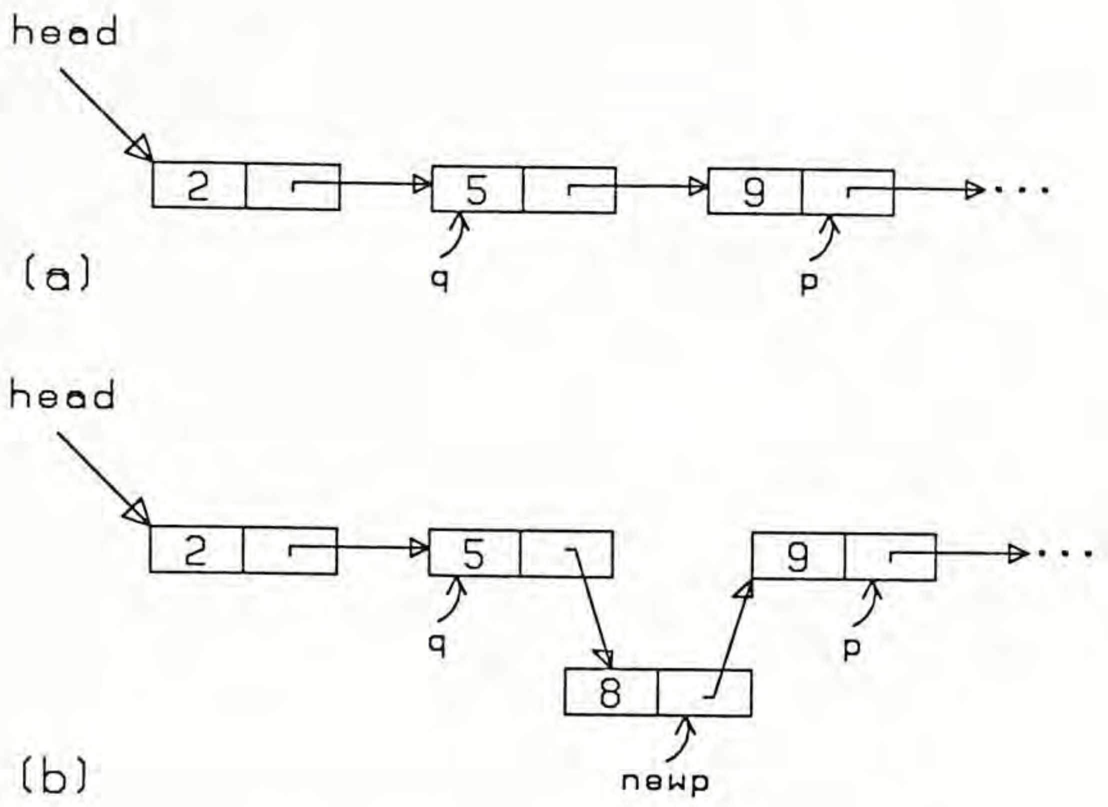

Figure 4-10 List Insertion using Pointers

In the above figure, (a) is the situation upon finding the correct location to insert the element, while (b) is the situation after inserting the new element into the linked list.

4.3.2 Dynamic Linking of Segments

In section 3.2.3.2.3, segments of the path joining c and ξ are linked by Dynamic Programming approach. Implementation of the approach is done by operations on linked lists.

Consider the following path segment:

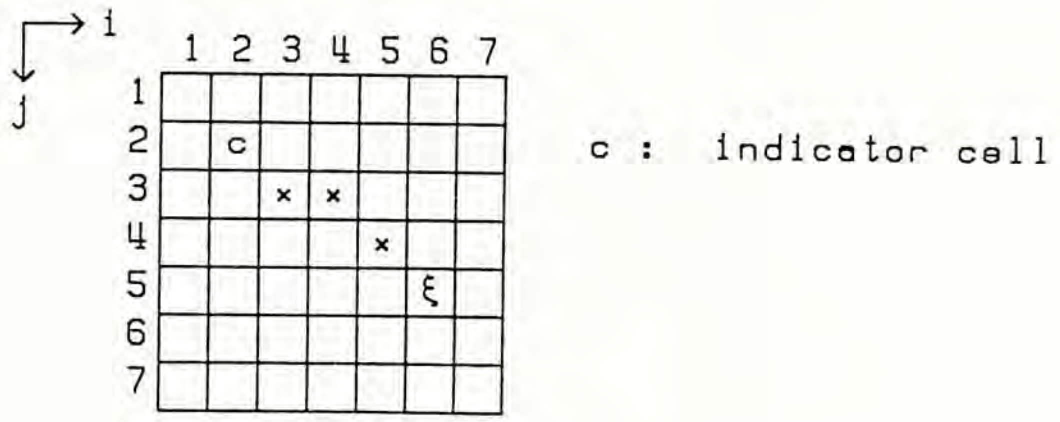

Figure 4-11 A Path Segment in the Array

It can be represented by the sequence: { (2,2), (3,3), (4,3), (5,4) }. The Manhattan Distances of the above locations to the indicator cell are { 0, 2, 3, 5 } respectively. In addition, each location may belong to a group with group number and address in the cluster tree. To effective record the data, the map variable is declared (Table 4-6). The structure is shown graphically in figure 4-12.

```
  mptr = ^maprec;     {map variable}
 maprec = record
            next     : mptr;     {pointer to the next variable}
            dist     : integer;  {Manhattan Distance from
                                   the indicator cell, c.}
            x        : integer;  {coordinate in x-axis (i-axis)}
            y        : integer;  {coordinate in y-axis (j-axis)}
            gpnum    : integer;  {group number}
            cator    : zptr      {indicator to the zisp tree
                                     (cluster tree)}
          end;
```

Table 4-6. Declaration on the map variable

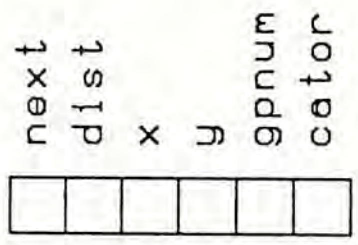

Figure 4-12 Structure of a Map Variable

Hence, the path segment in figure 4-11 is recorded as follows:

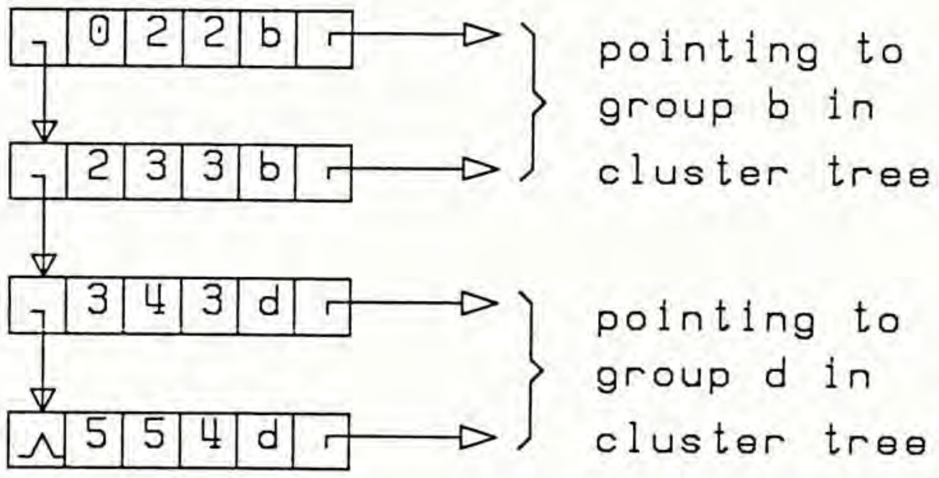
0 2 2 b
2 3 3 b
3 4 3 d
5 5 4 d
pointing to group b in cluster tree
pointing to group d in cluster tree

Figure 4-13 Map List of a Path Segment

where b and d are group numbers.

To link the segments, we have to join the segments and select the combination with the least cost. The cost is defined as Eq. 3-4 in section 3.2.3.2. An example on the linking of the path segments is illustrated in figure 4-14. All the segments are recorded in list form.

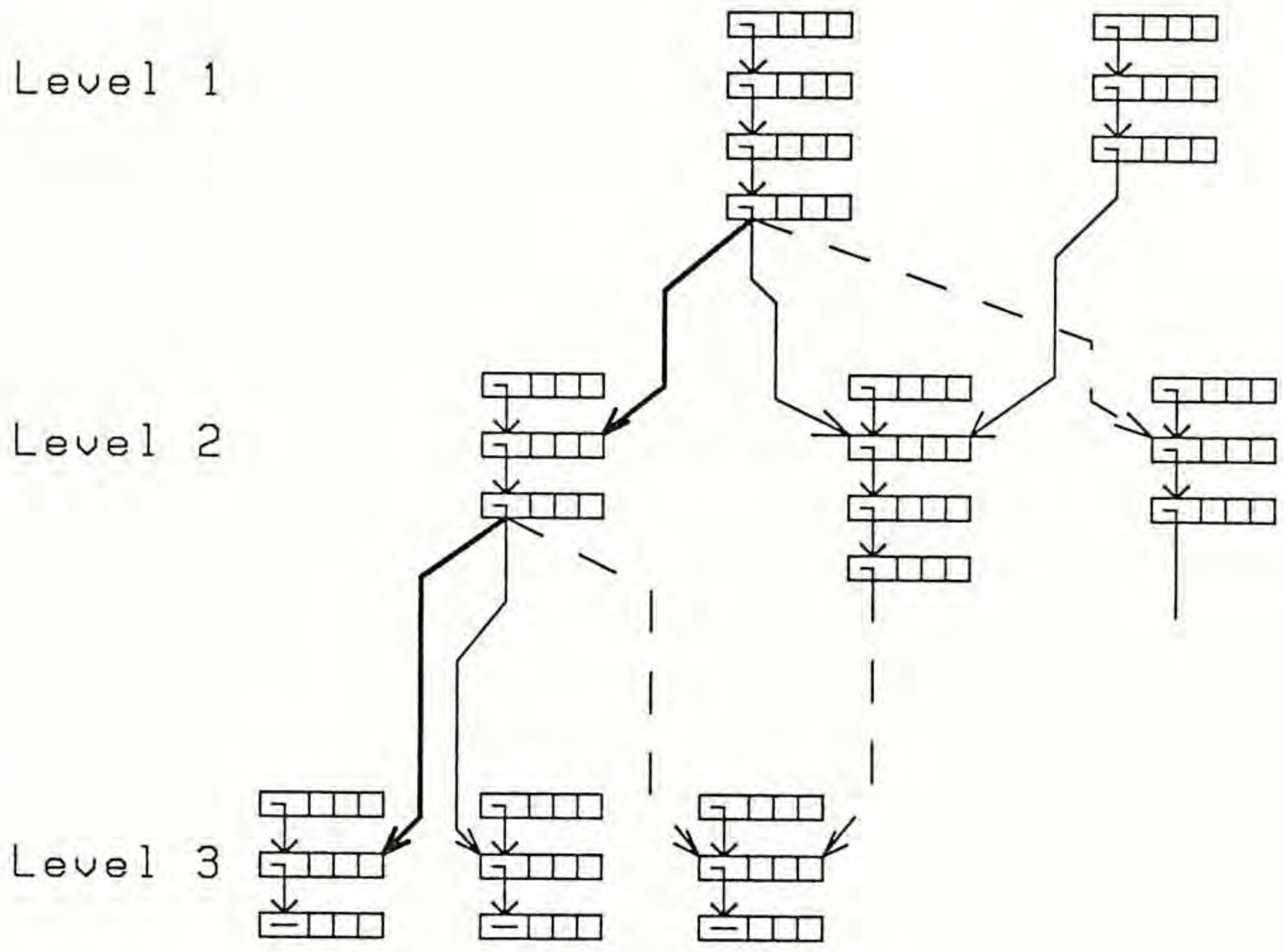

Figure 4-14 Dynamic Linking of Path Segments

In the above figure, each segment may link to more than one segments. The bold, solid and dashed lines distinguish among the linkages. The dynamic linking is by a top-down approach. Linking starts from level 1, i.e. segments oriented from the indicator cell, c. Once the best combination among two levels determined, it will propagate down to link the segments in the next level until the path reach the empty space, ξ. It is noticed that the end of each segment will point to the second member in the next segment. It is because the first member of a segment is duplicated with the last member in the previous segment, as shown in the segmentation of cross-cut (section 3.2.3.2.1). The "nil" pointer is represented by the symbol __.

4.2.3 Advantages of the Dynamic Data Structure

The linked list data structure is chosen for the following reasons:

Features of linked list:

1. Flexible (random access, insertion and deletion).
2. Fast access in information (e.g. x-y coordinates and connectivity attributes of a cell).
3. Easy implementation of critical path information (by fixing "pri" and "sec").
4. Substantial graph representation (compared with molecular and attractive force model).

Memory allocated:

minimized, especially compared with matrix/array representation.

Computation Time:

O(n)

compared with $O(n^2)$ in the matrix structure,

n is the number of cells in the circuit.

4.3 Data Manipulation and File Management

Since the computation on dynamic variables and management of files with this structure are relatively complicated as compared with those with formatted structure, we would like to describe some of the most important procedures and the layout of the programs.

4.3.1 The Connection Lists and the Group List

In the affinity clustering phase, information on the connections among nodes is stored in the connection lists. Clusters are formed by obtaining data on these lists. The procedure is illustrated in Figure 4-15. The directed connection list is originated from the connection array. Then, the undirected connection list and fan-out list is obtained from the directed connection list. Merging the fan-out list and the undirected connection list, the link-related list is formed. The group list is then extracted from the link-related list (Primary Grouping), and updated by the remained connection list (Element Appendage to Existing Groups). After final appendage (Loose Appendage of Ungrouped Elements and Single Element Group Formation), the adjusted group becomes completed group list. The completed group list can be further updated for adjustment of number of groups in the group list. This last procedure is optional.

The completed group list is constructed for the Alteration phase. Together with information in the new element array, computation goes to Element Assignment and finally Element Addition. The completed group list can also be used in the wiring optimization or critical path adjustment phase (Section 6).

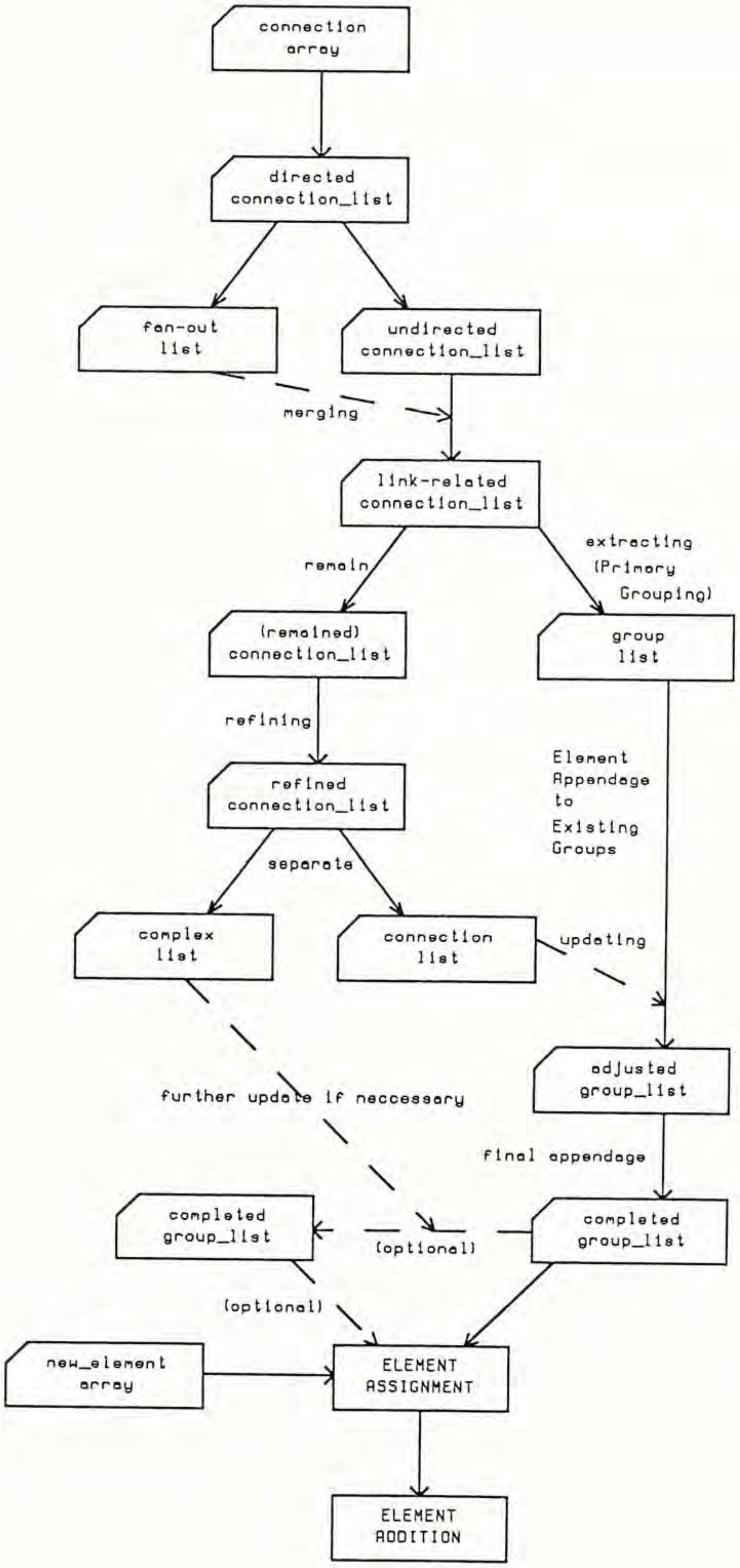

Figure 4-15 Manipulation on the Connection Lists and the Group List

4.3.2 Description on Programs and Data Files

There are 11 program files in the system. Their functions are described in the following list:

1.	fplace77.pas	- an initial placement phase based on simulated annealing algorithm. Details are discussed in Section 9: Appendix.
2.	fmolgrp7.pas	- the affinity clustering phase
3.	fmolasg7.pas	- the alteration phase
4.	smpladd7.pas	- a program on calculation of the total metal length on addition of new element without using the Incremental Alteration Placement Algorithm (IAPA)
* 5.	portion.pas	- program segment of the alteration phase.
* 6.	segments.pas	- library 1 and library 2 for the path segmentation procedure in the alteration phase.
7.	rst_base.pas	- concluding the placement result of test circuits
8.	grp_base.pas	- concluding the clustering result of test circuits
* 9.	conninit.pas	- connection data of test circuits
*10.	addlist.pas	- connection data of new elements to test circuits
11.	fnl_base.pas	- concluding the final results on both placement and clustering of the test circuits.

In the above list, files with * are program segments to be included. The following programs will include the files during compilation.

program	included file(s)
1. fplace77.pas	9
2. fmolgrp7.pas	9
3. fmolasg7.pas	5, 6, 9, 10
4. smpladd7.pas	9, 10

There are 8 types of data file in the system. They are classified as follows:

1.	xxxxrst7.dta	- ascii file on result from the initial placement in "fplace77.pas"
2.	xxxxbulk.dta	- record file on the attributes of groups (bulks) resulting from "fmolgrp7.pas"
3.	xxxxatom.dta	- record file on the attributes of elements (atoms) resulting from

"fmolgrp7.pas"

4. xxxxgrp7.dta - ascii file on group numbering resulting from "fmolgrp7.pas"
5. xxxxrst7.fnl - final result of placement from "fmolasg7.pas"
6. xxxxbulk.fnl - record file of final result of groups (bulks) resulting from "fmolasg7.pas"
7. xxxxatom.fnl - record file of final result of elements (atoms) resulting from "fmolasg7.pas"
8. xxxxgrp7.fnl - final result of group numbering from "fmolasg7.pas"

xxxx is the name of test circuit.

Layout of the programs on the affinity clustering phase and alteration phase is discussed in the following two sub-sections.

4.3.2.1 The Affinity Clustering Phase

Layout of the "fmolgrp7.pas" program is shown in figure 4-16. There are 26 subroutines. Subroutines under a tree is procedures or functions to be called by the parent. The following is a brief description on the subroutines. Connectivity attributes refers to "pri", "sec", "pc", and "sc" (details in Table 4-2).

1. convert : procedure; convert the directed connection array into an undirected connection array.
2. array_assign : procedure; assign the connections into a group array. This subroutine is for manual data check.
3. list_assign : procedure; assign data of the undirected connection array into the undirected connection list.
4. determ : procedure; to obtain the fan-out connection list from the directed connection list.
5. merge : procedure; merge the undirected connection list and fan-out connection list to form the link-related connection list.
6. fill_in_set : procedure; fill in element to the group set which contains element in the group list.
7. separate : procedure; to separate the grouped elements and ungrouped elements in the undirected connection list.
8. plant_l_root : procedure; initializes the root of a connection list with the list pointer variable "lptr".
9. constructive_build : procedure; to build the group list from the link-related

connection list. The group list is constructed with header indicating size of each group in the list and element variable with connectivity attributes.

10. locate_and_link : procedure; to find the location in the group list on which an element should be placed and to link the element in the location.
11. cmpd_adhesion : procedure; to adhere an element to the group list and create the connectivity attributes.
12. cmpd_activate : procedure; to activate a new element variable and set the corresponding x-y coordinates in the record.
13. affinity_group : procedure; this subroutine corresponds to the "Element Appendage to Existing Groups" in the algorithm (Table 3-4). It appends an element to the group list by measuring its belong tendencies to each group and justifying the values.
14. cmpd_append : procedure; it appends an element to the an existing group and update the connectivity attributes. (c.f. cmpd_adhesion)
15. loose_group : procedure; this subroutine corresponds to the step "Loose Appendage of Ungrouped Elements" in the algorithm (Table 3-5). It groups elements with loose connectivity to other elements (no. of connection = 1).
16. loose_attachment : procedure; this subroutine corresponds to the step "Single Element Group Formation" in the algorithm (Table 3-6). It forms "single element groups" on elements which do not belong to any group.

4.3.2.2 The Alteration Phase

Layout of the "fmolasg7.pas" program is shown in figure 4-17. There are 26 subroutines. Subroutines under a tree is procedures or functions to be called by the parent. The following is a brief description on the subroutines. Connectivity attributes refers to "pri", "sec", "pc", and "sc" (details in Table 4-3).

1. build_tree : procedure; builds the cluster tree from the group list read from the "xxxxbulk.dta" and "xxxxatom.dta" files. It constructs the skeleton of the tree as shown in figure 4-9, i.e. "pri" and "sec" pointers not yet linked. The linking of these pointers is done in the following subroutine.
2. link_tree : procedure; links the "pri" and "sec" pointers of each element in the cluster tree. It obtains the integer values of "pri" and "sec" in the group

list and links the corresponding elements, with assistance from the following subroutine.

3. search_for_link : procedure; obtains the instance no. of an instance and then searches for its location in the cluster tree.
4. fill_in_set : procedure; fills in elements to the group set which contains elements in the group list.
5. list_assign : procedure; assign data of the undirected connection array into the undirected connection list.
6. element_assign : procedure; determines to which group should an added element belongs to. The measure is by belong tendency. The connectivity attributes are also determined.
7. element_append : procedure; appends an element to the cluster tree, linking the "pri" and "sec" pointers and update "pc" and "sc" values as well.
8. init_brd_setting : procedure; initializes the values of a 7X7 boolean chessboard.
9. board_dbase : procedure; fills in the data in the boolean chessboard.
10. find_cneigh : procedure; to find an empty space for new element addition at the contacting neighbourhood of a group. A contacting neighbour of (x,y) is defined as one of the four adjacent neighbours to (x,y).
11. find_pneigh : procedure; to find an empty space for new element addition at the position neighbourhood of a group. A position neighbour of (x,y) is defined as one of the four corner neighbours to (x,y).
12. fcfs_element_linking : procedure; to place a new element to the first empty found. fcfs means First-Come-First-Serve.
13. location_assign : procedure; assigns the coordinates to an element in the cluster tree.
14. prelim_path_determine : procedure; determines the shortest path(s) joining a group and the nearest empty space(s) found (figure 3-4).
15. map_list_filling : procedure; fills data into a map list to represent the path determined in "cross_cut".
16. cross_cut : procedure; determines the shortest path joining the indicator cell and an empty space, epsilon (ep in short). Algorithm in Section 3.2.3.1.
17. update_distance : procedure; updates the "dist" values of a map list (Table 4-6).
18. present_conn_len : function; calculates the interconnection length of an element to the two mostly connected cells in the circuit by accessing connectivity attributes in the element's record (tree element variable in Table 4-3).

A. detail_element_linking : procedure; to find the empty space with the least cost to fill the added element and to place the element in the space.

19. total_length : function; to compute the total increased connection length due to the addition of the new element.

20. dynamic_conn_len : function; to calculate the interconnection length of an element to the two mostly connected cells in the circuit if it is placed at an assigned location (x,y).

B. path_segmentation : procedure; to find the optimal path joining the indicator cell and the empty space, epsilon by segmenting the cross-cut and then linking by Dynamic Programming approach. Algorithm in Section 3.2.3.2.

21. regular_path_determination : procedure; determine the path segments as shown in figure 3-9. The path templates is created in the following library, Library 1.

22. Library 1 : a library of subroutines (path templates in figure 3-9) called by subroutine 21: regular_path_determination. The library is summarized in the following table:

s_0_a	s_1_a	s_2_a	s_3_a	4_5_common_s
s_0_b	s_1_mid	s_2_b	s_3_b	[s_linear]
s_0_c	s_1_c	s_2_c	s_3_c	
			s_3_d	

```
s for "segment";  integer for pattern no.;
a,b,c,d corresponds to path 1,2,3,4;
s_1_mid for pattern 1,2,3 of pattern 1 template (figure 3-9);
s_linear is a subroutine called by 4_5_common_s.
```

Table 4-7. Path Template Library of regular_path_determination

23. map_root_plant : procedure; called by both Library 1 and Library 2. It creates the root of a map list (variable type in Table 4-6).

24. l_linking : procedure; called by both Library 1 and Library 2. It links variable to a map list (Table 4-6).

25. remained_path_determination : procedure; determine the path segments as shown in figure 3-10. The path templates is

created in the following library, Library 2.

26. Library 2 : a library of subroutines (path templates in figure 3-10) called by subroutine 25: remained_path_determination. The library is summarized in the following table:

sh_0	sh_1	2_3_common_sh	[s_linear]

sh for "short_segment"; integer for pattern no.;
s_linear is a subroutine called by 2_3_common_s.

Table 4-8. Path Template Library of remained_path_determination

27. attached_path_determination : procedure; determines the path segments with Manhattan Distance of 1 (case discussed in Section 3.2.3.2.2).
28. segment_concatenation : procedure; links the path segments obtained in the above procedures. The mechanism is shown in figure 4-14.
29. gp_cost_calculate : function; calculates the cost of two path segments if they are joined. The cost function is defined in Eq. 3-4 in Section 3.2.3.2.
30. accumulation : procedure; accumulates the group count number of a path segment. It is a subsidiary routine to the "gp_cost_calculate" function.
31. concatenate : procedure; to concatenate two path segments. It is called as the best combination of two path segments is determined.

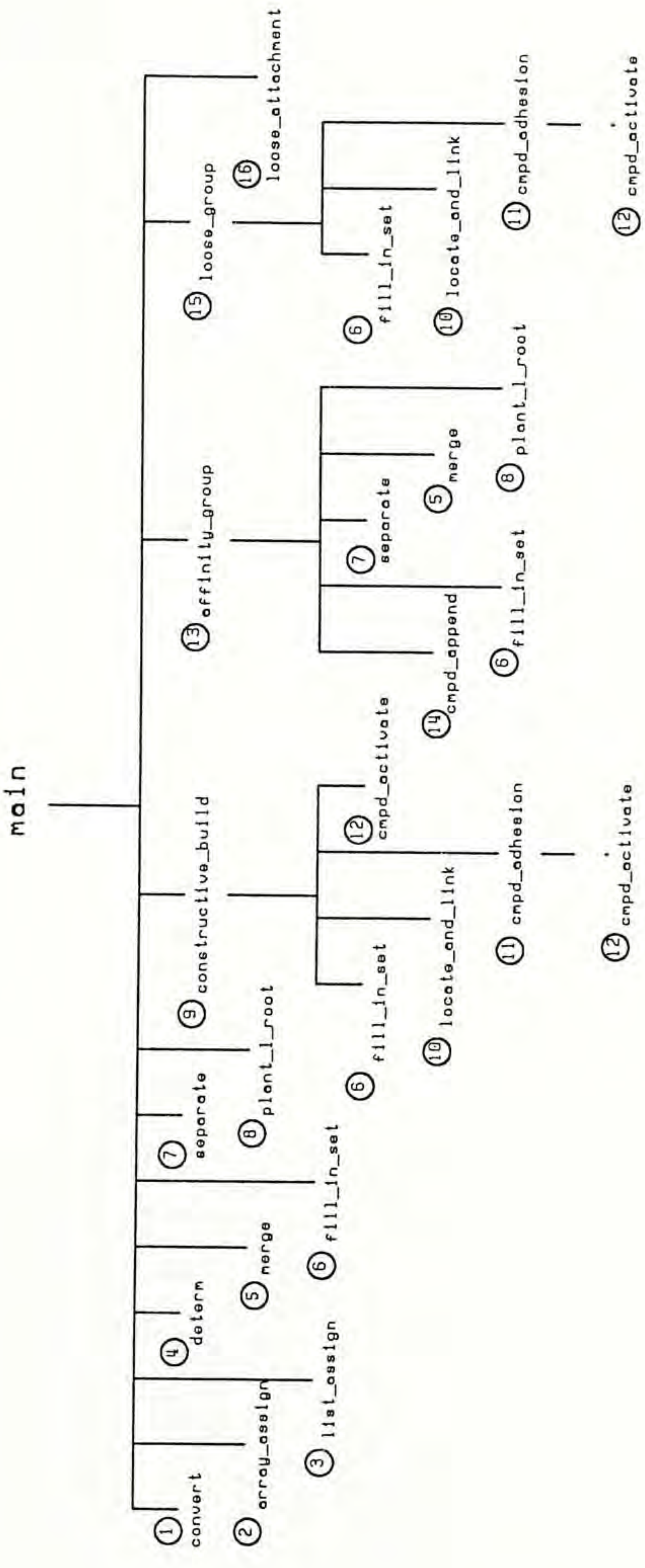

Figure 4-16 Layout of the "fmolgrp7.pas" File

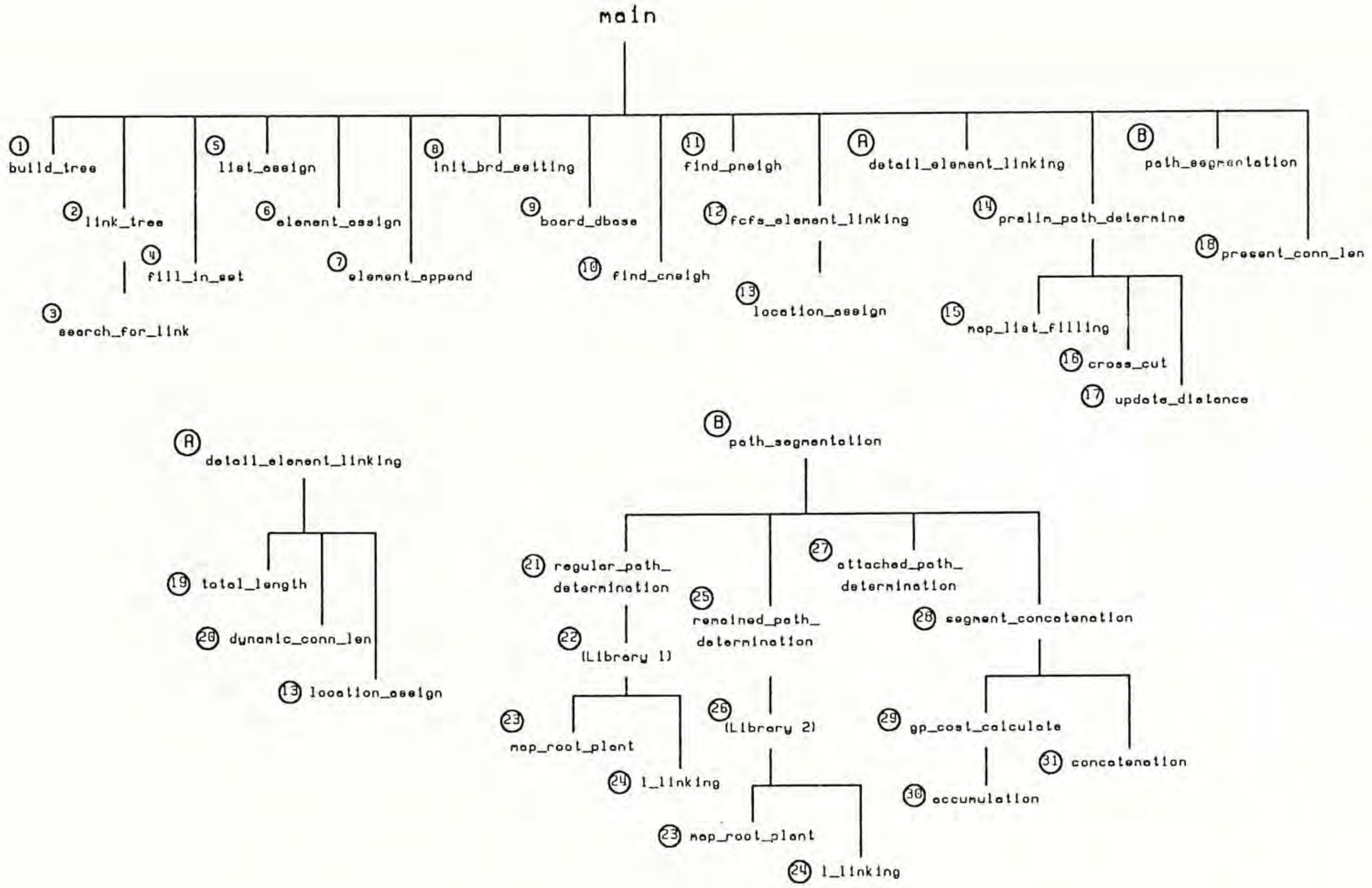

Figure 4-17 Layout of the "fmolasg7.pas" File

5. Results

Eight test circuits were used for evaluation. They are circuits of different characteristics so that the result obtained can be more generalized. Some of them are random logic circuits. Some of them are synchronous processing units. They are summarized as follows (the cell counts are shown in parentheses):

Ckt. 1 : A 4-bit synchronous counter and a 7-bit ring register (18 cells)
Ckt. 2 : A 8-bit serial data sequencer (46 cells)
Ckt. 3 : "Dividen", a module in a MPU circuit and "ICTckt", a module in a ICT (Integer Cosine Transform) archiving system (47 cells)
Ckt. 4 : Low order serial data synchronizer/desynchronizer (44 cells)
Ckt. 5 : High order serial data synchronizer/desynchronizer (46 cells)
Ckt. 6 : A 4-bit binary full adder with fast carry (42 cells) [Texas87]
Ckt. 7 : A 9-bit odd/even parity generators/checkers (48 cells) [Texas87]
Ckt. 8 : A 9-bit odd/even parity generators/checkers, with different numbering on instances (48 cells) [Texas87]

Ckt. 7 and Ckt. 8 are identical except the numbering on instances. This is to test the validation of the programs on different arrangement of the same circuitry. The circuit diagrams of Ckt. 1 to Ckt. 8 are shown in figure 5-1 to 5-13 respectively.

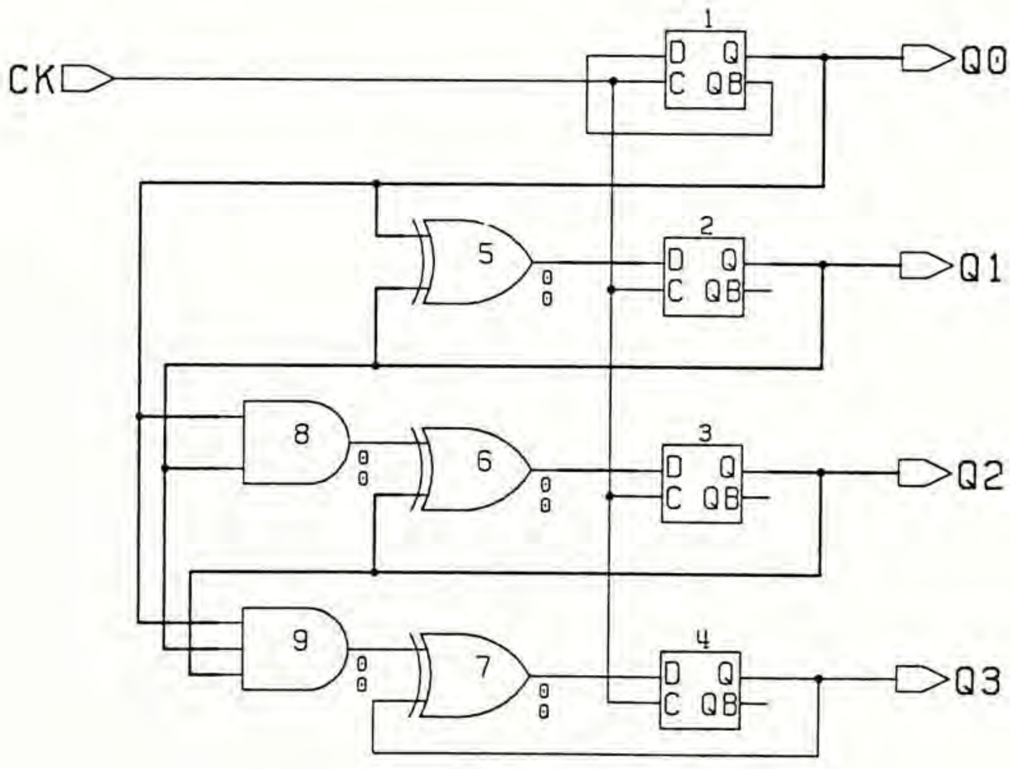

Figure 5-1 Ckt. 1: A 4-bit synchronous counter

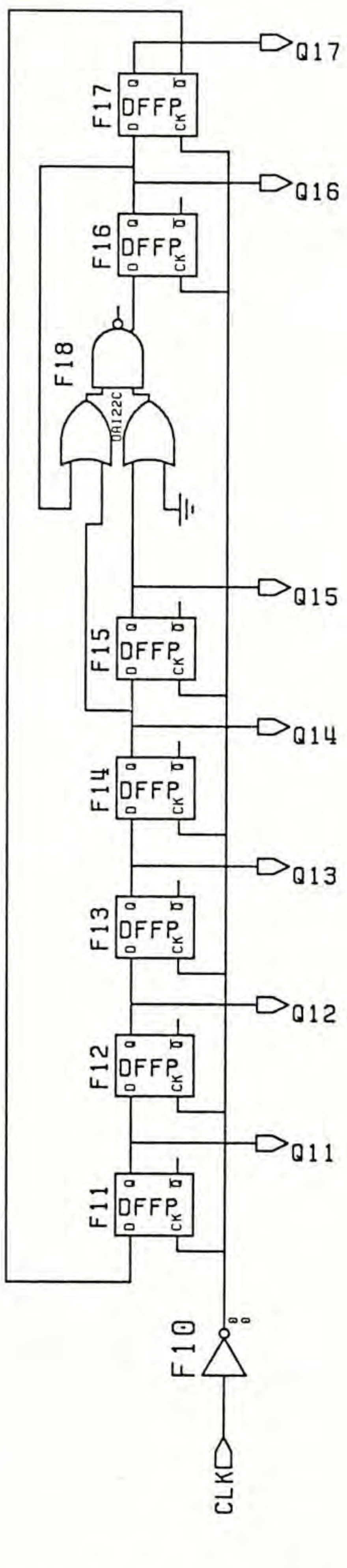

Figure 5-2 Ckt. 1: A 7-bit ring register

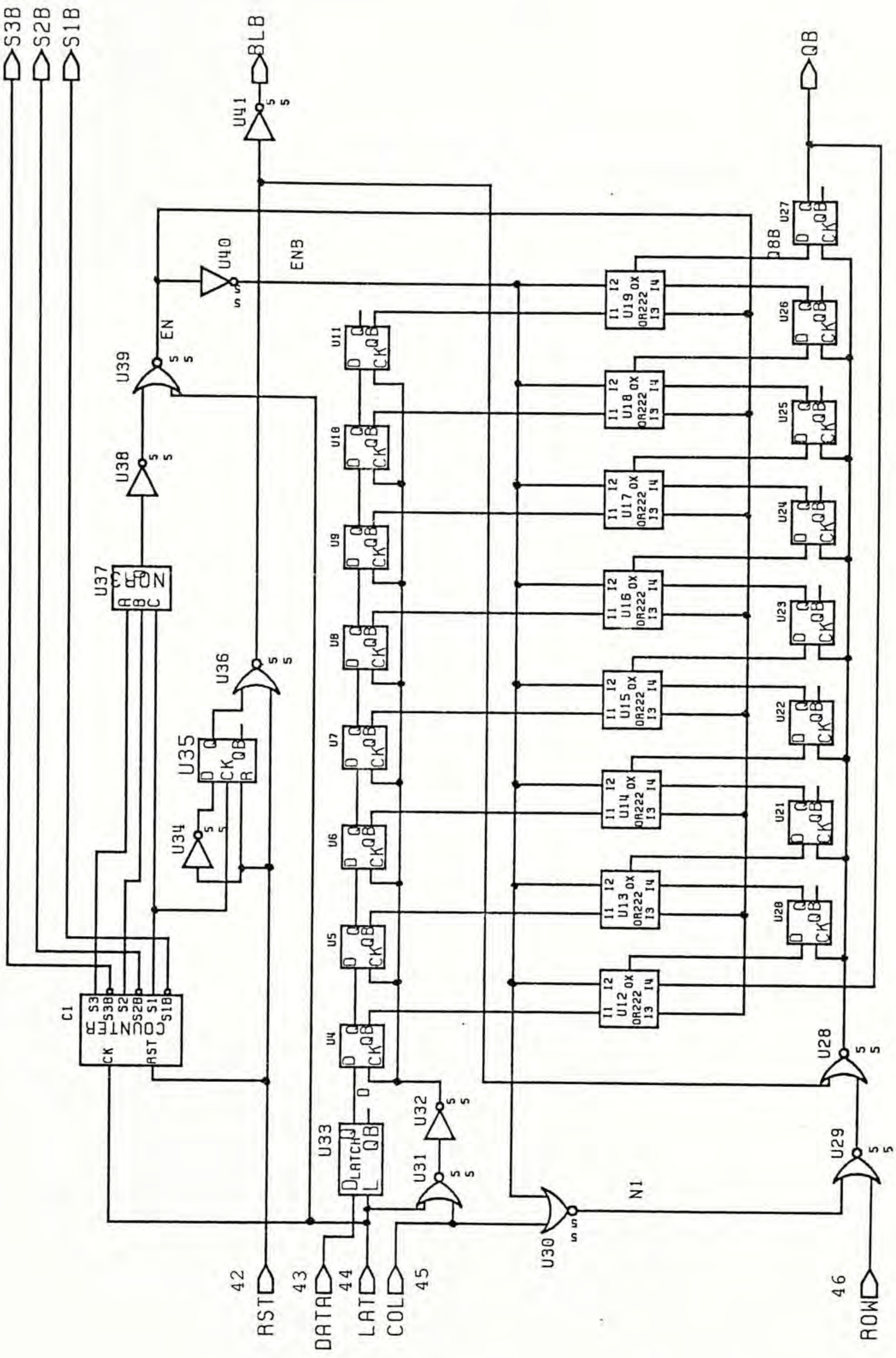

Figure 5-3 Ckt. 2: A 8-bit serial data sequencer

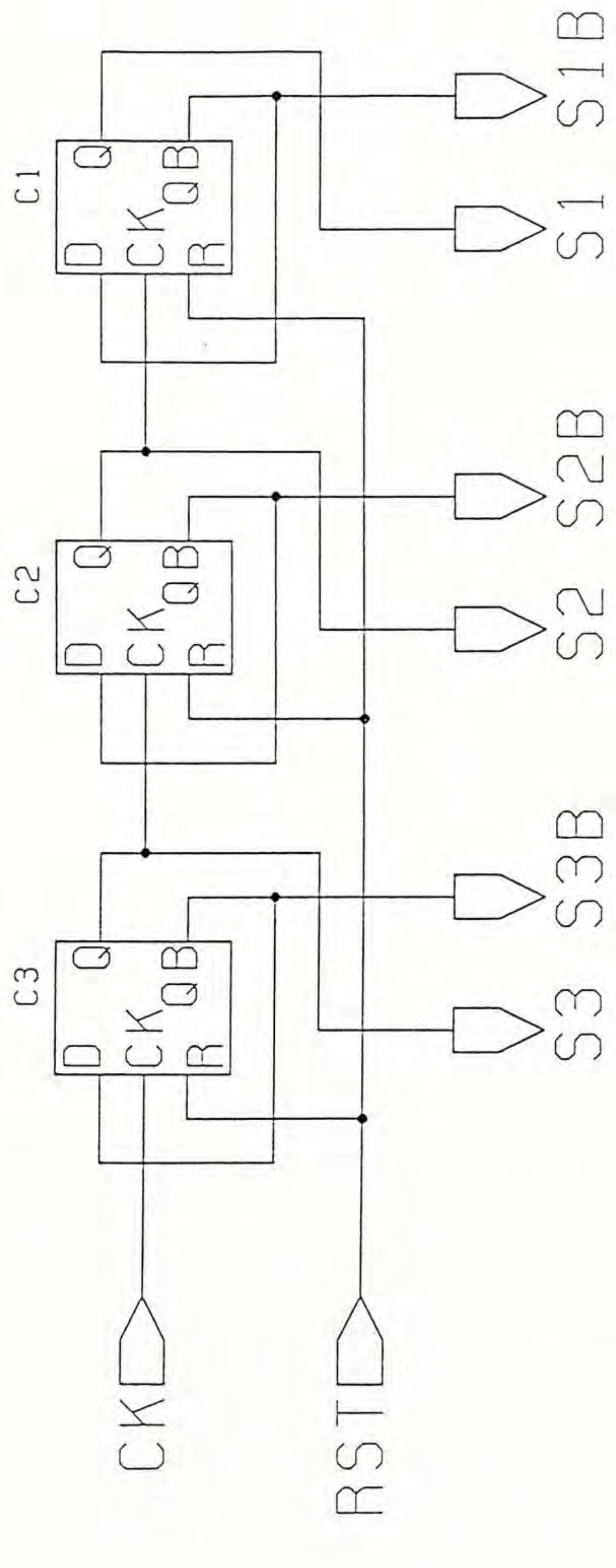

Figure 5-3 (a) Ckt. 2: "COUNTER" in ckt. 2

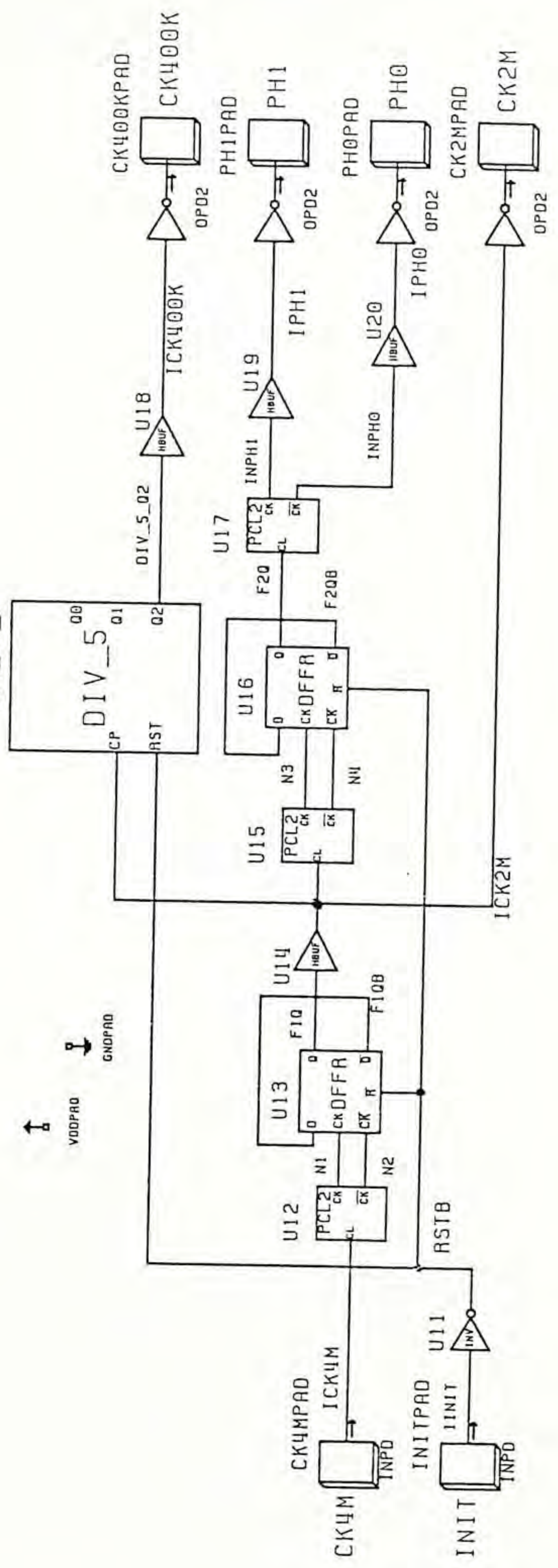

Figure 5-4 Ckt. 3: "Dividen", a module in a MPU circuit

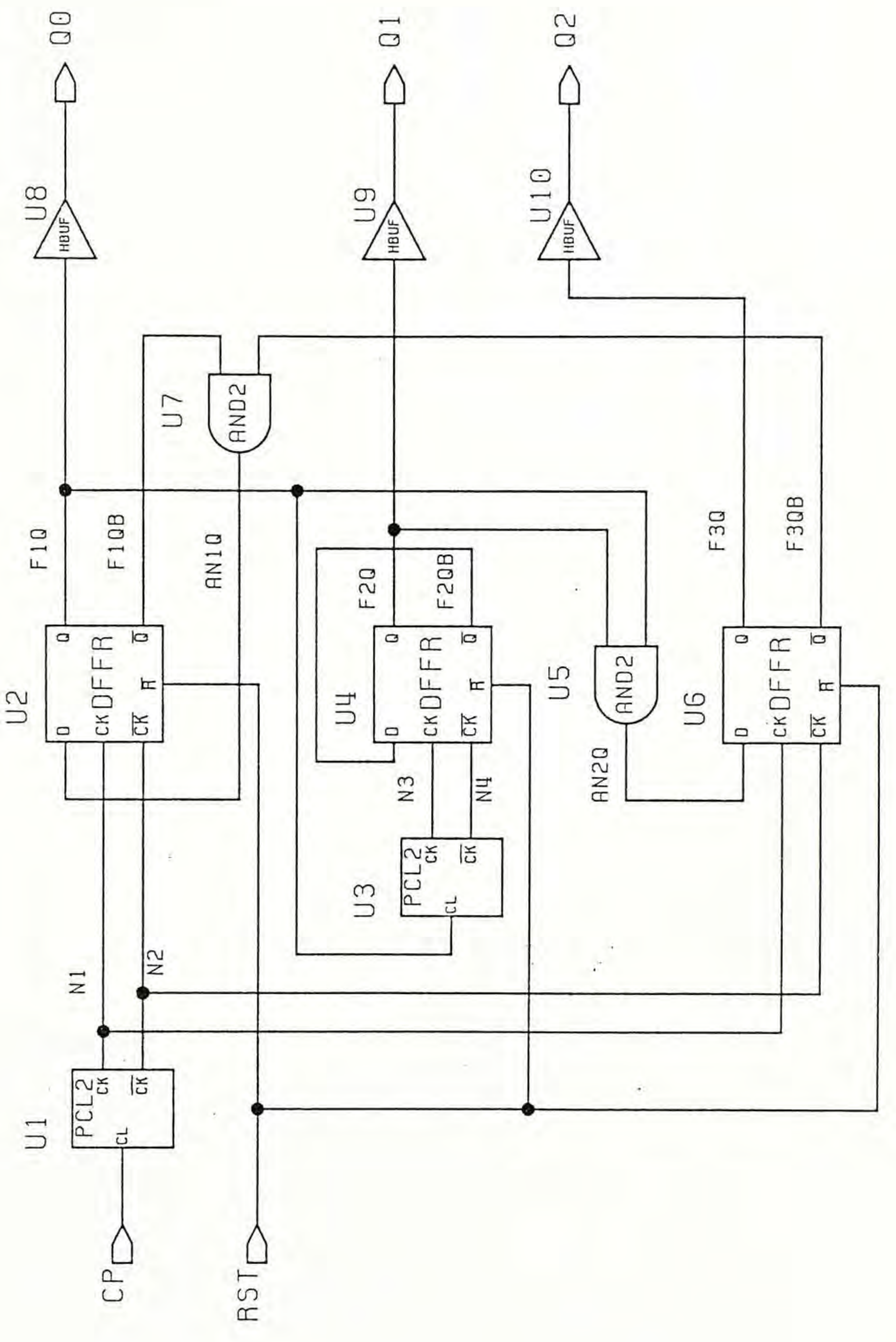

Figure 5-5 Ckt. 3: "Div_5", a module in "Dividen"

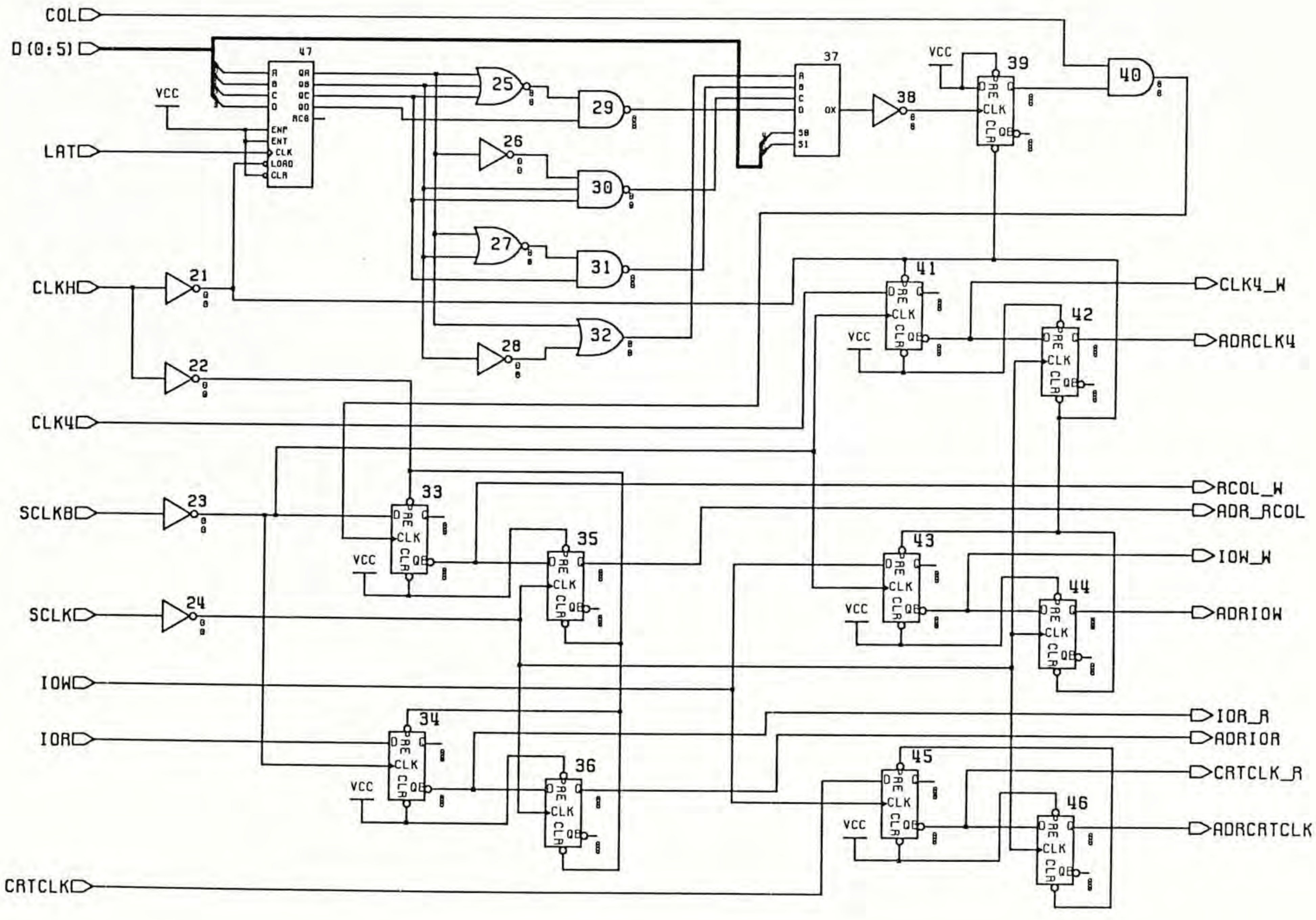

Figure 5-6 Ckt. 3: "ICTckt", a module in a ICT archiving system

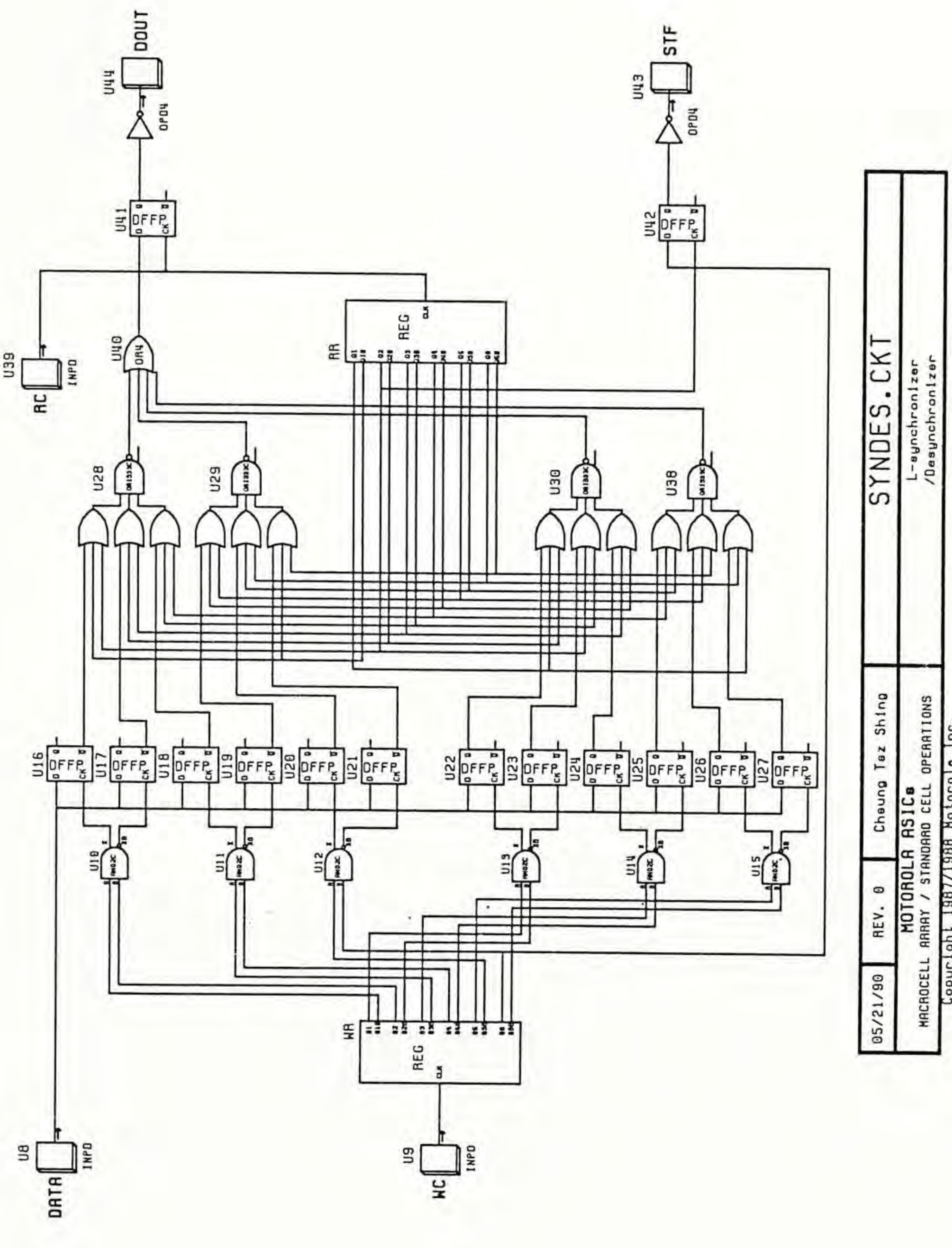

Figure 5-7 Ckt. 4: Low order serial data sychronizer/desynchronizer

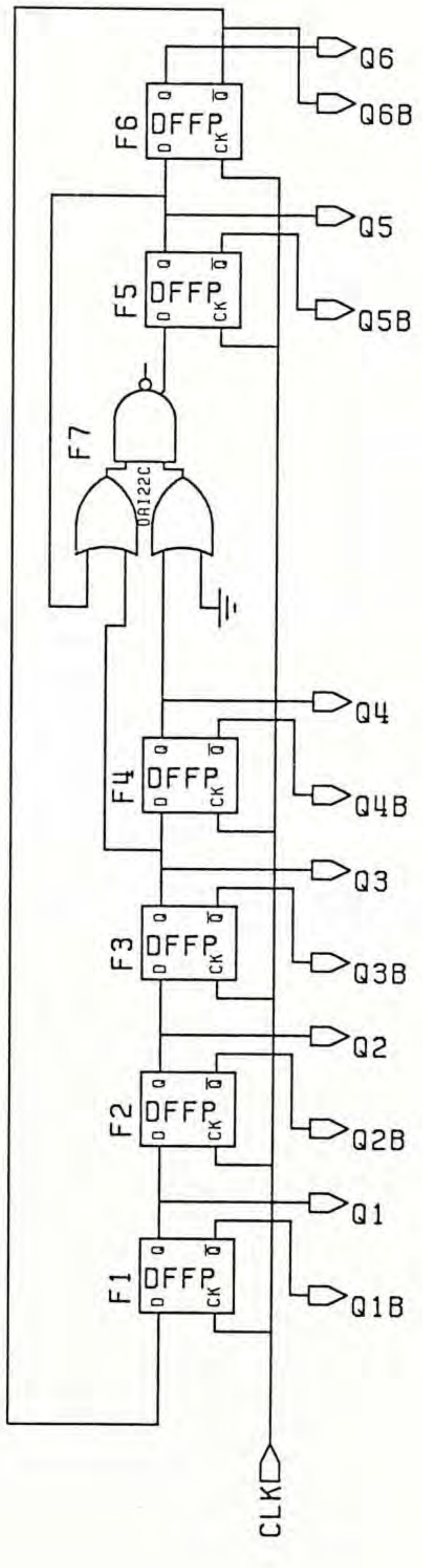

Figure 5-8 Ckt. 4: "REG" in ckt. 4

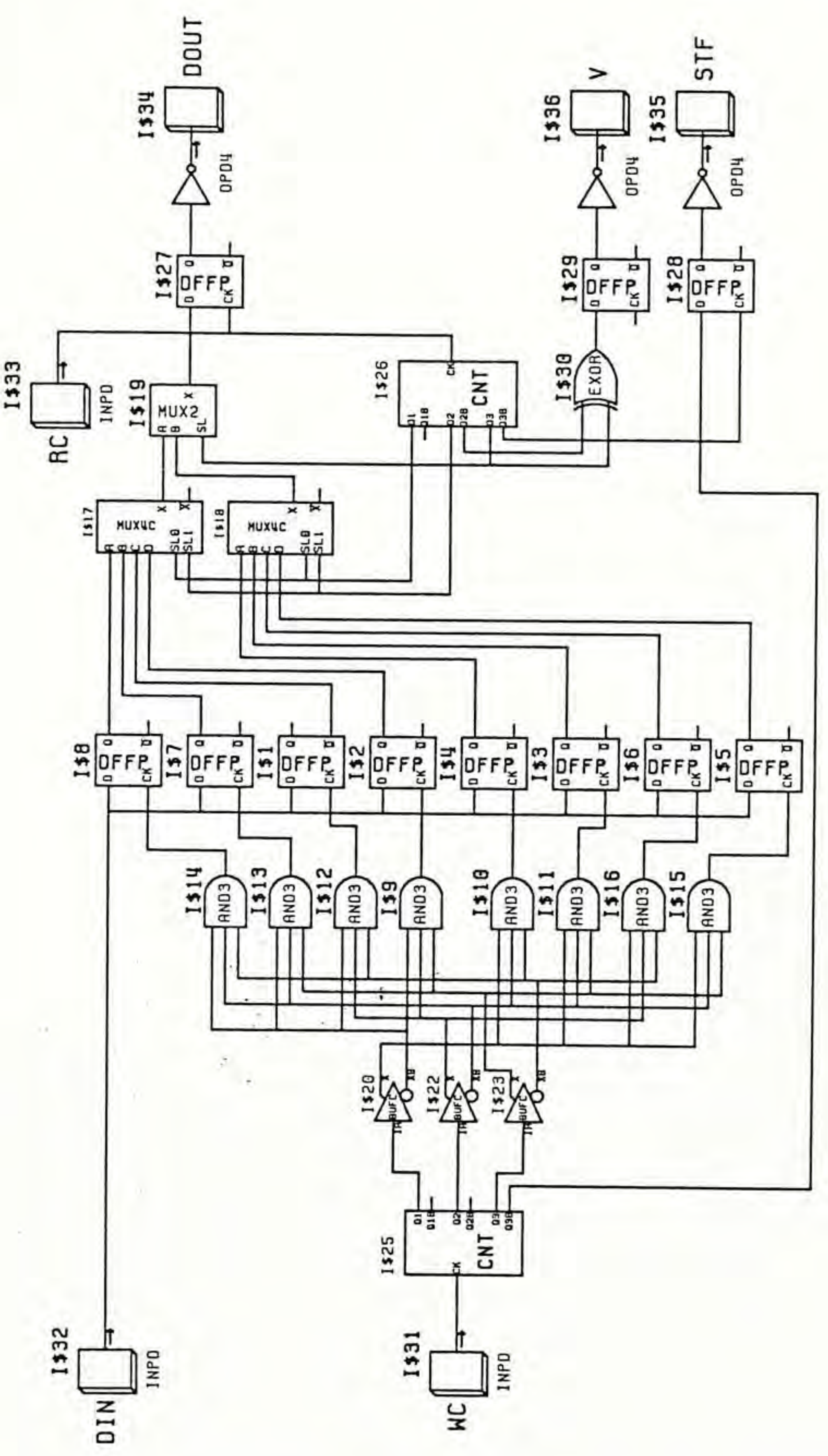

Figure 5-9 Ckt. 5: High order serial data sychronizer/desynchronizer

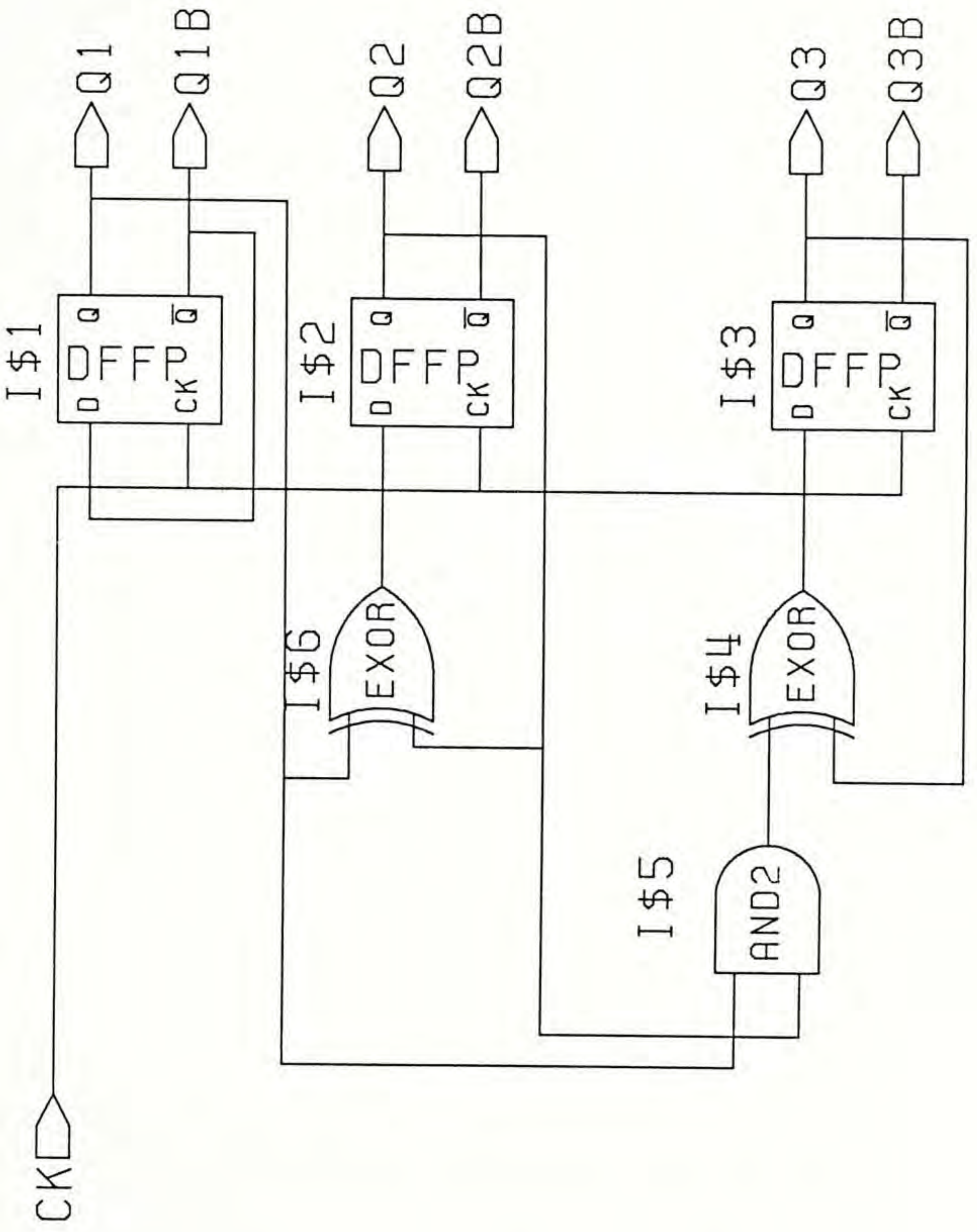

Figure 5-10 Ckt. 5: "CNT" in ckt. 5

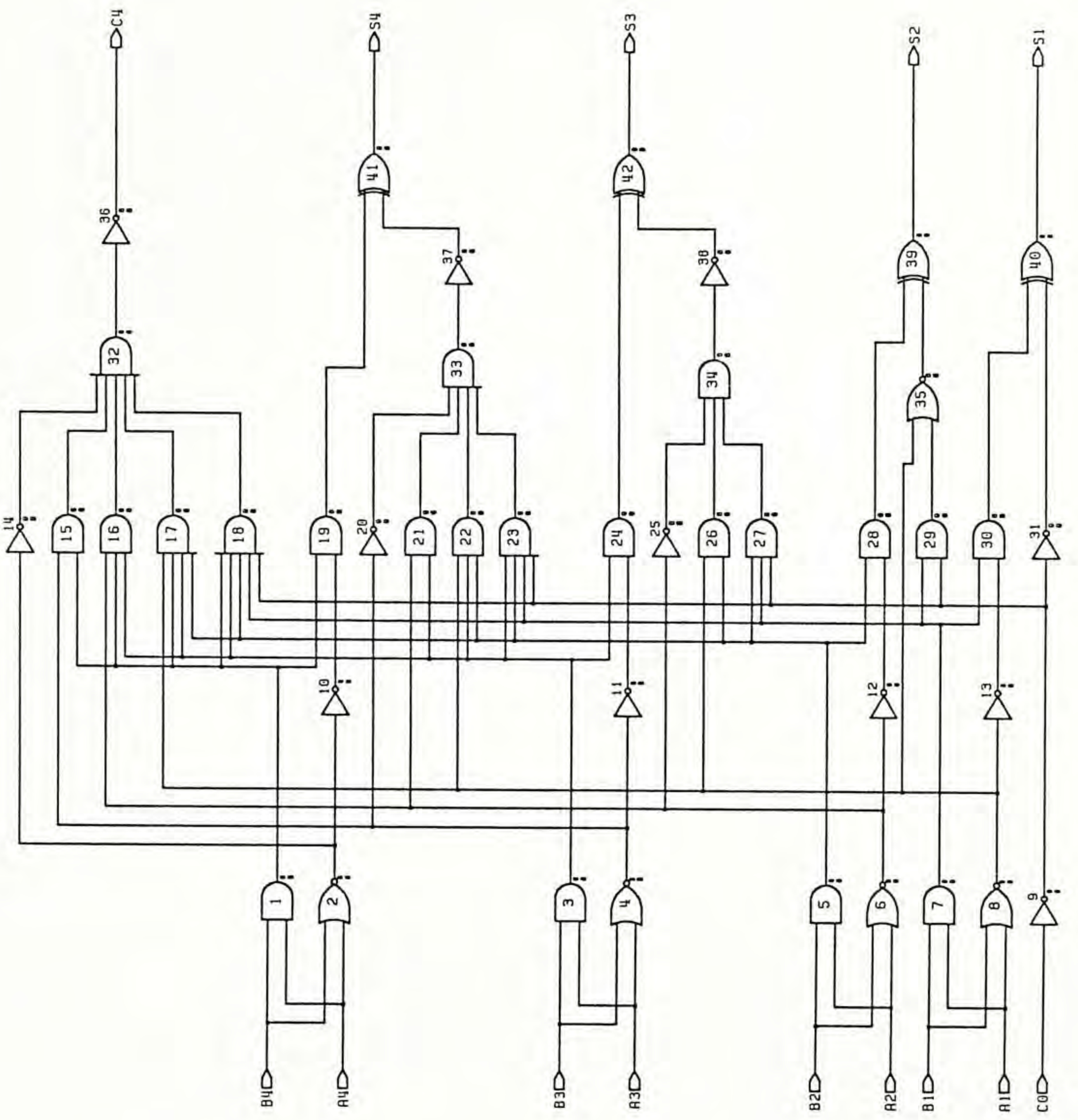

Figure 5-11 Ckt. 6: A 4-bit binary full adder with fast carry

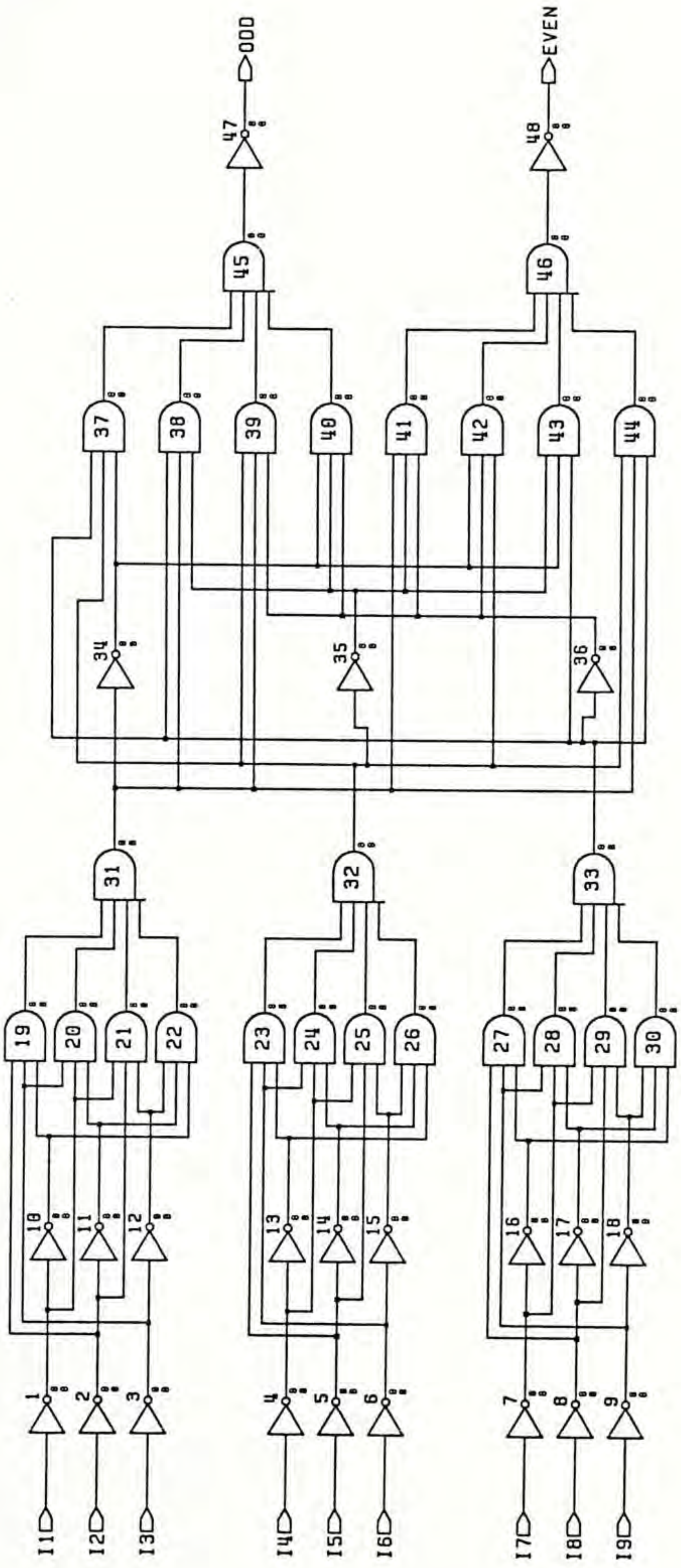

Figure 5-12 Ckt. 7: A 9-bit odd/even parity generators/checkers

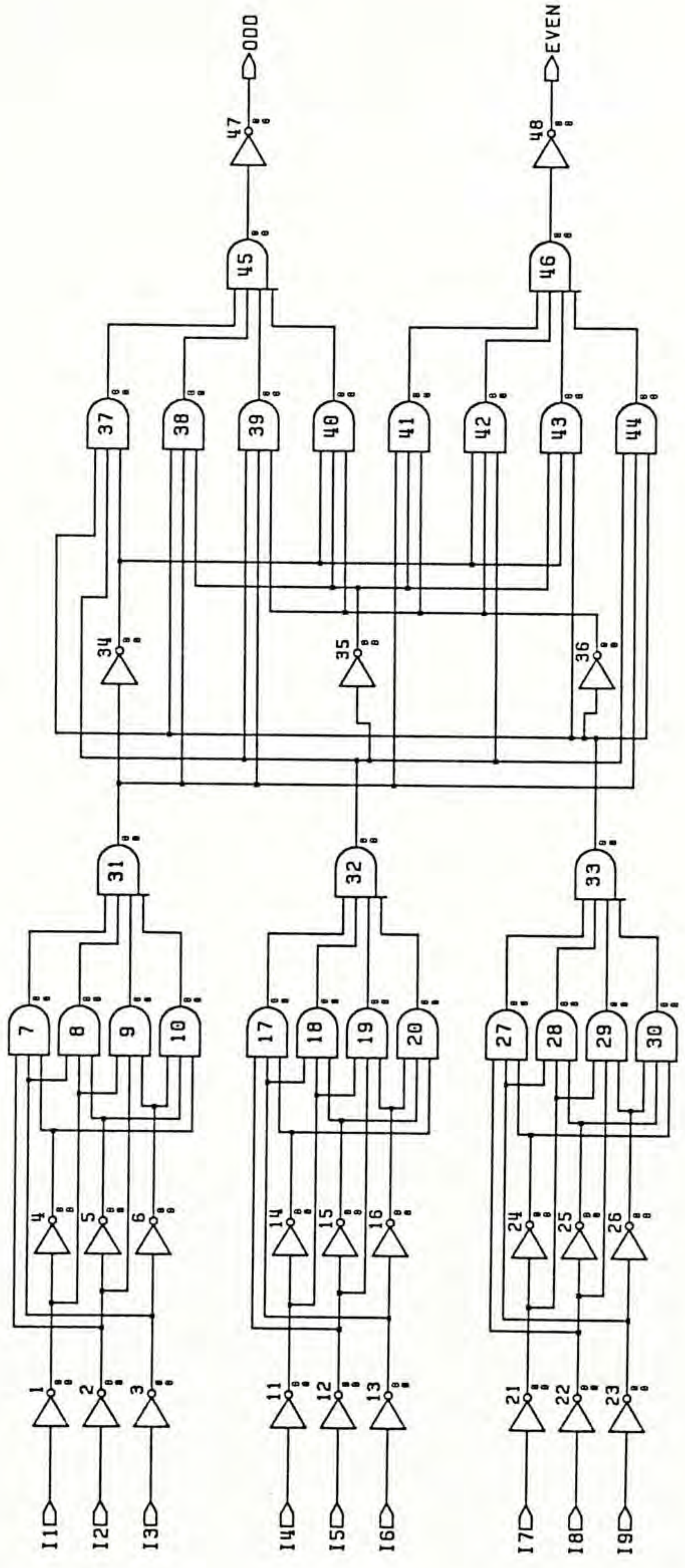

Figure 5-13 Ckt. 8: A 9-bit odd/even parity generators/checkers (different numbering)

5.1 Results on Affinity Clustering Phase

The results obtained after the affinity clustering phase are shown in figure 5-14 and 5-15. Matrices on the left are the placements of the instances, indicated by the numbers, resulting from a simulated annealing placement program. Matrices on the right show the location of individual groups by their respective group numbers. A zero group number indicates an empty space.

The group lists, together with the connectivity attributes of elements, of the ckt. 1, 3, 4, 6, 7, and 8 are shown in figure 5-16 to 5-21 (the group lists and details of affinity clustering procedure of ckt. 2 and ckt. 5 will be discussed in the Section 5.2). In the group list, "n" is the number of elements in the group (group size); "i" indicates element instance "inst"; "p" and "s" corresponds to the "pri" and "sec" attributes respectively. Integers in front of the slash "/" are the connected element instance while that behind are the connectivity "con" (format at Table 4-2).

It is worth mentioning that the results of clustering on ckt. 7 and ckt. 8 are the same (refer to Figure 5-20 and Figure 5-21). It proves the validation of the algorithm on different ordering of the instances.

```
Program start.
Circuit 1:
            4  7 11 17 10 12 15        3  3  5  5  4  5  5
            5  2  0 16 18 13 14        1  1  0  5  5  5  5
            1  9  8  3  6  0  0        1  1  1  2  2  0  0
            0  0  0  0  0  0  0        0  0  0  0  0  0  0
            0  0  0  0  0  0  0        0  0  0  0  0  0  0
            0  0  0  0  0  0  0        0  0  0  0  0  0  0
            0  0  0  0  0  0  0        0  0  0  0  0  0  0    RETURN:
Circuit 2:
           10  6 11 14 21 26 27        2  2  2  3  4  7  7
            9 17  4 33 18 19 43        2  3  2  1  3  3 11
            8 32 22  5 13 25 28        2  9  5  2  3  6  1
            0  2  7 15 40 12 20        0  1  2  3 10  3  4
            1 38 39 36 30 23 29        1  1  1  1  8  5  8
           37  3 42 44 46  0 24        1  1  1  1 13  0  6
           34 35 41 31 45 16  0        1  1  1  1 12  3  0    RETURN:
Circuit 3:
            9  2  3  5  6 12  7        2  1  2  1  1  3  1
            6  1 11  4 10 13 14        1  1  1  2  1  3  1
           15 16 17 20 18 19 25        4  4  4  6  5  6  7
           38 31 24 35 36 27 47       11  7  6  6  6  7  7
           39 21 44 33 34 42 41        7  7 12  8  8 12 12
           32 37 30 23 22 40 28       10  9  9  8  8  8  7
           26 46 45 43  0 26  0        7  8  8 12  0  7  0    RETURN:


Circuit 4:
           17 18  3  2  1  9  7        5  6  1  1  1  2  1
           14 10 11 26 15 19  4        3  1  3 10  4  6  1
           16  0 24 13 23  6  5        5  0  9  1  8  1  4
           26 42 38 25  8 21 12       11  4 12  9  5  7  4
           33 37 35 41 29 44 20       11 11 12 11 12 14  7
           30 32 22 31 36  0  0       11 11  8 12 12  0  0
           34 40  0 43 39 27  0       11 11  0 13 11 10  0    RETURN:
Circuit 5:
           27 26  2 40 11  9  0        6  6  2 10  1  1  0
            3  5 24  6 15 20  0        2  2  5  2  1  3  0
            0 16  4 18 22 21 34        0  2  2  1  4  3  4
            8  1  7 12 14 10 33        2  2  2  1  1  1  4
           28 19 17 46 13 32 35        4  2  2  2  1  3  3
           38 43 42 41 25 31 23        8 11  2  2  2  3  3
           39 44 45 30 37 36 29        9 11  2  2  7  3  7    RETURN:
Circuit 6:
            9 29 35  8 28 39  0        9  7 16  8  5 16  0
           40 34 27  5 26 18 14       15 14 11  5 11 11 10
            0 23  7 17  4 38 32        0 11  7 11  4 17 10
           36 30 25 22 20  1  2       19 15 14 11 13  1  2
           31  6 33  3 42 15 37        9  6 13  3 19  1 18
           12 21 13 16 24  0 19        6 11  6 11  3  0 12
            0  0  0  0 11 10 41        0  0  0  0  4  2 12    RETURN:
```

Figure 5-14 Initial Placement and Affinity Clustering Results of Ckt. 1 to 6

```
Circuit 7:
          9 18 30 28 13 48  7       9  9 15 14  4 19  7
         27  8 23  4  1 16 10      14  8 12  4  1  7  1
         25 17 29  6 21  2 11      13  8 15  6 11  2  2
         15  5 32 45  3 19 24       6  5 16 16  3 10 12
         47 37 20 12 34 14 22      18 16 10  3 16  5 11
         26 33 42 40 36 41 46      13 16 16 17 16 17 16
         35 44 38 31 43 39  0      16 16 17 16 16 17  0      RETURN:
Circuit 8:
          3  4 10  5  6  9 25       3  1  5  2  3  5 12
         11  1 19  2 15 18 22       6  1 10  2  7  9 12
         12 37 14 20 16 26 28       7 16  6 10  8 13 14
          8 17 39 32 13 23 29       4  9 17 16  8 13 15
          7 43 31 27 34 36 30       4 16 16 14 16 16 15
         35 33 47 40 44 42 21      16 16 18 17 16 16 11
         38 41 45 46 24 48  0      17 17 16 16 11 19  0      RETURN:
```

Figure 5-15 Initial Placement and Affinity Clustering Results of Ckt. 7 and 8

```
Single Element Groups Formation:
Group 1 n=5 i: 1 2 5 8 9
Group 2 n=2 i: 3 6
Group 3 n=2 i: 4 7
Group 4 n=1 i: 10
Group 5 n=8 i: 11 12 13 14 15 16 17 18
Press RETURN if finished investigation:
```

```
Group List with connectivity attributes:
G=1 n:5 i:1 p 5/1 s 8/1 i:2 p 5/2 i:5 p 2/2 s 8/2 i:8 p 5/2 i:9 p 5/2
G=2 n:2 i:3 p 6/2 i:6 p 3/2
G=3 n:2 i:4 p 7/2 i:7 p 4/2
G=4 n:1 i:10 p -1/0
G=5 n:8 i:11 p 12/2 s 17/2 i:12 p 11/2 s 13/2 i:13 p 12/2 s 14/2 i:14 p 13/2 s
15/2 i:15 p 14/2 s 18/2 i:16 p 17/2 i:17 p 11/2 s 16/2 i:18 p 15/2
PRESS RETURN:
```

Figure 5-16 Group List of Ckt. 1

```
 Single Element Groups Formation:
 Group 1 n=9 i: 1 2 5 6 7 8 10 11 14
 Group 2 n=3 i: 3 4 9
 Group 3 n=2 i: 12 13
 Group 4 n=3 i: 15 16 17
 Group 5 n=1 i: 18
 Group 6 n=2 i: 19 20
 Group 7 n=9 i: 21 25 26 27 28 29 31 39 47
 Group 8 n=10 i: 22 23 24 33 34 35 36 40 45 46
 Group 9 n=2 i: 30 37
 Group 10 n=1 i: 32
 Group 11 n=1 i: 38
 Group 12 n=4 i: 41 42 43 44
 Press RETURN if finished investigation:
```

```
 Group List with connectivity attributes:
 G=1 n:9 i:1 p 2/2 s 6/2 i:2 p 1/2 s 5/1 i:5 p 2/1 s 6/1 i:6 p 1/2 s 10/1 i:7 p
2/1 s 6/1 i:8 p 2/1 i:10 p 6/1 i:11 p 2/1 s 6/1 i:14 p 1/1
 G=2 n:3 i:3 p 4/2 i:4 p 3/2 s 9/1 i:9 p 4/1
 G=3 n:2 i:12 p 13/2 i:13 p 12/2
 G=4 n:3 i:15 p 16/2 i:16 p 15/2 s 17/1 i:17 p 16/1
 G=5 n:1 i:18 p -1/0
 G=6 n:2 i:19 p 20/1 i:20 p 19/1
 G=7 n:9 i:21 p 47/1 i:25 p 47/3 s 26/1 i:26 p 25/1 s 27/1 i:27 p 47/2 s 31/1 i:
28 p 25/1 s 27/1 i:29 p 25/1 i:31 p 27/1 i:39 p 47/1 i:47 p 25/3 s 27/2
 G=8 n:10 i:22 p 33/1 s 34/1 i:23 p 33/1 s 34/1 i:24 p 35/1 s 36/1 i:33 p 34/2 s
 35/2 i:34 p 33/2 s 36/2 i:35 p 33/2 s 46/2 i:36 p 34/2 i:40 p 33/1 i:45 p 33/2
i:46 p 35/2
 G=9 n:2 i:30 p 37/1 i:37 p 30/1
 G=10 n:1 i:32 p -1/0
 G=11 n:1 i:38 p -1/0
 G=12 n:4 i:41 p 42/2 s 43/2 i:42 p 41/2 s 44/2 i:43 p 41/2 i:44 p 42/2
 PRESS RETURN:
```

Figure 5-17 Group List of Ckt. 3

```
 Single Element Groups Formation:
 Group 1 n=8 i: 1 2 3 4 6 7 10 13
 Group 2 n=1 i: 9
 Group 3 n=2 i: 11 14
 Group 4 n=4 i: 5 12 15 42
 Group 5 n=3 i: 8 16 17
 Group 6 n=2 i: 18 19
 Group 7 n=2 i: 20 21
 Group 8 n=2 i: 22 23
 Group 9 n=2 i: 24 25
 Group 10 n=2 i: 26 27
 Group 11 n=9 i: 28 30 32 33 34 37 39 40 41
 Group 12 n=5 i: 29 31 35 36 38
 Group 13 n=1 i: 43
 Group 14 n=1 i: 44
 Press RETURN if finished investigation:
```

```
 Group List with connectivity attributes:
 G=1 n:8 i:1 p 2/2 s 6/2 i:2 p 1/2 s 3/2 i:3 p 2/2 s 4/2 i:4 p 3/2 s 7/2 i:6 p 1
/2 i:7 p 4/2 i:10 p 2/2 i:13 p 2/2
 G=2 n:1 i:9 p -1/0
 G=3 n:2 i:11 p 14/2 i:14 p 11/2
 G=4 n:4 i:5 p 12/1 s 15/1 i:12 p 15/2 s 5/1 i:15 p 12/2 i:42 p 12/1 s 15/1
 G=5 n:3 i:8 p 16/1 s 17/1 i:16 p 17/2 s 8/1 i:17 p 16/2
 G=6 n:2 i:18 p 19/2 i:19 p 18/2
 G=7 n:2 i:20 p 21/2 i:21 p 20/2
 G=8 n:2 i:22 p 23/2 i:23 p 22/2
 G=9 n:2 i:24 p 25/2 i:25 p 24/2
 G=10 n:2 i:26 p 27/2 i:27 p 26/2
 G=11 n:9 i:28 p 30/4 s 32/2 i:30 p 28/4 i:32 p 28/2 s 39/1 i:33 p 28/2 i:34 p 2
8/2 i:37 p 28/2 i:39 p 32/1 s 33/1 i:40 p 28/1 s 30/1 i:41 p 32/1 s 33/1
 G=12 n:5 i:29 p 38/4 s 31/2 i:31 p 29/2 i:35 p 36/2 i:36 p 29/2 s 35/2 i:38 p 2
9/4
 G=13 n:1 i:43 p -1/0
 G=14 n:1 i:44 p -1/0
 PRESS RETURN:
```

Figure 5-18 Group List of Ckt. 4

```
Single Element Groups Formation:
Group 1 n=2 i: 1 15
Group 2 n=2 i: 2 10
Group 3 n=2 i: 3 24
Group 4 n=2 i: 4 11
Group 5 n=2 i: 5 28
Group 6 n=2 i: 6 12
Group 7 n=2 i: 7 29
Group 8 n=2 i: 8 13
Group 9 n=2 i: 9 31
Group 10 n=2 i: 14 32
Group 11 n=8 i: 16 17 18 21 22 23 26 27
Group 12 n=2 i: 19 41
Group 13 n=2 i: 20 33
Group 14 n=2 i: 25 34
Group 15 n=2 i: 30 40
Group 16 n=2 i: 35 39
Group 17 n=1 i: 36
Group 18 n=1 i: 37
Group 19 n=2 i: 38 42
Press RETURN if finished investigation:
```

```
 Group List with connectivity attributes:
 G=1 n:2 i:1 p 15/1 i:15 p 1/1
 G=2 n:2 i:2 p 10/1 i:10 p 2/1
 G=3 n:2 i:3 p 24/1 i:24 p 3/1
 G=4 n:2 i:4 p 11/1 i:11 p 4/1
 G=5 n:2 i:5 p 28/1 i:28 p 5/1
 G=6 n:2 i:6 p 12/1 i:12 p 6/1
 G=7 n:2 i:7 p 29/1 i:29 p 7/1
 G=8 n:2 i:8 p 13/1 i:13 p 8/1
 G=9 n:2 i:9 p 31/1 i:31 p 9/1
 G=10 n:2 i:14 p 32/1 i:32 p 14/1
 G=11 n:8 i:16 p 17/2 s 21/2 i:17 p 18/3 s 22/3 i:18 p 23/4 s 17/3 i:21 p 16/2 i
:22 p 17/3 i:23 p 18/4 i:26 p 17/2 i:27 p 18/3
 G=12 n:2 i:19 p 41/1 i:41 p 19/1
 G=13 n:2 i:20 p 33/1 i:33 p 20/1
 G=14 n:2 i:25 p 34/1 i:34 p 25/1
 G=15 n:2 i:30 p 40/1 i:40 p 30/1
 G=16 n:2 i:35 p 39/1 i:39 p 35/1
 G=17 n:1 i:36 p -1/0
 G=18 n:1 i:37 p -1/0
 G=19 n:2 i:38 p 42/1 i:42 p 38/1
 PRESS RETURN:
```

Figure 5-19 Group List of Ckt. 6

```
 Single Element Groups Formation:
 Group 1 n=2 i: 1 10
 Group 2 n=2 i: 2 11
 Group 3 n=2 i: 3 12
 Group 4 n=2 i: 4 13
 Group 5 n=2 i: 5 14
 Group 6 n=2 i: 6 15
 Group 7 n=2 i: 7 16
 Group 8 n=2 i: 8 17
 Group 9 n=2 i: 9 18
 Group 10 n=2 i: 19 20
 Group 11 n=2 i: 21 22
 Group 12 n=2 i: 23 24
 Group 13 n=2 i: 25 26
 Group 14 n=2 i: 27 28
 Group 15 n=2 i: 29 30
 Group 16 n=12 i: 31 32 33 34 35 36 37 42 43 44 45 46
 Group 17 n=4 i: 38 39 40 41
 Group 18 n=1 i: 47
 Group 19 n=1 i: 48
 Press RETURN if finished investigation:
```

```
 Group List with connectivity attributes:
 G=1 n:2 i:1 p 10/1 i:10 p 1/1
 G=2 n:2 i:2 p 11/1 i:11 p 2/1
 G=3 n:2 i:3 p 12/1 i:12 p 3/1
 G=4 n:2 i:4 p 13/1 i:13 p 4/1
 G=5 n:2 i:5 p 14/1 i:14 p 5/1
 G=6 n:2 i:6 p 15/1 i:15 p 6/1
 G=7 n:2 i:7 p 16/1 i:16 p 7/1
 G=8 n:2 i:8 p 17/1 i:17 p 8/1
 G=9 n:2 i:9 p 18/1 i:18 p 9/1
 G=10 n:2 i:19 p 20/1 i:20 p 19/1
 G=11 n:2 i:21 p 22/1 i:22 p 21/1
 G=12 n:2 i:23 p 24/1 i:24 p 23/1
 G=13 n:2 i:25 p 26/1 i:26 p 25/1
 G=14 n:2 i:27 p 28/1 i:28 p 27/1
 G=15 n:2 i:29 p 30/1 i:30 p 29/1
 G=16 n:12 i:31 p 44/1 i:32 p 37/1 s 42/1 i:33 p 37/1 s 43/1 i:34 p 37/1 s 42/1
i:35 p 37/1 s 42/1 i:36 p 37/1 s 42/1 i:37 p 42/2 s 43/2 i:42 p 37/2 s 46/1 i:43
 p 37/2 i:44 p 37/2 s 31/1 i:45 p 37/1 i:46 p 42/1 s 43/1
 G=17 n:4 i:38 p 41/2 i:39 p 41/2 i:40 p 41/2 i:41 p 38/2 s 39/2
 G=18 n:1 i:47 p -1/0
 G=19 n:1 i:48 p -1/0
 PRESS RETURN:
```

Figure 5-20 Group List of Ckt. 7

```
Single Element Groups Formation:
Group 1 n=2 i: 1 4
Group 2 n=2 i: 2 5
Group 3 n=2 i: 3 6
Group 4 n=2 i: 7 8
Group 5 n=2 i: 9 10
Group 6 n=2 i: 11 14
Group 7 n=2 i: 12 15
Group 8 n=2 i: 13 16
Group 9 n=2 i: 17 18
Group 10 n=2 i: 19 20
Group 11 n=2 i: 21 24
Group 12 n=2 i: 22 25
Group 13 n=2 i: 23 26
Group 14 n=2 i: 27 28
Group 15 n=2 i: 29 30
Group 16 n=12 i: 31 32 33 34 35 36 37 42 43 44 45 46
Group 17 n=4 i: 38 39 40 41
Group 18 n=1 i: 47
Group 19 n=1 i: 48
Press RETURN if finished investigation:
```

```
 Group List with connectivity attributes:
 G=1 n:2 i:1 p 4/1 i:4 p 1/1
 G=2 n:2 i:2 p 5/1 i:5 p 2/1
 G=3 n:2 i:3 p 6/1 i:6 p 3/1
 G=4 n:2 i:7 p 8/1 i:8 p 7/1
 G=5 n:2 i:9 p 10/1 i:10 p 9/1
 G=6 n:2 i:11 p 14/1 i:14 p 11/1
 G=7 n:2 i:12 p 15/1 i:15 p 12/1
 G=8 n:2 i:13 p 16/1 i:16 p 13/1
 G=9 n:2 i:17 p 18/1 i:18 p 17/1
 G=10 n:2 i:19 p 20/1 i:20 p 19/1
 G=11 n:2 i:21 p 24/1 i:24 p 21/1
 G=12 n:2 i:22 p 25/1 i:25 p 22/1
 G=13 n:2 i:23 p 26/1 i:26 p 23/1
 G=14 n:2 i:27 p 28/1 i:28 p 27/1
 G=15 n:2 i:29 p 30/1 i:30 p 29/1
 G=16 n:12 i:31 p 44/1 i:32 p 37/1 s 42/1 i:33 p 37/1 s 43/1 i:34 p 37/1 s 42/1
i:35 p 37/1 s 42/1 i:36 p 37/1 s 42/1 i:37 p 42/2 s 43/2 i:42 p 37/2 s 46/1 i:43
 p 37/2 i:44 p 37/2 s 31/1 i:45 p 37/1 i:46 p 42/1 s 43/1
 G=17 n:4 i:38 p 41/2 i:39 p 41/2 i:40 p 41/2 i:41 p 38/2 s 39/2
 G=18 n:1 i:47 p -1/0
 G=19 n:1 i:48 p -1/0
 PRESS RETURN:
```

Figure 5-21 Group List of Ckt. 8

5.2 Details of Affinity Clustering Procedure on Ckt. 2 and Ckt. 5

The result of affinity clustering on ckt. 2 is shown in figure 5-22 and 5-23. The maximum connectivity of the circuit is 2. Hence, there was only one pass in the "Primary Grouping" step (Table 3-3). After this step, three groups were formed with sizes 7, 8 and 8 respectively. From ckt. 2, it is noted that the three groups are with larger number of connections. Then, after the Element Appendage to Existing Groups step, the first group became a group with size 15, while the other two had no change. After the third step "Loose Appendage of Ungrouped Elements", grouping increases to 8, with 5 groups of size 2. After the last step "Single Element Groups Formation", all the elements were included in the group list and there were 5 single element groups. As a result, ckt. 2 was partitioned into 13 groups.

The result on ckt. 5 is shown in figure 5-24 and 5-25. The maximum connectivity of this circuit is 3. Hence, there were two passes in the "Primary Grouping" step. In the pass with conn. (no. of connections) 2*, there was only one group with 8 elements. Each element in this group has 3 connections to other elements in the group. In other words, they are elements with the highest connectivity in the circuit. In the pass with conn. 1, five groups formed with sizes 8, 5, 3, 2, and 2 respectively. At the "Element Appendage to Existing Groups" step, group 2, 3 and 4 were expanded to include 17, 7 and 4 elements respectively. At the "Loose Appendage of Ungrouped Elements" step, there were 3 new groups with group size of 2. At the last step, 4 single element groups were formed. As a result, ckt. 5 was partitioned into 11 groups.

* In the pass with conn. i, elements with number of connections greater than i is grouped.

```
MAXCONN=2
Press RETURN if finished investigation:


Round: 1  End of pass with conn 1.
Group 1 n=7 i: 1 2 3 34 35 36 37
Group 2 n=8 i: 4 5 6 7 8 9 10 11
Group 3 n=8 i: 12 13 14 15 16 17 18 19
Press RETURN if finished investigation:

Primary Grouping:
Group 1 n=7 i: 1 2 3 34 35 36 37
Group 2 n=8 i: 4 5 6 7 8 9 10 11
Group 3 n=8 i: 12 13 14 15 16 17 18 19
Press RETURN if finished investigation:




Element Appendage to Existing Groups:
Group 1 n=15 i: 1 2 3 28 31 33 34 35 36 37 38 39 41 42 44
Group 2 n=8 i: 4 5 6 7 8 9 10 11
Group 3 n=8 i: 12 13 14 15 16 17 18 19
Press RETURN if finished investigation:




Loose Appendage of Ungrouped Elements:
Group 1 n=15 i: 1 2 3 28 31 33 34 35 36 37 38 39 41 42 44
Group 2 n=8 i: 4 5 6 7 8 9 10 11
Group 3 n=8 i: 12 13 14 15 16 17 18 19
Group 4 n=2 i: 20 21
Group 5 n=2 i: 22 23
Group 6 n=2 i: 24 25
Group 7 n=2 i: 26 27
Group 8 n=2 i: 29 30
Press RETURN if finished investigation:
```

Figure 5-22 Affinity Clustering Procedure on Ckt. 2

```
Single Element Groups Formation:
Group 1 n=15 i: 1 2 3 28 31 33 34 35 36 37 38 39 41 42 44
Group 2 n=8 i: 4 5 6 7 8 9 10 11
Group 3 n=8 i: 12 13 14 15 16 17 18 19
Group 4 n=2 i: 20 21
Group 5 n=2 i: 22 23
Group 6 n=2 i: 24 25
Group 7 n=2 i: 26 27
Group 8 n=2 i: 29 30
Group 9 n=1 i: 32
Group 10 n=1 i: 40
Group 11 n=1 i: 43
Group 12 n=1 i: 45
Group 13 n=1 i: 46
Press RETURN if finished investigation:
```

```
 Group List with connectivity attributes:
 G=1 n:15 i:1 p 2/2 s 35/2 i:2 p 1/2 s 3/2 i:3 p 2/2 s 31/1 i:28 p 36/1 i:31 p 3
/1 i:33 p 3/1 i:34 p 35/2 i:35 p 1/2 s 34/2 i:36 p 35/2 s 28/1 i:37 p 1/2 s 36/1
 i:38 p 37/1 i:39 p 3/1 i:41 p 36/1 i:42 p 1/1 s 2/1 i:44 p 3/1
 G=2 n:8 i:4 p 5/2 i:5 p 4/2 s 6/2 i:6 p 5/2 s 7/2 i:7 p 6/2 s 8/2 i:8 p 7/2 s 9
/2 i:9 p 8/2 s 10/2 i:10 p 9/2 s 11/2 i:11 p 10/2
 G=3 n:8 i:12 p 13/2 s 14/2 i:13 p 12/2 i:14 p 12/2 i:15 p 12/2 i:16 p 12/2 i:17
 p 12/2 i:18 p 12/2 i:19 p 12/2
 G=4 n:2 i:20 p 21/1 i:21 p 20/1
 G=5 n:2 i:22 p 23/1 i:23 p 22/1
 G=6 n:2 i:24 p 25/1 i:25 p 24/1
 G=7 n:2 i:26 p 27/1 i:27 p 26/1
 G=8 n:2 i:29 p 30/1 i:30 p 29/1
 G=9 n:1 i:32 p -1/0
 G=10 n:1 i:40 p -1/0
 G=11 n:1 i:43 p -1/0
 G=12 n:1 i:45 p -1/0
 G=13 n:1 i:46 p -1/0
 PRESS RETURN:
```

Figure 5-23 Affinity Clustering Procedure on Ckt. 2 (cont.)

```
MAXCONN=3
Press RETURN if finished investigation:


Round: 2  End of pass with conn 2.
Group 1 n=8 i: 9 10 11 12 13 14 15 16
Press RETURN if finished investigation:


Round: 1  End of pass with conn 1.
Group 1 n=8 i: 9 10 11 12 13 14 15 16
Group 2 n=5 i: 17 18 42 45 46
Group 3 n=3 i: 32 35 36
Group 4 n=2 i: 33 34
Group 5 n=2 i: 43 44
Press RETURN if finished investigation:

Primary Grouping:
Group 1 n=8 i: 9 10 11 12 13 14 15 16
Group 2 n=5 i: 17 18 42 45 46
Group 3 n=3 i: 32 35 36
Group 4 n=2 i: 33 34
Group 5 n=2 i: 43 44
Press RETURN if finished investigation:


Element Appendage to Existing Groups:
Group 1 n=8 i: 9 10 11 12 13 14 15 16
Group 2 n=17 i: 1 2 3 4 5 6 7 8 17 18 19 25 30 41 42 45 46
Group 3 n=7 i: 20 21 23 31 32 35 36
Group 4 n=4 i: 22 28 33 34
Group 5 n=2 i: 43 44
Press RETURN if finished investigation:



Loose Appendage of Ungrouped Elements:
Group 1 n=8 i: 9 10 11 12 13 14 15 16
Group 2 n=17 i: 1 2 3 4 5 6 7 8 17 18 19 25 30 41 42 45 46
Group 3 n=7 i: 20 21 23 31 32 35 36
Group 4 n=4 i: 22 28 33 34
Group 5 n=2 i: 26 27
Group 6 n=2 i: 29 37
Group 7 n=2 i: 43 44
Press RETURN if finished investigation:
```

Figure 5-24 Affinity Clustering Procedure on Ckt. 5

```
 Single Element Groups Formation:
 Group 1 n=8 i: 9 10 11 12 13 14 15 16
 Group 2 n=17 i: 1 2 3 4 5 6 7 8 17 18 19 25 30 41 42 45 46
 Group 3 n=7 i: 20 21 23 31 32 35 36
 Group 4 n=4 i: 22 28 33 34
 Group 5 n=1 i: 24
 Group 6 n=2 i: 26 27
 Group 7 n=2 i: 29 37
 Group 8 n=1 i: 38
 Group 9 n=1 i: 39
 Group 10 n=1 i: 40
 Group 11 n=2 i: 43 44
 Press RETURN if finished investigation:
```

```
 Group List with connectivity attributes:
 G=1 n:8 i:9 p 10/3 s 11/3 i:10 p 9/3 i:11 p 9/3 i:12 p 9/3 i:13 p 9/3 i:14 p 9/
3 i:15 p 9/3 i:16 p 9/3
 G=2 n:17 i:1 p 17/1 i:2 p 17/1 i:3 p 18/1 i:4 p 18/1 i:5 p 18/1 i:6 p 18/1 i:7
p 17/1 i:8 p 17/1 i:17 p 18/2 s 45/2 i:18 p 17/2 s 3/1 i:19 p 17/1 s 18/1 i:25 p
 42/1 i:30 p 17/1 s 18/1 i:41 p 17/1 s 18/1 i:42 p 46/2 s 25/1 i:45 p 17/2 i:46
p 17/2 s 42/2
 G=3 n:7 i:20 p 35/1 s 36/1 i:21 p 32/1 s 35/1 i:23 p 32/1 i:31 p 32/1 s 35/1 i:
32 p 36/2 s 21/1 i:35 p 36/2 s 20/1 i:36 p 32/2 s 35/2
 G=4 n:4 i:22 p 33/1 s 34/1 i:28 p 33/1 s 34/1 i:33 p 34/2 s 22/1 i:34 p 33/2
 G=5 n:1 i:24 p -1/0
 G=6 n:2 i:26 p 27/1 i:27 p 26/1
 G=7 n:2 i:29 p 37/1 i:37 p 29/1
 G=8 n:1 i:38 p -1/0
 G=9 n:1 i:39 p -1/0
 G=10 n:1 i:40 p -1/0
 G=11 n:2 i:43 p 44/2 i:44 p 43/2
 PRESS RETURN:
```

Figure 5-25 Affinity Clustering Procedure on Ckt. 5 (cont.)

5.3 Results on Alteration Phase

The results on alteration is summarized in the following table. Ckt. 1 is not included in this table since its cell count is too small (18) and the result is not suitable to be compared with other circuits (cell count > 40).

Ckt	2	3	4	5	6	7	8
N_o	46	47	44	46	42	48	48
ΔN	3	2	5	3	3	1	1
C_o	257	184	294	189	182	295	271
C_d	291	223	333	248	196	314	299
C_s	292	216	341	245	201	305	295
C_a	284	215	332	242	198	306	295
P_s	277	191	327	224	193	302	289
P_a	264	186	307	207	186	299	289
ΔC_s (%)	0.34	-3.14	2.40	-1.21	2.55	-2.87	-1.34
$\Delta C_s/\Delta N$ (%)	0.11	-1.57	0.48	-0.40	0.85	-2.87	-1.34
ΔC_a (%)	-2.41	-3.59	-0.30	-2.42	1.02	-2.55	-1.34
$\Delta C_a/\Delta N$ (%)	-0.80	-1.80	-0.06	-0.81	0.34	-2.55	-1.34
ΔP_s (%)	7.78	3.80	11.22	18.52	6.04	2.37	6.64
ΔP_a (%)	2.72	1.09	4.42	9.52	2.20	1.36	6.64

where N_o = Original cell count,
ΔN = Change in cell count,
C_o = Original cost on the placement,
C_d = Cost on MDAE (Method of Direct Addition of new Element),
C_s = Cost on MSPS (Method of Simple Propagation of empty-Space),
C_a = Cost on IAPA (Incremental Alteration Placement Algorithm),
P_s = Cost on MSPS without considering added elements,
P_a = Cost on IAPA without considering added elements,
ΔC_s = Change in cost on MSPS = $(C_s-C_d)/C_d$,
$\Delta C_s/\Delta N$ = Change in cost on MSPS per added element,
ΔC_a = Change in cost on IAPA = $(C_a-C_d)/C_d$,
$\Delta C_a/\Delta N$ = Change in cost on IAPA per added element,
ΔP_s = $(P_s-C_o)/C_o$, and
ΔP_a = $(P_a-C_o)/C_o$,

Table 5-1. Results on Alteration of Ckt.2 to 8

MDAE is a method of updating the placement by placing the added element in an empty space with the least cost. Hence, the original placement of all the other elements would not be changed. This method is the most straightforward way to update the placement and its results are used as reference for other methods.

MSPS is a method to change the original placement by simply propagating (shifting) elements along the cross-cut (Section 3.2.3.1). Firstly, the mostly connected element, c, of the added element, a, will be determined. Secondly, the nearest empty space to 'c' will be determined. Thirdly, the cross-cut between c and the empty space will be found. Then, the cells will be shifted along the cross-cut towards the empty space until the empty space is adjacent to 'c' (as in Figure 3-13). Finally, the added element will be placed at the evacuated space.

From the above table, it is seen that ΔC_a reduces the cost much more than ΔC_s. This implies IAPA make smaller disturbance to the placements than both MDAE and MSPS. Calculation on the average values of ΔC are as follows:

Average $\Delta C_s = -0.47\%$

Average $\Delta C_a = -1.66\%$

Hence, improvement on ΔC by IAPA = 1.19% with respect to MSPS
= 1.66% with respect to MDAE

$\Delta C/\Delta N$ is a measure on change in cost per added element. It gives normalized index on the result of the algorithms.

Average $\Delta C_s/\Delta N = -0.68\%$ per added element

Average $\Delta C_a/\Delta N = -1.00\%$ per added element

Hence, improvement on $\Delta C/\Delta N$ by IAPA
= 0.32% per added element with respect to MSPS
= 1.00% per added element with respect to MDAE

P_s and P_a are the costs on MSPS and IAPA respectively without taking the added elements into account. In other words, they considers only the connections among the original circuitry. These two values are to indicate the degree of disturbance on the original placement subject to the alteration. That is, if the P value of a method is much larger than C_o, it implies that the method has caused much disturbance on the original placement. From Table 5-1, it is also noticed that IAPA gives better result than MSPS : the increases in P values of the circuits are less. Calculation on the average values of ΔP are as follows:

Average $\Delta P_s = 8.05\%$

```
Average ΔPa = 3.99%
Hence, improvement on ΔP by IAPA = 4.06%
```

The above results are summarized in the graphs of figure 5-26 to 5-28.

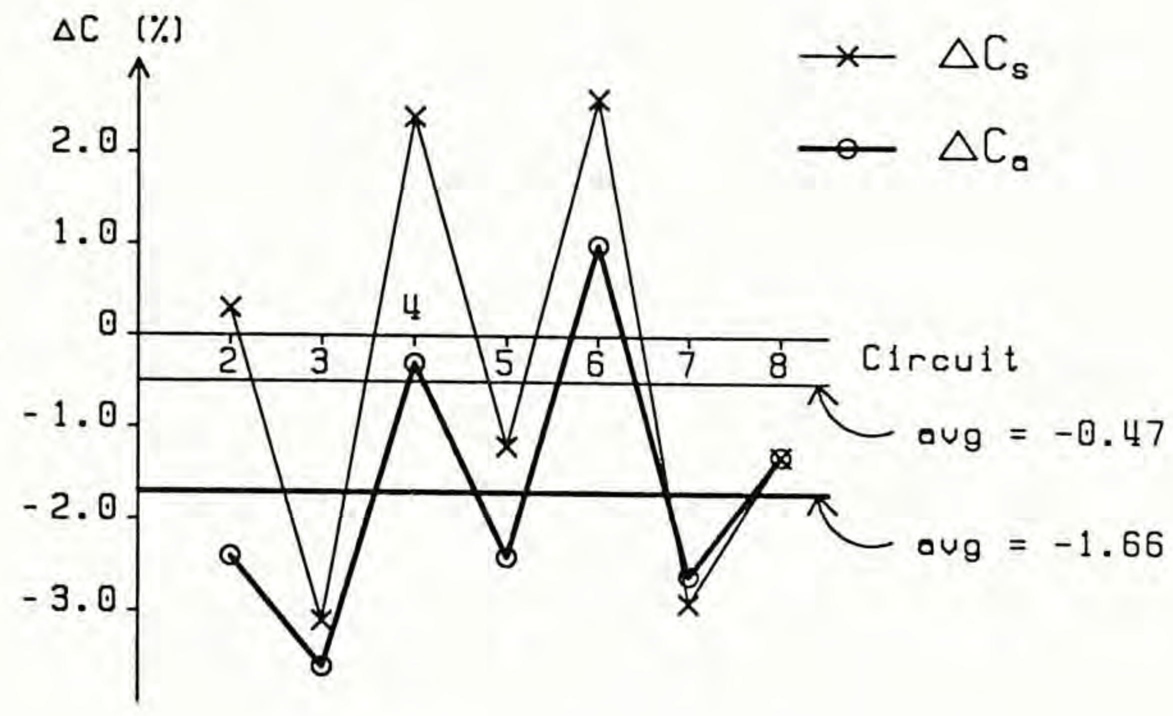

Figure 5-26 Results on Total Change in Cost

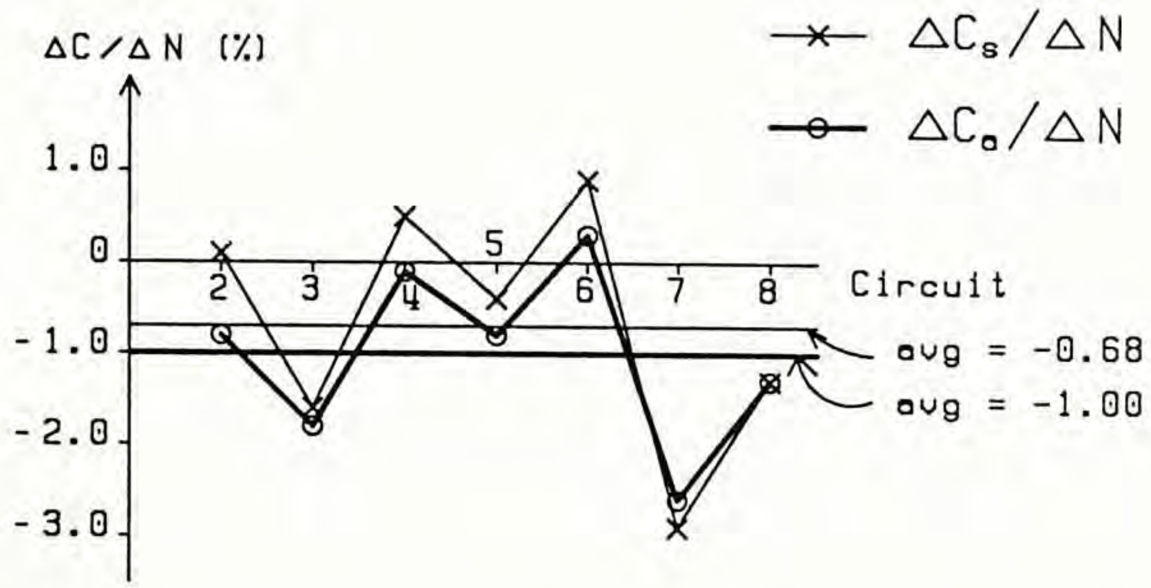

Figure 5-27 Results on Change in Cost Per Added Element

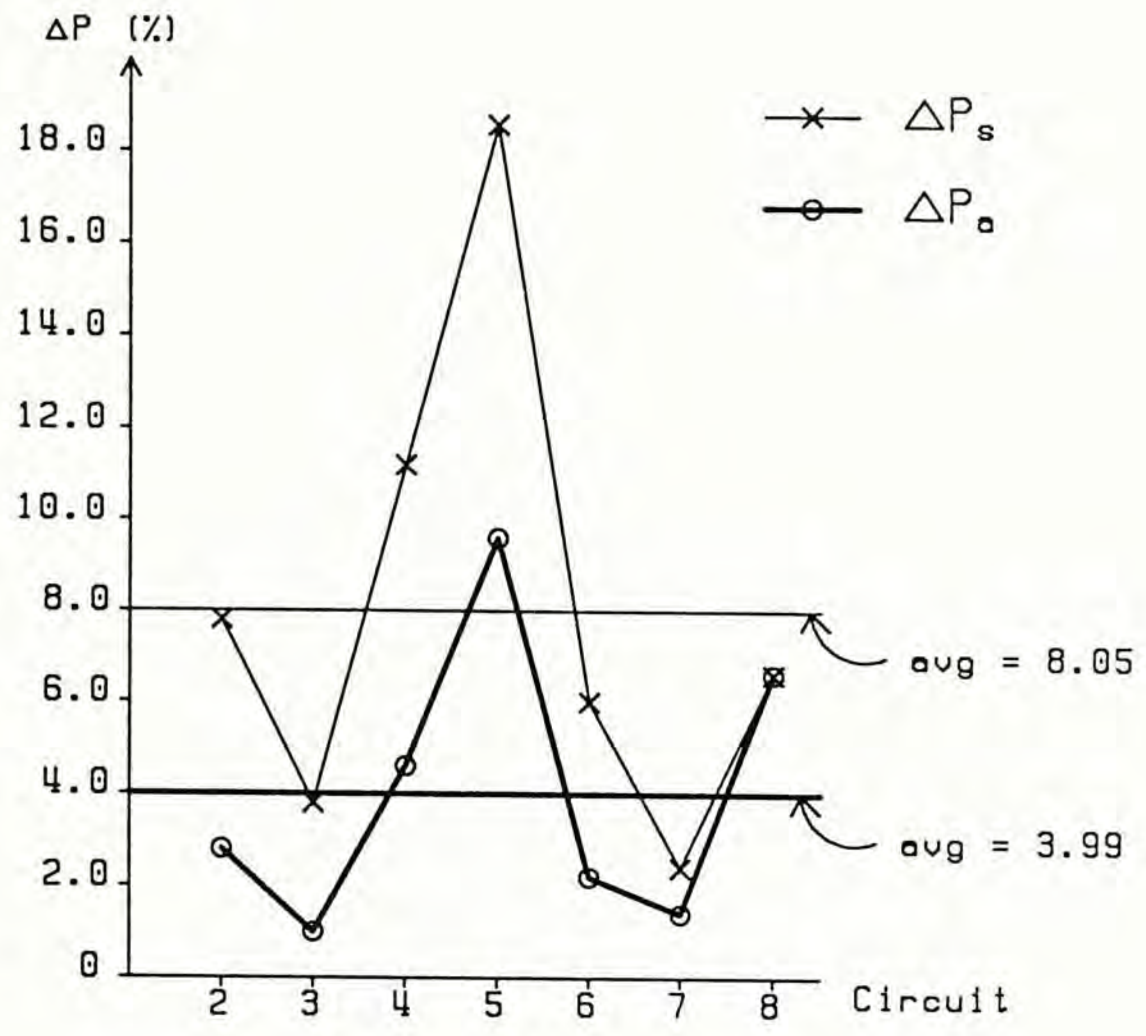

Figure 5-28 Results on Change in Cost Without Considering Added Elements

In the three figures, solid lines represent the results of MSPS while bold lines represent those of IAPA. Areas between the lines indicate the improvement of placement by using IAPA instead of MSPS.

5.4 Details of Alteration Procedure on Ckt. 2 and Ckt. 5

The processing history on ckt. 2 in the alteration procedure is summarized as follows:

The alterations on ckt. 2 are shown in figure 5-29. There are 3 new elements added to the circuit, namely, 47, 48, 49. e47 (element 47) is assigned to G1 = {1, 2, 3, 28, 31, 33, 34, 35, 36, 37, 38, 39, 41, 42, 44}. In the processing of e47 (figure 5-29a), the left matrix is the original placement, the middle one is the grouping , and the right one gives the location(s) of adjacent empty space(s) to G1. A "T" represents an empty space adjacent to G1. In this case, an empty space adjacent to G1 is found on (1,4). The new element will be assigned to this space and the length of connection of e47 to G1 is 3 units.

After calculating e48's belong tendencies to the groups, it is assigned to G3 = {12, 13, 14, 15, 16, 17, 18, 19}. In addition, e48 will be linked to the cluster tree. In the left matrix, it can be observed that there are two adjacent empty spaces to G3, namely (6,6) and (7,7). The connection length at the first one is 11 units and that at the second is 15. Hence, the first one is chosen as the empty space for e48.

Similarly, e49 is assigned to G9 = {32} but there is no adjacent empty space to this group. Hence, the optimal propagation path joining c and ξ is determined and shown as "T" items on the right matrix (in this matrix, the meaning of "T" and "F" is different from those in figure a and b). It can be observed that this path passes through 3 groups: G1 (size=16), G13 (size=1), and G3 (size=9), and is the optimal solution to the alteration problem.

The processing history on ckt. 5 is summarized as follows:

The alterations on ckt. 5 is shown in figure 5-30. There are 3 new elements added to the circuit, namely, 47, 48, 49. e47 is assigned to G9 = {39}. The nearest empty space is at (1,3). The optimal propagation path is shown on the right matrix. It passes through G2 (size=17) and G11 (size=2).

e48 is assigned to G9 = {39, 47} (updated). The nearest empty space is at (2,7). The optimal propagation path passes through G2 (size=17), G1 (size=8), and G4 (size=4).

e49 is assigned to G2 = {1, 2, 3, 4, 5, 6, 7, 8, 17, 18, 19, 25, 30, 41, 42, 45, 46} and the nearest empty space is at (1,7). The optimal propagation path passes through G1 (size=8).

```
 Looking for C Neigh ...
n=16 i: 1 2 3 28 31 33 34 35 36 37 38 39 41 42 44 47
 C Neigh found.

10  6 11 14 21 26 27     2  2  2  3  4  7  7     F F F F F F F
 9 17  4 33 18 19 43     2  3  2  1  3  3 11     F F F F F F F
 8 32 22  5 13 25 28     2  9  5  2  3  6  1     F F F F F F F
 0  2  7 15 40 12 20     0  1  2  3 10  3  4     T F F F F F F
 1 38 39 36 30 23 29     1  1  1  1  8  5  8     F F F F F F F
37  3 42 44 46  0 24     1  1  1  1 13  0  6     F F F F F F F
34 35 41 31 45 16  0     1  1  1  1 12  3  0     F F F F F F F
detail_link (d) or fcfs_link (f): d

Inst: 47 Erg:3 Len:3 MIN_ENERGY=3 LOC assigned

End of Element 47.
Press RETURN to continue:
```

(a)

```
 Looking for C Neigh ...
n=9 i: 12 13 14 15 16 17 18 19 48
 C Neigh found.

10  6 11 14 21 26 27     2  2  2  3  4  7  7     F F F F F F F
 9 17  4 33 18 19 43     2  3  2  1  3  3 11     F F F F F F F
 8 32 22  5 13 25 28     2  9  5  2  3  6  1     F F F F F F F
47  2  7 15 40 12 20     1  1  2  3 10  3  4     F F F F F F F
 1 38 39 36 30 23 29     1  1  1  1  8  5  8     F F F F F F F
37  3 42 44 46  0 24     1  1  1  1 13  0  6     F F F F F T F
34 35 41 31 45 16  0     1  1  1  1 12  3  0     F F F F F F T
detail_link (d) or fcfs_link (f): d

Inst: 48 Erg:11 Len:11
Inst: 48 Erg:15 Len:15 MIN_ENERGY=11 LOC assigned

End of Element 48.
Press RETURN to continue:
```

(b)

```
 Min cost=48
Press RETURN to continue:
 MIN_BOARD:
10  6 11 14 21 26 27     2  2  2  3  4  7  7     F F F F F F F
 9 17  4 33 18 19 43     2  3  2  1  3  3 11     F F F F F F F
 8 32 22  5 13 25 28     2  9  5  2  3  6  1     F T F F F F F
47  2  7 15 40 12 20     1  1  2  3 10  3  4     F T F F F F F
 1 38 39 36 30 23 29     1  1  1  1  8  5  8     F F T T F F F
37  3 42 44 46 48 24     1  1  1  1 13  3  6     F F F F T T F
34 35 41 31 45 16  0     1  1  1  1 12  3  0     F F F F F F T
 Want to insert? (y/n):y
 LOC assigned .
 End of Element 49.
Press RETURN to continue:
```

(c)

Figure 5-29 Alteration Procedure on Ckt. 2

```
 Min cost=36
Press RETURN to continue:
 MIN_BOARD:
 27 26  2 40 11  9  0     6  6  2 10  1  1  0     F F F F F F F
  3  5 24  6 15 20  0     2  2  5  2  1  3  0     F F F F F F F
  0 18  4 16 22 21 34     0  2  2  1  4  3  4     T F F F F F F
  8  1  7 12 14 10 33     2  2  2  1  1  1  4     F T F F F F F
 28 19 17 46 13 32 35     4  2  2  2  1  3  3     F T F F F F F
 38 43 42 41 25 31 23     8 11  2  2  2  3  3     F T F F F F F
 39 44 45 30 37 36 29     9 11  2  2  7  3  7     T F F F F F F
 Want to insert? (y/n):y
 LOC assigned
 End of Element 47.
Press RETURN to continue:
```

(a)

```
 Min cost=52
Press RETURN to continue:
 MIN_BOARD:
 27 26  2 40 11  9  0     6  6  2 10  1  1  0     F F F F F F F
  3  5 24  6 15 20  0     2  2  5  2  1  3  0     F F F F F F T
  1 18  4 16 22 21 34     2  2  2  1  4  3  4     F F F F F F T
  8 19  7 12 14 10 33     2  2  2  1  1  1  4     F F F T T T F
 28 43 17 46 13 32 35     4 11  2  2  1  3  3     F F T F F F F
 38 47 42 41 25 31 23     8  9  2  2  2  3  3     F T F F F F F
 39 44 45 30 37 36 29     9 11  2  2  7  3  7     F F F F F F F
 Want to insert? (y/n):y
 LOC assigned
 End of Element 48.
Press RETURN to continue:
```

(b)

```
 Min cost=36
Press RETURN to continue:
 MIN_BOARD:
 27 26  2 40 11  9  0     6  6  2 10  1  1  0     F F F F F T T
  3  5 24  6 15 20 34     2  2  5  2  1  3  4     F F F T T F F
  1 18  4 16 22 21 10     2  2  2  1  4  3  1     F F F F F F F
  8 19  7 17 12 14 33     2  2  2  2  1  1  4     F F F F F F F
 28 43 48 46 13 32 35     4 11  9  2  1  3  3     F F F F F F F
 38 47 42 41 25 31 23     8  9  2  2  2  3  3     F F F F F F F
 39 44 45 30 37 36 29     9 11  2  2  7  3  7     F F F F F F F
 Want to insert? (y/n):y
 LOC assigned
 End of Element 49.
Press RETURN to continue:
```

(c)

Figure 5-30 Alteration Procedure on Ckt. 5

The final results of placement, grouping, and the cluster trees of ckt. 2 and 5 are shown in figure 31 and 32 respectively.

```
Ckt 2:        Placement                      Grouping
        10  6 11 14 21 26 27          2  2  2  3  4  7  7
         9 17  4 33 18 19 43          2  3  2  1  3  3 11
         8 32 22  5 13 25 28          2  9  5  2  3  6  1
        47 49  7 15 40 12 20          1  9  2  3 10  3  4
         1 38  2 39 30 23 29          1  1  1  1  8  5  8
        37  3 42 44 36 46 24          1  1  1  1  1 13  6
        34 35 41 31 45 16 48          1  1  1  1 12  3  3
 RETURN:
```

(a)

```
Final Cluster Tree:
G=1 n:16 i: 1 2 3 28 31 33 34 35 36 37 38 39 41 42 44 47
G=2 n:8 i: 4 5 6 7 8 9 10 11
G=3 n:9 i: 12 13 14 15 16 17 18 19 48
G=4 n:2 i: 20 21
G=5 n:2 i: 22 23
G=6 n:2 i: 24 25
G=7 n:2 i: 26 27
G=8 n:2 i: 29 30
G=9 n:2 i: 32 49
G=10 n:1 i: 40
G=11 n:1 i: 43
G=12 n:1 i: 45
G=13 n:1 i: 46
  RETURN:
```

(b)

```
Final Cluster Tree with connectivity attributes:
 G=1 n:16 i:1 p 2/2 s 35/2 i:2 p 1/2 s 3/2 i:3 p 2/2 s 31/1 i:28 p 36/1 i:31 p 3
/1 i:33 p 3/1 i:34 p 35/2 i:35 p 1/2 s 34/2 i:36 p 35/2 s 28/1 i:37 p 1/2 s 38/1
 i:38 p 37/1 i:39 p 3/1 i:41 p 36/1 i:42 p 1/1 s 2/1 i:44 p 3/1 i:47 p 2/2 s 1/1

 G=2 n:8 i:4 p 5/2 i:5 p 4/2 s 6/2 i:6 p 5/2 s 7/2 i:7 p 6/2 s 8/2 i:8 p 7/2 s 9
/2 i:9 p 8/2 s 10/2 i:10 p 9/2 s 11/2 i:11 p 10/2
 G=3 n:9 i:12 p 13/2 s 14/2 i:13 p 12/2 i:14 p 12/2 i:15 p 12/2 i:16 p 12/2 i:17
 p 12/2 i:16 p 12/2 i:19 p 12/2 i:48 p 14/1 s 15/1
 G=4 n:2 i:20 p 21/1 i:21 p 20/1
 G=5 n:2 i:22 p 23/1 i:23 p 22/1
 G=6 n:2 i:24 p 25/1 i:25 p 24/1
 G=7 n:2 i:26 p 27/1 i:27 p 26/1
 G=8 n:2 i:29 p 30/1 i:30 p 29/1
 G=9 n:2 i:32 p -1/0 i:49 p 32/2 s 45/1
 G=10 n:1 i:40 p -1/0
 G=11 n:1 i:43 p -1/0
 G=12 n:1 i:45 p -1/0
 G=13 n:1 i:46 p -1/0
   RETURN:
```

(c)

Figure 5-31 Final Results of Placement, Grouping and the Cluster Tree of Ckt. 2

```
    Ckt 5:        Placement                        Grouping
           27 26  2 40 11 15  9          6  6  2 10  1  1  1
            3  5 24  6 49 20 34          2  2  5  2  2  3  4
            1 18  4 16 22 21 10          2  2  2  1  4  3  1
            8 19  7 17 12 14 33          2  2  2  2  1  1  4
           28 43 48 46 13 32 35          4 11  9  2  1  3  3
           38 47 42 41 25 31 23          8  9  2  2  2  3  3
           39 44 45 30 37 36 29          9 11  2  2  7  3  7
   RETURN:
```

(a)

```
 Final Cluster Tree:
 G=1 n:8 i: 9 10 11 12 13 14 15 16
 G=2 n:18 i: 1 2 3 4 5 6 7 8 17 18 19 25 30 41 42 45 46 49
 G=3 n:7 i: 20 21 23 31 32 35 36
 G=4 n:4 i: 22 28 33 34
 G=5 n:1 i: 24
 G=6 n:2 i: 26 27
 G=7 n:2 i: 29 37
 G=8 n:1 i: 38
 G=9 n:3 i: 39 47 48
 G=10 n:1 i: 40
 G=11 n:2 i: 43 44
    RETURN:
```

(b)

```
 Final Cluster Tree with connectivity attributes:
 G=1 n:8 i:8 p 10/3 s 11/3 i:10 p 9/3 i:11 p 9/3 i:12 p 9/3 i:13 p 9/3 i:14 p 9/
3 i:15 p 9/3 i:16 p 9/3
 G=2 n:18 i:1 p 17/1 i:2 p 17/1 i:3 p 18/1 i:4 p 18/1 i:5 p 18/1 i:6 p 18/1 i:7
p 17/1 i:8 p 17/1 i:17 p 18/2 s 45/2 i:18 p 17/2 s 3/1 i:19 p 17/1 s 18/1 i:25 p
 42/1 i:30 p 17/1 s 18/1 i:41 p 17/1 s 18/1 i:42 p 46/2 s 25/1 i:45 p 17/2 i:46
p 17/2 s 42/2 i:49 p 1/2 s 41/2
 G=3 n:7 i:20 p 35/1 s 36/1 i:21 p 32/1 s 35/1 i:23 p 32/1 i:31 p 32/1 s 35/1 i:
32 p 36/2 s 21/1 i:35 p 36/2 s 20/1 i:36 p 32/2 s 35/2
 G=4 n:4 i:22 p 33/1 s 34/1 i:28 p 33/1 s 34/1 i:33 p 34/2 s 22/1 i:34 p 33/2
 G=5 n:1 i:24 p -1/0
 G=6 n:2 i:26 p 27/1 i:27 p 26/1
 G=7 n:2 i:29 p 37/1 i:37 p 29/1
 G=8 n:1 i:38 p -1/0
 G=9 n:3 i:39 p -1/0 i:47 p 39/1 i:48 p 39/1
 G=10 n:1 i:40 p -1/0
 G=11 n:2 i:43 p 44/2 i:44 p 43/2
    RETURN:
```

(c)

Figure 5-32 Final Results of Placement, Grouping and the Cluster Tree of Ckt. 5

In this thesis, only the results of 7X7 grid plane is done because the maximum size that can be afforded by the initial placement program "fplace77.pas" (Appendix I) is 7X7 variables. If a larger plane is to be process, the initial placement should be done manually. A new algorithm on the initial placement for larger circuit is discussed in Section 6.5. This new algorithm will utilize the constructed cluster tree data structure in the Affinity Clustering phase and will solve placement problem on a larger grid plane.

6. Discussion

In this section, the computation time of IAPA will be analyzed (section 6.1); three alternative methods on the determination of propagation path will be discussed and compared (section 6.2); an algorithm on wiring optimization (section 6.3) and the method to generalize the cluster tree data structure (section 6.4) are introduced. In addition, a new placement algorithm and an alternative method on element allocation are described in section 6.5 and 6.6 respectively.

6.1 Computation Time of the Algorithm

The computation time of the algorithm is summarized as follows (n is the number of cells in the circuit and $|e|$ is the number of connections). Derivation is in Appendix II.

1. Affinity Clustering:

step	computation cycle
i. construction of connection lists	$|e|$
ii. primary grouping	$|e| + n^2/16$
iii. element appendage to existing groups	$|e|/2 + 5n^2/64$
iv. loose appendage of ungrouped elements	$|e|/4 + 13n^2/256$
v. single element group formation	$|e|/8 + 15n^2/256$

Total computation time is

$T1 = 23|e|/8 + n^2/4$ cycles .... Eq. 6-1

Assume that each cell has an average of 2 connections to other cells, $|e|=2n$. Then, the equation becomes:

$T1 = 23n/4 + n^2/4$ cycles .... Eq. 6-2

2. Alteration:

step	computation cycle
i. element assignment to a group	n
ii. empty space searching	$2(n/2)^{1/2}$
iii. determination of direction of element allocation	
- cross-cut direction of allocation	$(n/2)^{1/2}$
- dynamic determination of path based on size functions	
- segmentation of cross-cut	$(n/2)^{1/2}/4$
- partial optimization of segments	$3n^{1/2}/2$
- dynamic linking of segments	$3^{[(n^{1/2}/6.36)+1]}$

iv. element allocation $(n/2)^{1/2}$

Total computation time is

$$T2 = n + 4^*(n/2)^{1/2} + (n/2)^{1/2}/4 + 3n^{1/2}/2 + 3^{[(n^{1/2}/6.36)+1]}$$

$$= n/2 + 4.5n^{1/2} + 3^{[(n^{1/2}/6.36)+1]} \quad \text{cycles} \quad \text{.................. Eq. 6-3}$$

The graphs on these two equations are shown in figure 6-1.

Equation 6-2 and 6-3 indicates acceptable computation time since the computation time for a whole placement process should be at least of complexity $O(n^2)$ while the present algorithm is of about $O(n^2)$ (the n terms, $n^{1/2}$ terms and the last term in Eq. 6-3 is small as compared with the n^2 term as n is less than 10000).

The values of T1 and T2 for n=9 to 100 is summarized in the following table:

n	9	16	25	36	49	64	81	100
T1	73	156	300	531	882	1392	2106	3075
T2	27.5	40	54.6	71.5	90.6	111.9	135.7	161.9

Table 6-0. Values of T1 and T2

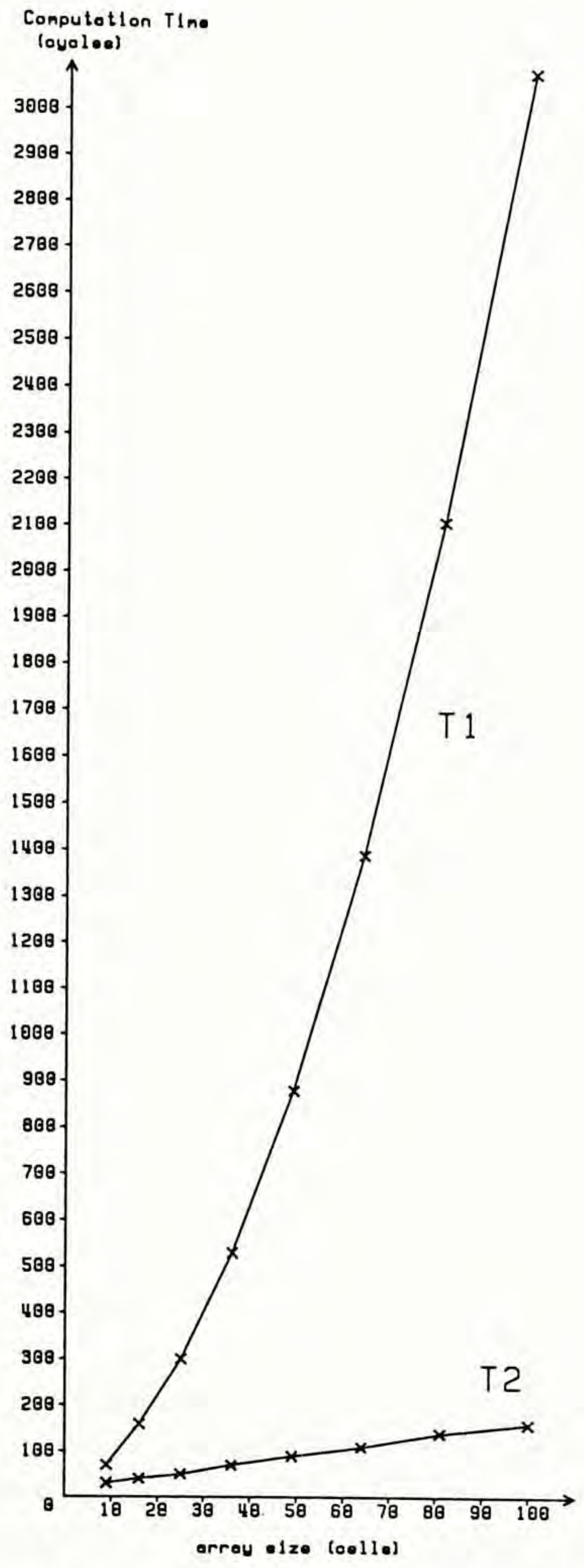

Figure 6-1 Computation Time of IAPA

6.2 Alternative Methods on the Determination of Propagation Path

In this thesis, the propagation path on alteration of placement is determined by segmentation of cross-cut and linking of optimal segments. The following are three methods proposed on the determination of propagation path. The first one is an exhaustive search on the optimal propagation path. It will go step by step on the determination of path direction. The second one executes a thorough search on path segments. It will use more computation time but the search is more detailed. The third one is a method with different approach. It will use progressive accessing method and recursive functions is required.

6.2.1 Method 1

This method will execute an exhaustive search on a region between c and ξ. Consider the following examples:

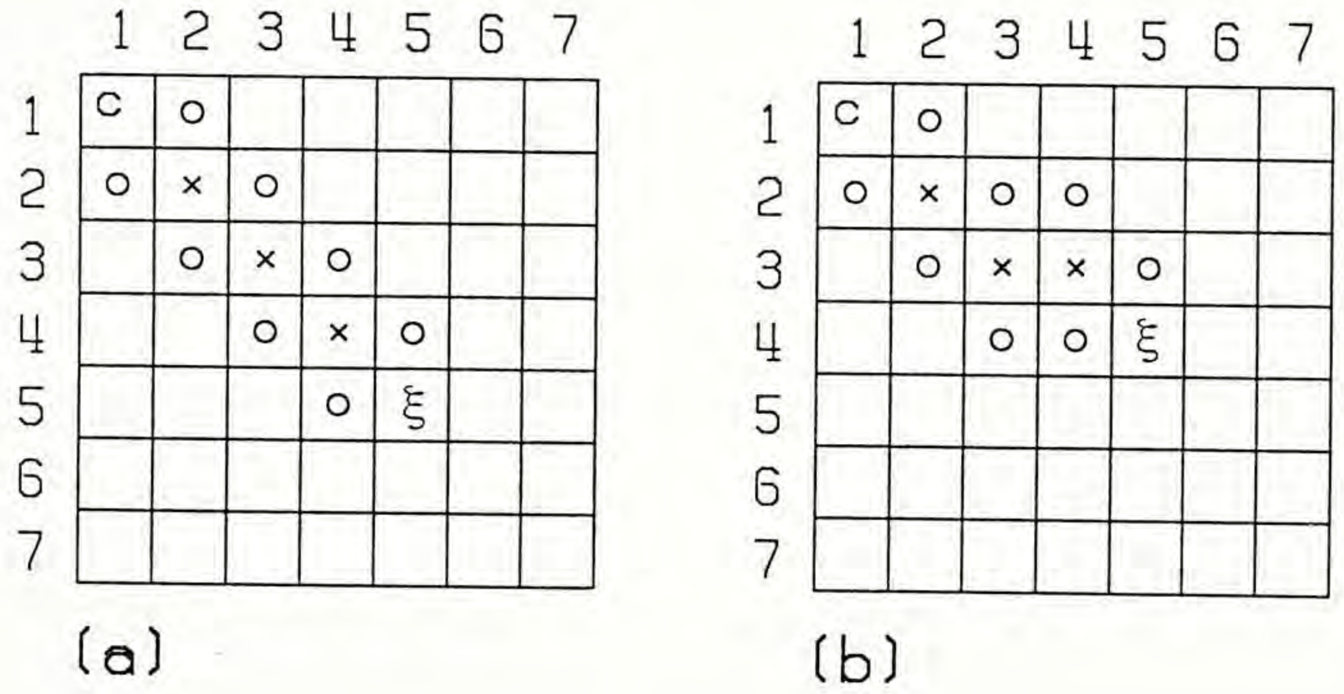

Figure 6- 2 Ranges for Exhaustive Search

The paths with crosses are the cross-cuts. The adjacent neighbours to the cross-cuts will be marked with circles. Then, the optimal propagation paths are determined within the regions with marks by progressively linking towards the target, ξ, starting from c. The directions of linking are illustrated in figure 6-3. On the slot with circles, there will be two directions on linking, while on that with crosses, there will be three directions. Linkage of the propagation path is based on the criterion to minimize the cost functions and the length of the path.

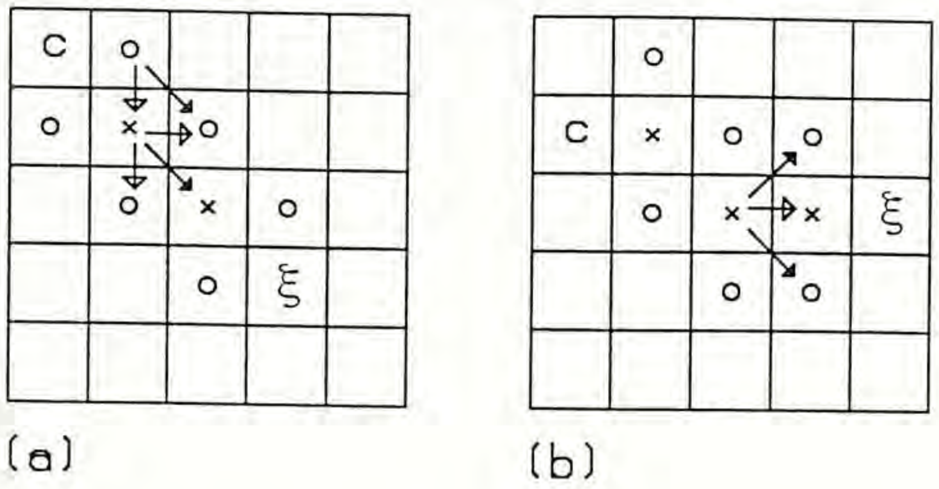

Figure 6-3 Directions for Linking Propagation Path

The number of execution cycles on this method is:

$T_{M1} = (A.M.)^x$

where A.M. is the arithmetic mean of the number of directions of linking, and

x is the number of steps on the path.

Since A.M. = (2+3)/2 and $x = (n/2)^{1/2}$,

$T_{M1} = (1.9)^{n^{1/2}}$ ----------- Eq. 6-4.

For an 7X7 array, the average number of execution time is

$T_{M1} =$ 89.4 cycles

which is a large number as compared with that in section 3.2.3.2.3 (9 cycles).

6.2.2 Method 2

Method 2 is similar to the dynamic determination of path in section 3.2.3.2. However, the search for optimal path is done on more path segments. Consider the examples in figure 6-4. The small squares mark the points on the cross-cut with L_{MHT} of 4 or 5 from c or the last point. The slots with circles are points to determine path segment templates (as in section 3.2.3.2.2). These circles may be on the diagonal or adjacent (horizontal or vertical) neighbours to the crosses. It is classified by the inclined angle of the cross-cut. In algorithmic form,

```
xdiff:= xe-xs;
ydiff:= ye-ys;
L_MHT := xdiff+ydiff;
if (L_MHT=4 or L_MHT=5) then
```

```
begin
  set_cross_point;
  if (xdiff>=2) or (ydiff>=2) then
    set_circle_points_diagonally
  else if (ydiff<2) then
    set_circle_points_horizontally
  else
    set_circle_points_vertically;
end;  {if}
```

where (xs,ys) and (xe,ye) is the start and end points of the segment respectively.

Table 6-1. An Algorithm to Determine the Circle Points on Path Segmentation in Method 2

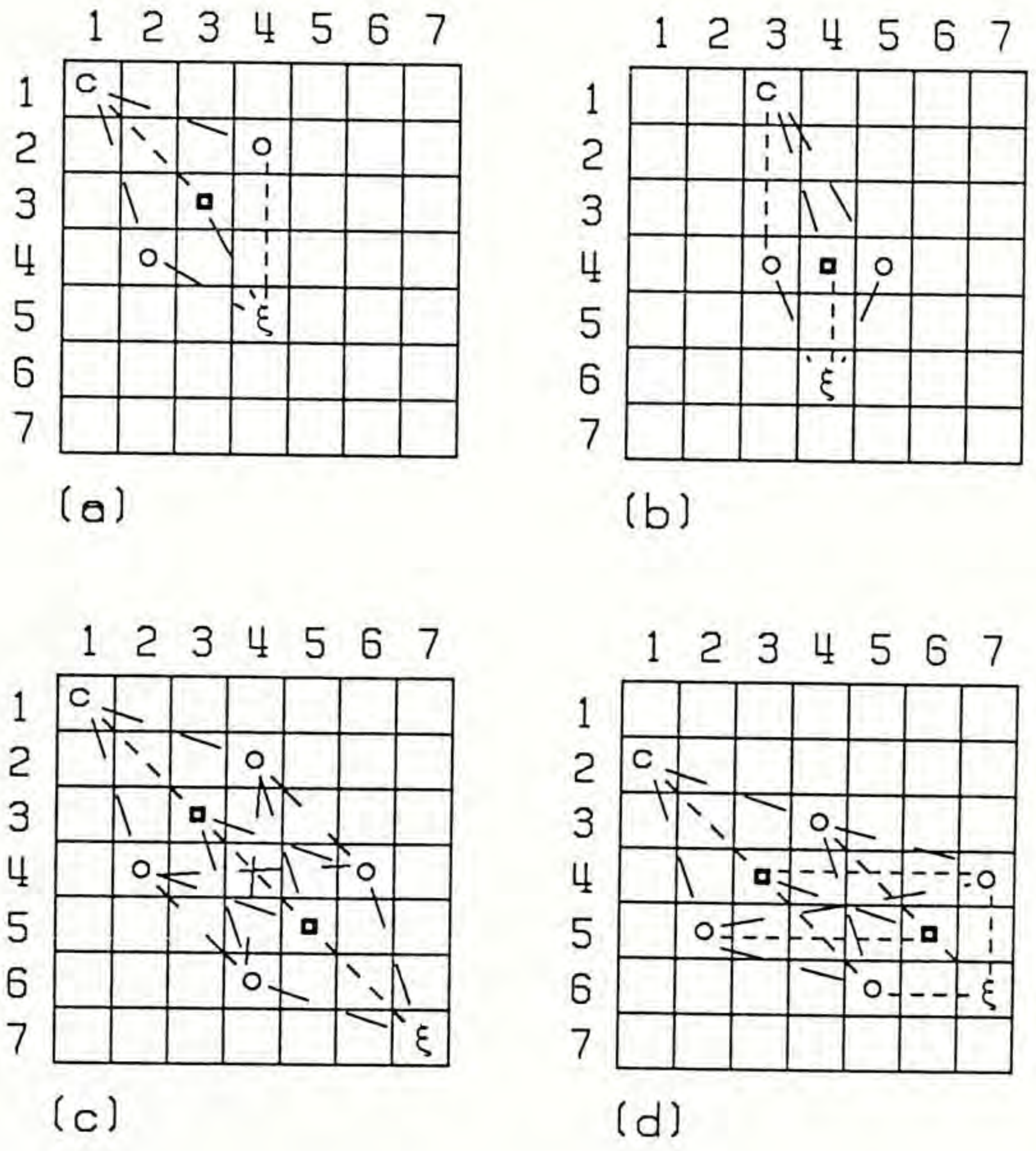
Figure 6-4 Path Segmentation of Method 2

There are more combinations on the linkage of the segments as compared with the method in section 3.2.3.2.3. In figure (a) and (b), there is a level of 3 division points (1 with square and 2 with circles) and each segment have about 3 templates (section 3.2.3.2.2). Hence, the number of combinations is approximately

(number of combinations of path) * (number of combinations of templates in a path)

$= (3)*(3^2)$

$= 3^3$

which is case for a path with $L_{MHT} \geq 6$.

In figure (c) and (d), there are two levels of 3 division points. The number of combinations of path segments is approximately

$(3^2)*(3^3)$

$= 3^5$

which is case for a path with $L_{MHT} \geq 10$.

In general, for a length of L, the number of combinations is given by

$$f(L) = 3^{[2*(L/A.M.) + 1]}$$

where A.M. is the arithmetic mean of the length of a path segment.

Since the arithmetic mean is (4+5)/2 = 4.5,

$$f(L) = 3^{[2*(L/4.5) + 1]} \qquad \text{------------- Eq. 6-5.}$$

Since the average length of path in an array with size n is $(n/2)^{1/2}$, Eq. 6-2 is given by

$$T_{M2} = 3^{[2*(n^{1/2}/6.36) + 1]} \qquad \text{------------- Eq. 6-6.}$$

For an 7X7 array, the average number of combination is $T_{M2} = 27$.

6.2.3 Method 3

An alternative method to determine the propagation path in section 3.2.3.2 is proposed as follows. The propagation path is found out by progressively approaching the target empty space, ξ, instead of segmenting the cross-cut.

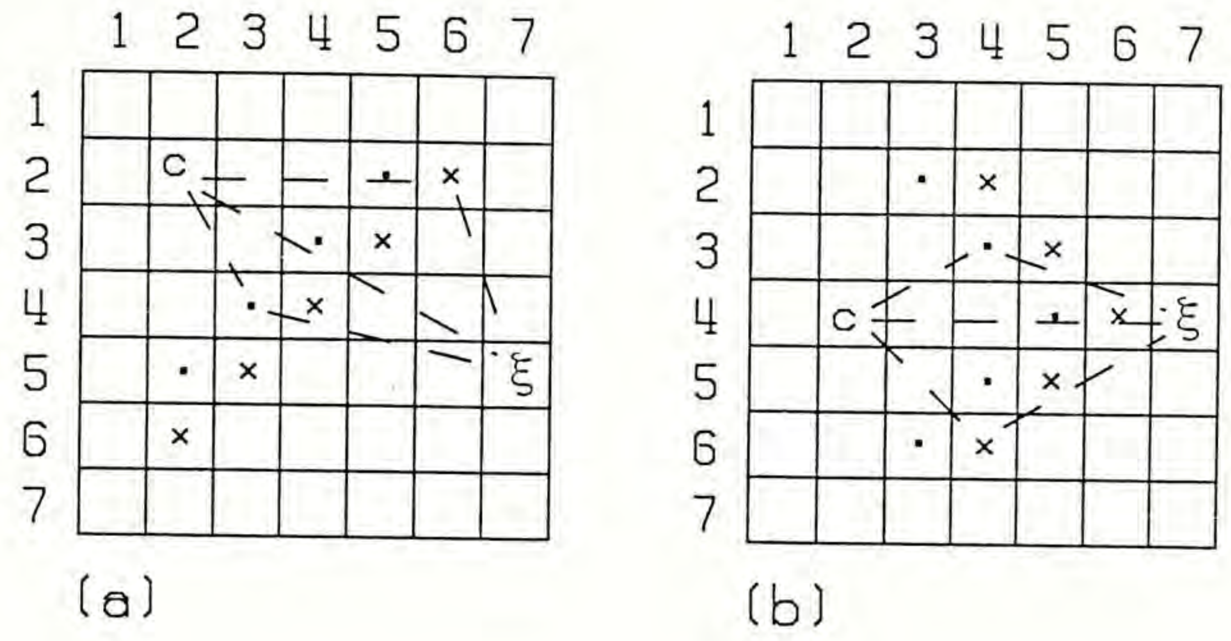

Figure 6-5 Examples on Propagation Path

The optimal propagation path is determined by the following steps:

(1) - A straight line (cross-cut) is drawn between the 2 cells.

- Store the number of "passing" groups; this value is the initial value.

(2) 5 cells at a L_{MHT} of 4 with the one nearest to the cross-cut on the cross-cut are chosen as the middle of the 5 cells. These 5 cells are marked with crosses "x" and those cells with L_{MHT} of 4 and adjacent to the crossed cells are marked with dots ".".

(3) - Check those cells with a cross or a dot

- Store the number of "passing" groups.
- Find the one with the smallest value.
  - If more than 1 path are with the smallest value, the summation of size of the "passing" groups should be taken into account as well.
  - If that is still incapable of determining the difference, additional straight lines are drawn between the cells concerned and the empty cell, the one with shorter L_{MHT} is chosen.

(4) Check cells with a circle inside, see whether they are in the same groups with their adjacent cells. In the example on the Figure 6-6 , there are two criteria on the choice of the optimal path:

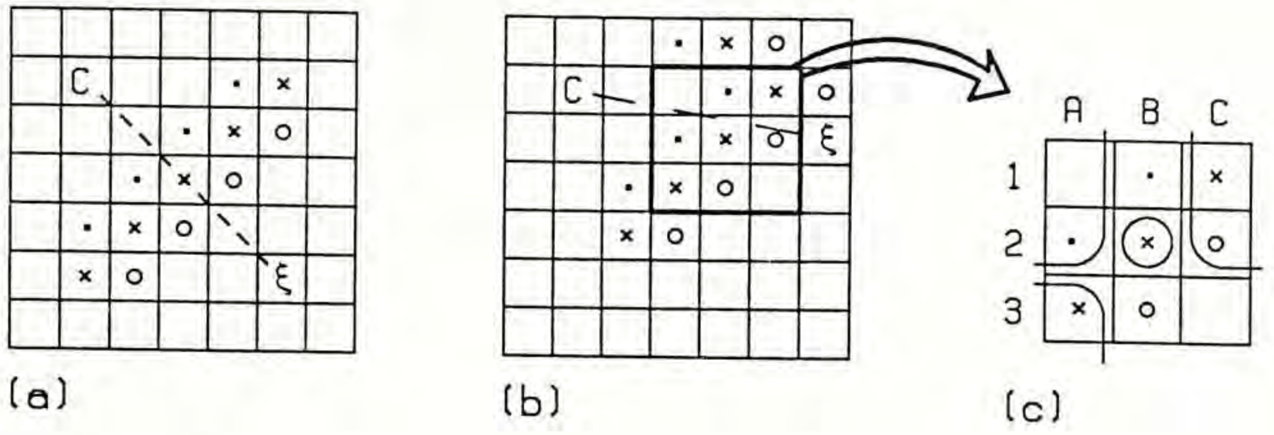

Figure 6-6 Examples on the Choice of the Next Start Point on Path Determination

(a) If C1 is the path that passing through the smallest no. of groups then C2 will be chosen as the next starting point since C1 and C2 are in the same group.

(b) Let A2 be the end of the chosen path and with G_{A2} groups passed while C1 with G_{C1}.

```
Now, C1 and C2 are in the same group,
if $G_{C1} = G_{A2} + 1$ then if summation of size (C2) < summation of size (A2) then
                                  C2 chosen
                              else A2 chosen
else A2 is still the chosen one
```

```
(5) if Chosen cell < > empty space then
        the process starts again          {chosen cell --> starting point }
    else end.
```

Remark:

(i) When a L_{MHT} of 3 is used instead of 4, we are actually checking each and every possible path. It, in some way, is a more general check but on the other hand, the more iteration or shorter the path is, the more likely that we may omit some "good" path with complicated initial start but straightforward ending.

Without loss of generality, choosing 5 cells on each search is also appropriate for cell arrays with size greater than 7X7.

(ii) When iterative paths are selected, linkage of all the optimal path segments is not necessarily the optimal path. The case can be illustrated by the mathematical equation on vector summation:

$| \sum \mathbf{v}_i | \leq \sum |\mathbf{v}_i|$, where $\mathbf{v}_i$ is a vector. ------------------ Eq. 6-7.

The above method can be implemented with recursive subroutines [Monro87, ScWePe82]

The average number of execution cycles of this method is:

$T_{M3} = 9^{[(n^{1/2}/6.36) + 1]}$ cycles ------------------ Eq. 6-8.

6.2.4 Comparison on Execution Time of the Four Methods

Comparison on the execution time of IAPA (Incremental Alteration Placement Algorithm), Method 1, Method 2 and Method 3 is shown in the following figure. It is noticed that IAPA is the one with the least number of execution cycles.

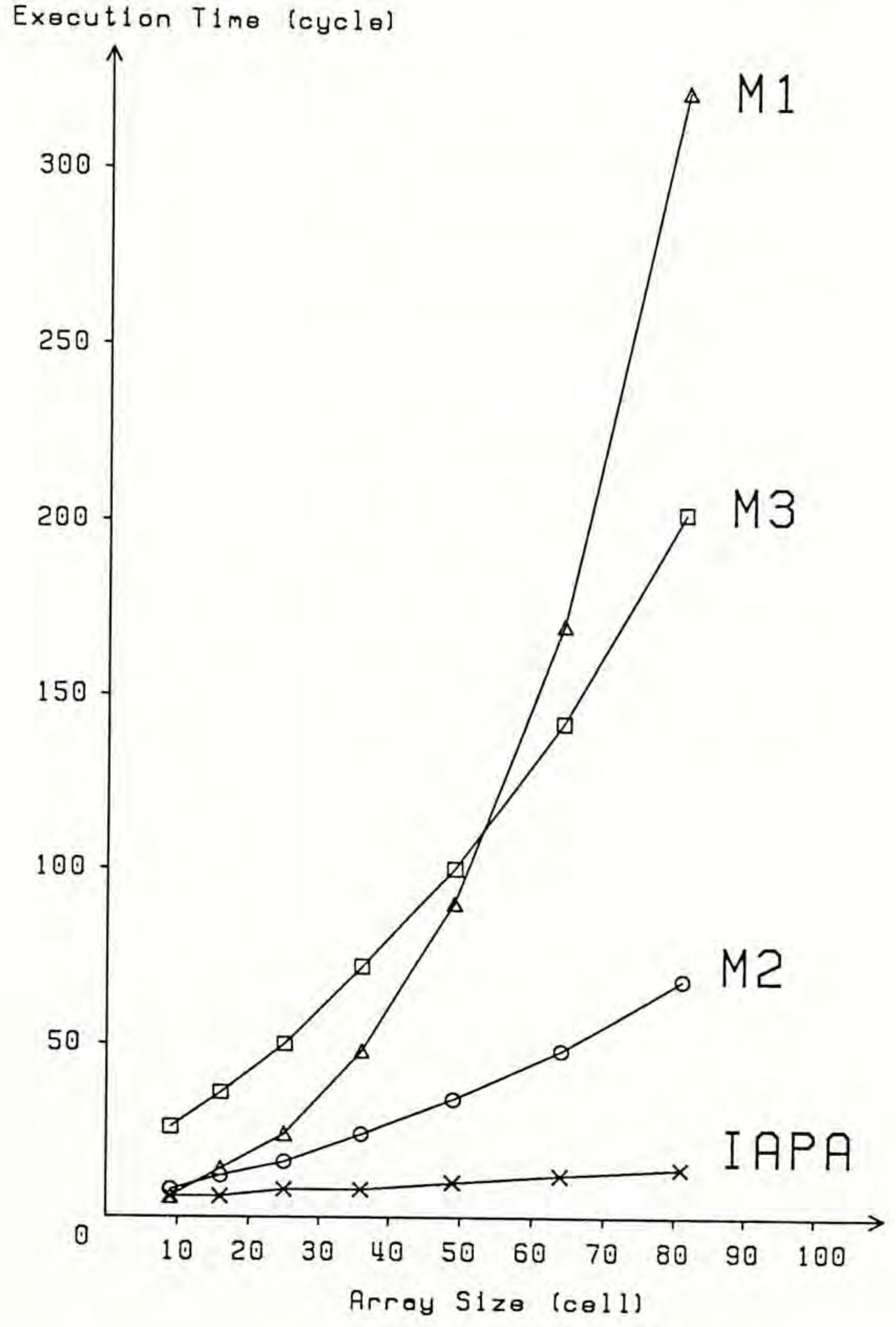

Figure 6-7 Comparison on Execution Time of IAPA, M1, M2 and M3

6.3 Wiring Optimization

In section 4.1, it was stated that one of the function of the affinity clustering phase is to minimize crossing congestion across channels. The method is described as follows:

The number of crossing count can be estimated by the following formula (criteria) : Consider the example in figure 6-7. The i-th net is represented by (s_i,t_i), where s and t are the start and end points of the net, and are real numbers.

At channel 1, considering only net 1 and net 2, t_1 and s_2 are on the same column and it will increase the channel width by 1. In algorithmic form,

```
if (($s_1 \le t_2$) and ($s_2 \le t_1$) and ($s_1 <> t_1$) and ($s_2 <> t_2$)) then
  increment cross_count[channel_num];
```

Hence, the cross count formed by the pair {net 1, net 2} is 2. The cases of {net 2, net 3} and {net 3, net 4} are similar, and the cross counts are 2 as well. Since the channel width is the maximum of the cross counts, the channel width = 2.

At channel 2, the maximum cross count occurs at the third column and is 3. Hence, the channel width = 3.

At channel 3, net 1 and net 3 are pure vertical connections and thus $s_1=t_1$, $s_3=t_3$. Even though there is crossing at net 2 and net 3, the channel width should not be increased. Hence, the channel width = 1.

In general, the algorithm is as follows:

```
for j:= 1 to n do
begin
  at channel j,
  arrange end points of nets such that $s_1<s_2<s_3$ ... $<s_n$,
                                       and $s_i>t_i$ $\forall$ i ;
  if (($s_i \le t_j$) and ($s_j \le t_i$)) and ($s_i <> t_i$) and ($s_j <> t_j$)) $\forall$ j > i then
    increment cross_count[channel_num];
  channel_width[channel_num]:= max{cross_count};
end; {for}
```

Table 6-2. An Algorithm on Channel Width Estimation

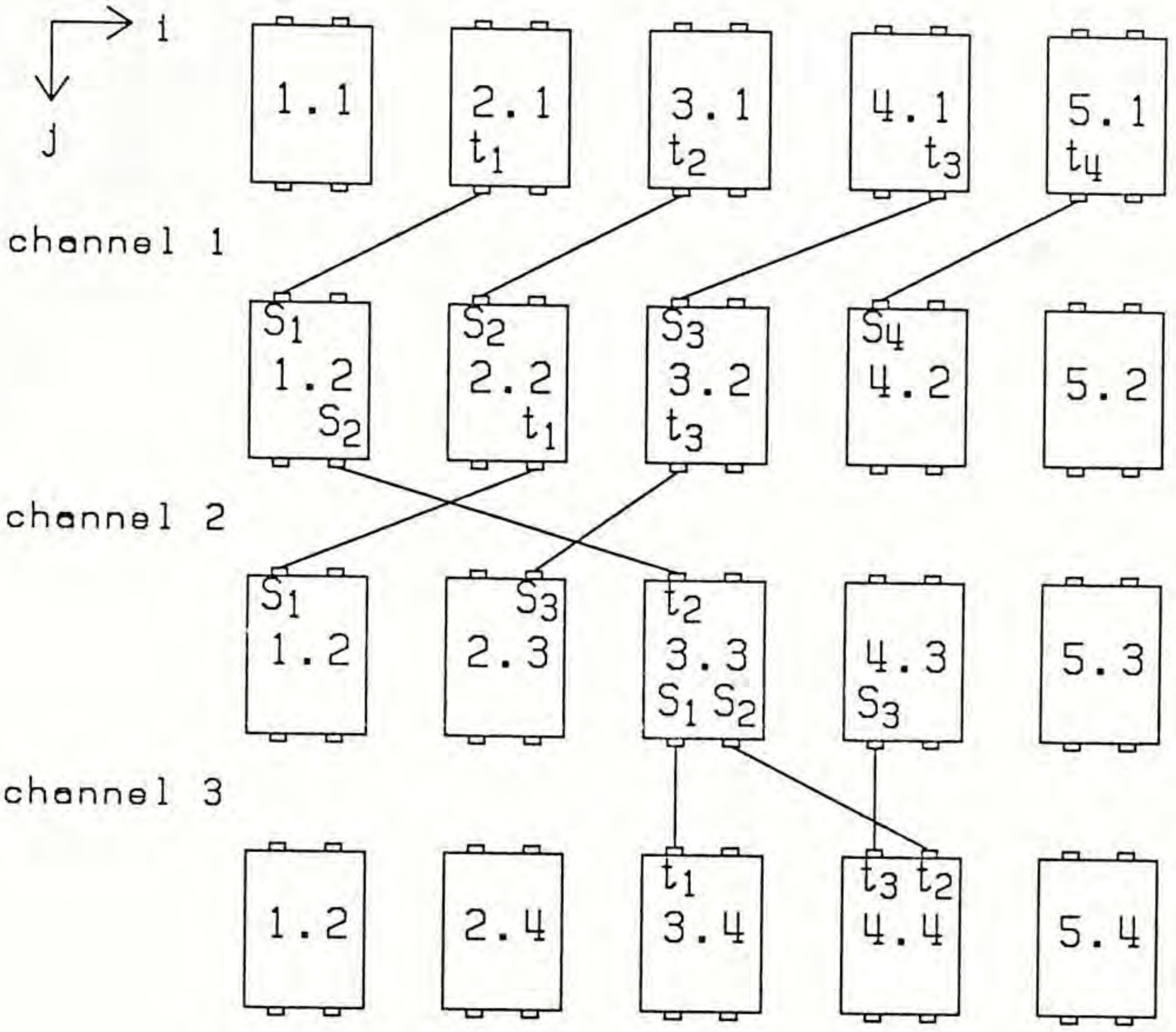

Figure 6-8 A Wiring Model on Macrocell Array

It is worth noting that the coordinates of x and y (or i and j) can be readily obtained from the data (coox and cooy) on each record of the cluster tree.

6.3.1 Data Structure

Figure 6-9 shows how data are accessed by an array for each axis of the chip. "PTR" stands for pointer. The entry in the Y array for a given y coordinate points to the head of the list for all data at that y coordinate. For example, wiring information at y_n coordinate is accessed through the pointer in to the storage pool.

For processing efficiency, entries are sorted according to their starting coordinates (i.e. $s_1 < s_2 < s_3 \; \ldots \; < s_n$). Feature type states the type of crossing of the net (i.e. case in channel 1, 2 or 3). Feature PTR is for the case of non-planar crossings (e.g. cyclic constraint). It is optional.

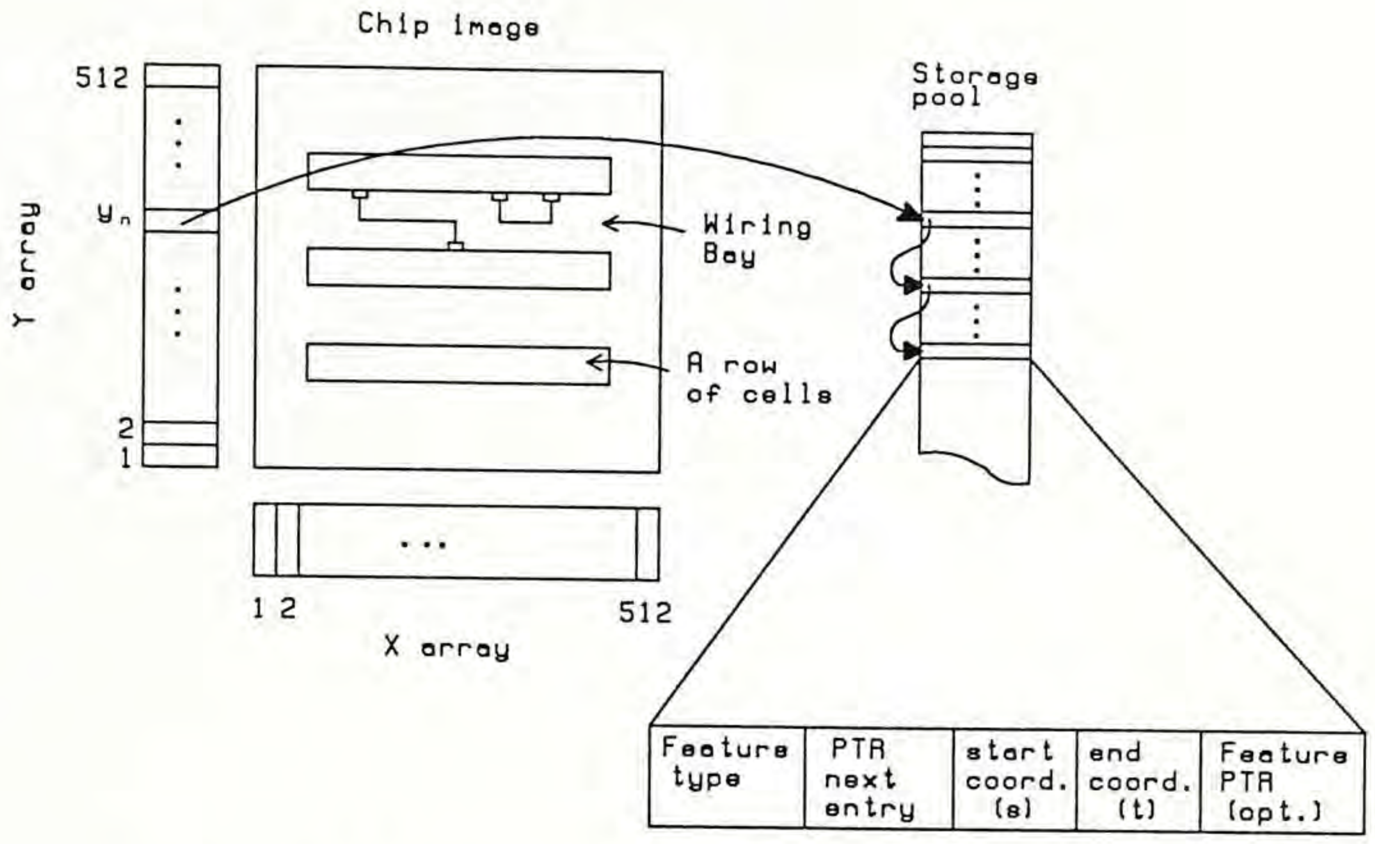

Figure 6-9 Data Structure Scheme for Modeling Chip Image

6.3.2 Overlapping and Separate Bounding Boxes

Alternatively, the number of crossings can be determined by the number of overlapping of bounding boxes of connections at each channel.

In Figure 6-10,bounding boxes A and B are separate while C and D are overlapped. At channel 1, the channel width should be increased due to the overlapping boxes. At channel 2, however, the width of channel should not be increased because the boxes are separate .

However, the case is different in figure 6-11. Although the bounding boxes X and Y are overlapping, it occurs between two channels. In this case, width of neither channel should be increased.

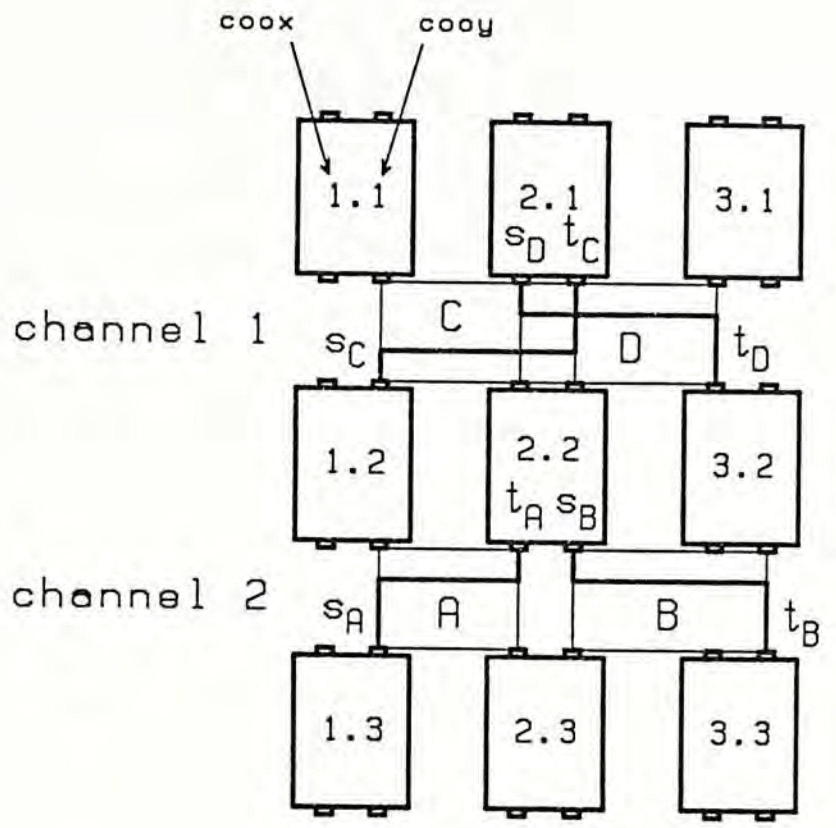

Figure 6-10 Case of Overlapping and Separate Bounding-boxes on Single Channels

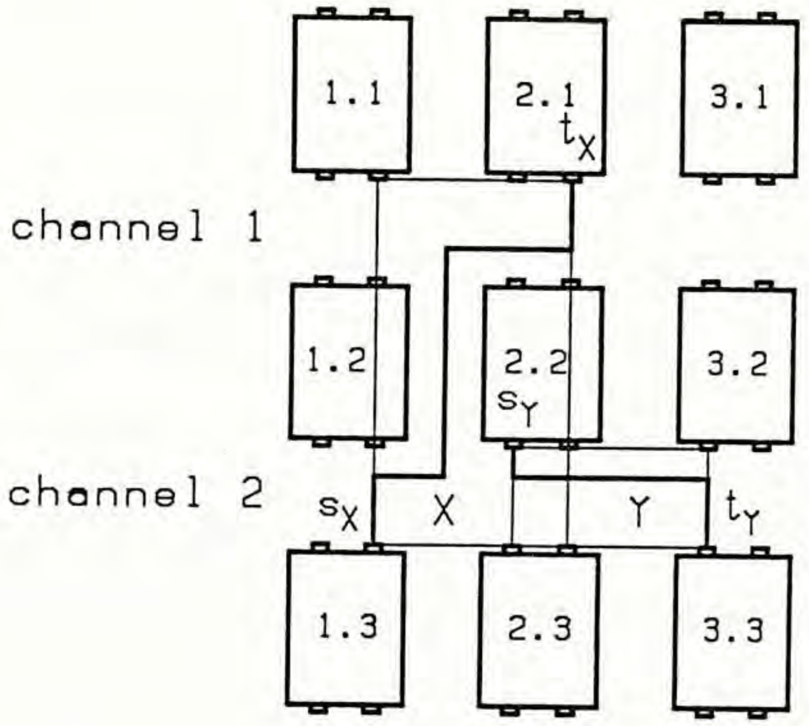

Figure 6-11 Case of Overlapping Bounding-boxes on Two Channels

To optimize wiring of a circuit, we should minimize number of crossings at each channel so that the channel widths and thus the interconnection lengths are minimized.

6.4 Generalization of the Data Structure

The data structure of group list and cluster tree described in section 4.3 is only a simplified model of a circuit. For a more detailed representation of a circuit, the record structure should include information on a few items: (1) cell type of the element, (2) more detailed information on the adhesive elements, (3) indicator on block macrocell, and (4) indicator on the critical path.

The record structure of in figure 6-12 is proposed to replace the element variables in Table 4-2 and 4-3 for the generalization. The "bulkrec" record in Table 4-2 and "zisprec" record in Table 4-3 will be used as usual the "head of a group".

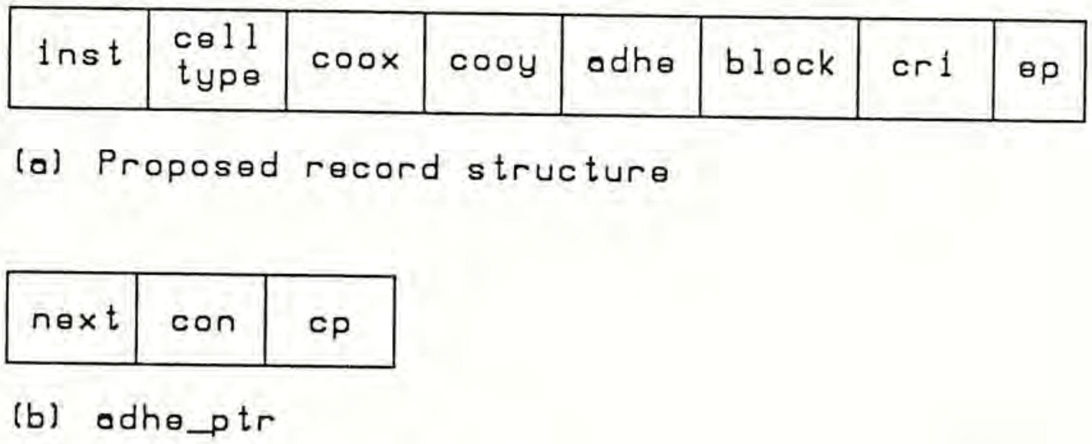

Figure 6-12 Generalized Record Structure

In the record at figure (a), "inst", "coox", "cooy" and "ep" are the same as those declared in Table 4-2 and 4-3. Four new variables (cell_type, adhe, block, and cri) and a new pointer variable (adhe_ptr) are introduced in the record.

1. cell_type : integer; {indicates the type of cell an element belongs to}

2. adhe : adhe_ptr; {(as in figure b) to record ALL the adhesive elements to the instance. It is more detailed than the two variables ("pri" and "sec") in the "treerec" record.}

```
[ adhe_ptr:- next : adhe_ptr;   {pointer to the next adhesive element}          ]
[            con  : integer;    {connectivity of the adhesive element}          ]
[            cp   : integer in group list;  {instance of the adhesive element}  ]
[                 : tptr in cluster tree;   {points to the adhesive element}    ]
```

3. block : set of integer in group list; {set of instances of elements in a block}
   : tptr in cluster tree; {points to other element(s) of a block, if any}

4. cri : integer in group list; {instance of the next element on the critical path}
   : tptr in cluster tree; {points to the next element on the critical path}

Details of the above items will be discussed in the following sub-sections.

6.4.1 Cell Types

All the macrocells can be classified into types of cell. Three of the cell types are shown in the following figure:

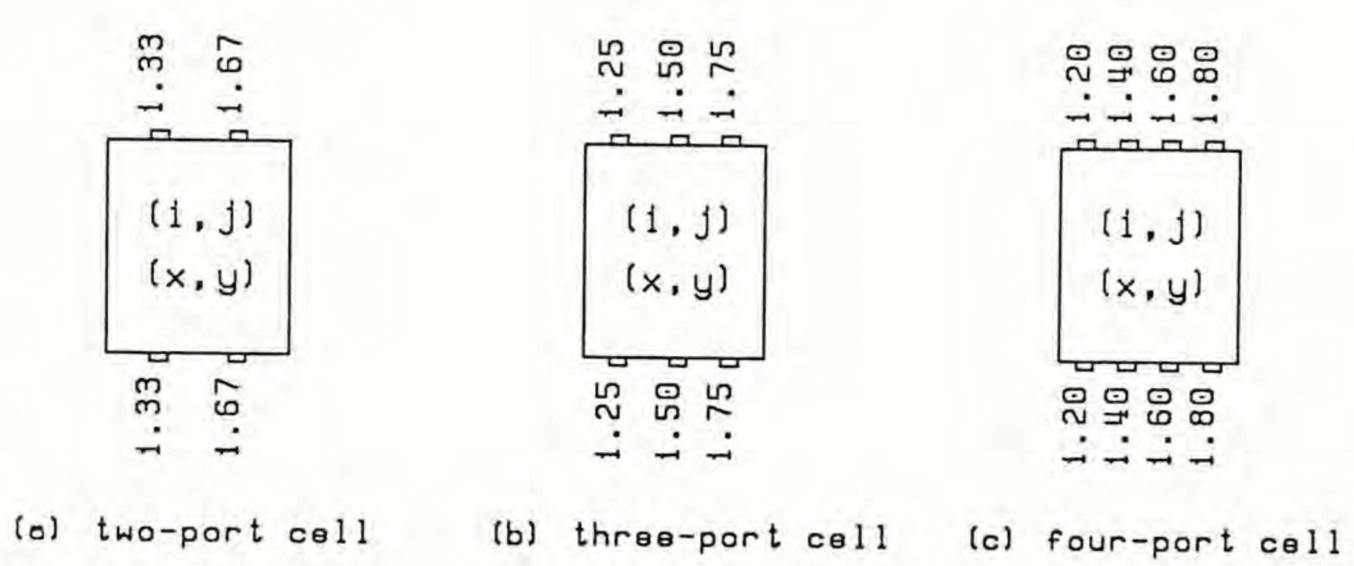

Figure 6-13 Cell Types in the Generalized Data Structure

Cell types not only indicates the classification of cells but also give the locations of each ports on the cells. Port locations together with net connections give information on the wiring in each channel (section 6.3). The locations are given by the relative position of a port and the absolute position of the cell. For example, the relative position of the two ports in a two-port cell are 0.33 unit and 0.67 unit respectively and the absolute position of the cell is (i, j). Then, the locations of the two ports in the i-axis are:

$i + 0.33 = i.33$, and

$i + 0.67 = i.67$ respectively.

The cases in other types of cell are similar.

6.4.2 Adhesive Attributes

Unlike the element records in Table 4-2 and 4-3, there is no limit on the number of adhesive elements in the generalized data record. The way to link the adhesive elements is illustrated in figure 6-14.

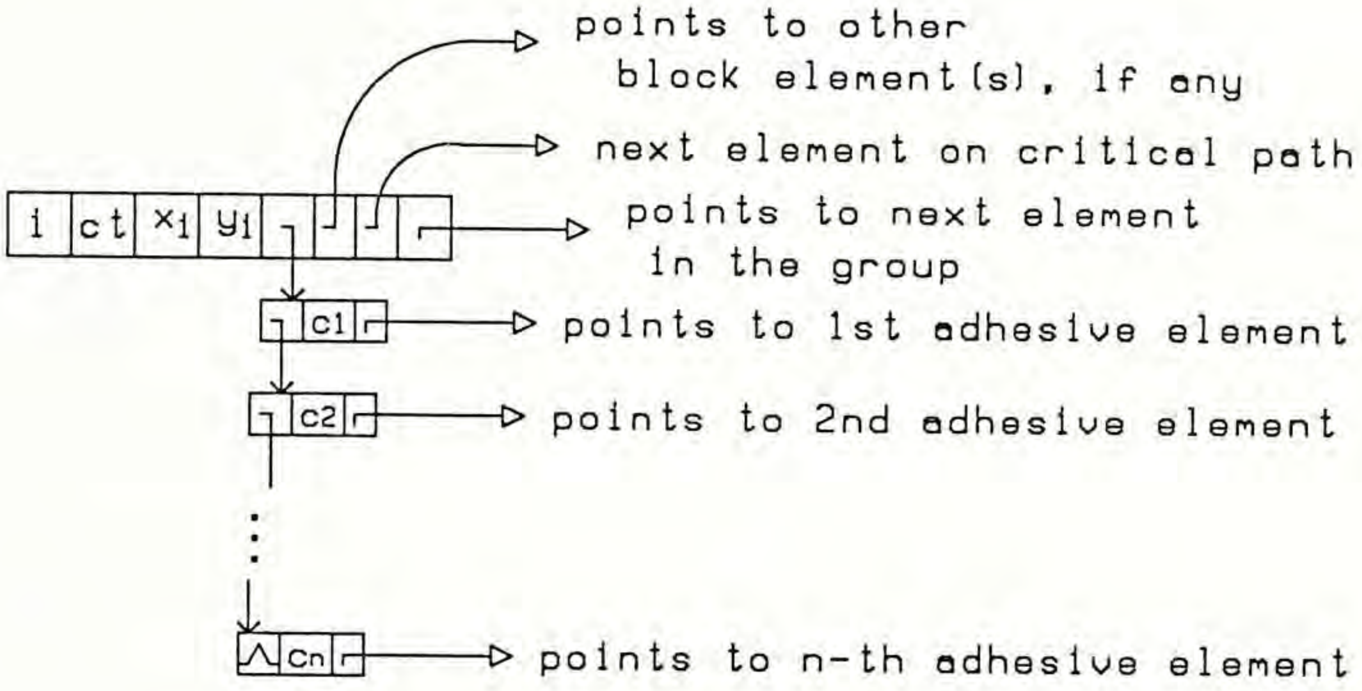
points to other block element(s), if any
next element on critical path
points to next element in the group
points to 1st adhesive element
points to 2nd adhesive element
points to n-th adhesive element
i
ct
x1
y1
c1
c2
cn

Figure 6-14 Linkage to Adhesive Elements by the "adhe" Pointer

In the above figure, the element record with instance "i" is shown. "ct" stands for the cell type of the element. (x_i, y_i) are the coordinates. The pointers of the "adhe_ptr" records link the "tptr" record of the adhesive elements and c1, c2, ... , cn are the connectivities of the corresponding adhesive elements. As in section 4-3, the "ep" pointer links the next element in the group. The "block" pointer links the block element(s) and the "cri" pointer links the next element on critical path.

6.4.3 Blocks Representation

Some logic parts (e.g. multiplexers, JK flip-flops, arithmetic logic unit) are composed of several macrocells and they will form blocks of macrocells with certain sizes and shapes. To distinguish these blocks in the circuit so that the placement alteration is done properly, one more attribute should be added to the element record, naming "block", to identify existence of the blocks.

If such blocks are added to the circuit, we should find a space with the same size and shape to the block so that placement is possible. In the case that any block of cells exists on the path of propagation, each move (shift of cell) should be done after checking the "block" attribute of all the elements related, so as not to break the structure of any block. Thus, checking "block" attributes of all the elements related is necessary before allocating empty spaces and executing additions.

The attribute "block" is suggested to be a pointer for linkage of element(s) in the same block. Since pointer is a dynamic variable, direct access is possible and it will facilitate the computation.

6.4.4 Critical Path Adjustment

Consider the following circuit:

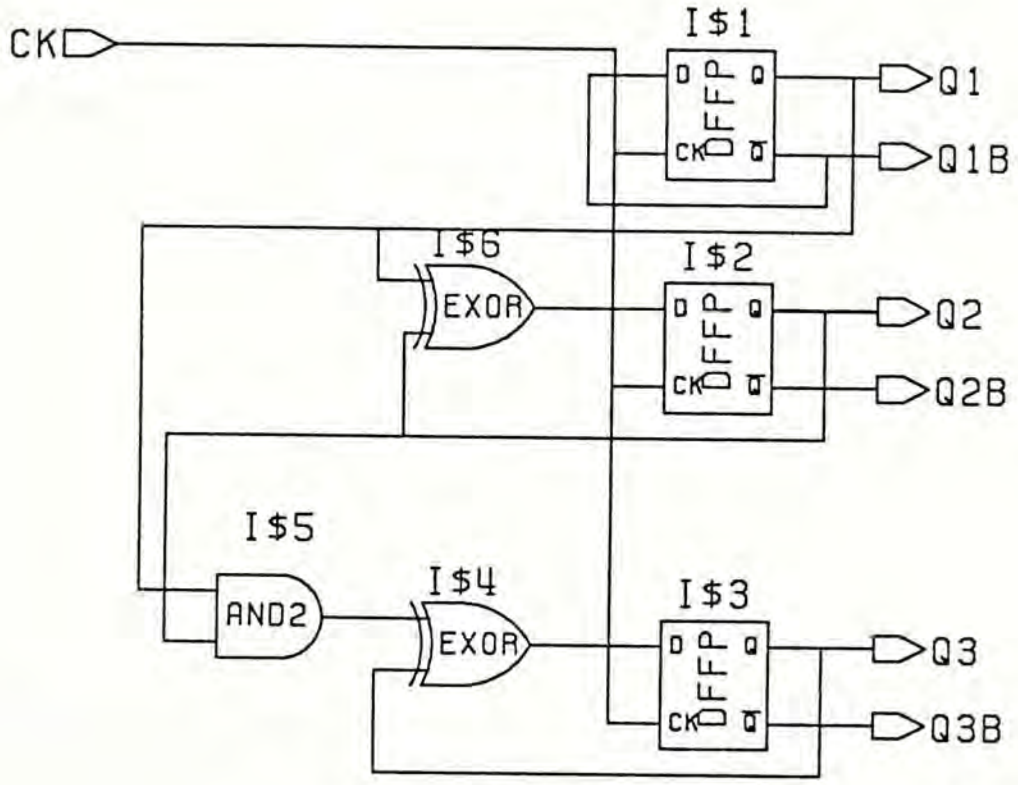

Figure 6-15 A 3-bit Synchronous Counter

The connection lists of the circuit (algorithm in section 3.1.1) are:

Directed Connection List	Undirected Connection List	Fan-out Connection List	Link-related Connection List
1,5: 1	1,5: 1	5,6: 2	1,5: 1
1,6: 1	1,6: 1		1,6: 1
2,5: 1	2,5: 1		2,5: 1
2,6: 1	2,6: 2		2,6: 2
3,4: 1	3,4: 2		3,4: 2
4,3: 1	4,5: 1		4,5: 1
5,4: 1			5,6: 2
6,2: 1			

Table 6-3. Connection Lists of the 3-bit Synchronous Counter

The grouping is N = {1, 2, 5, 6} U {3,4}. The cluster tree together with connectivity attributes are shown in figure 6-16.

In the circuit, the longest delay path is from input CK, via I$1, I$5, I$4, and I$3 to output Q3 (I$x is the notation for Instance x). This critical path can be represented by the sequence {1, 5, 4, 3}, where the integers denote the instances. In the cluster tree of figure 6-16, the critical path is outlined by bold lines. Delay on this critical path is given by the following equation.

Let Delay(e, s) denotes the delay from s to e, where s, e are start point and end point respectively.

$$\text{Delay}(3, 1) = (|x_3-x_4| + |x_4-x_5| + |x_5-x_1|) + (|y_3-y_4| + |y_4-y_5| + |y_5-y_1|)$$

To reduce the delay on the critical path, we should adjust the location of elements on the critical path such that the above value is minimized.

In general, the delay on a critical path $\{e_1, e_2, \dots, e_n\}$ is:

$$\text{Delay}(e_n, e_1) = \sum (|x_{e\,i+1}-x_{e\,i}| + |y_{e\,i+1}-y_{e\,i}|) \qquad \text{------------ Eq. 6-9.}$$

The sequence $\{e_1, e_2, \dots, e_n\}$ can be traced by the "cri" pointers on the elements. The values on x and y coordinates can be read from the "coox" and "cooy" attributes. In algorithmic form, calculation on critical path delay is:

```
p:= start_point;
delay:= 0;
while (p^.cri<>nil) do
  begin
    delay:= delay + abs(p^.cri^.coox-p^.coox)
              + abs(p^.cri^.cooy-p^.cooy);
    p:= p^.cri
  end;  {while}
```

Table 6-4. Algorithm of Critical Path Delay Calculation

And, the aim of critical path adjustment is to minimize Delay(e_n, e_1).

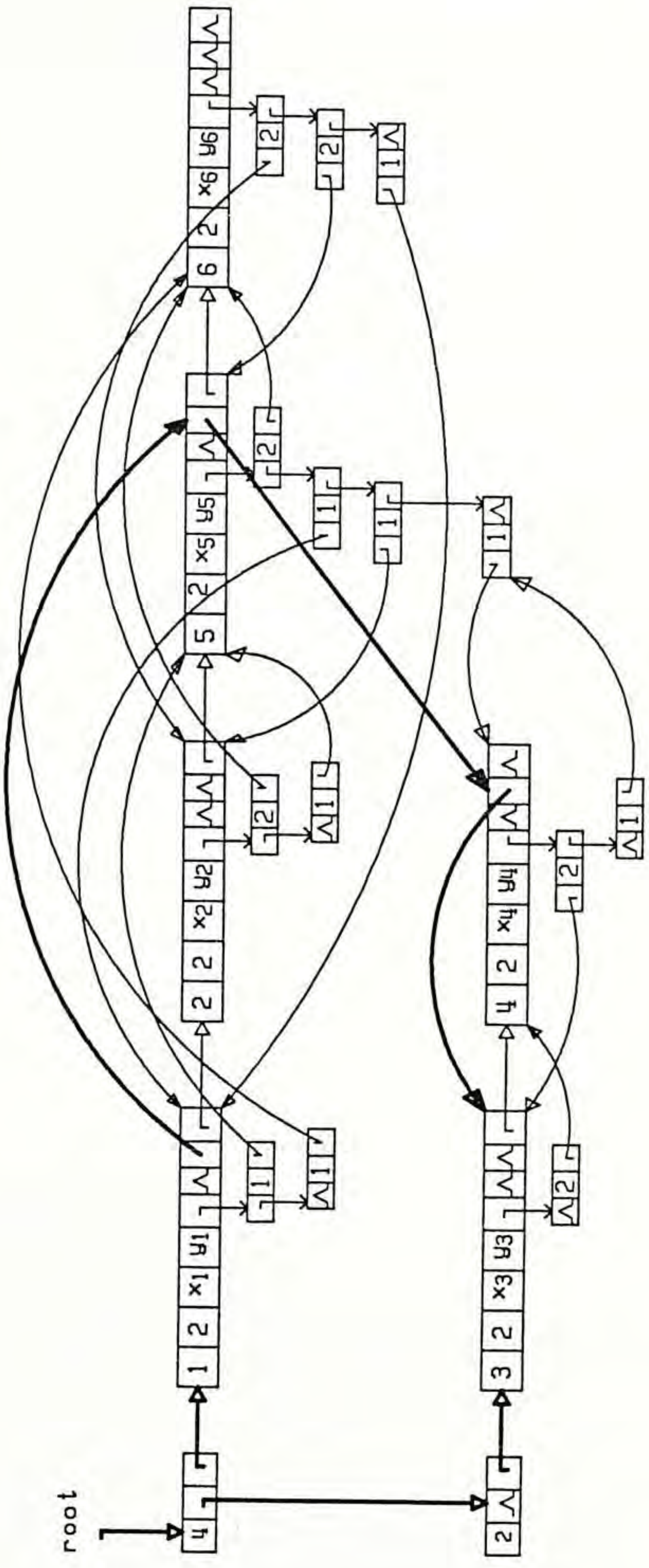

Figure 6-16 Generalized Cluster Tree of the 3-bit Synchronous Counter

6.4.5 Total Interconnection Length Estimation

In the cluster tree of section 4.3, the total interconnection length of a circuit can be estimated by the following equation:

Total Interconnection Length
$= \sum (| p\hat{}.pri\hat{}.coox - p\hat{}.coox | + | p\hat{}.pri\hat{}.cooy - p\hat{}.cooy |$
$+ | p\hat{}.sec\hat{}.coox - p\hat{}.coox | + | p\hat{}.sec\hat{}.cooy - p\hat{}.cooy |)$ ----------- Eq. 6-10.
where "p" is the tree element variable (Table 4-3).

However, the above term is only an rough estimation because only the two mostly connected elements of each instance are taken into account. With the generalized data structure, the total interconnection length can be estimated more accurately:

Total Interconnection Length
$= \sum \sum (| ap\hat{}.cp\hat{}.coox - p\hat{}.coox | + | ap\hat{}.cp\hat{}.cooy - p\hat{}.cooy |)$ -------- Eq. 6-11.
where "ap" is the adhe_ptr variable (Figure 6-11).

In algorithmic form, the above equation can be implemented as follows.

```
zp:= zisp_root;
total_length:= 0;
while (zp<>nil) do
begin
   p:= zp^.atom;
   while (p<>nil) do
   begin
      ap:= p^.adhe;
      while (ap<>nil) do
      begin
         total_length:= total_length
            + ap^.con*(abs(ap^.cp^.coox-p^.coox)+abs(ap^.cp^.cooy-p^.cooy));
        ap:= ap^.next;
      end; {3rd while}
      p:= p^.ep;
   end; {2nd while}
   zp:= zp^.molecule;
end; {1st while}
```

Table 6-5. Algorithm on Total Interconnection Length Estimation

6.5 A New Placement Algorithm

A new algorithm on the initial placement for larger circuit is discussed in this section . Since this new algorithm will utilize the constructed cluster tree data structure in the Affinity Clustering phase, it may solve placement problems on grid plane with size up to 10X10 (estimated from the declaration on array variables for a program with this algorithm). The flowchart of the algorithm is as follows:

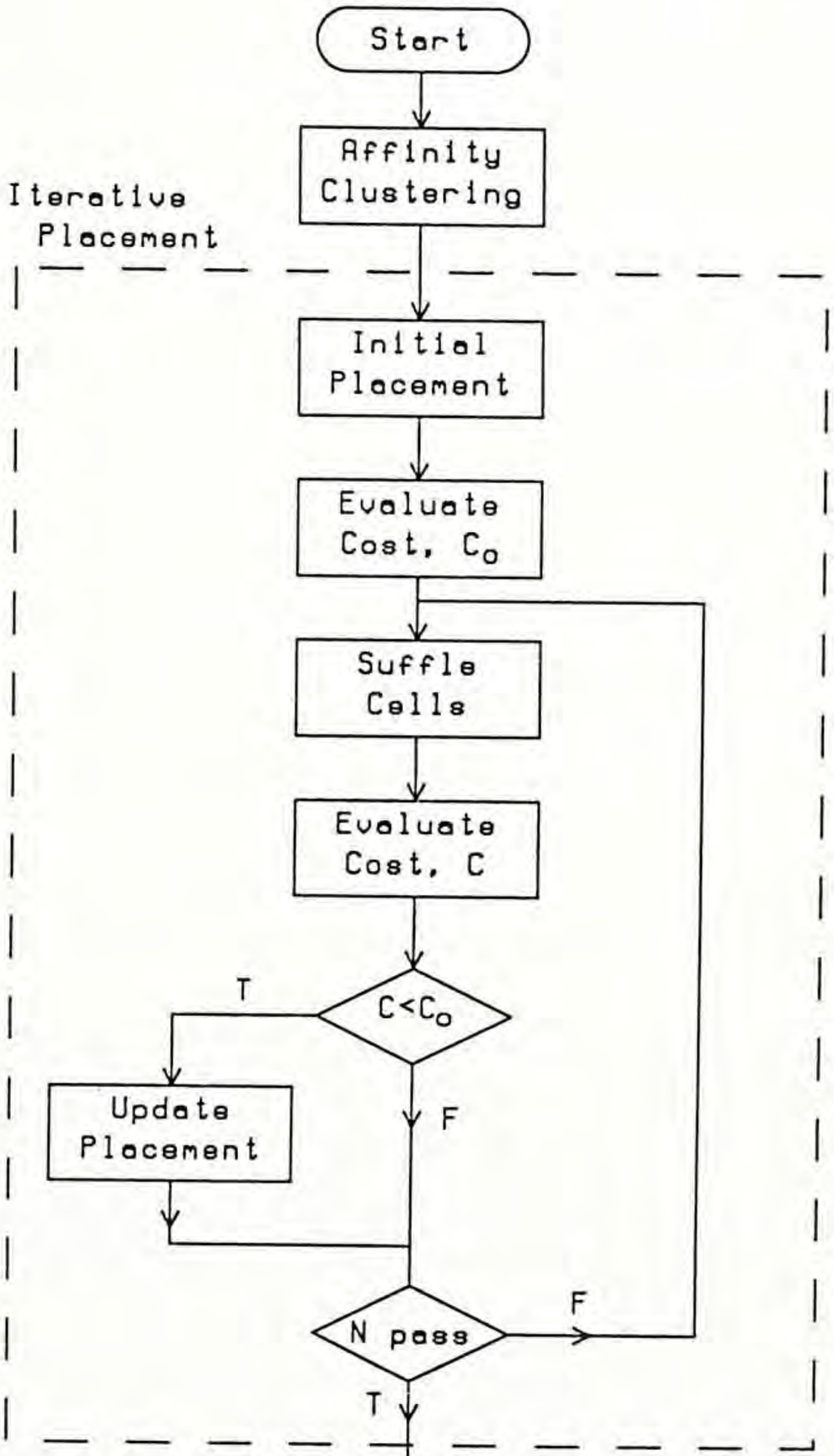

Figure 6-17 Flowchart of the New Placement Algorithm

In this algorithm, the affinity clustering phase is done before iterative placement. The affinity clustering phase is classified as a partitioning task as those described in section 2.1.

The iterative placement composed mainly of 5 step:

1. The initial placement step is one which assign location for element by their grouping. Elements belonging to the same group will be placed around an area. Each time an element is placed at the array, its interconnection length to the adhesive elements should be measured so as to minimize the wiring length.

2. The cost is evaluated by three parameters. The first one is wiring optimization. Width of each channel should be minimized. The measure of channel widths was suggested in section 6.3. The second parameter is the total interconnection length and the estimation algorithm was stated in section 6.4.5. The third parameter is the delays in the critical paths (section 6.4.4).

3. Cell shuffling is done by interchanging two or more adjacent cells. The selection of these cells are based on information in the cluster tree. Usually, cells with higher connectivities would be selected in the beginning of the iteration so that the change in cost value is more significant. However, at the later stage of iteration, shuffling cells with lower connectivities is more likely to reduce the cost value.

4. Evaluation of the new cost after cell shuffling. If the value of the new cost is lower, the placement will be updated. Else, another cell shuffling would be perturbed.

5. Step 4 will be repeated until the placement reach an equilibrium state or the number of iteration reach a pre-defined value, N.

This placement algorithm is recommended because the cluster tree formed by the affinity clustering phase gives a detailed data base on the circuit, and operations on the data structure is flexible and will save both computation time and computer memory. The resulting layout by this algorithm will be similar to that by the Simulated Annealing algorithm in Appendix since the evaluation in cost and method of cell shuffling in the two algorithms are similar.

6.6 An Alternative Method on Element Allocation

The method used in Element Allocation in section 3.2.4 shifts elements along the optimal propagation path. An alternative method which uses information on the cluster tree is suggested. Considering the slots among the two end points in the six templates in figure 6-18.

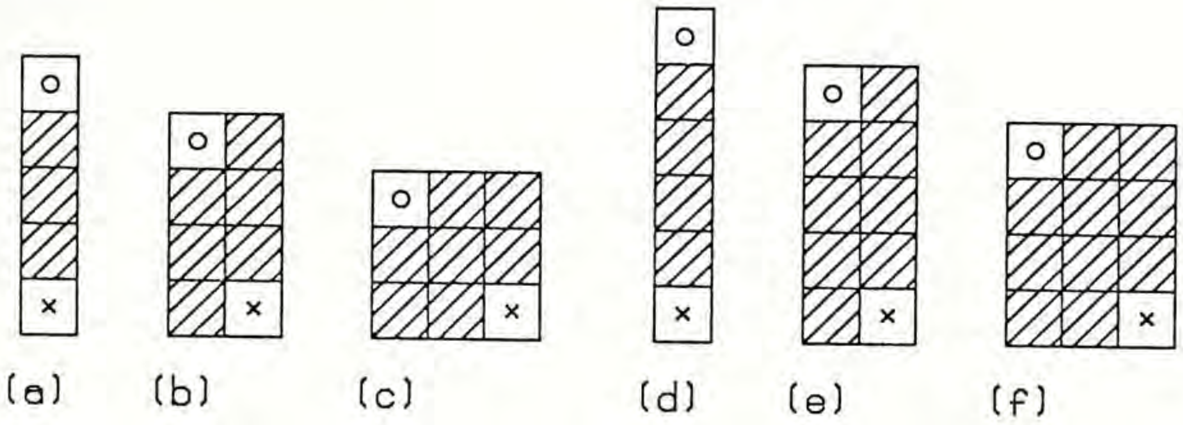

Figure 6-18 Slots Among End Points in the Six Templates

The above 6 templates were defined in section 3.2.3.2. The shaped parts are the slots among the end points "o" and "x". (a), (b), and (c) are the templates for L_{MHT} (Manhattan Length) of 4, while (d), (e), and (f) are those of L_{MHT} of 5. Mathematically, the set of these "enclosed slots" is defined by

$$S_{ES}(o, x) = \{(i, j)\} - \{(i_o, j_o), (i_x, j_x)\},$$
$$i \in \{i_o, \ldots, i_x\} \text{ and } j \in \{j_o, \ldots, j_x\}. \qquad \text{------------ Eq. 6-12.}$$

At figure (a), there are 3 slots between "o" and "x". However, since only an average of 1.5 cells may belong to the same group, there are 1.5 combinations for a change in placement between these two end points. The number of combinations of placement change and the number of occurence of the six templates are summarized in Table 6-6.

Template	a	b	c	d	e	f
No. of Enclosed Slots, N_{ES}	3	6	7	4	8	10
Avg. no. of combination, N_C	1.5	3	3.5	2	4	5
No. of Occurrence, N_O	4	8	4	4	8	8

Table 6-6. Statistics on the Six Templates

The occurrence of the above templates is illustrated by figure 6-19.

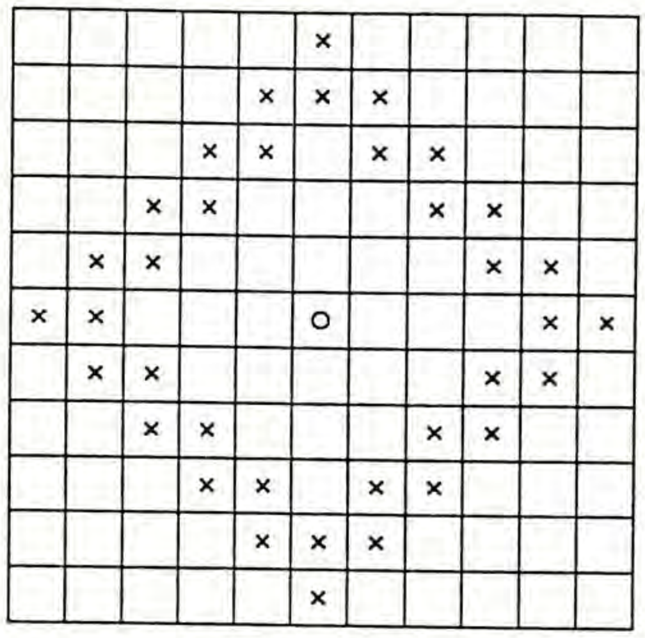

Figure 6-19 Occurrence of the Six Templates

Crosses on the outer ring are locations with L_{MHT} = 5 units to the circle, while those on the inner ring with L_{MHT} = 4 units. There are totally 40 locations with L_{MHT} = 4 or L_{MHT} = 5 to the circle. However, the numbers of occurrence of the templates are not equal. For example, there is 8 cases in which template (b) matches with the space, but there is only 4 case in which template (a) matches.

The average number of combinations of each set of templates is given by

$$\mathrm{Avg}(L_{MHT}) = \sum (N_C * N_O) / \sum N_O \qquad \text{------------- Eq. 6-13.}$$

The average number of combinations of templates a, b, and c is given by

$$\begin{aligned}\mathrm{Avg}(4) &= (1.5\mathrm{x}4 + 3.5\mathrm{x}4 + 3\mathrm{x}8) / 16 \\ &= 2.75\end{aligned}$$

The average number of combinations of templates d, e, and f is given by

$$\begin{aligned}\mathrm{Avg}(5) &= (2\mathrm{x}4 + 4\mathrm{x}8 + 5\mathrm{x}8) / 24 \\ &= 3.33\end{aligned}$$

As an average, the number of combinations of the six templates is

$(2.75+3.33)/2 = 3.04$

and the average number of the combination in an array with size n is:

$N_{com}(n) = 3.04*[(n/2)^{1/2}/4.5]$

$\Rightarrow N_{com}(n) = 0.478n^{1/2}$ ------------- Eq. 6-14.

The values of $N_{com}(n)$ for n=9 to 100 cells is given as follows:

n	9	16	25	36	49	64	81	100
$N_{com}(n)$	1.43	1.91	2.39	2.87	3.35	3.82	4.30	4.78

Table 6-7. No. of Combination in Arrays with Size from 3X3 to 10X10

The values of $N_{com}(n)$ for n=100 to 10000 cells is given as follows:

n	100	400	900	1600	2500	3600	4900	6400	8100	10000
$N_{com}(n)$	4.8	9.6	14.3	19.1	23.9	28.7	33.5	38.2	43.0	47.8

Table 6-8. No. of Combination in Arrays with Size from 10X10 to 100X100

Plots of the above values is shown in figure 6-20 and 6-21. Since $N_{com}(n)$ is proportional to the square root of n, it is noticed that an average of only 47.8 combinations is need in the change of placement in an 100X100 macrocell array. Thus, the algorithm is also applicable for large scale circuits.

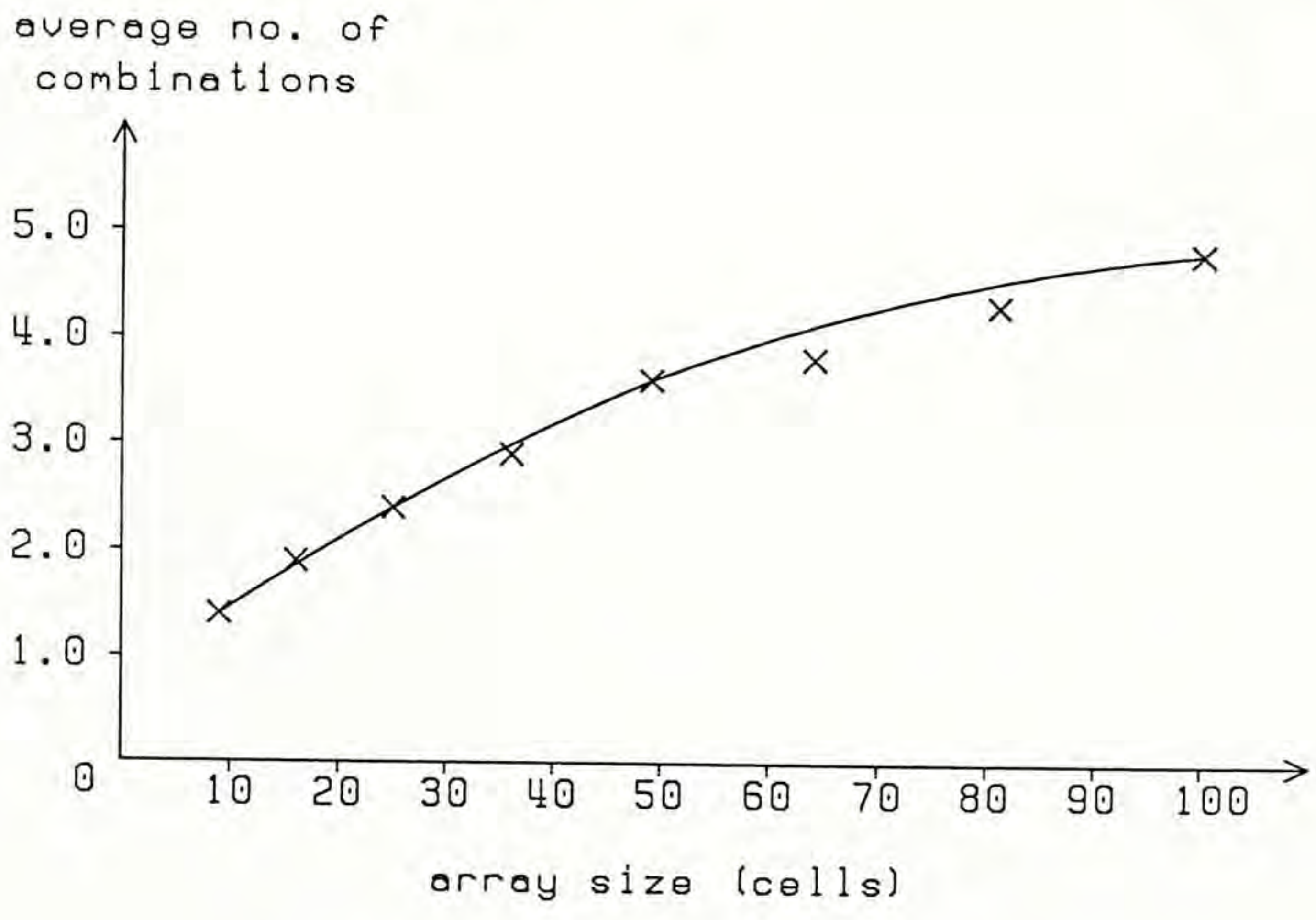

Figure 6-20 Plot of Ncom(n) of Medium Scale Arrays

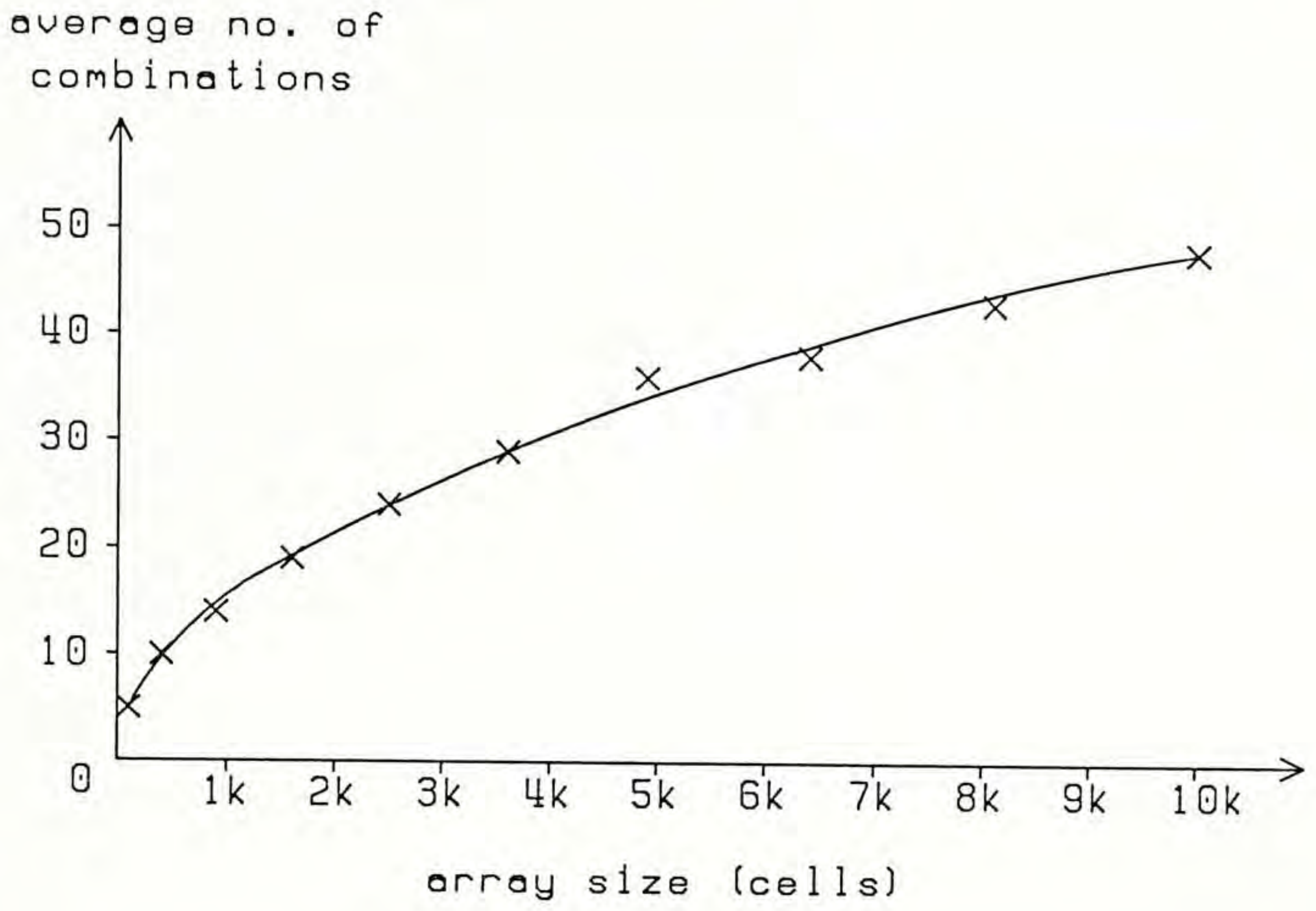

Figure 6-21 Plot of Ncom(n) of Large Scale Arrays

7. *Conclusion*

In this thesis, a new alteration placement algorithm on macrocell array design is presented. This algorithm is aimed at automatic adjustment of the placement due to minor change on a design. Thus, the designer need not re-execute the placement phase or manually correct the placement. Using dynamic variables to form the data structure, the design is represented in a flexible form. It is proved that the computation time is reduced by implementing such data structure. Testing results from eight test circuits have been very encouraging. Alternative techniques which may improve the result of the algorithm are also suggested and investigated. In addition, another new placement algorithm which bases on the already developed data structure is recommended for fully utilization of the programs. These proposed methodologies are aiming at improving dimensions and performance of designs on macrocell array.

In section 1, Introduction: the problem of layout automation, the proposed algorithm to solve placement alteration, the modeling of macrocell array, and the measure of cost function was introduced.

In section 2, Reviews of Partitioning and Placement Methods: the problems of partitioning and placement, and some of the State-of-Art methods were described.

In section 3, Algorithm: the steps of the Incremental Alteration Placement Algorithm (IAPA) was described and a simple example circuit was used to illustrate the procedure.

In section 4, Implementation: techniques and details on the implementation of IAPA was investigated. The construction of the data structure and the layout of programs and data files were also described.

In section 5, Results: the results of simulation on 8 test circuits are analyzed and summarized. Details of simulation procedure of two circuits were investigated for more detailed understanding of effectiveness of the algorithm.

In section 6, Discussion: the computation time of IAPA was investigated. Three alternative methods on the determination of propagation path were introduced and compared for a deeper investigation on the problem and a better understanding of the computational complexities. In addition, an algorithm together with the data structure on the wiring optimization problem, a method to generalize the data structure to solve problems with more sophisticated circuits, and a new placement algorithm which bases on the already developed data structure in the affinity clustering phase were discussed for further research on the topic. At last, an alternative method on element allocation which will improve the performance of the algorithm was introduced. These methods not only gave substantial suggestions for a more realistic software but will also provide insight on creative automatic layout techniques.

In section 7 and 8, the conclusion and list of references were presented.

In the last section, Appendix: the principles, implementation, and results of a placement program using Simulated Annealing Algorithm were summarized. The program was used as the initial placement phase for the Incremental Alteration Placement Algorithm and its program file was named "fplace77.pas" (section 4.4.2).

In general, the main idea brought out in this thesis is the application of dynamic data structure to a new placement algorithm, IAPA, which will help solving a common IC design problem: change of placement due to minor change in schematic. Design of the algorithm not only emphasized the validation of methodology but also considered realistic implementation possibility. A well-organized software system was built and suitable test circuits were selected for evaluation of the algorithm. Results of simulation and computational complexity of the algorithm were also carefully studied and analyzed. In addition, constructive suggestions to modify the algorithm were examined for further exploration of the topic.

8. References

[1] E. R. Barnes, "Algorithm for Partitioning the Nodes of a Graph", SIAM J. Algebraic and Discrete Methods, Vol. 3, No. 4, Dec. 1982, pp.541-550.

[2] E. R. Barnes, A. Vannelli, J. Q. Walker, "A New Procedure for Partitioning the Nodes of a Graph", IBM Research Report RC 10561, June 1984.

[3] E. R. Barnes, "Partitioning the Nodes of a Graph", Graph Theory with Application to Algorithms and Computer Science, Wiley-Interscience Publication, 1985, pp.57-72.

[4] Dimitri P. Bertsekas, "Dynamic Programming: Deterministic and Stochastic Models", Prentice-Hall Inc., 1987, pp. 26-28.

[5] M. A. Breuer, " Min-Cut Placement", J. Design Automation & Fault Tolerant Computing 1, No. 4, 343-382, 1977.

[6] Charney, H. R., and D. L. Plato, "Efficient partitioning of components", in Proceedings of the 5th Annual Design Automation Workshop, pp. 16-0 to 16-21, 1968.

[7] K. K. Cheng, "A Pattern Recognition Circuit using CMOS Standard Cell Array Technology", BSc Thesis, Dept. of Electronic Engineering, The Chinese University of Hong Kong, 1989.

[8] T.S. Cheung, "Serial Data Synchronizers/Desynchronizers implemented on Macrocell Arrays", BSc Thesis, Dept. of Electronic Engineering, The Chinese University of Hong Kong, 1988.

[9] L. Corrigan, "A placement capability based on partitioning", Proceedings of the 16th Design Automation Conference, 1979, pp. 406-413.

[10] Eric V. Denardo, "Dynamic Programming: Models and Application", Prentice-Hall, 1982, pp. 16-18.

[11] Donath, W. E., and A. J. Hoffman, "Algorithms for partitioning of graphs and computer logic based on eigenvectors of connection matrices", IBM Technical Disclosure Bulletin 15, pp. 938-944, 1972.

[12] Donath, W. E. and A. J. Hoffman, "Lower bounds for the partitioning of graphs", IBM Journal of Research and Development, vol. 17, pp. 420-425, 1973.

[13] Donath, W. E., "Physical Design Automation of VLSI Systems", 1988, chap. 3, pp. 65-86, Editor: Bryan T. Preas and Michael J. Lorenzetti Pub: The Benjamin/Cummings Publishing Company, Inc.

[14] M. Dannie Durand, "Parallel simulated annealing: accuracy vs. speed in

placement", IEEE Design & Test of Computers Magazine, June 1989, pp. 8-34.

[15] Alfred E. Dunlop and Brian W. Kernighan, "A procedure for placement of standard-cell VLSI circuits", IEEE Trans. on CAD, Vol. CAD-4, No.1, Jan. 1985, pp. 92-98.

[16] W. H. Elder, P. P. Zenewicz, R. R. Alvarodiaz. "An interactive system for VLSI chip physical design", IBM Journal of Research and Development, Vol. 28, No. 5, Sept., 1984, pp. 524-536.

[17] M. Feuer, W. R. Heller, et al., "EMERALPS - Automatic Partitioning and Placement of Logic Circuits on Weinberger Images", Proceedings of the IBM Design Automation Conference, 1977, pp. 9-22.

[18] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", Freeman, 1979.

[19] Satoshi Goto and Tsuneo Matsuda, "Partitioning, Assignment and Placement", Advances in CAD for VLSI Vol. 4: Layout Design and Verification, North Holland, 1986, pp. 55-97.

[20] Satoshi Goto, " An Efficient Algorithm for the Two-Dimensional Placement Problem in Electrical Circuit Layout", IEEE Transactions on Circuits and Systems, Vol. CAS-28, No. 1, Jan., 1981, pp. 12-18.

[21] Maurice Hannan and Jerome M. Kurtzberg, "Placement Techniques", in Design Automation of Digital Systems, M. A. Breuer, Ed. Englewood Cliffs, NJ : Prentice-Hall, 1972 Chap. 5, pp. 213-282.

[22] Tony Holden, "Knowledge based CAD and Micro-Electronics", 1987 Pub: North-Holland.

[23] Ernest E. Hollis, "Design of VLSI Gate Array ICs", Prentice-Hall, 1986, pp. 6-12.

[24] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", Bell Systems Technical Journal, Vol. 49, Feb. 1970, pp. 291-307.

[25] Kodres, U. R., Partitioning and Card Selection. Design Automation of Digital Systems (edited by M. A. Breuer), Prentice-Hall, Inc., Englewood Cliffs, New Jersey, pp. 173-212, 1972.

[26] D. LaPotin and S. Director, "Mason: A Global Floor-planning Approach to VLSI Design", IEEE Trans. on CAD of ICAS, Vol. CAD-5, pp. 477-489, Oct. 1986.

[27] K. W. Lallier and R.K. Jackson, "A new circuit placement program for FET chips", Proceedings of the 16th Design Automation Conference, 1979, pp. 109-113.

[28] U. Lauther, "A min cut placement algorithm for general cell assemblies based on a graph representation", Proc. 16th Design Automation Conf. (San Diego), pp. 1-10, 1979.

[29] C. Y. Lee, "An algorithm for path connections and its applications", IRE Trans. Electron. Comput. EC10(3), pp. 346-364, 1961.

[30] Kurt Mehlhorn, "Data structures and algorithms 2: Graph algorithms and NP-Completeness", Springer-Verlag, 1984, pp. 1-3.

[31] A. Mennone and R. L. Russo, "Selecting seed vertices for multiple mappings using the automated logic mapping system", in IBM Technical Disclosure Bulletin 13, pp. 3202-3204, April 1971.

[32] D. Milford and N. Kingswood, "CAD tools for semicustom IC design", Microprocessors & Microsystems, Vol. 12, No. 7, Sept., 1988, pp. 363-372.

[33] M. Monro, "A Crash Course in PASCAL", Pub: Edward Arnold, 1987, pp. 160-170.

[34] Tatsuo Ohtsuki, "Maze-running and line-search algorithms", Advances in CAD for VLSI Vol. 4: Layout Design and Verification, North Holland, 1986, pp. 99-131.

[35] Thomas Payne, Robert Wells, and Werner Gundel, "A Study of Automatic Placement Strategies for Very Large Gate Array Designs", IEEE International Conference on Computer-Aided Design (ICCAD-87), 1987, IEEE Computer Society, pp. 194-197.

[36] Bryan T. Preas and Patrick G. Karger, "Physical Design Automation of VLSI Systems", 1988, chap. 4, pp. 87-155, Editor: Bryan T. Preas and Michael J. Lorenzetti, Pub: The Benjamin/Cummings Publishing Company, Inc.

[37] Jerry Priste, "Design Manual: MCA600ECL and MCA1200ECL, MECL 10,000 Macrocell Arrays Design Guidelines", Rev. 1, Motorola Semiconductor Products Inc., 1984.

[38] G. Russell, D. J. Kinniment, E. G. Chester, and M. R. McLauchlan, "CAD for VLSI", Van Nostrand Reinhold (UK) Co. Ltd, 1985.

[39] A. Sangiovanni-Vincentelli, "Automatic Layout of Integrated Circuits", 1987 in "Design Systems for VLSI Circuits: Logic Synthesis and Silicon Compilation" Pub: NATO Advanced Science Institute Series.

[40] G. Michael Schneider, Steven W. Weingart, David M. Perlman, "An Introduction to Programming and Problem Solving with Pascal", 2ed, 1982, Pub: John Wiley and sons, pp. 377-378.

[41] Carl Sechen, "Average Interconnection Length Estimation for Random and

Optimized Placements", IEEE International Conference on Computer-Aided Design (ICCAD-87), 1987, IEEE Computer Society, pp. 190-193.

[42] Carl Sechen, "VLSI Placement and Global Routing Using Simulated Annealing", Kluwer Academic Publishers, 1988, pp. 255-266.

[43] Tsuneta Sudo, Tatsuo Ohtsuki, and Satoshi Goto, "CAD Systems for VLSI in Japan", Proceedings of the IEEE, Vol. 71, No. 1, Jan., 1983.

[44] Texas Instruments, "Standard Cell Data Book" (SCJ1208), Texas Instruments Inc., 1987.

[45] Jean-Paul Tremblay and Richard B. Bunt, "An introduction to computer science: an algorithmic approach", McGraw-Hill Ltd., 1979, pp. 432-627.

[46] Niklaus Wirth, "Algorithms and data structures", Prentice-Hall Inc., 1986, pp. 171-268.

9. Appendix I

The problem of placement on macrocell array is investigated in this section. The solution will be found out by a Simulated Annealing algorithm together with several common measures on the performance of a circuit. The algorithm in this section is used as the placement phase for IAPA. The program file is named "fplace77.pas" (section 4.4.2).

9.1 Definition of the Problem

The problem is to place cells (logic gates or flip-flops) into a cell array such that the following three purposes can be fulfilled:
(1) the total interconnection metal length should be minimized to a certain extent,
(2) the metal connection length on the critical path should be small for good performance of the circuit, and
(3) the skew in propagation delays on the input-to-output paths should be small as well.

A 3X3 cell array (Figure 9-1) and a digital circuit with 9 logic units (Figure 9-2) are used as examples in the simulation program.

Figure 9-1 A 3X3 cell array

9.2 The Simulated Annealing Algorithm

Simulated annealing is a process assembling the heat annealing of doped semiconductor wafer or crystals. Since the natural formation of bondings among molecules in these substances is always looking for a minimum potential energy, the process can be imitated to find the global minimum in a multi-objective problem. The algorithm of simulated annealing is shown in Table 9-1.

```
Start with some state, So;
T = To;
repeat
  while (not at equilibrium) do
     begin
        Perturb S to get a new state Sn;
        ΔE := E(Sn) - E (S);
        if ΔE < 0 then
          replace S with Sn
        else
          with probability exp(-ΔE/kT) replace S with Sn
      end;
   T := c X T;      { 0 < c < 1 }
 until (frozen);
```

k : Kelvin constant, T : temperature,
E(S) : Energy of state S
c : a constant (decrease rate of T).

Table 9-1. A Simulated Annealing Algorithm

9.3 Example Circuit

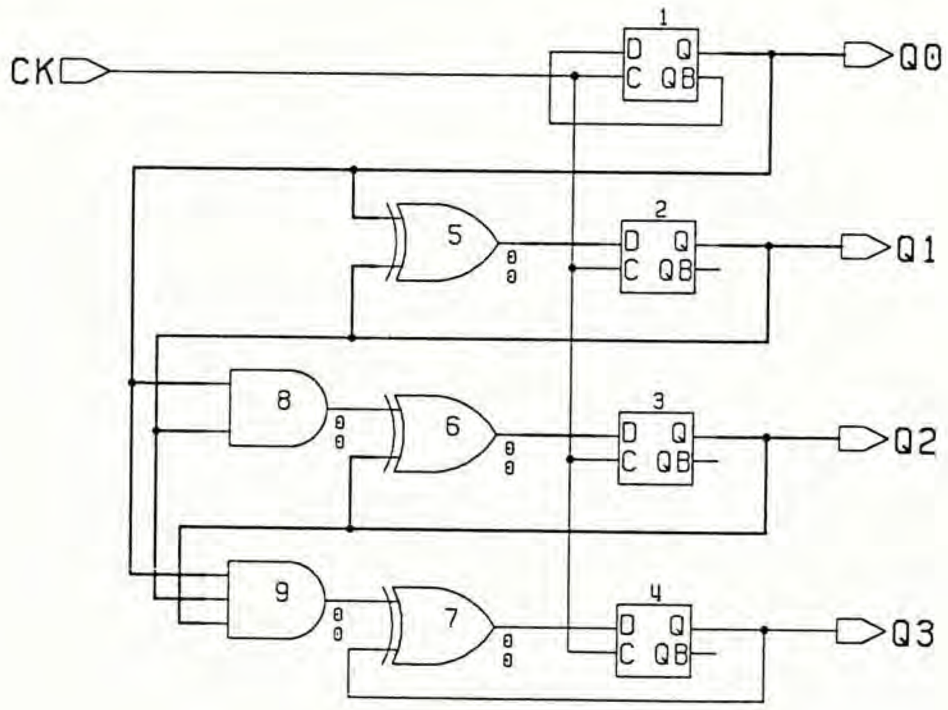

Figure 9-2 A 4-bit Synchronous Counter

The counter has the following 'directed' connections:
[1,5], [1,8], [1,9], [2,5], [2,8], [2,9], [3,6], [3,9],
[4,7], [5,2], [6,3], [7,4], [8,6], [9,7].

The connection in cell 1 is not counted because it can be viewed as internal connection.

9.4 Performance Indices and Energy Value

There are three performance indices in this problem, namely, (1) total interconnection length, (2) the delay of the critical path, and (3) the skew in the input-to-output delays. And, the energy is given by a weighted sum of the three indices.

9.4.1 Total Interconnection Length

For simpler manipulations of data, the interconnection length in connection is measured in Manhattan Distance and the unit of length is assumed unity:

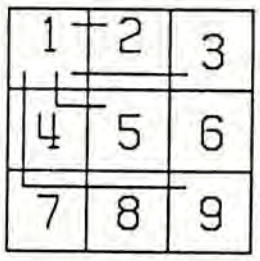

Figure 9-3 Manhattan Distance Among Slots

For example, distance between cell 1 and 2 is D(1,2)=1, D(1,3)=2, D(1,5)=2 and D(1,9)=4.

9.4.2 Delay on Critical Paths

Dynamic Programming approach [1], [2] is used in the calculation of delay on critical paths. The connection flow diagram of the synchronous counter is shown in figure 9-4.

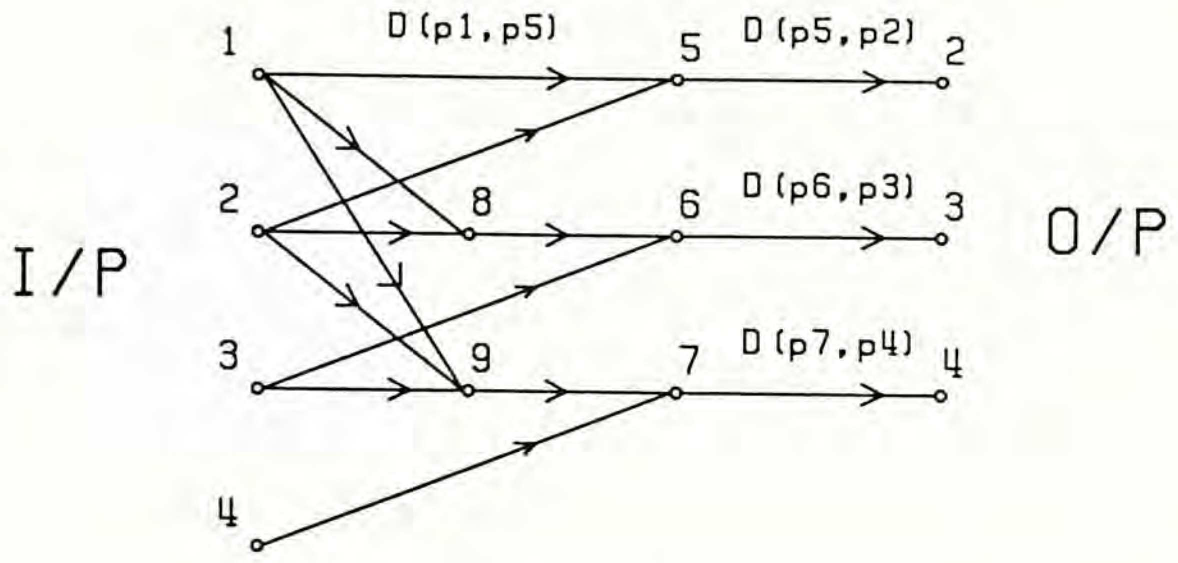

Figure 9-4 Connection Flow Diagram of the Synchronous Counter

The critical path delay (i.e. greatest delay in this circuit) determines the maximum operating frequency of the circuit. Notice that the flow is directed, i.e. from inputs to outputs. To illustrate the principle of Dynamic Programming, let's analyze a simpler connection flow diagram:

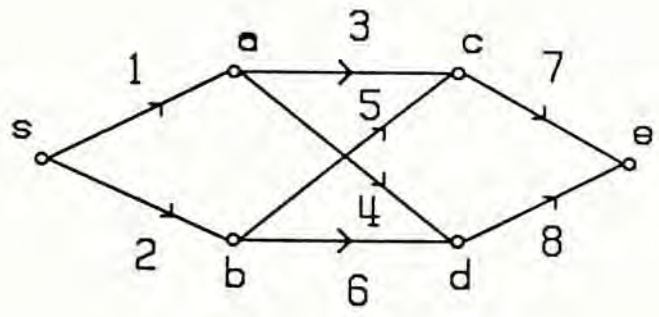

Figure 9-5 An Example Connection Flow Diagram

```
delay in  ce : 7 --> Pce        , where Pxy is the maximum delay from x to y.
          de : 8 --> Pde

          ae : ace : Pce + 3 = 10
               ade : Pde + 4 = 12  --> Pae
          be : bce : Pce + 5 = 12
               bde : Pde + 6 = 14  --> Pbe

          se : sae : Pae + 1 = 13
               sbe : Pbe + 2 = 16  --> Pse
```

Hence, the maximum delay is 16 units in the critical path {s, b, d, e}.

9.4.3 Skew in Input-to-Output Delays

In many synchronous circuit, the propagation delays, t_{pd}, of outputs relative to inputs are required to be approximate. In the example circuit, it is required that the time delay from CK to Q's be close. Since the propagation delay of the D-type Flip Flops (DFF) are the same, the skews are contributed by the difference in metal length and number of fan-outs. Hence, the propagation delay degradation is given by

$$\Delta t_{pd} = w1 * l + w2 * N_{fo} \qquad \text{---------------------- Eq. 9-1.}$$

where w1 = time delay per unit length of metal,
w2 = time delay per fan-out,
l = metal-length connecting to an O/P,
N_{fo} = number of fan-outs of an O/P.

In this example, l is measured in unit integer and both w1 and w2 are set 1 for simplicity.

9.4.4 Energy Value

Since the delay in critical path is considered much important, the energy value is given by

$$E = \text{round}[(L/Lo + S/So + 2D/Do) \times 10] \qquad \text{--------------------- Eq. 9-2}$$

where L = the total metal length,
S = Skew in I/P-O/P paths,
D = Delay in the critical path, and the 'o's denote initial values.
The multiplier, 10, is to adjust E to an integer such that the subsequent calculation can be simpler. In addition, operations on integer variables are faster and will save much storage.

9.5 The Simulation Program

Pascal is used as the programming language of the simulation algorithm. The program consists of 4 functions, namely, (1) alise, (2) max_delay, (3) replace, and (4) total_length; and 4 procedures: (1) init_weight, (2) inverse, (3) initial, and (4) shuffle.

9.5.1 The "function" Subroutines

9.5.1.1 alise

This function calculates the maximum delay difference in the input-to-output paths. Firstly, it is reasoned that the maximum difference can be obtained by finding the difference between the maximum and minimum delay paths. Both the minimum and maximum delays are obtained by iterative calculations using Eq. 9-1.

9.5.1.2 max_delay

This function calculates the maximum delay path (critical path) in the circuit. The Reaching Method of Dynamic Programming [2] is used. The algorithm is:

```
1. Set $V_j = 0$ for $j = 1,2, \ldots , N$.
2. for i:= 1 to N-1 do
    3. for j:= i+1 to N do
        $V_j := \max\{V_j, V_i + D_{ij}\}$
```

where Vx is maximum delay from x to terminal node,

D_{xy} is delay from x to y.

Table 9-2. Reaching Method of Dynamic Programming

9.5.1.3 replace

This function determines whether to replace the state or not if the energy difference is positive. A boolean value would be returned: true for replace, false for not. The kelvin constant, $k = 0.05$.

9.5.1.4 total_length

This function calculates the total interconnection length among cells. The statements on computation are:

```
for i:= 1 to 9 do
  for j:= i to 9 do
    length := length + $C_{ij}.Dp_i p_j$;
```

where C_{ij} is the number of connection between i and j,

$Dp_i p_j$ is the distance between the positions of i and j.

9.5.2 The "procedure" Subroutines

9.5.2.1 init_weight

This procedure assigns the interconnection distance (w-distance) of slots. The characteristic equation is given by

w-distance(A,B) := abs(ii-i) + abs (jj-j) -------------------- Eq. 9-3.
where (i,j) is the coordinate of cell A and (ii,jj) coordinate of B.

9.5.2.2 inverse

Since the original cell order is expressed in position-of-cell form, it is more precise to expressed the cells in cell-in-position form. Hence, this procedure reverses the order of position of cells.

9.5.2.3 initial

This procedure assigns the connection of nodes. The connections are according to the directed connection of the synchronous counter.

9.5.2.4 shuffle

The purpose of this subroutine is to make perturbation on the placement. It shuffles the positions of either (1) any two arbitrary cells, or (2) any two adjacent cells, by the choice of the designer.

9.5.3 The Main Program

The main program is the implementation of the simulated annealing algorithm (Table 9-1). The initial temperature is To=1000 and the rate of decrease of time is c (adjustable), the Kelvin constant is defined in the function 'replace'. The frozen temperature is 100.

9.6 Results and Discussion

The initial placement configuration is :

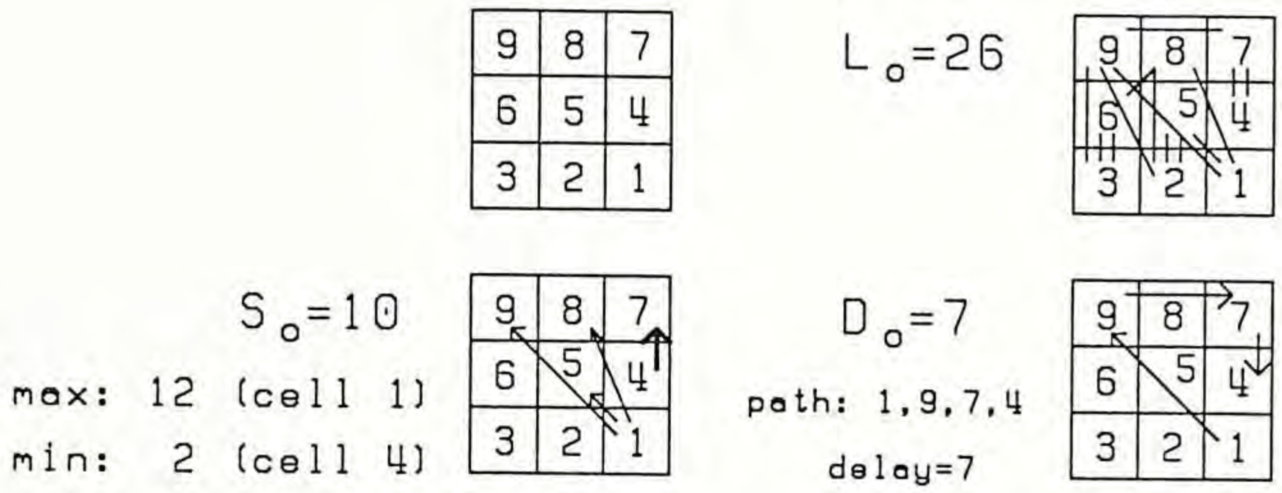

Figure 9-6 The Initial Placement Configuration

The energy value, E, is 40 by default.

The best placement configuration (with minimum energy) ever found is:

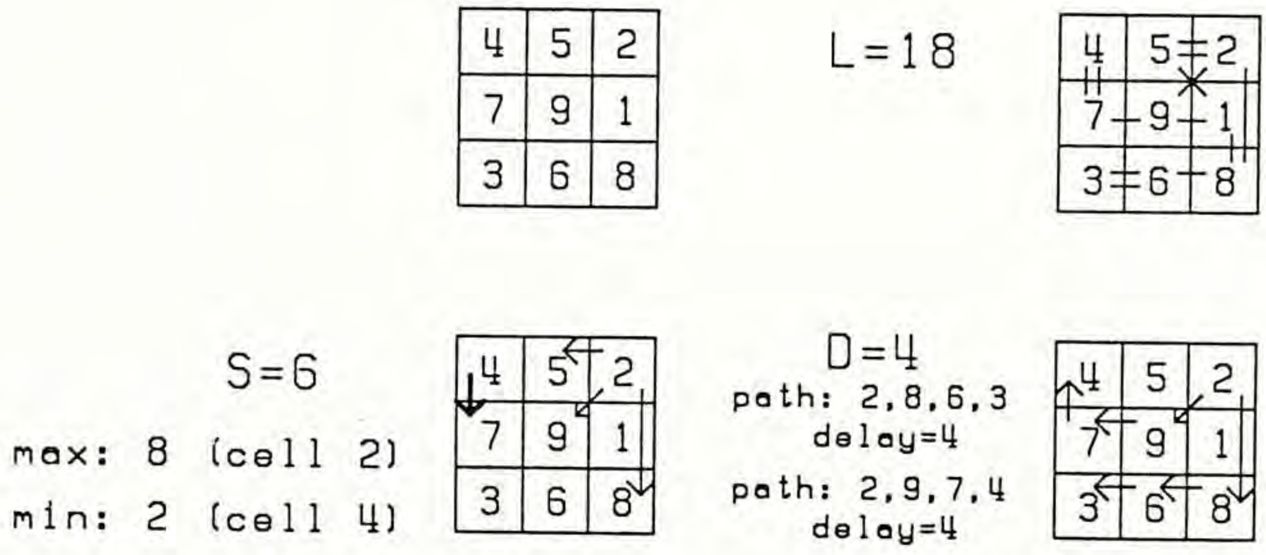

Figure 9-7 The Best Placement Configuration Ever Found

The energy value, E = round [(18/26 + 6/10 + 2 (4)/7) X 10]
= 24

Several values of c (0.72, 0.9, 0.99) are used in the simulation and the sequence of final energy is as follows (30 consecutive executions of the program are done on c=0.72 and 16 executions were done on c=0.9 and c=0.99):
When c=0.72, the sequence is:

34, 40, 34, 39, 33, 33, 36, 36, 37, 33, 45, 42, 37, 35, 36, 32, 36, 41
45, 36, 39, 35, 39, 41, 42, 36, 30, 38, 34, 38

mean = 37.07, STD = 3.695, max = 45, min = 30.

When c=0.9, the sequence is:

35, 38, 37, 39, 29, 33, 36, 30, 38, 36, 37, 37, 37, 46, 33, 38, 32, 34

mean = 35.83, STD = 3.823, max = 46, min = 29.

When c=0.99, the sequence is:

32, 28, 32, 31, 28, 32, 35, 35, 33, 36, 41, 33, 39, 37, 33, 38, 34, 41

mean = 34.33, STD = 3.819, max = 41, min = 28.

It is found that the one with c=0.9 can give a quite good set of solution and is thus fully investigated.

The first 6 simulations with c=0.72 are listed in Figure 9-8 to 9-13, and those with c=0.9 in Figure 9-14 to 9-19. It is found that some valuable configurations, i.e. those with quite well performance indices, were obtained in the process of simulation. If these values are collected into a set, the member in this set can be used as reference for further minimization, initial placement, or options for special requirement. For instance, configurations with particular good performance in S may be useful in circuit design of accurate synchronous machine, while a small D is preferred in high speed systems.

Moreover, executions of the program with linear decrease in temperature were also done. However, the result is unsatisfactory. It is because the simulated annealing algorithm is a process with properties of exponential function. Linear parameters may cause conflict in the simulation.

```
Running
 12 2 Initial energy=40 L=26 S=10 D= 7 Decrease rate of temperature=0.720
T=1000 Su54 p:987645321 11 3 L=29 S= 8 D= 8 E=42 E.Df= 2 Rd=0.318 Pb=0.961 Nw
T= 720 Su58 p:957648321 11 3 L=29 S= 8 D= 8 E=42 E.Df= 0 Nw
T= 518 Su85 p:987645321 11 3 L=29 S= 8 D= 8 E=42 E.Df= 0 Nw
T= 373 Su62 p:987245361 11 3 L=27 S= 8 D= 8 E=41 E.Df=-1 Nw
T= 269 Su15 p:987241365  9 3 L=27 S= 6 D= 7 E=36 E.Df=-5 Nw
T= 193 Su17 p:981247365  9 2 L=25 S= 7 D= 6 E=34 E.Df=-2 Nw
T= 139 Su76 p:981246375  9 2 L=29 S= 7 D= 7 E=38 E.Df= 4 Rd=0.938 Pb=0.563 Od
T= 100 Su16 p:986247315  9 2 L=31 S= 7 D= 8 E=42 E.Df= 8 Rd=0.231 Pb=0.203 Od
 Final position is : 9,8,1,2,4,7,3,6,5,# Final energy=34
>
```

Figure 9-8 1st Simulation with c=0.72

```
Running
 12 2 Initial energy=40 L=26 S=10 D= 7 Decrease rate of temperature=0.720
T=1000 Su47 p:984657321 12 2 L=27 S=10 D= 8 E=43 E.Df= 3 Rd=0.739 Pb=0.942 Nw
T= 720 Su17 p:984651327  9 3 L=27 S= 6 D= 9 E=42 E.Df=-1 Nw
T= 518 Su85 p:954681327  9 3 L=27 S= 6 D= 9 E=42 E.Df= 0 Nw
T= 373 Su65 p:984651327  9 3 L=27 S= 6 D= 9 E=42 E.Df= 0 Nw
T= 269 Su18 p:914658327  9 3 L=25 S= 6 D= 9 E=41 E.Df=-1 Nw
T= 193 Su62 p:914258367  7 3 L=23 S= 4 D= 8 E=36 E.Df=-5 Nw
T= 139 Su65 p:914268357  8 3 L=27 S= 5 D= 8 E=38 E.Df= 2 Rd=0.804 Pb=0.750 Od
T= 100 Su42 p:912458367  8 4 L=27 S= 4 D= 9 E=40 E.Df= 4 Rd=0.114 Pb=0.450 Nw
 Final position is : 9,1,2,4,5,8,3,6,7,# Final energy=40
>
```

Figure 9-9 2nd Simulation with c=0.72

```
Running
 12 2 Initial energy=40 L=26 S=10 D= 7 Decrease rate of temperature=0.720
T=1000 Su26 p:987254361 12 2 L=24 S=10 D= 7 E=39 E.Df=-1 Nw
T= 720 Su57 p:985274361 12 2 L=28 S=10 D= 7 E=41 E.Df= 2 Rd=0.974 Pb=0.946 Od
T= 518 Su89 p:897254361 12 2 L=25 S=10 D= 8 E=42 E.Df= 3 Rd=0.393 Pb=0.891 Nw
T= 373 Su59 p:857294361 12 2 L=26 S=10 D= 8 E=43 E.Df= 1 Rd=0.061 Pb=0.948 Nw
T= 269 Su76 p:856294371 12 3 L=32 S= 9 D=10 E=50 E.Df= 7 Rd=0.472 Pb=0.594 Nw
T= 193 Su16 p:851294376  8 3 L=26 S= 5 D= 8 E=38 E.Df=-12 Nw
T= 139 Su65 p:861294375  9 3 L=28 S= 6 D= 6 E=34 E.Df=-4 Nw
T= 100 Su47 p:861297345  9 3 L=28 S= 6 D= 6 E=34 E.Df= 0 Nw
 Final position is : 8,6,1,2,9,7,3,4,5,# Final energy=34
>
```

Figure 9-10 3rd Simulation with c=0.72

```
Running
 12 2 Initial energy=40 L=26 S=10 D= 7 Decrease rate of temperature=0.720
T=1000 Su43 p:987653421 12 5 L=35 S= 7 D=10 E=49 E.Df= 9 Rd=0.431 Pb=0.835 Nw
T= 720 Su34 p:987654321 12 2 L=26 S=10 D= 7 E=40 E.Df=-9 Nw
T= 518 Su78 p:978654321 11 3 L=28 S= 8 D= 7 E=39 E.Df=-1 Nw
T= 373 Su56 p:978564321 12 3 L=32 S= 9 D= 7 E=41 E.Df= 2 Rd=0.216 Pb=0.898 Nw
T= 269 Su42 p:978562341 12 3 L=30 S= 9 D= 7 E=41 E.Df= 0 Nw
T= 193 Su52 p:978265341 10 3 L=28 S= 7 D= 7 E=38 E.Df=-3 Nw
T= 139 Su65 p:978256341 11 3 L=28 S= 8 D= 7 E=39 E.Df= 1 Rd=0.622 Pb=0.866 Nw
T= 100 Su12 p:978156342 11 3 L=28 S= 8 D= 7 E=39 E.Df= 0 Nw
 Final position is : 9,7,8,1,5,6,3,4,2,# Final energy=39
>
```

Figure 9-11 4th Simulation with c=0.72

```
Running
 12 2 Initial energy=40 L=26 S=10 D= 7 Decrease rate of temperature=0.720
T=1000 Su48 p:947658321 10 2 L=24 S= 8 D= 7 E=37 E.Df=-3 Nw
T= 720 Su79 p:749658321  9 2 L=24 S= 7 D= 7 E=36 E.Df=-1 Nw
T= 518 Su47 p:479658321  9 2 L=23 S= 7 D= 6 E=33 E.Df=-3 Nw
T= 373 Su24 p:279658341 10 3 L=27 S= 7 D= 7 E=37 E.Df= 4 Rd=0.123 Pb=0.807 Nw
T= 269 Su58 p:279685341 10 3 L=27 S= 7 D= 7 E=37 E.Df= 0 Nw
T= 193 Su29 p:972685341 10 3 L=23 S= 7 D= 7 E=36 E.Df=-1 Nw
T= 139 Su58 p:972658341 10 3 L=25 S= 7 D= 7 E=37 E.Df= 1 Rd=0.988 Pb=0.866 Od
T= 100 Su84 p:972645381  9 2 L=22 S= 7 D= 6 E=33 E.Df=-3 Nw
 Final position is : 9,7,2,6,4,5,3,8,1,# Final energy=33
>
```

Figure 9-12 5th Simulation with c=0.72

```
Running
 12 2 Initial energy=40 L=26 S=10 D= 7 Decrease rate of temperature=0.720
T=1000 Su26 p:987254361 12 2 L=24 S=10 D= 7 E=39 E.Df=-1 Nw
T= 720 Su14 p:987251364  9 3 L=23 S= 6 D= 7 E=35 E.Df=-4 Nw
T= 518 Su85 p:957281364  9 3 L=23 S= 6 D= 7 E=35 E.Df= 0 Nw
T= 373 Su89 p:857291364  9 3 L=25 S= 6 D= 7 E=36 E.Df= 1 Rd=0.731 Pb=0.948 Nw
T= 269 Su36 p:857291634  9 3 L=23 S= 6 D= 6 E=32 E.Df=-4 Nw
T= 193 Su84 p:457291638  9 3 L=23 S= 6 D= 6 E=32 E.Df= 0 Nw
T= 139 Su58 p:487291635  9 3 L=25 S= 6 D= 6 E=33 E.Df= 1 Rd=0.337 Pb=0.866 Nw
T= 100 Su87 p:478291635 10 2 L=23 S= 8 D= 8 E=40 E.Df= 7 Rd=0.431 Pb=0.248 Od
 Final position is : 4,8,7,2,9,1,6,3,5,# Final energy=33
>
```

Figure 9-13 6th Simulation with c=0.72

```
Initial energy = 40 L=26 S=10 D= 7 Decrease rate of temperature = 0.900
T=1000 Su57 p:985674321 12 2 L=30 S=10 D= 7 E=42 E.Df= 2 Rd=0.129 Pb=0.961 Nw
T= 900 Su48 p:945678321 11 2 L=28 S= 9 D= 7 E=40 E.Df=-2 Nw
T= 810 Su69 p:645978321 10 2 L=27 S= 8 D= 7 E=38 E.Df=-2 Nw
T= 729 Su38 p:645973821 10 2 L=29 S= 8 D= 7 E=39 E.Df= 1 Rd=0.625 Pb=0.973 Nw
T= 656 Su84 p:685973421 11 3 L=32 S= 8 D= 7 E=40 E.Df= 1 Rd=0.808 Pb=0.970 Nw
T= 590 Su12 p:685973412 11 3 L=31 S= 8 D= 7 E=40 E.Df= 0 Nw
T= 531 Su26 p:285973416 10 3 L=25 S= 7 D= 6 E=34 E.Df=-6 Nw
T= 478 Su36 p:285976413 10 3 L=25 S= 7 D= 6 E=34 E.Df= 0 Nw
T= 430 Su49 p:285476913  9 2 L=23 S= 7 D= 5 E=30 E.Df=-4 Nw
T= 387 Su76 p:285467913  9 3 L=27 S= 6 D= 7 E=36 E.Df= 6 Rd=0.352 Pb=0.734 Nw
T= 349 Su56 p:286457913  8 3 L=25 S= 5 D= 7 E=35 E.Df=-1 Nw
T= 314 Su79 p:286459713  9 2 L=24 S= 7 D= 7 E=36 E.Df= 1 Rd=0.484 Pb=0.938 Nw
T= 282 Su46 p:284659713  9 5 L=33 S= 4 D=10 E=45 E.Df= 9 Rd=0.815 Pb=0.529 Od
T= 254 Su42 p:486259713  8 3 L=24 S= 5 D= 7 E=34 E.Df=-2 Nw
T= 229 Su52 p:486529713  9 3 L=23 S= 6 D= 7 E=35 E.Df= 1 Rd=0.853 Pb=0.916 Nw
T= 206 Su67 p:487529613  9 3 L=23 S= 6 D= 7 E=35 E.Df= 0 Nw
T= 185 Su37 p:483529617  9 5 L=31 S= 4 D= 9 E=42 E.Df= 7 Rd=0.015 Pb=0.470 Nw
T= 167 Su75 p:483729615  8 2 L=27 S= 6 D= 9 E=42 E.Df= 0 Nw
T= 150 Su85 p:453729618  8 2 L=25 S= 6 D= 8 E=38 E.Df=-4 Nw
T= 135 Su82 p:453789612  9 2 L=29 S= 7 D= 8 E=41 E.Df= 3 Rd=0.924 Pb=0.641 Od
T= 122 Su62 p:453769218 11 2 L=27 S= 9 D= 6 E=37 E.Df=-1 Nw
T= 109 Su19 p:453761298  9 2 L=27 S= 7 D= 6 E=35 E.Df=-2 Nw
 Final position is : 4,5,3,7,6,1,2,9,8,# Final energy=35
>
```

Figure 9-14 1st Simulation with c=0.9

```
Initial energy = 40 L=26 S=10 D= 7 Decrease rate of temperature = 0.900
T=1000 Su36 p:987354621 12 2 L=26 S=10 D= 7 E=40 E.Df= 0 Nw
T= 900 Su37 p:983754621 12 3 L=34 S= 9 D=10 E=51 E.Df=11 Rd=0.608 Pb=0.783 Nw
T= 810 Su76 p:983654721 12 4 L=34 S= 8 D= 9 E=47 E.Df=-4 Nw
T= 729 Su83 p:938654721 11 4 L=32 S= 7 D= 9 E=45 E.Df=-2 Nw
T= 656 Su75 p:938674521 11 2 L=28 S= 9 D= 8 E=43 E.Df=-2 Nw
T= 590 Su54 p:938675421 11 3 L=31 S= 8 D= 8 E=43 E.Df= 0 Nw
T= 531 Su58 p:935678421 11 3 L=31 S= 8 D= 8 E=43 E.Df= 0 Nw
T= 478 Su85 p:938675421 11 3 L=31 S= 8 D= 8 E=43 E.Df= 0 Nw
T= 430 Su62 p:938275461 10 3 L=29 S= 7 D= 8 E=41 E.Df=-2 Nw
T= 387 Su15 p:938271465 10 3 L=29 S= 7 D= 8 E=41 E.Df= 0 Nw
T= 349 Su17 p:938217465 10 4 L=33 S= 6 D= 8 E=42 E.Df= 1 Rd=0.412 Pb=0.944 Nw
T= 314 Su45 p:938217564  9 2 L=25 S= 7 D= 8 E=39 E.Df=-3 Nw
T= 282 Su16 p:938267514 10 2 L=23 S= 8 D= 7 E=37 E.Df=-2 Nw
T= 254 Su36 p:968237514 10 2 L=23 S= 8 D= 7 E=37 E.Df= 0 Nw
T= 229 Su34 p:968247513 10 2 L=29 S= 8 D= 8 E=42 E.Df= 5 Rd=0.533 Pb=0.646 Nw
T= 206 Su32 p:968347512 11 2 L=28 S= 9 D= 8 E=43 E.Df= 1 Rd=0.336 Pb=0.907 Nw
T= 185 Su85 p:965347812 11 2 L=30 S= 9 D= 8 E=43 E.Df= 0 Nw
T= 167 Su18 p:965347182 10 2 L=28 S= 8 D= 8 E=42 E.Df=-1 Nw
T= 150 Su62 p:925347186 10 2 L=25 S= 8 D= 6 E=35 E.Df=-7 Nw
T= 135 Su65 p:926347185  9 2 L=29 S= 7 D= 8 E=41 E.Df= 6 Rd=0.804 Pb=0.411 Od
T= 122 Su42 p:945327186 10 3 L=29 S= 7 D= 7 E=38 E.Df= 3 Rd=0.114 Pb=0.610 Nw
T= 109 Su26 p:945367182 10 3 L=27 S= 7 D= 9 E=43 E.Df= 5 Rd=0.682 Pb=0.401 Od
 Final position is : 9,4,5,3,2,7,1,8,6,# Final energy=38
>
```

Figure 9-15 2nd Simulation with c=0.9

```
Initial energy = 40 L=26 S=10 D= 7 Decrease rate of temperature = 0.900
T=1000 Su38 p:837654821 11 2 L=24 S= 9 D= 7 E=38 E.Df=-2 Nw
T= 900 Su39 p:837654921 11 2 L=28 S= 9 D= 8 E=43 E.Df= 5 Rd=0.393 Pb=0.395 Nw
T= 810 Su59 p:837694521 11 2 L=24 S= 9 D= 7 E=38 E.Df=-5 Nw
T= 729 Su53 p:857694321 12 2 L=26 S=10 D= 6 E=37 E.Df=-1 Nw
T= 656 Su42 p:857692341 12 4 L=30 S= 8 D= 7 E=40 E.Df= 3 Rd=0.110 Pb=0.913 Nw
T= 590 Su45 p:847692351 10 2 L=24 S= 6 D= 6 E=34 E.Df=-6 Nw
T= 531 Su47 p:874692351 10 2 L=23 S= 8 D= 6 E=34 E.Df= 0 Nw
T= 478 Su43 p:873692451 10 4 L=31 S= 6 D= 8 E=41 E.Df= 7 Rd=0.431 Pb=0.746 Nw
T= 430 Su34 p:874692351 10 2 L=23 S= 8 D= 6 E=34 E.Df=-7 Nw
T= 387 Su78 p:784692351  9 3 L=25 S= 6 D= 6 E=33 E.Df=-1 Nw
T= 349 Su56 p:784592361 11 3 L=27 S= 8 D= 6 E=36 E.Df= 3 Rd=0.216 Pb=0.842 Nw
T= 314 Su42 p:782594361 11 4 L=31 S= 7 D= 7 E=39 E.Df= 3 Rd=0.910 Pb=0.826 Od
T= 282 Su84 p:748592361 10 2 L=24 S= 8 D= 6 E=34 E.Df=-2 Nw
T= 254 Su58 p:745892361 10 2 L=22 S= 8 D= 6 E=34 E.Df= 0 Nw
T= 229 Su12 p:745891362 10 2 L=23 S= 8 D= 6 E=34 E.Df= 0 Nw
T= 206 Su48 p:785491362 10 2 L=23 S= 8 D= 6 E=34 E.Df= 0 Nw
T= 185 Su79 p:985471362 12 2 L=27 S=10 D= 7 E=40 E.Df= 6 Rd=0.243 Pb=0.523 Nw
T= 167 Su67 p:985461372 12 3 L=31 S= 9 D= 9 E=47 E.Df= 7 Rd=0.642 Pb=0.432 Od
T= 150 Su64 p:985671342 12 2 L=27 S=10 D= 7 E=40 E.Df= 0 Nw
T= 135 Su58 p:958671342 12 2 L=29 S=10 D= 7 E=41 E.Df= 1 Rd=0.395 Pb=0.862 Nw
T= 122 Su15 p:918675342 10 2 L=23 S= 8 D= 7 E=37 E.Df=-4 Nw
T= 109 Su23 p:918675243 12 2 L=33 S=10 D=10 E=51 E.Df=14 Rd=0.694 Pb=0.077 Od
 Final position is : 9,1,8,6,7,5,3,4,2,# Final energy=37
>
```

Figure 9-16 3rd Simulation with c=0.9

```
Initial energy = 40 L=26 S=10 D= 7 Decrease rate of temperature = 0.900
T=1000 Su28 p:927654381 10 2 L=22 S= 8 D= 7 E=36 E.Df=-4 Nw
T= 900 Su68 p:927854361 12 2 L=24 S=10 D= 7 E=39 E.Df= 3 Rd=0.661 Pb=0.936 Nw
T= 810 Su14 p:927651364  9 3 L=23 S= 6 D= 7 E=35 E.Df=-4 Nw
T= 729 Su85 p:927581364  9 3 L=23 S= 6 D= 7 E=35 E.Df= 0 Nw
T= 656 Su89 p:827591364  9 3 L=25 S= 6 D= 7 E=36 E.Df= 1 Rd=0.731 Pb=0.970 Nw
T= 590 Su36 p:827591634  9 3 L=23 S= 6 D= 6 E=32 E.Df=-4 Nw
T= 531 Su84 p:427591638  9 3 L=23 S= 6 D= 6 E=32 E.Df= 0 Nw
T= 478 Su58 p:427891635  9 3 L=23 S= 6 D= 6 E=32 E.Df= 0 Nw
T= 430 Su12 p:417892635  9 3 L=21 S= 6 D= 5 E=28 E.Df=-4 Nw
T= 387 Su39 p:417832695 10 3 L=26 S= 7 D= 7 E=37 E.Df= 9 Rd=0.225 Pb=0.628 Nw
T= 349 Su79 p:419832675  9 4 L=27 S= 5 D= 8 E=38 E.Df= 1 Rd=0.812 Pb=0.944 Nw
T= 314 Su67 p:419832765  9 3 L=25 S= 6 D= 8 E=38 E.Df= 0 Nw
T= 282 Su66 p:419632785  9 3 L=25 S= 6 D= 8 E=38 E.Df= 0 Nw
T= 254 Su57 p:419632587  9 5 L=31 S= 4 D= 8 E=39 E.Df= 1 Rd=0.269 Pb=0.924 Nw
T= 229 Su25 p:419635287 11 5 L=32 S= 6 D=10 E=47 E.Df= 8 Rd=0.519 Pb=0.497 Od
T= 206 Su65 p:419532687  8 5 L=29 S= 3 D= 8 E=37 E.Df=-2 Nw
T= 185 Su36 p:419562387  8 5 L=31 S= 3 D=10 E=43 E.Df= 6 Rd=0.226 Pb=0.523 Nw
T= 167 Su42 p:219564387  9 2 L=25 S= 7 D= 7 E=37 E.Df=-6 Nw
T= 150 Su63 p:219534687  9 2 L=23 S= 7 D= 6 E=33 E.Df=-4 Nw
T= 135 Su79 p:217534689 11 2 L=27 S= 9 D= 7 E=39 E.Df= 6 Rd=0.978 Pb=0.411 Od
T= 122 Su54 p:219435687 11 4 L=31 S= 7 D= 7 E=39 E.Df= 6 Rd=0.013 Pb=0.373 Nw
T= 109 Su79 p:217435689 13 4 L=35 S= 9 D= 9 E=48 E.Df= 9 Rd=0.254 Pb=0.193 Od
 Final position is : 2,1,9,4,3,5,6,8,7,# Final energy=39
>
```

Figure 9-17 4th Simulation with c=0.9

```
Initial energy = 40 L=26 S=10 D= 7 Decrease rate of temperature = 0.900
T=1000 Su33 p:987354621 12 2 L=26 S=10 D= 7 E=40 E.Df= 0 Nw
T= 900 Su37 p:983754621 12 3 L=34 S= 9 D=10 E=51 E.Df=11 Rd=0.324 Pb=0.783 Nw
T= 810 Su68 p:963754821 11 3 L=26 S= 8 D= 7 E=38 E.Df=-13 Nw
T= 729 Su57 p:963574821 12 2 L=28 S=10 D= 7 E=41 E.Df= 3 Rd=0.350 Pb=0.921 Nw
T= 656 Su73 p:967534821 12 2 L=28 S=10 D= 7 E=41 E.Df= 0 Nw
T= 590 Su67 p:976534821 12 3 L=32 S= 9 D= 8 E=44 E.Df= 3 Rd=0.621 Pb=0.903 Nw
T= 531 Su87 p:936534721 13 4 L=34 S= 9 D= 9 E=48 E.Df= 4 Rd=0.829 Pb=0.860 Nw
T= 478 Su29 p:286534791 10 4 L=26 S= 6 D= 7 E=36 E.Df=-12 Nw
T= 430 Su13 p:286514793  8 4 L=22 S= 4 D= 7 E=32 E.Df=-4 Nw
T= 387 Su89 p:296514783  8 4 L=28 S= 4 D= 9 E=40 E.Df= 8 Rd=0.336 Pb=0.662 Nw
T= 349 Su27 p:796514283  8 4 L=26 S= 4 D= 7 E=34 E.Df=-6 Nw
T= 314 Su28 p:796514823  8 4 L=29 S= 4 D= 8 E=38 E.Df= 4 Rd=0.077 Pb=0.775 Nw
T= 282 Su59 p:756914823  8 4 L=29 S= 4 D= 8 E=38 E.Df= 0 Nw
T= 254 Su56 p:765914823  9 4 L=33 S= 5 D= 8 E=41 E.Df= 3 Rd=0.279 Pb=0.790 Nw
T= 229 Su19 p:765194823  8 4 L=32 S= 4 D= 7 E=36 E.Df=-5 Nw
T= 206 Su26 p:725194863  8 4 L=24 S= 4 D= 7 E=33 E.Df=-3 Nw
T= 185 Su23 p:735194862  9 4 L=27 S= 5 D= 7 E=35 E.Df= 2 Rd=0.953 Pb=0.806 Od
T= 167 Su28 p:735194263 12 4 L=33 S= 8 D= 8 E=44 E.Df=11 Rd=0.437 Pb=0.267 Od
T= 150 Su17 p:125794863  9 3 L=22 S= 6 D= 5 E=29 E.Df=-4 Nw
T= 135 Su76 p:125694873  9 3 L=26 S= 6 D= 7 E=36 E.Df= 7 Rd=0.600 Pb=0.355 Od
T= 122 Su29 p:195724863  8 3 L=24 S= 5 D= 7 E=34 E.Df= 5 Rd=0.614 Pb=0.439 Od
T= 109 Su87 p:125894763  8 4 L=24 S= 4 D= 7 E=33 E.Df= 4 Rd=0.602 Pb=0.481 Od
 Final position is : 1,2,5,7,9,4,8,6,3,# Final energy=29
>
```

Figure 9-18 5th Simulation with c=0.9

```
Initial energy = 40 L=26 S=10 D= 7 Decrease rate of temperature = 0.900
T=1000 Su66 p:967854321 12 2 L=30 S=10 D= 8 E=44 E.Df= 4 Rd=0.958 Pb=0.923 Od
T= 900 Su32 p:987654231 12 2 L=31 S=10 D= 7 E=42 E.Df= 2 Rd=0.603 Pb=0.957 Nw
T= 810 Su52 p:987624531 12 2 L=29 S=10 D= 7 E=41 E.Df=-1 Nw
T= 729 Su23 p:987634521 12 2 L=26 S=10 D= 7 E=40 E.Df=-1 Nw
T= 656 Su45 p:987635421 11 5 L=33 S= 6 D=10 E=47 E.Df= 7 Rd=0.129 Pb=0.808 Nw
T= 590 Su62 p:987235461 11 5 L=31 S= 6 D=10 E=46 E.Df=-1 Nw
T= 531 Su29 p:287935461 10 4 L=31 S= 6 D=10 E=46 E.Df= 0 Nw
T= 478 Su36 p:287965431 10 5 L=31 S= 5 D=10 E=45 E.Df=-1 Nw
T= 430 Su12 p:187965432 10 5 L=29 S= 5 D=10 E=45 E.Df= 0 Nw
T= 387 Su53 p:187963452 10 5 L=29 S= 5 D=10 E=45 E.Df= 0 Nw
T= 349 Su18 p:817963452 11 5 L=31 S= 6 D=10 E=46 E.Df= 1 Rd=0.027 Pb=0.944 Nw
T= 314 Su17 p:871963452 11 4 L=31 S= 7 D= 8 E=42 E.Df=-4 Nw
T= 282 Su76 p:861973452 11 3 L=29 S= 8 D= 7 E=39 E.Df=-3 Nw
T= 254 Su79 p:861793452 10 2 L=24 S= 8 D= 7 E=37 E.Df=-2 Nw
T= 229 Su74 p:861493752 10 2 L=25 S= 8 D= 7 E=38 E.Df= 1 Rd=0.105 Pb=0.916 Nw
T= 206 Su52 p:861493725  9 2 L=22 S= 7 D= 6 E=33 E.Df=-5 Nw
T= 185 Su87 p:761493825 11 2 L=24 S= 9 D= 9 E=44 E.Df=11 Rd=0.571 Pb=0.305 Od
T= 167 Su69 p:891463725  9 2 L=23 S= 7 D= 6 E=33 E.Df= 0 Nw
T= 150 Su38 p:391468725  8 2 L=21 S= 6 D= 6 E=31 E.Df=-2 Nw
T= 135 Su38 p:891463725  9 2 L=23 S= 7 D= 6 E=33 E.Df= 2 Rd=0.380 Pb=0.744 Nw
T= 122 Su25 p:891463752 11 2 L=26 S= 9 D= 7 E=39 E.Df= 6 Rd=0.743 Pb=0.373 Od
T= 109 Su15 p:895463721 12 2 L=31 S=10 D= 7 E=42 E.Df= 9 Rd=0.502 Pb=0.193 Od
 Final position is : 8,9,1,4,6,3,7,2,5,# Final energy=33
>
```

Figure 9-19 6th Simulation with c=0.9

9.7 Summary

There are some findings in the simulation of this problem:

1. About at least 15 iterations (perturbations by cell shuffling) should be done before a reasonable minimum is found. Hence, the program execution with c>=0.9 give better result.
2. Exponential decrease in temperature gives much better result than that by linear decrease rate.
3. At fast temperature, T, decrease rate, i.e. simulation with small c (e.g. 0.72), many times of execution should be done to obtain an acceptable minimal, while at slow T decrease rate (c=0.9, 0.99), less number of iterations is required. However, in the case of very slow T decrease rate, the minimal is mainly determined in the range of T=200 down to 100.

9.8 References

[1] Dimitri P. Bertsekas, 'Dynamic Programming: Deterministic and Stochastic Models', pp.26-28, Prentice-Hall Inc, 1987.

[2] Eric V. Denardo, 'Dynamic Programming: Models and application', pp.16-18, Prentice-Hall, 1982.

10. Appendix II

The derivation of the algorithm's computation time (section 6.1) is summarized in this section.

1. Affinity Clustering Phase:

i. Construction of Connection Lists

The number of computation cycles is identical to the length of the connection list which is equal to the number of connections, $|e|$, of the circuit.

In step (ii) to (v), it is necessary to construct an ordered group list. Hence, the number of computation cycle of a step is approximately given by

$1/2 *$ mean of L_o and $L_f * N$ -------- Eq. 10-1

where L_o is the length of group list before execution of the step,

L_f is the length of group list after execution of the step, and

N is the number of grouped items (i.e. number of added items to the group list) in the step.

In step (ii) to (v), it is expected that the number of grouped items in each consecutive steps are n/2, n/4, n/8, and n/8 respectively.

ii. Primary Grouping

In this step, it is necessary to scan once the connection list with length $|e|$. Then, assuming that half of the cells are clustered in this step, to construct the group list with length n/2, the following number of computation is need:

$|e| + 1/2 * (0+n/2)/2 * n/2 = |e| + n^2/16$ -------- Eq. 10-2

iii. Element Appendage to Existing Groups

In this step, the length of the connection list left is $|e|/2$, L_o is n/2 and L_f is 3n/4. Hence, the number of computation is:

$|e|/2 + 1/2 * (n/2+3n/4)/2 * n/4 = |e|/2 + 5n^2/64$ -------- Eq. 10-3

iv. Loose Appendage of Ungrouped Elements

In this step, the length of the connection list left is $|e|/4$, L_o is $3n/4$ and L_f is $7n/8$. Hence, the number of computation is:

$$|e|/4 + 1/2 * (3n/4+7n/8)/2 * n/8 = |e|/4 + 13n^2/256 \quad \text{--------} \text{ Eq. 10-4}$$

v. Single Element Group Formation

In this step, the length of the connection list left is $|e|/8$, L_o is $7n/8$ and L_f is n. Hence, the number of computation is:

$$|e|/8 + 1/2 * (7n/8+n)/2 * n/8 = |e|/8 + 15n^2/256 \quad \text{--------} \text{ Eq. 10-5}$$

2. Alteration Phase:

i. Element Assignment to a Group

It is necessary to scan through the group list once to determine which group the added element belongs to. Hence, the number of computation is n.

ii. Empty Space Searching

To find the empty space, the step consists of two parts: 1) to search the nearest empty space, 2) to locate the nearest group element. The expected length between an empty space and the group element is equal to the expected length between any two points in an nXn grid array. That is

$$1/2 * (2)^{1/2} * (n)^{1/2} = (n/2)^{1/2} \quad \text{--------} \text{ Eq. 10-6}$$

where $(2)^{1/2}$ is diagonal distance in a grid, and
$(n)^{1/2}$ is the length of the array.

Since this step consists of two parts with the method, the number of computation time is

$$2 * (n/2)^{1/2} = 2(n/2)^{1/2} \quad \text{--------} \text{ Eq. 10-7}$$

iii. Determination of Direction of Element Allocation:

- Cross-cut Direction of Allocation

To determine the cross-cut direction of allocation, it is necessary to scan through the element between two points in the nXn array. Hence, the number of computation is equal to that in Eq. 10-6.

- Dynamic Determination of Path Based on Size Functions
  - Segmentation of Cross-cut

In this part, the path with length given by Eq. 10-6 is divided into segment with length 4, the number of computation time is equal to the number of segments:

$(n/2)^{1/2}/4$ -------- Eq. 10-8

- Partial Optimization of Segments

In this part, there is an average of 3 templates for each segment, the average of length of each segment is $2(2)^{1/2}$, and the number of segment is given by Eq. 10-8. Hence, the number of computation is

$3 * 2(2)^{1/2} * (n/2)^{1/2}/4 = 3n^{1/2}/2$ -------- Eq. 10-9

- Dynamic Linking of Segments

In this part, since each segment have about 3 templates (section 3.2.3.2.2), the number of computations is approximately the number of combinations of templates in a path.

For a path with $L_{MHT} \geq 6$, there is two parts of segments each with 3 templates. Hence, the number of combinations of path segments is 3^2.

For a path with $L_{MHT} \geq 10$, there is three parts of segments and the number of combinations of path segments is 3^3.

In general, for a length of L, the number of combinations is given by

$$f(L) = 3^{[(L/A.M.) + 1]}$$

where A.M. is the arithmetic mean of the length of a path segment.

Since the arithmetic mean is (4+5)/2 = 4.5,

$$f(L) = 3^{[(L/4.5) + 1]}$$ --------- Eq. 10-10

Since the average length of path in an array with size n is $(n/2)^{1/2}$, Eq. 6-2 is given by

$$T = 3^{[(n^{1/2}/6.36) + 1]}$$ --------- Eq. 10-11

- Element Allocation

Since the number of moves (shifting) of cells is equal to the expected length between any two points in an nXn grid array. The equation is identical that in Eq. 10-6:

$(n/2)^{1/2}$ -------- Eq. 10-12