To the Graduate Council:

I am submitting herewith a thesis written by James E. Hilger entitled "Hierarchical Clustering Of Digital Circuits". I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master ofScience, with a major in Electrical Engineering.

Donald W. Bouldin

Donald W. Bouldin, Major Professor

We have read this thesis
and recommend its acceptance:

Marshall O. Pace

Jeffrey D Case

Accepted for the Council:

Vice Provost
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's/Electrical Engineering degree at the University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in her absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature James E. Hilger

Date July 13, 1988

HIERARCHICAL CLUSTERING OF DIGITAL CIRCUITS

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

James E. Hilger
August 1988

ACKNOWLEDGMENTS

I express many thanks to Dr. Donald W. Bouldin for his guidance and patience throughout this thesis project. Thanks go to Dr. J. Case and Dr. M. O. Pace for serving as committee members and for their suggestions and input. I would also like to thank Dr. Case for lending me the PCAD hardware and manuals needed to make this project possible. Thanks go to Ken Key for answering questions about PCAD.

I express sincere appreciation and thanks to my grandmother, Mrs. Lillian Harrison for her encouragement and support which made this degree possible. I express many thanks and sincere appreciation to my beloved and wonderful wife, Jeannette H. Hilger, for her proofreading, encouragement, support, and assistance in completing this thesis.

ABSTRACT

The increased complexity of schematics has generated the need for comprehensible representation of the schematics. An algorithm to create a hierarchical representation of a flat digital schematic is presented. The algorithm is implemented in the 'C' programming language. The input is a text file representation of a schematic database used by the P-CAD schematic capture package. Schematics are partitioned into modules containing 7 ± 2 blocks to facilitate comprehension. Clustering of logic symbols is done to make the algorithm independent of the person drawing the schematic. A preclustering step is incorporated to aid in the development of module partitions. The algorithm implementation operates on a personal computer.

A quality measure is used to grade the schematic partitioned. The quality measure is based on a Guassian distribution function and the average number of logic symbols per page. Several digital schematics are used as a test cases. The hierarchical algorithm's performance is analyzed by interpreting the quality measure.

TABLE OF CONTENTS

FIGURES

CHAPTER I

INTRODUCTION

As the complexity of digital logic systems continue to increase, the need arises for efficient design aides and methodologies. Automated design packages allow digital logic systems to increase exponentially in the number of components and complexity. Continually increasing complexity without efficient design methodologies may produce human errors due to lack of comprehending details. By grouping a set of logic components and abstracting to a higher-level structure, complexity is reduced.

In order to create a hierarchical system, two design methods exist, top down and bottom up. Top down design involves breaking the design into less complex modules. Using less complex modules to form a complex structure is known as bottom up design. By grouping sets of logic components into functional modules, partitions in the system are created. Grouping more components and modules together forms different levels of abstraction. Thus, the viewer is allowed to view any level of abstraction desired.

Grouping or clustering of logic symbols into modules allows for the repeated use of the modules. Designers are not required to relearn the specifics of a group of gates, but must remember the functionality of groups. Productivity is increased because the modules are designed once.

The purpose of clustering a digital circuit schematic is to generate a schematic that is easy to both read and comprehend. Schematics that are more concise and comprehendible lead to fewer engineering errors, contribute to ease of reading, and help to expedite engineering changes. Cluttered schematics are difficult to read and cause confusion that leads to less understanding of how the system operates, delays in system development, and increased costs.

This thesis details the work undertaken to develop and implement an algorithm that partitions a digital schematic. The digital schematic is entered graphically with the schematic capture system by P-CAD®. The algorithm is implemented in the 'C' programming language using the Microsoft® C compiler in the MS-DOS® environment.

The theoretical and previous work in hierarchical clustering of digital schematics is presented in Chapter II. Included in Chapter II are the basic theoretical considerations and

latest developments in partitioning. Important equations and theories are stated and discussed also. Chapter III details the algorithm development and flow, including all of the alternatives considered and the final algorithm. The results of test cases are discussed in Chapter IV. Conclusions of this project are presented in Chapter V.

CHAPTER II

THEORETICAL CONSIDERATIONS

When certain items of a set exhibit the same characteristics, this set can be divided into subsets. These subsets or clusters have the same overall properties and have strong similarities. Several algorithms [3], [11] exist to perform clustering based on these similarities. Many applications [8] use a measure of similarity based on distance and thresholds. Some of these methods [1], [5], [9], [10], [12] are used as the basis for the hierarchical clustering of digital circuit schematics.

I. Related Work

The latest area of study is that related to computer aided design (CAD) systems in the layout of very large scale integration (VLSI). The algorithms use clustering techniques that pertain to hierarchical clustering of digital circuit schematics. The same idea is used in each in that the partition is made according to the number of connections between components. The process of generating a hierarchy of the physical elements is called partitioning [6]. There are several types of partitioning algorithms. They are sequential, parallel and those that utilize hierarchical information.

Payne and vanCleemput [6] presented a heuristic algorithm for automatic partitioning of digital systems. The heuristic algorithm used high level information to increase the overall effectiveness of it. A quality function was utilized to measure the partition's performance to find the optimum solution. Prepartitioning was used to set up the system for the actual partitioning step. The algorithm prepartitioning adapted a quality function and processed design constraints. The adoption of a quality function is the generation of a function that best fits the specific system being partitioned. The algorithm selected the smallest logic block and constructed a physical block by combining the physical designs of its parts. This design was improved iteratively and the whole process repeated until the system was partitioned completely.

A sequential algorithm took 4 times longer to execute and had physical blocks that contained 14 percent less logic than a hierarchical algorithm presented by Payne [5]. The

results of the sequential algorithm were considered of lower quality by Payne [5] due to the fact that in the implementation of the sequential algorithm, hierarchical information was not utilized.

An important question in clustering concerns the optimum number of clusters in the partitioning process. Another equally important question is deciding where are the cluster centers. Without some *a priori* knowledge of the data set, these questions seem impossible. The optimum method is to have the computer answer these questions on a schematic to schematic basis with no *a priori* knowledge. Several algorithms for clustering have been presented [3], [11].

In an answer to the clustering questions, Davies and Bouldin [2] presented a cluster separation measure that does not depend on the number of clusters or the method of partitioning them. This method did not use the common objective function by itself, but utilized a general cluster separation measure R. This allowed the user to input only the exponents for the type of distance measure wanted. The types included Euclidean and "city block" distance measures [2].

Schuler and Ulrich [9] used the concept of clustering trees in conjunction with sets of interconnected nodes. Clustering was done in a pairwise fashion. This required several cycles of calculating and updating a clustering value (CV). As pairwise elements became clusters, the CV was updated and the newly formed cluster was then considered an element for more clustering. This continued until all elements belonged to a cluster. The results obtained were difficult to verify due to the fact that the only way to obtain the optimum solution for the test cases used was to exhaustively search the solution space [9]. Sha and Dutton [10] also used a method similar to clustering trees in the placement of arbitrarily sized rectangular blocks.

Preas and Karger [8] presented a paper on the current techniques of automatic placement. The paper listed and described all the current techniques being utilized for a placement function within automatic layout systems. A technique of simulated annealing was described. This technique was the answer to the problems that caused other heuristic algorithms to stop in a minimum rather than finding the overall minimum. Simulated annealing allows the program to explore the solution space until it is stopped externally or a threshold is satisfied. An advantage to this type of algorithm is that it will not stop even after several iterations have caused the problem to become worse.

More recent work in schematic partitioning includes the work of Chun, Chang and McNamee [1]. This algorithm utilized the net-list description of a digital schematic to do the partitioning. The primary area of interest was feedback loops in the schematics. The algorithm used an incidence matrix specifically for feedback loops. An incidence matrix is a square matrix with rows and columns designated as source of signals and target of signals respectively. For each row, a '1' is entered into all columns for which the column element is a target of the source signal [1]. This matrix also helps solve the problem of identifying which element is in the forward path and which is in the reverse feedback path. The knowledge of what path an element is in made the placement of these elements easier. The algorithm employed a combination of heuristic and algorithmic methods to accomplish the partitioning. The algorithm was presented in the 1987 Design Automation Conference.

II. K-Means Algorithm

The k-means algorithm [11] is based on the minimization of an objective function which is defined as the sum of the squared distances from all points in the cluster to the cluster center. This is a sequential clustering algorithm that is influenced by the order of input data, number of clusters specified, and clusters which are selected as the initial clusters.

The algorithm proceeds by arbitrary selection of k initial cluster centers. At the n^{th} step, the data samples are distributed such that

$$\mathbf{X} \text{ belongs to } U_j(n) \text{ if } \|\mathbf{X} - \mathbf{V}_j(n)\| \leq \|\mathbf{X} - \mathbf{V}_i(n)\| \qquad (1)$$

for all $i = 1, 2, ..., k$, $i \neq j$, where $U_j(n)$ represents the set of data samples whose cluster center is $\mathbf{V}_j(n)$.

After all the data samples have been distributed, new cluster centers are calculated by

$$\mathbf{V}_j(n+1) = (N_j)^{-1} \sum \mathbf{X}, \quad j = 1, 2, ..., k \qquad (2)$$

where N_j is the number of samples in $U_j(n)$. The new cluster center calculated by Equation 2 minimizes the objective function defined above. If $V_j(n+1) = V_j(n)$ for j = 1, 2, ..., k then the algorithm has converged and the procedure is terminated. Otherwise the algorithm goes back to distribute all the samples on the $n^{th} + 1$ step.

III. Quality Measure

The quality of the hierarchical partitioning of a schematic is based on the criteria of 7 ± 2 blocks [5]. The optimum condition is to have seven blocks on each page of the schematic. The farther away from seven, the lower the partitioning value. The worst case is to have the entire schematic of n blocks on one page. Hence, a quality measure is developed to grade the partitioning.

Considering a Gaussian distribution with a mean of seven and standard deviation of two, the closer to the mean the better the partitioning. A quality performance (QP) factor is calculated using the ideal solution of seven blocks. The QP factor is calculated using Equation 3. Whenever ß equals seven, QP equals 100%. In this calculation, ß is the mean number of blocks on a page after the schematic has been partitioned. The magnitude comes from considering a Gaussian distribution where ß can be greater or less than seven.

$$QP(\%) = 100e^{-\{(|ß - 7|) / 8\}} \qquad (3)$$

Notice that as the value of ß gets larger, QP becomes smaller. With the criteria that each page will contain only 7 ± 2 blocks, the QP factor has the limits shown in Equation 4. This quality measure was chosen so that the higher the QP percentage, the closer the partitioning is to seven blocks.

$$0\% < QP \leq 100\% \qquad (4)$$

The schematic is partitioned hierarchically, and the number of blocks per level on a page, ß, is calculated. With the values of ß, the QP percentage is calculated for each level in the hierarchy using Equation 3. The final QP associated for the schematic is calculated by averaging together all the QP factors. The partitioning process can then be adjusted by human intervention in an attempt to achieve a higher QP percentage.

CHAPTER III

THE HIERARCHICAL ALGORITHM

In order to partition the digital schematic into a hierarchy, a measure of cluster similarity has to be established. Similar digital component symbols are clustered together to form a partition that contains 7 ± 2 blocks. The clustering of these symbols is based on their x and y coordinates. These coordinates are stored by the schematic capture system software when the user graphically enters the schematic.

Partitions of 7 ± 2 blocks constitute a page of the schematic. After all the digital component symbols on a page have been partitioned, these partitions are themselves clustered together to form a hierarchy. This process continues until there is one page that contains a single block representing the entire system. Thus, the original flat schematic is transformed into a hierarchical schematic. Any hierarchical schematic can be flattened and then hierarchically partitioned by the algorithm for comparison purposes.

I. System Flowchart

The original schematic is contained in a file with the extension .SCH. This file contains the database for the schematic in the schematic capture system. The next step is to use the PDIF-OUT program of the schematic capture system. This transforms the database into a text file that can be read by the hierarchical clustering algorithm. The output of this algorithm is in the format that the PDIF-IN program recognizes. This part of the schematic capture system transforms the text file into the format of the schematic database that can be viewed on the screen. Figure 1 shows the overall system flow described above.

II. Design Alternatives

Various alternatives were considered in the development of the hierarchical algorithm. Alternatives considered included region growing and preset page sizes. The final working algorithm cannot be completed until careful consideration of design

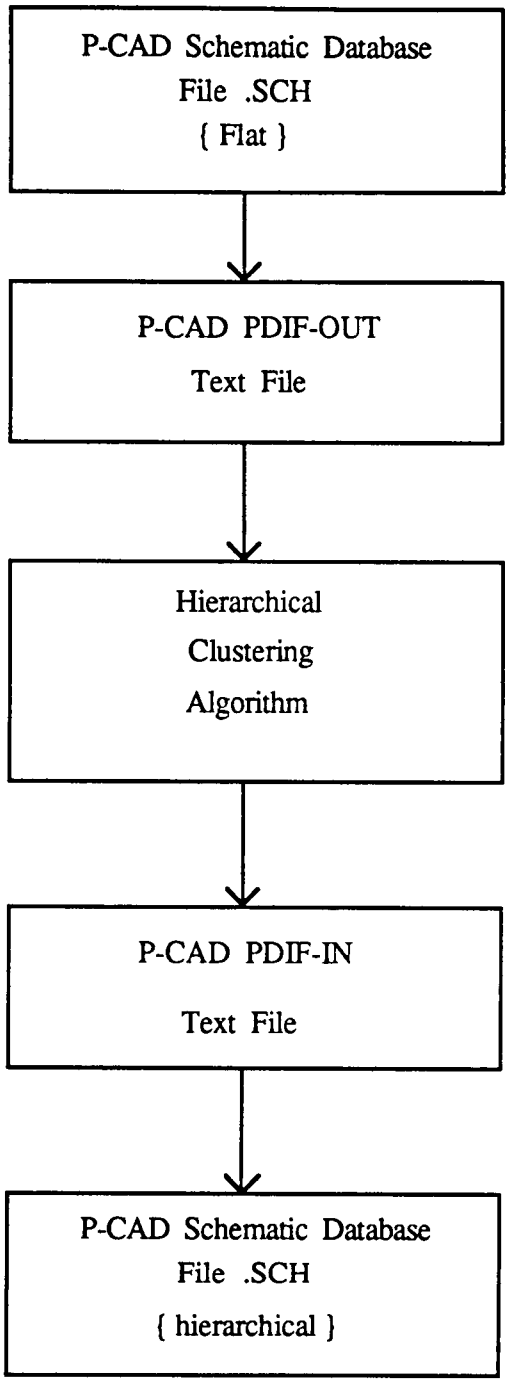

Figure 1. The overall system flowchart.

alternatives. Effects of design alternatives on the resulting output schematic directed selection of alternatives for use in the hierarchical algorithm.

Region Growing

Starting in a particular area of the schematic and moving outwards away from the area, the schematic can be partitioned into pages of 7 ± 2 blocks. Region growing continues until the whole schematic is partitioned. The algorithm for this method starts at a point in the schematic and traces all the nets until 7 logic symbols are contained within a boundary.

The advantage of region growing is that when a starting point is chosen, the growth continues until all the component symbols are partitioned into pages. The logical starting point is a corner of the schematic. Considering the nets and gates at the same time has the advantage in that the schematic text file is read once for each page. Hence, less time is required since fewer file openings and closings are necessary.

On the other hand, the disadvantages of growing a region using the schematic capture system text format as the input are numerous. The structure of a schematic capture system text file must be considered. In the creation of a hierarchy, a graphical symbol must be defined. The definition includes the input and output logic designations, pin names, and all graphical information needed to draw the symbol. The symbol definition is written in the text file first, thus this information has to be known first. In looking for information to write the definition, the region is grown and nets leaving and entering the region are marked. A logical approach is to store all of the information found, then write the information to a file at the appropriate time. The problem is the storage required to save all the information. The growth starts to spread out as nets are followed generating large amounts of information. The structure of a schematic capture system text file makes region growing an unattractive approach to the partitioning problem.

Other problems with region growing include deciding which nets to trace and how far to trace the nets. Nets and logic symbols are written in separate areas of the text file as is shown in Figure 2 [7]. The task of deciding how far to trace the nets to get 7 ± 2 blocks is difficult for as the nets are traced, a search of the logic symbols for net connections must be done. Net tracing and searching for logic symbols are both complex and time consuming.

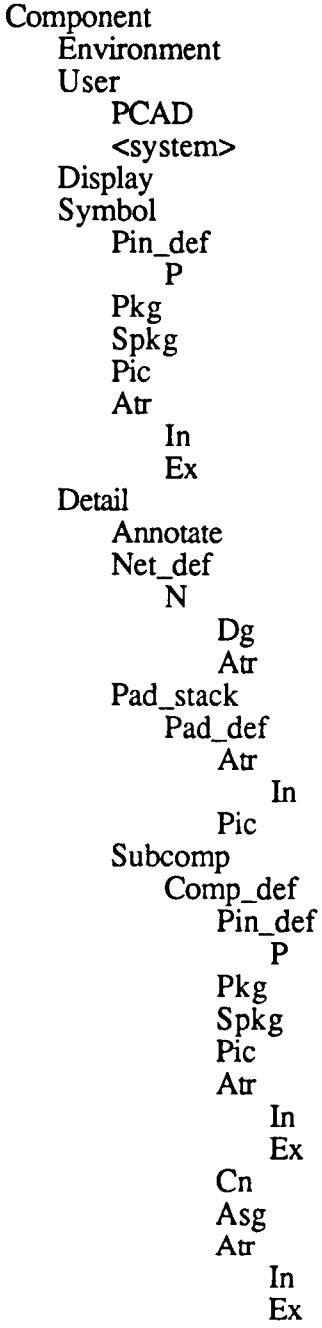

Figure 2. Outline of a PDIF File.

Preset Page Sizes

Preset page sizes is another design alternative. This method consists of using preset x and y coordinates to mark the boundary of each page. All of the logic symbols and nets within the preset page boundaries are considered as part of a page. The page sizes remain the same for all schematics, therefore algorithm performance is independent of the schematic partitioned.

Using preset page sizes avoids the time consuming and complex task of net tracing and searching for logic symbols. With preset page sizes, the x and y coordinates of a page boundary are known before the algorithm begins. The knowledge of page boundaries before starting the algorithm is an advantage since only a search for nets and logic symbols within the page boundary is necessary. This approach to partitioning is less time consuming and not as complex to implement as region growing.

A disadvantage of preset page sizes is that there are no guarantees that each page will contain 7 ± 2 blocks. The schematic might contain areas of nothing but nets. Preset page sizes allow pages containing only nets and blank areas and logic symbols outside of the 7 ± 2 block criteria. Without flexibility in page sizes, the resulting output schematic, along with algorithm performance is dependent on the person drawing the original schematic.

III. Algorithm Flowchart

Flexibility and the schematic capture system text file format guided development of the hierarchical algorithm. A variation of preset page sizes is used to make the hierarchical algorithm performance independent of the input schematic text file and the person drawing the schematic. Clustering is used to make the hierarchical algorithm independent of the person drawing the schematic, and a preclustering step is developed to allow flexible page sizes. Figure 3 shows the flowchart for the hierarchical clustering algorithm being presented.

Preclustering

The hierarchical clustering algorithm begins with a preclustering step. This step reads the text file and looks for the x and y coordinates of certain digital logic symbols. The number of component symbols is tabulated and is used to find the number of cluster centers. The number of cluster centers used to start the algorithm is found by dividing the

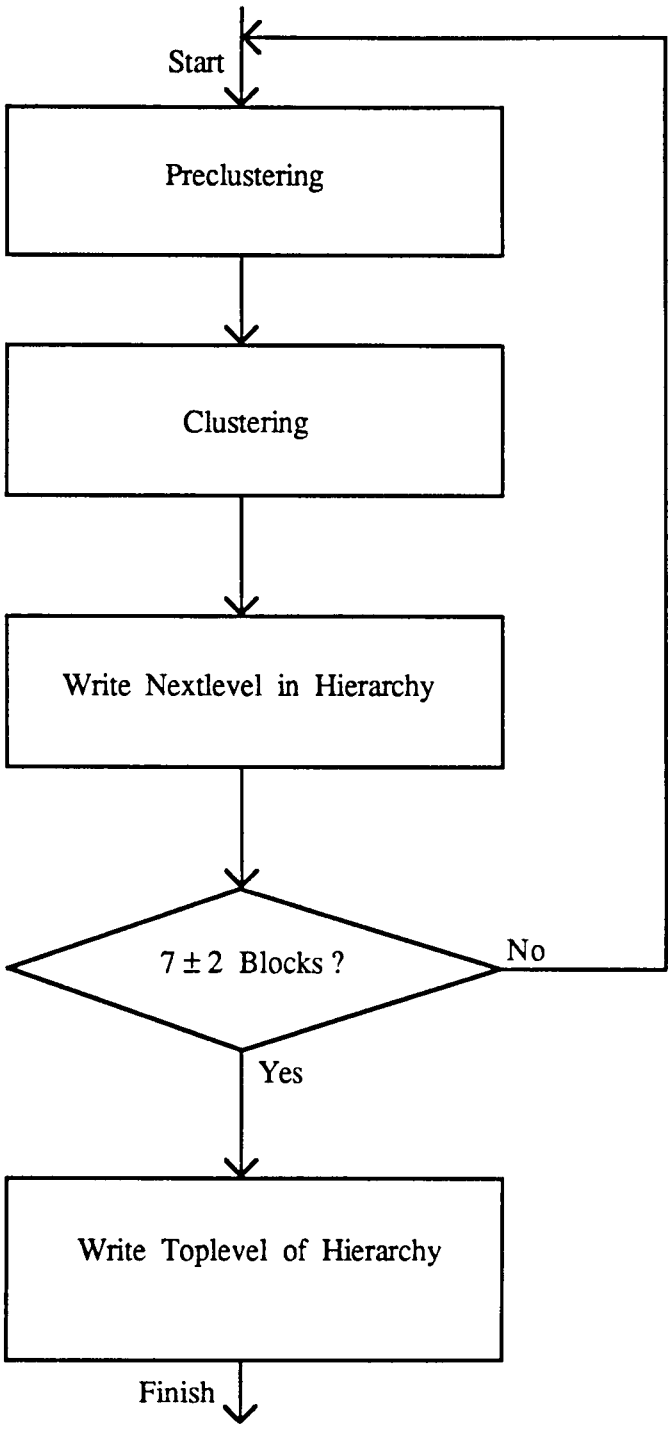

Figure 3. The overall algorithm flow.

total number of component symbols by seven. This will allow the algorithm to have partitions that are as close as possible to the criteria of 7 ± 2 blocks. When the cluster centers are found, the preclustering step is complete.

Clustering

The algorithm next begins the clustering step in which all of the component symbols are considered. The component symbols are partitioned into groups using the cluster centers found in the preclustering step. These groups have boundaries that extend into neighboring groups. The boundaries are trimmed such that no groups overlap. This step partitions each group into rectangular shape. The clustered component symbols are now defined as pages.

Partitioning

Each page has nets that connect component symbols and nets that are routed through the page with no connections to the component symbols. Consideration of all nets within a page insures that all inter-page connections are accounted for. The pages are then put into the PDIF text format and written to separate files. Figure 2 shows the outline of a PDIF text file. The pages are then considered as modules.

Next Level

The modules are combined in a schematic capture system text format file to form the next higher level of the hierarchy. The next level has the modules and connector symbols as components. The nets between modules and between connector symbols and modules are included in the file. If there are more than 9 modules, the algorithm is repeated with the next level file as the input data. This cycle continues until there is only one page with 7± 2 blocks that represents the entire system. When this stage is reached, the hierarchical clustering is complete.

Top Level

The final step in the algorithm is to create a text file in the schematic capture system format that represents the top level of the hierarchy. The top level has a box that

represents the entire system and connectors as components. The nets included in the top level are between the connectors and the box. When the next level is written in a schematic capture system text formatted file, the hierarchical clustering algorithm is finished.

IV. Functions

Functions written in the 'C' programming language accomplish all of the manipulations to form the hierarchy. The algorithm is comprised of twenty-two specialized functions. *Main* is the primary calling function that controls the algorithm flow by calling functions in the correct order. Also, *main* calculates the QP factor for each level in the hierarchy and the final QP factor for the entire schematic. Figure 4 shows the communication between functions. Figure 5 shows the flowchart for the function *main*.

Preclustering Functions

The preclustering is accomplished with the 'C' function *train*. The preclustering starts with reading in the coordinates of only certain logic symbols of the schematic. The other symbols such as resistors, capacitors and connectors are not considered due to the additional clusters that are created by including them. These additional clusters lead to pages that contain only connectors or other symbols with little information.

The initial cluster centers and cluster memberships are calculated using the k-means algorithm in the function *train*. The clusters contain a variable number of symbols. The cluster members are then reordered in increasing distance from the cluster center. Clusters that contain eight or more members are reduced to only seven members by reclustering members that are farthest away from the cluster center. The members are added to the next closest cluster that contains less than seven members.

New cluster centers are calculated after the member rearrangement. These are the cluster centers that are used in the cluster step. The preclustering is finished after calculating the new cluster centers.

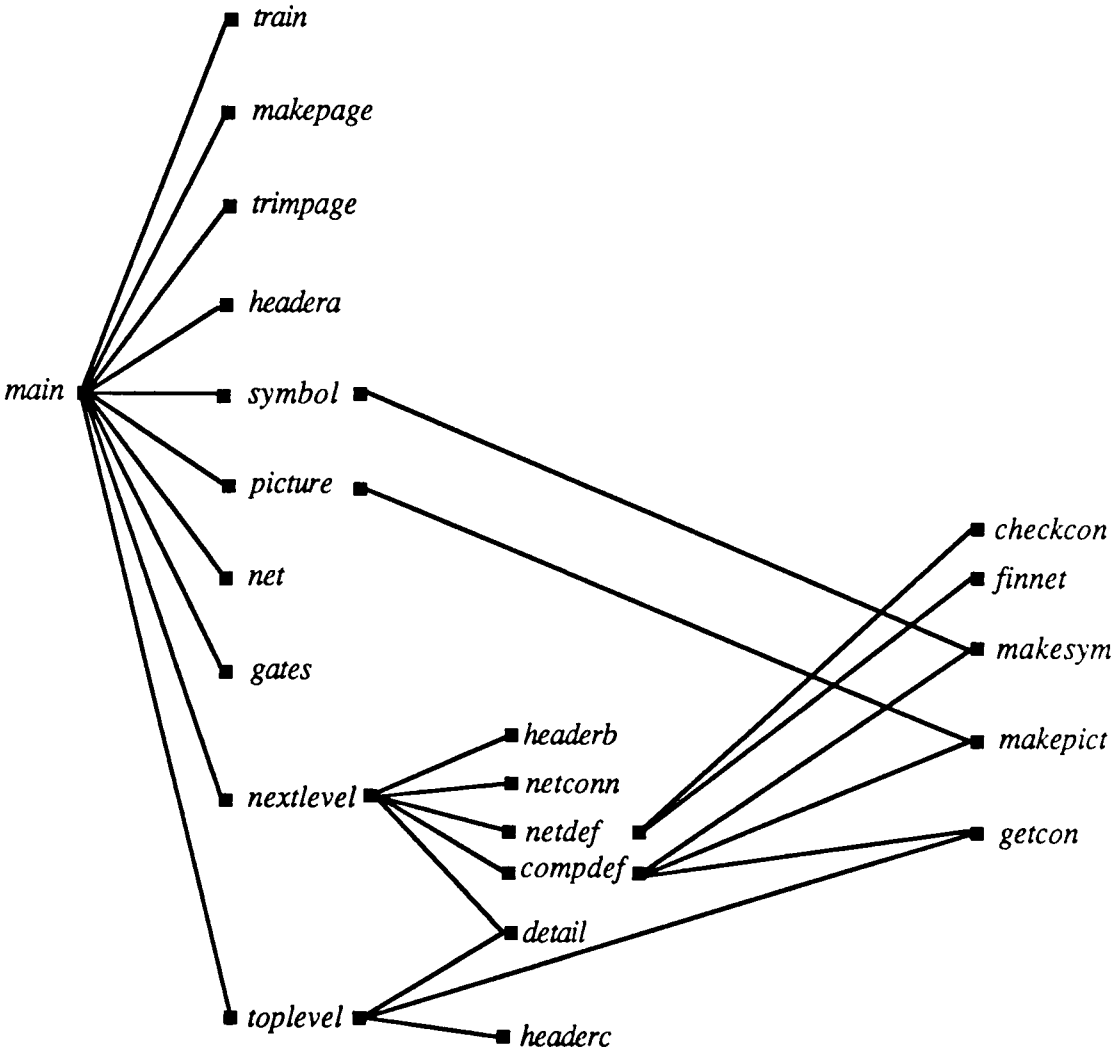

Figure 4. Communication between functions.

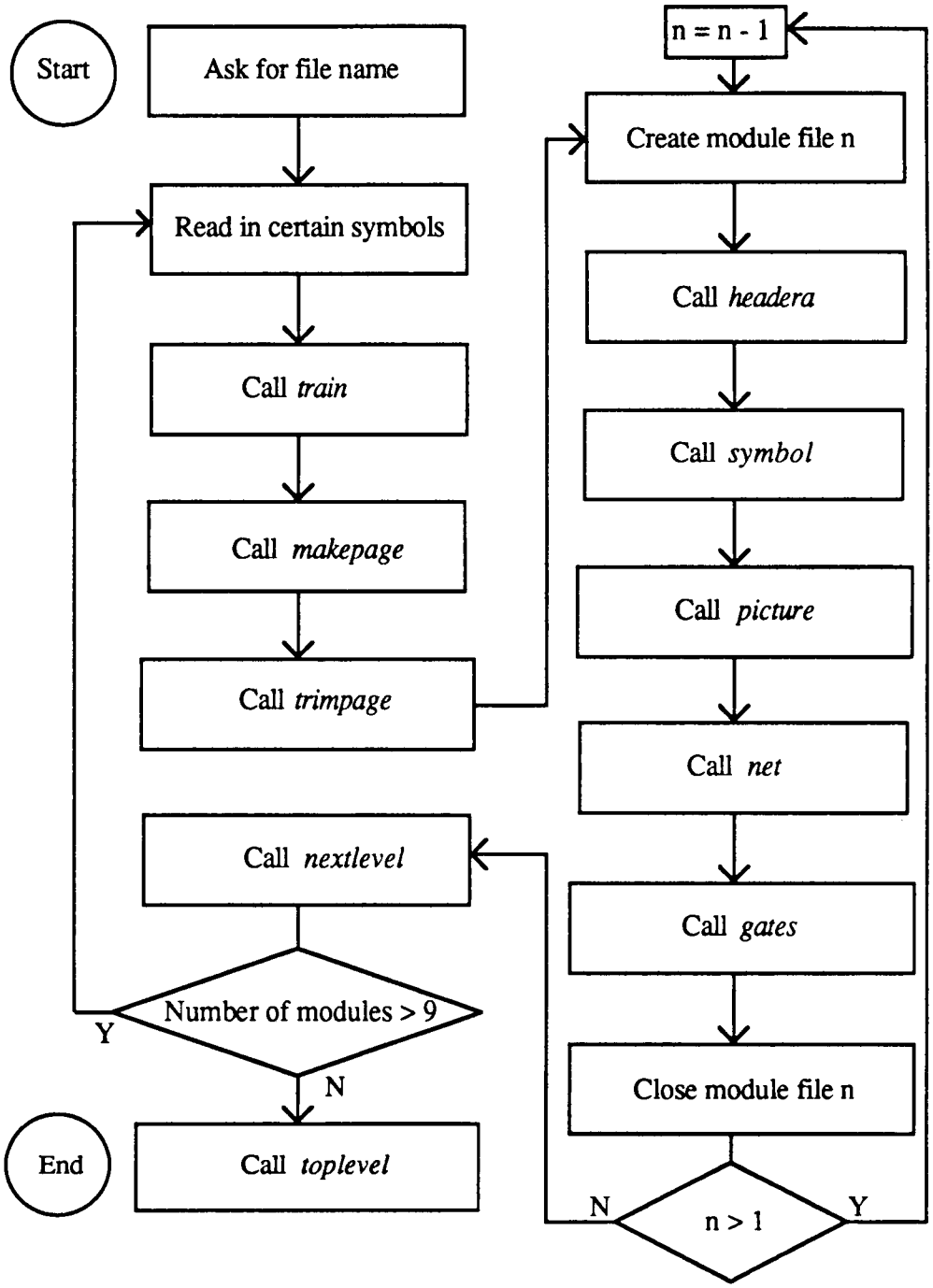

Figure 5. The flowchart of function *main* .

Clustering Functions

The clustering is accomplished within the 'C' functions *makepage* and *trimpage*. Clustering the schematic begins by reading in and clustering each symbol with the closest cluster center. As each symbol is read in, a number is assigned to the symbol for identification at a later time. The symbol number is to be used in writing a schematic capture system text file for each cluster. After reading in all the symbols, the maximum and minimum values of x and y for each page are determined. Clustering all the symbols is accomplished in the function *makepage*.

The 'C' function *trimpage* checks for any overlap of pages. A rectangle is formed by using the minimum and maximum x and y coordinates. Page overlaps are determined by checking for any boundary overlap of two pages using x and y coordinates. When two pages are determined to overlap, the page with the longest y boundary is reduced to five units past the other page. After elimination of overlaps, all of the component symbols are grouped according to the new x and y coordinates. Thus, the clustering of component symbols is completed.

Partitioning Functions

After clustering, partitioning functions create schematic capture system text files that consist of user environments, symbols, and details. The schematic capture system text file is an ASCII representation of the schematic capture system design database. The 'C' functions *headera*, *symbol*, *picture*, *net*, and *gates* write portions of a schematic capture system text file for each page.

Headera is the 'C' function writing the user environment section of the schematic capture system text file. Each page must be put into a separate file. Files for each page are distinguished by placing a page number in each file name. The page number is placed also in the component name within the file.

Symbol writing is accomplished by the 'C' functions *symbol*. The symbol consists of a rectangle representing the page with pins referring to nets leaving and entering the page. Pins are located by searching through all nets to find nets that cross the rectangular boundary. The function *symbol* calls the function *makesym* to do the search. If a net crosses the boundary, the net is broken and is considered a pin of the rectangle. All pin information found is written to the schematic capture system text file of the appropriate page.

After defining the symbol, a graphical representation of the symbol is written to the schematic capture system text file. *Picture* is the function that writes the coordinates of all graphics needed to draw the symbol. *Picture* calls another function called *makepict* to find all the graphical information needed to draw the symbol.

Details consist of all net and gate definitions and locations. 'C' functions *net* and *gates* write the needed definitions for each page in the schematic capture system text files. All nets within the rectangular boundary of a page are written in the schematic capture system text file. Using the component numbers assigned to components during the clustering step, the function *gates* copies all of the gate definitions within the page boundary to the schematic capture system text file. Upon completion of writing gate definitions, files for each page are complete. The pages are now defined as modules.

Next Level Functions

Nextlevel, *headerb*, *netconn*, *detail*, *netdef*, *checkcon*, *finnet*, *compdef*, and *getcon* are the functions that create and write to a file, the next level of the hierarchy. *Nextlevel* is called by the primary calling function *main*. Figure 6 shows the flowchart for the function *nextlevel*. The modules written to separate files by the partitioning functions and connector logic symbols from the original input file are the logic symbols in the next level of the hierarchy. The schematic capture system text file has the same format as shown in Figure 2.

Headerb writes the user environment section to a schematic capture system text file with the extension .SYM. *Netconn* searches the connector definitions to get a list of nets associated with the connector symbols. *Detail* writes the symbol definition needed for the next level of the hierarchy. The symbol definition is written by using the net list found by *netconn* to write the correct number of pins for the rectangle representing the next level in the hierarchy. When *detail* has completed writing the symbol definition for the next level in the hierarchy, the detail section of the schematic capture system text file is written next.

The function *netdef* routs and writes the nets between modules. *Netdef* calls *checkcon* to check for connections to the logic symbols of the current being defined. *Finnet* is called last to search the modules for any net connections missed by *netdef* and *checkcon*. The reason for this last check is to be sure that all of the connections needed to define the net are found due to the fact that a net must be defined in only one complete definition and not several split definitions.

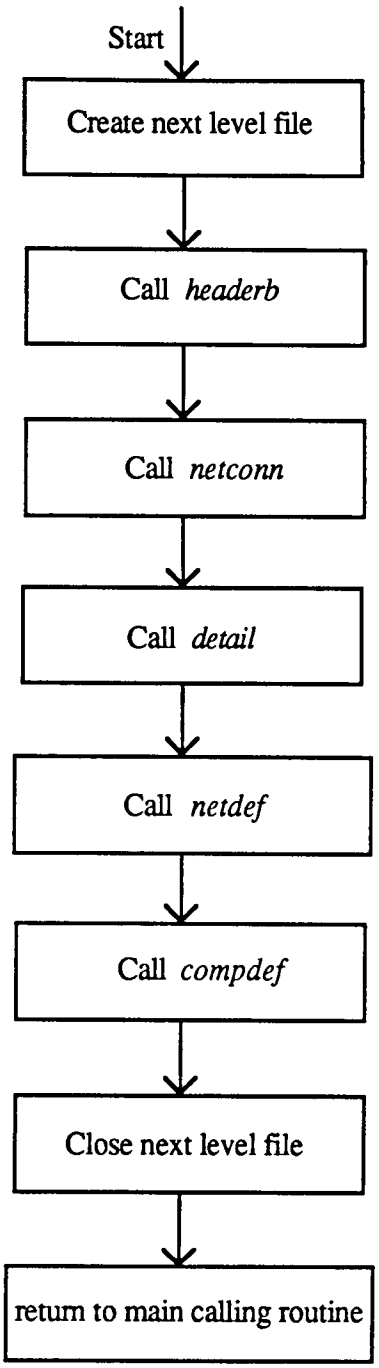

Figure 6. The flowchart of function *nextlevel*.

Compdef writes the module definitions. The definitions include all graphical information needed to draw the module logic symbol and net connections to the module. Functions *makesym* and *makepict* are called to find all information needed to write the module definitions. After defining all modules, *comdef* calls the function *getcon*. *Getcon* copies the connector symbols definitions into the schematic capture system text file. When the function *compdef* is finished, the next level in the hierarchy is completed.

Top Level Functions

The 'C' functions *toplevel*, *headerc*, *detail*, and *getcon* create the top level in the hierarchy. *Toplevel* is called by the function *main*. Figure 7 shows the flowchart for *toplevel*. *Headerc* writes the user environment section to a schematic capture system text file with the extension .SCH. *Toplevel* calls *detail* and *getcon* to complete the detail portion of the hierarchical top level. The nets connecting a rectangle, which represents the entire system, and connector logic symbols are written to the schematic capture system text file by *toplevel*. *Detail* then writes the rectangular logic symbol used to represent the top level in the hierarchy in the schematic capture system text file. The definitions of all connector logic symbols are written to the schematic capture system text file by the function *getcon*. After completion of all component logic symbol definition writing, the file for the top level in the hierarchy is closed. Thus, the algorithm is completed and the final QP factor is calculated for the complete schematic by the calling function *main*.

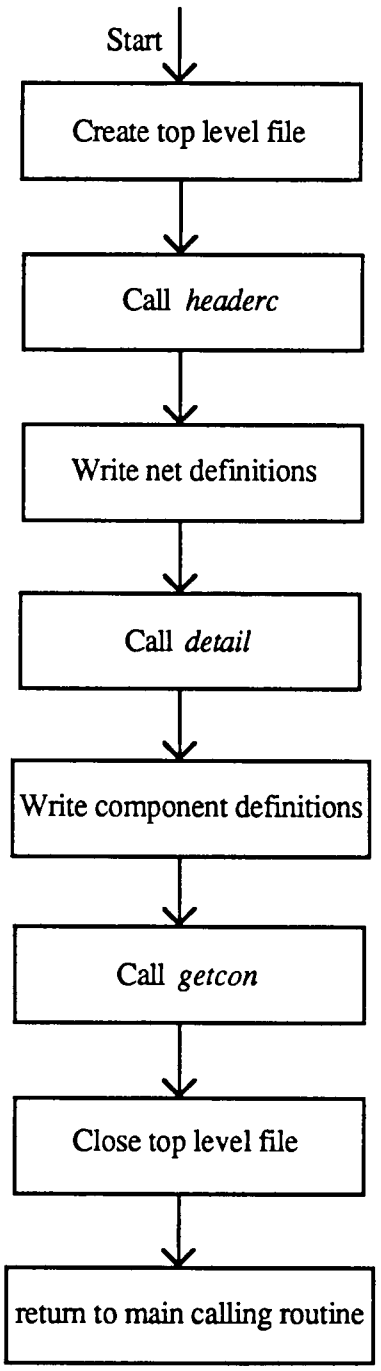

Figure 7. The flowchart of function *toplevel*.

CHAPTER IV

RESULTS

The hierarchical algorithm is started by typing 'hpart' at the system prompt. The working display of the hierarchical algorithm is similar to the schematic capture system. User familiarity with the schematic capture system is not essential to use the hierarchical algorithm implementation for only file names are needed as input. Figure 8 illustrates the computer screen during a sample run. Results presented in the following sections were obtained using the Texas Instruments (TI) professional computer with MS-DOS® version 2.13.

I. Design Constraints

Constraints of the hierarchical algorithm are attributable to two systems, the operating system of the TI professional computer and the Microsoft® C compiler and linker. The limitations imposed on the hierarchical algorithm allowed the algorithm to be computer independent. Figure 9 summarizes the design constraints and algorithm limitations.

Operating System Limitations

Algorithm development on the TI professional computer limited the size of the 'C' programming language implementation. In order to keep the algorithm portable, the 'C' programming language is used and all display graphics are kept to a minimum. Limiting the display graphics to a minimum allows the algorithm to be computer independent since graphics on the TI professional computer are different than graphics on other models of computers. The present 'C' implementation is portable to all computers with one megabyte or 640 kilobytes of random access memory (RAM).

With one megabyte of RAM, the hierarchical algorithm can have 200 pages per level in the hierarchy. Each page can have a maximum of 30 logic symbols. The 30 logic symbols include the 7 ± 2 block criteria, resistors, capacitors, and other schematic calibration symbols. More pages can be obtained with a larger RAM. Use of a hard disk

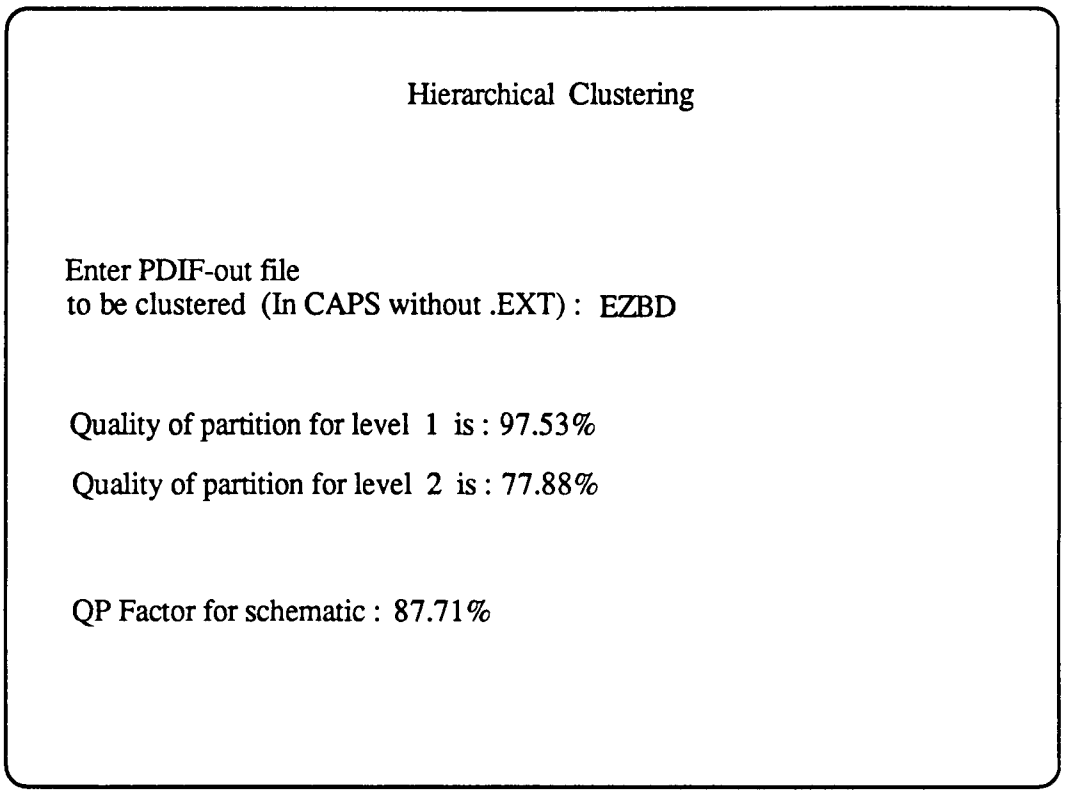

Figure 8. Sample of screen during test run with EZBD.

Attributes	Limitations
1. Number of pages	200 maximum-1Mbyte RAM 100 maximum-640kbyte RAM
2. Number of levels	∞
3. Number of logic symbols	30 per page
4. Number of pins per module	100 maximum
5. Number of component types	600 maximum
6. System memory required	1 megabyte RAM 640 kilobyte RAM
7. System requirements	Hard disk
8. Operating system requirements	DOS 2.0 or later
9. Compile requirements	/AH option Microsoft® C 4.0 or later
10. Computer	All computers

Figure 9. Algorithm constraints and features.

as temporary storage is not done because of speed, number of file reads and writes, and increased algorithm complexity. The number of levels in the hierarchy is unlimited.

The number and size of files containing the 'C' programming language implementation of the hierarchical algorithm is limited on the TI professional computer. The line editor limited the file size to 1300 lines. In addition, the MS-DOS® operating system limits the number of files opened at one time. Thus, the 'C' implementation had to be carefully planned in advance as to which files contain what functions.

Compiler and Linker Limitations

Microsoft® C compiler and linker version 4.0 is the development tool used for the hierarchical algorithm. Limitations imposed on the algorithm by the use of Microsoft® C are only those imposed by the operating system. The 'C' compiler is an implementation of the original definition of the 'C' language.

Each file compiled must be less than 64K bytes in size. All functions that are over 325 lines in length are not optimized. Optimization can reduce the storage space and execution time required by eliminating unnecessary instructions and rearranging code. Several options in the 'C' compiler direct the compiler to generate more efficient code or smaller, but perhaps less efficient code. To use the optimization features of the compiler, all functions for the hierarchical algorithm are coded to contain less than 325 lines. Allowing each function to consist of only 325 lines or less, along with limiting each file size to 1300 lines, and limiting the number of files allowed to be opened at once, the algorithm implementation is adjusted to conform to these constraints.

The 'C' compiler has four memory model options: small, medium, large, and huge. Choice of a memory model is done after consideration of the amount of memory needed for the algorithm. Choosing the huge memory model allows for a 'C' program to contain arrays larger than 64K bytes. The amount of memory available and ability of the linker limit the size of arrays. A small memory model is used for algorithm implementation development, while the huge memory model is used for compilation of the final version. The huge memory model allows for the maximum of 200 pages of 30 logic symbols. Before compiling the program, the header file *hphead.h* must be adjusted to the amount of RAM available. Thus, the final implementation must be compiled using the /AH option of the compiler to invoke the huge memory model.

II. Test Case Analysis

A schematic of a digital circuit titled EZBD is used for testing. In developing the algorithm implementation, the choice of EZBD as an example proved to be invaluable. EZBD is provided by P-CAD to demonstrate all of the schematic capture capabilities. Thus many features of the schematic capture system are included in EZBD. Figures 10, 11, and 12 show different layers and levels of complexity available on the schematic EZBD. All three figures are contained in one schematic text file used as input for the hierarchical clustering algorithm. Several different schematics are used in testing, Figure 13 summarizes the results obtained for each test case.

Figure 10 has wire crossings, numerous inputs and outputs, several types of logic symbols and text. The wire crossings and small text can lead to confusion. Figures 14, 15, 16, 17, and 18 show EZBD partitioned into modules which make up the lowest level in the hierarchy.

Figure 19 shows the next level of the hierarchy. The next level uses rectangles to represent the modules in Figures 14 through 18. This level represents the entire system. Emphasis is on communication between modules instead of the symbol logic that makes up the modules.

The top level of the hierarchy is shown in Figure 20. The entire schematic is represented by a single rectangle. Figure 20 emphasizes the input and output of the entire system. Compared to Figure 10, Figure 20 shows an improvement in clarity.

Quality Measure

The QP factor for level one of EZBD is 97.53%. Level two has a QP factor of 77.88%. The average number of logic symbols per module in the preclustering step is 6.80. A grade for level one of 97.53% indicates EZBD is partitioned into modules of which are 2.47% away from having 7 blocks per module. The QP product for the entire EZBD schematic is 87.71%.

In the preclustering step, 34 logic symbols are used as input to the k-means algorithm. The addition of one logic symbol to EZBD to make 35 logic symbols does not give an overall QP factor of 100.00%. Generating additional logic to increase the overall QP factor has no substantial effect due to the second level having five blocks. The

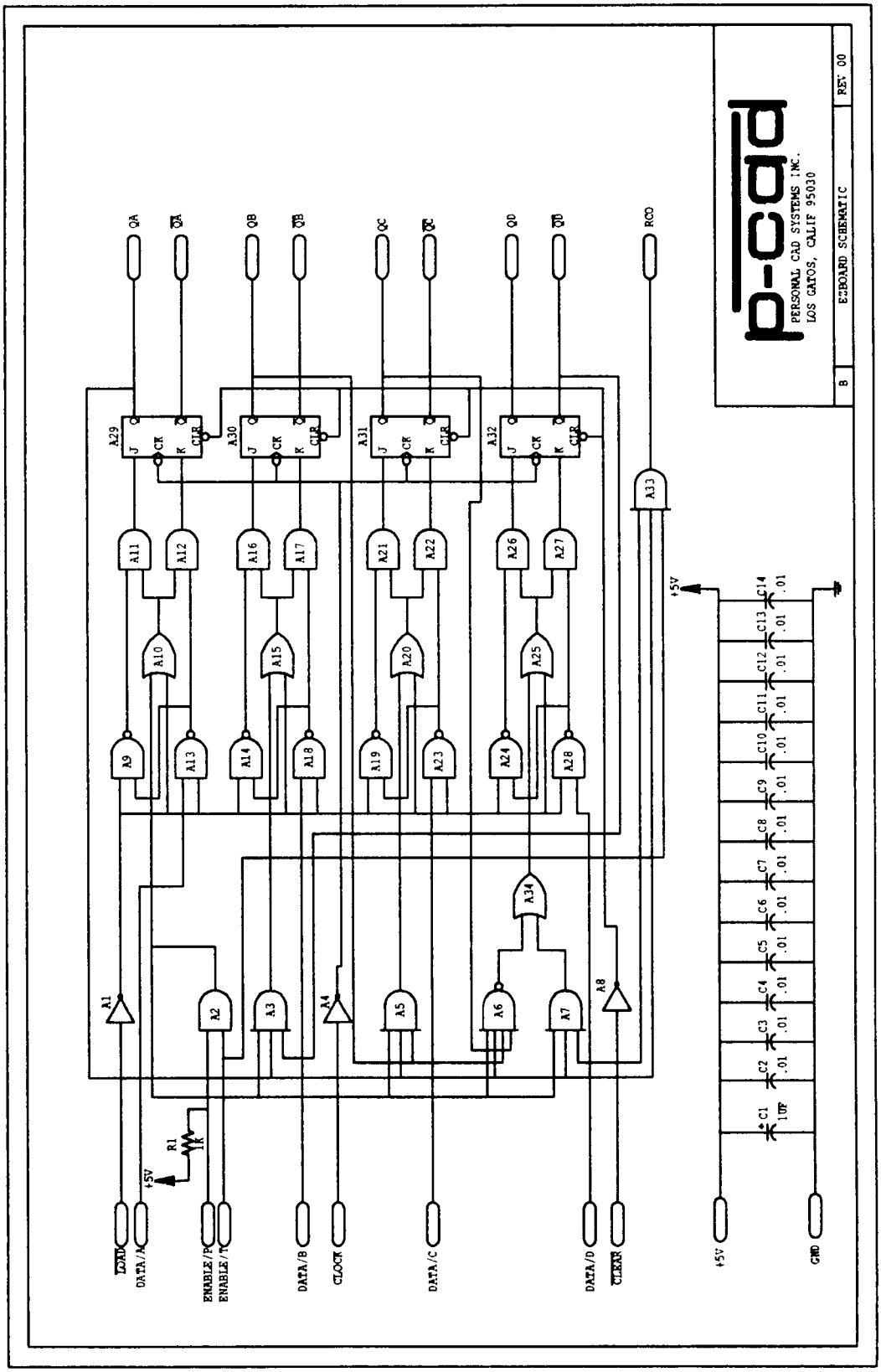

Figure 10. Basic schematic of EZBD.

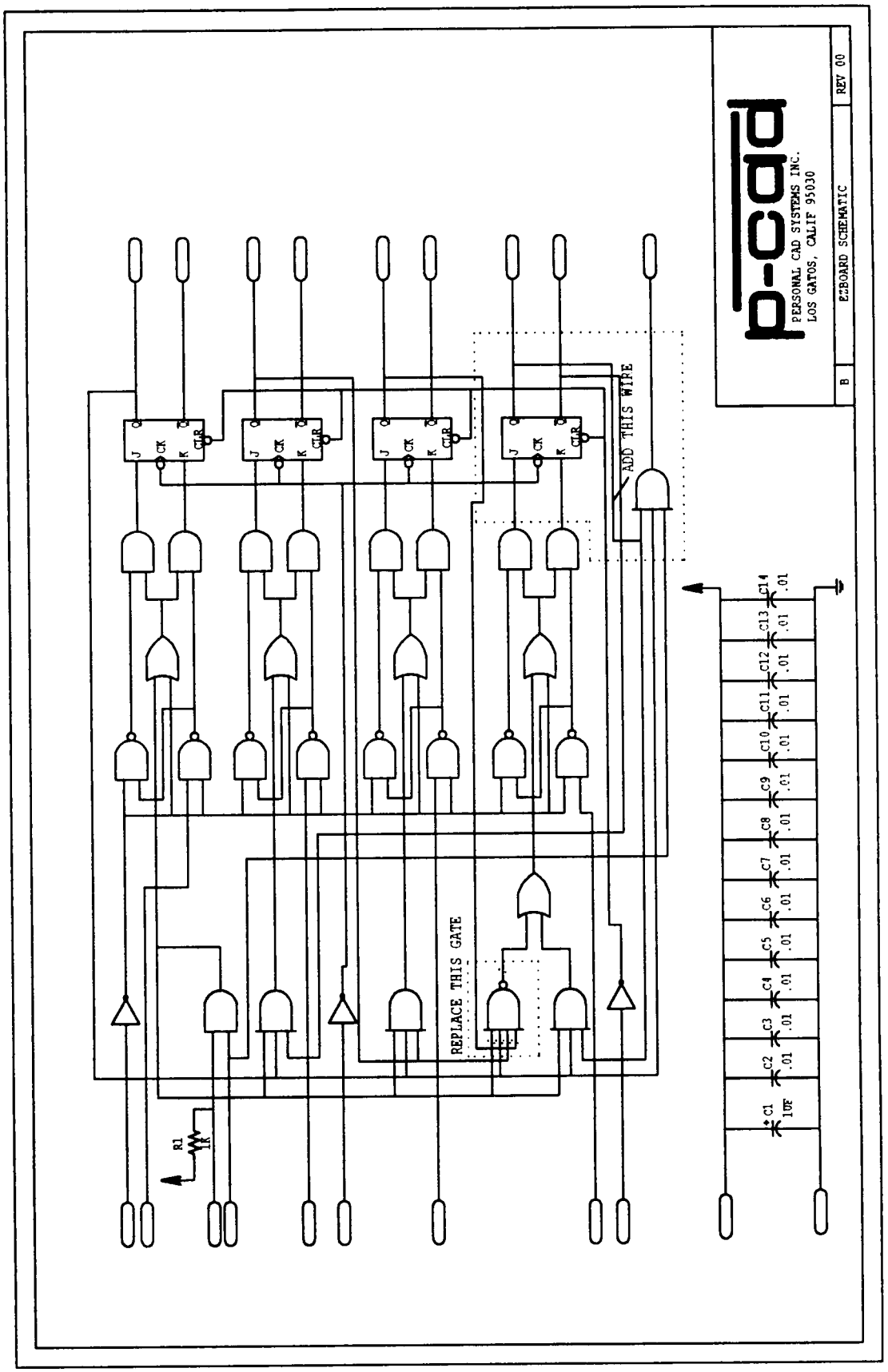

Figure 11. EZBD with layer help turned on.

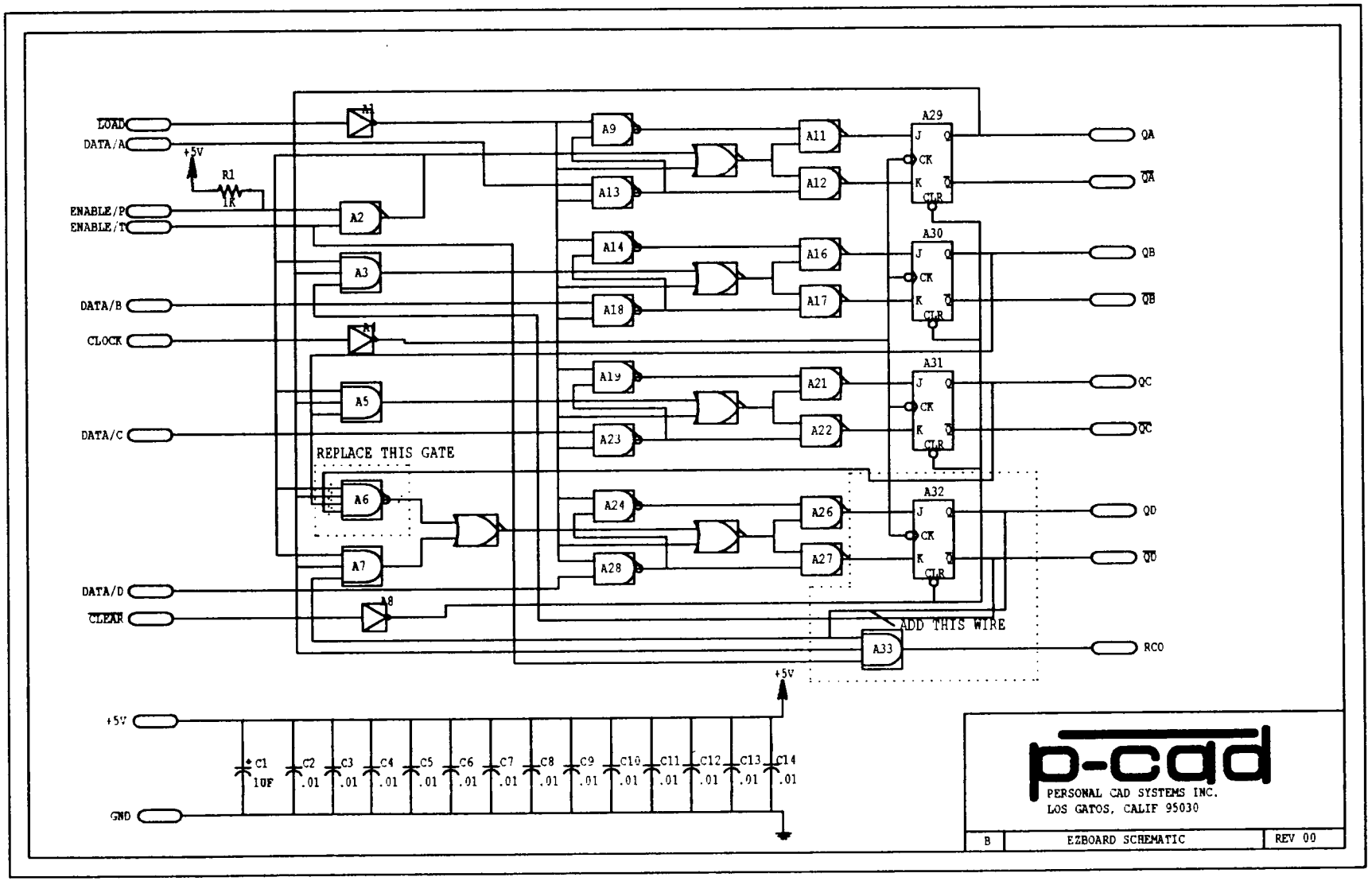

Figure 12. EZBD with all layers turned on.

	QP Factor per Level			QP Factor Overall	
Example	1	2	3	hierarchically	flat
BD1	98.76 %	68.73 %	47.24 %	71.57 %	0.04%
BD2	98.69 %	92.00 %	60.65 %	83.78 %	2.1×10^{-5} %
EZBD	97.53%	77.88%		87.71%	3.42%

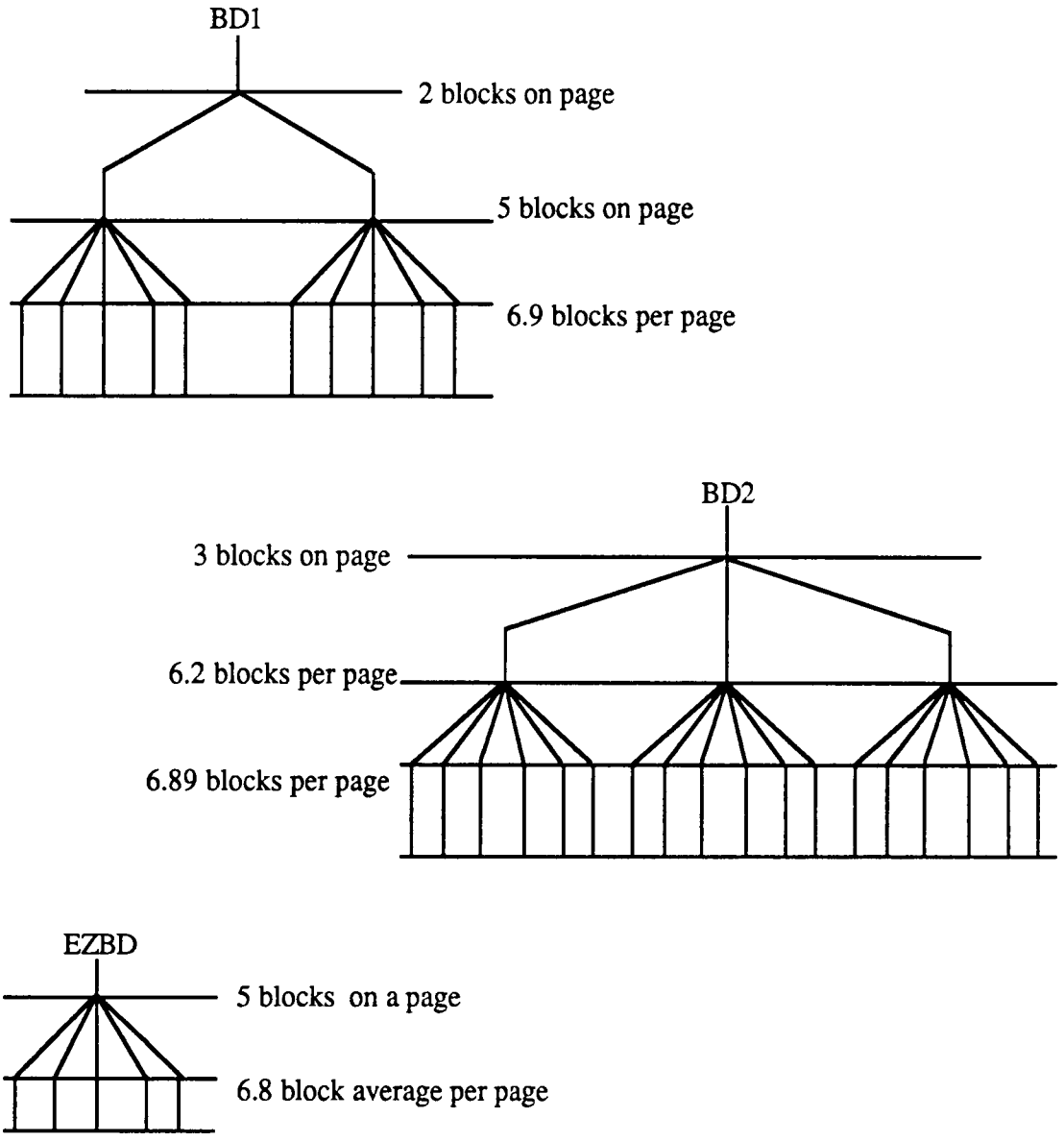

Figure 13. Test case summary.

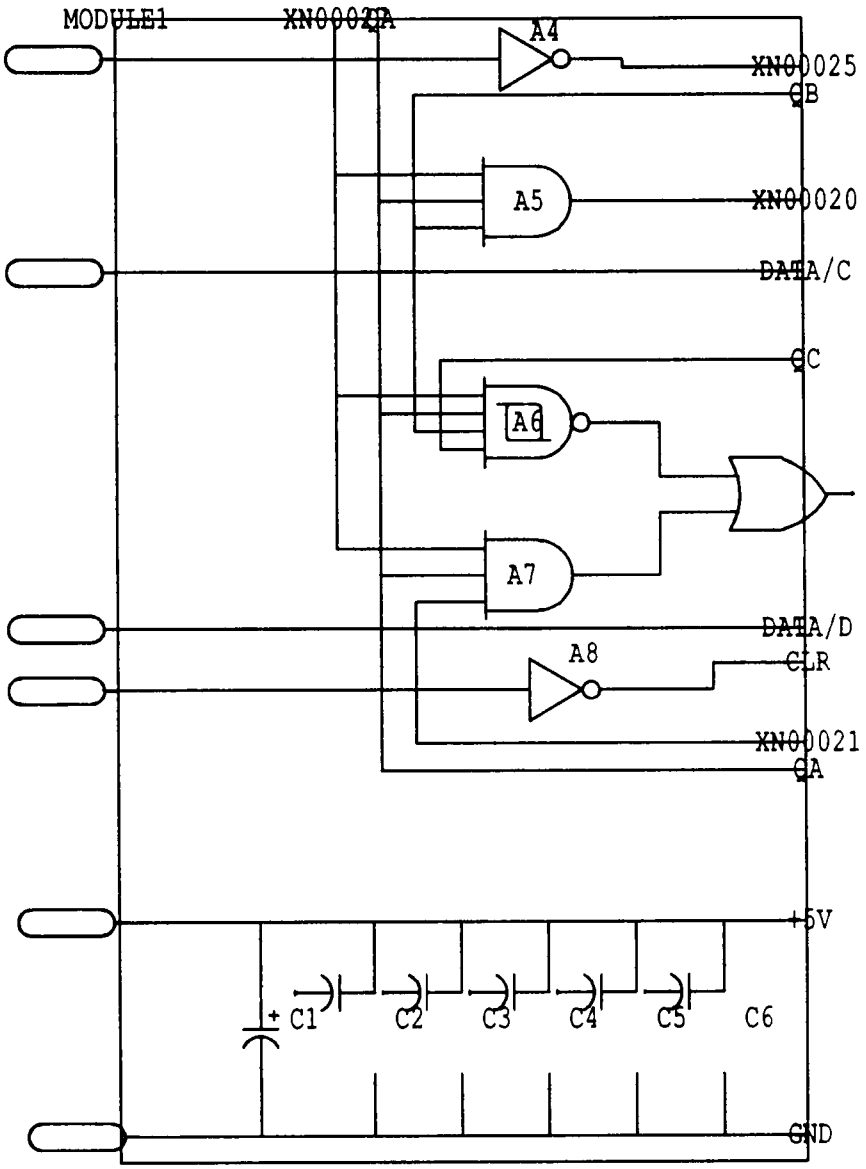

Figure 14. Module 1.

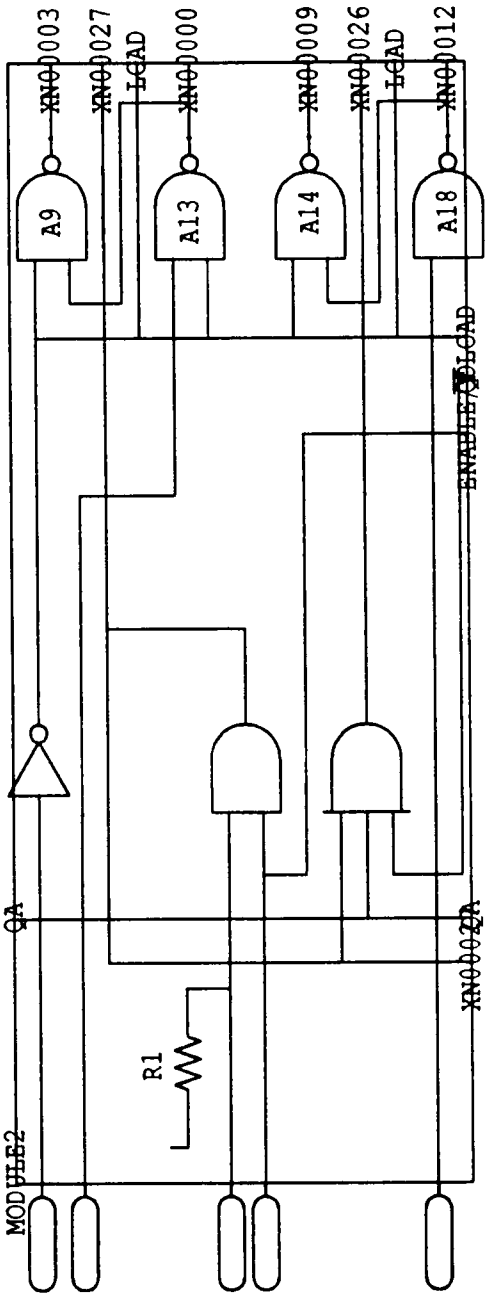

Figure 15. Module 2.

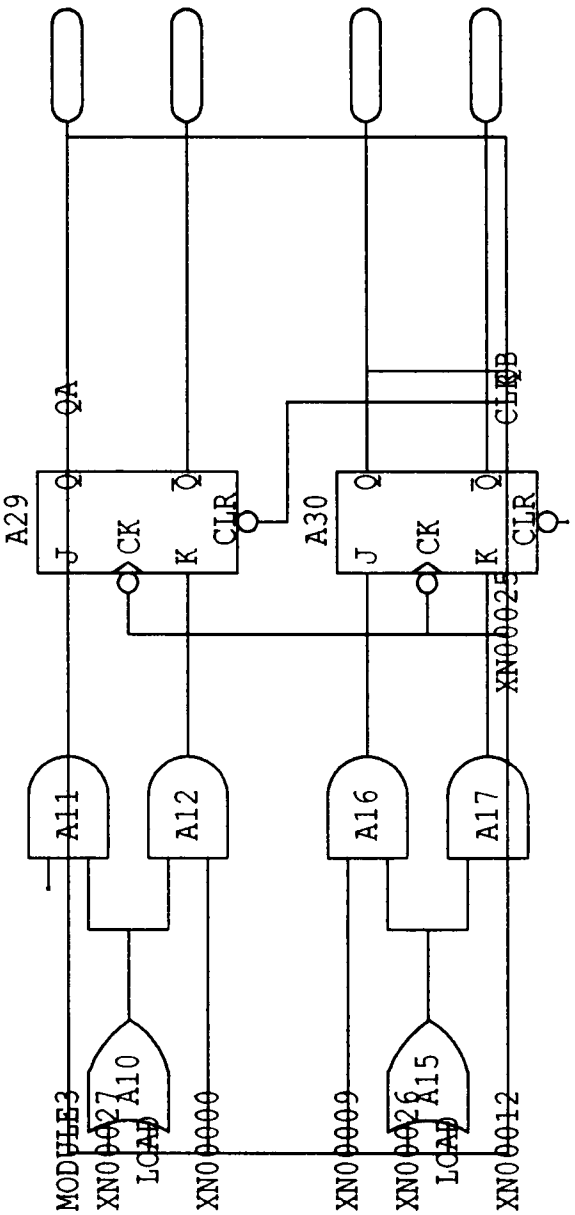

Figure 16. Module 3.

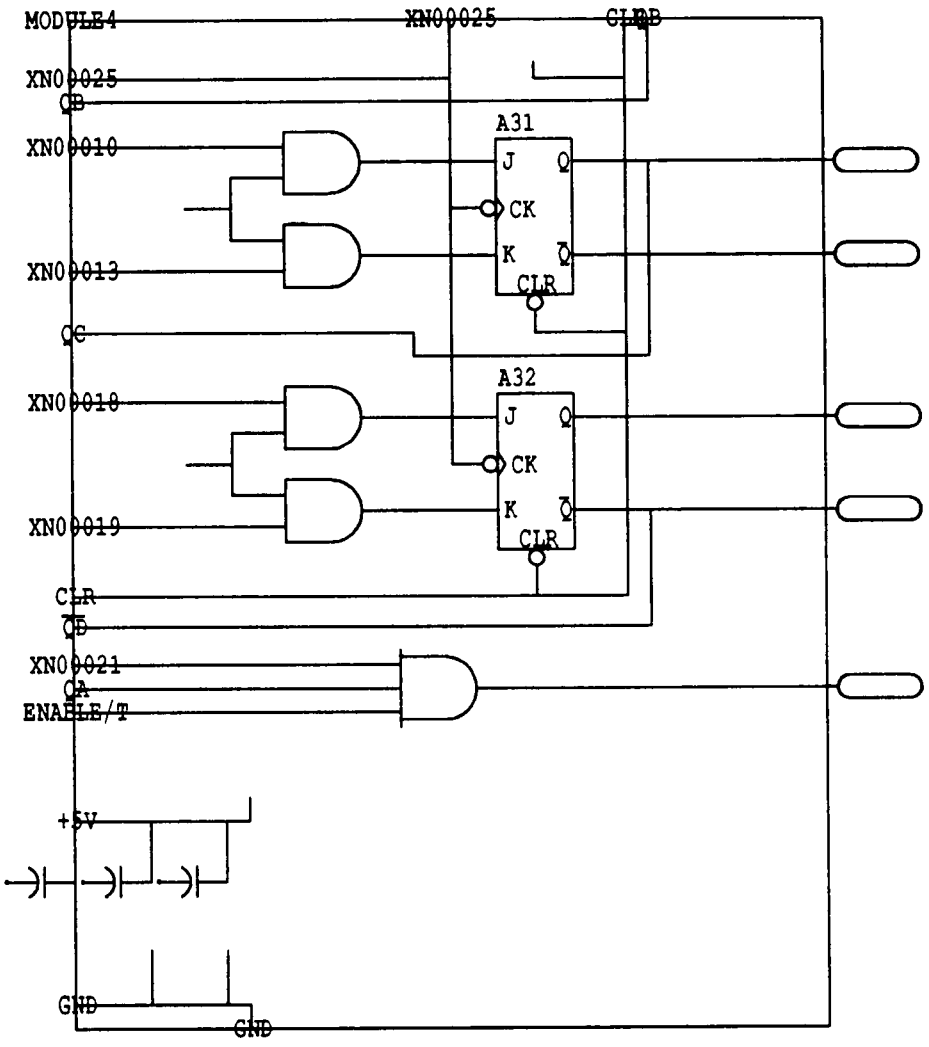

Figure 17. Module 4.

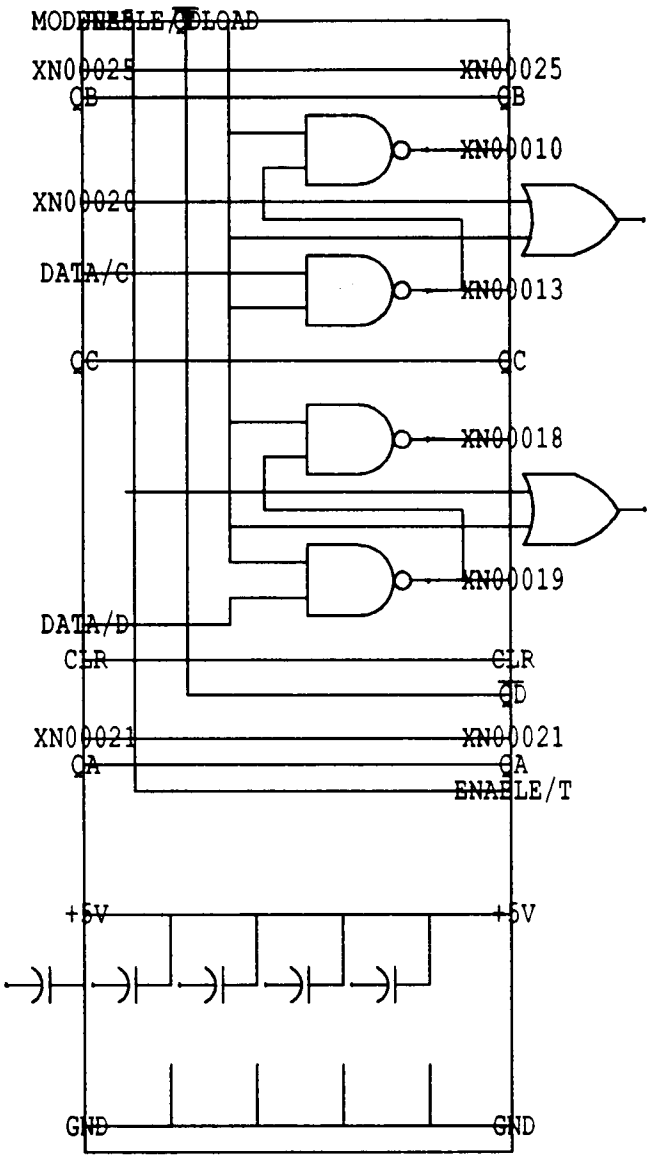

Figure 18. Module 5.

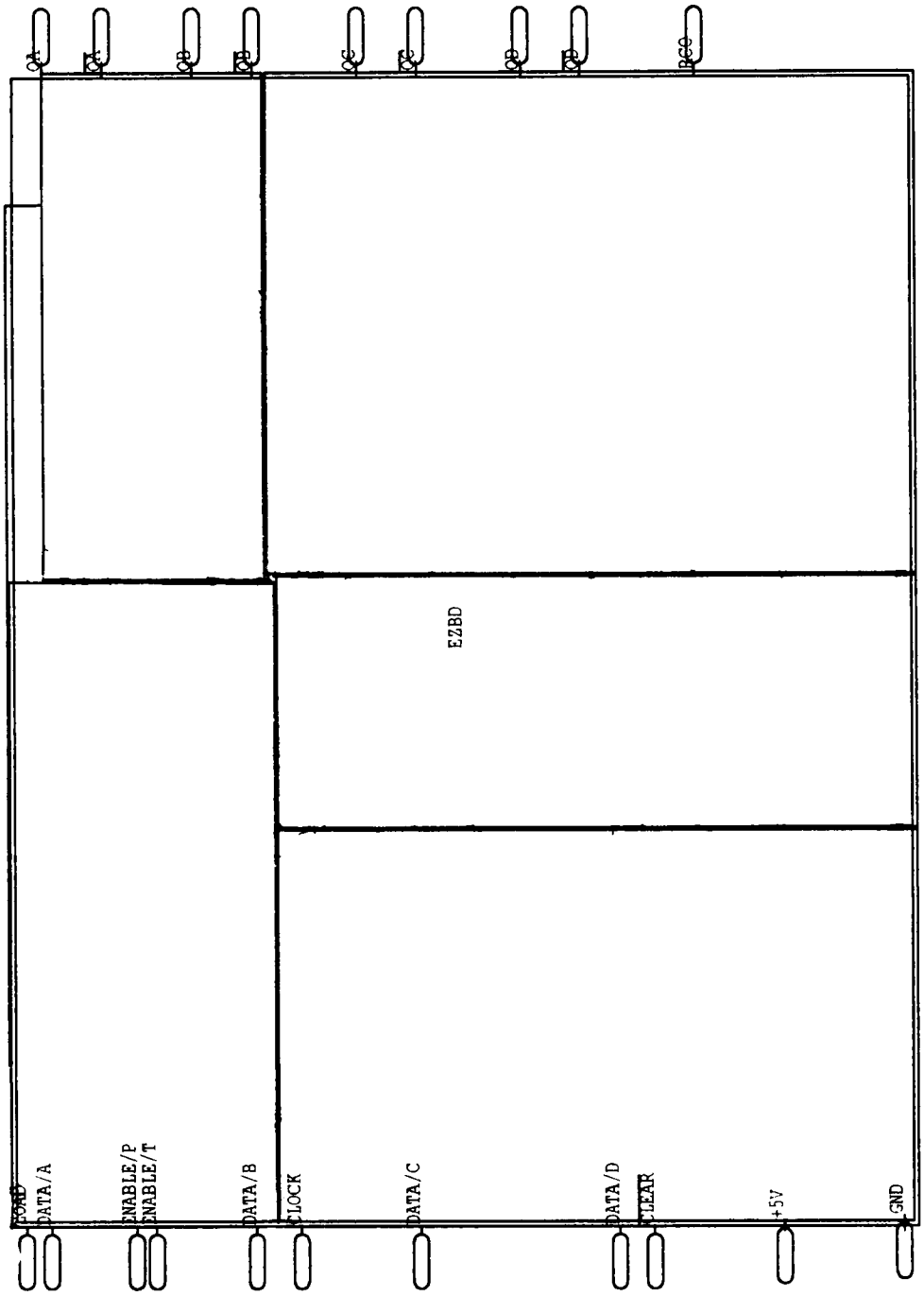

Figure 19. The second level in the hierarchy.

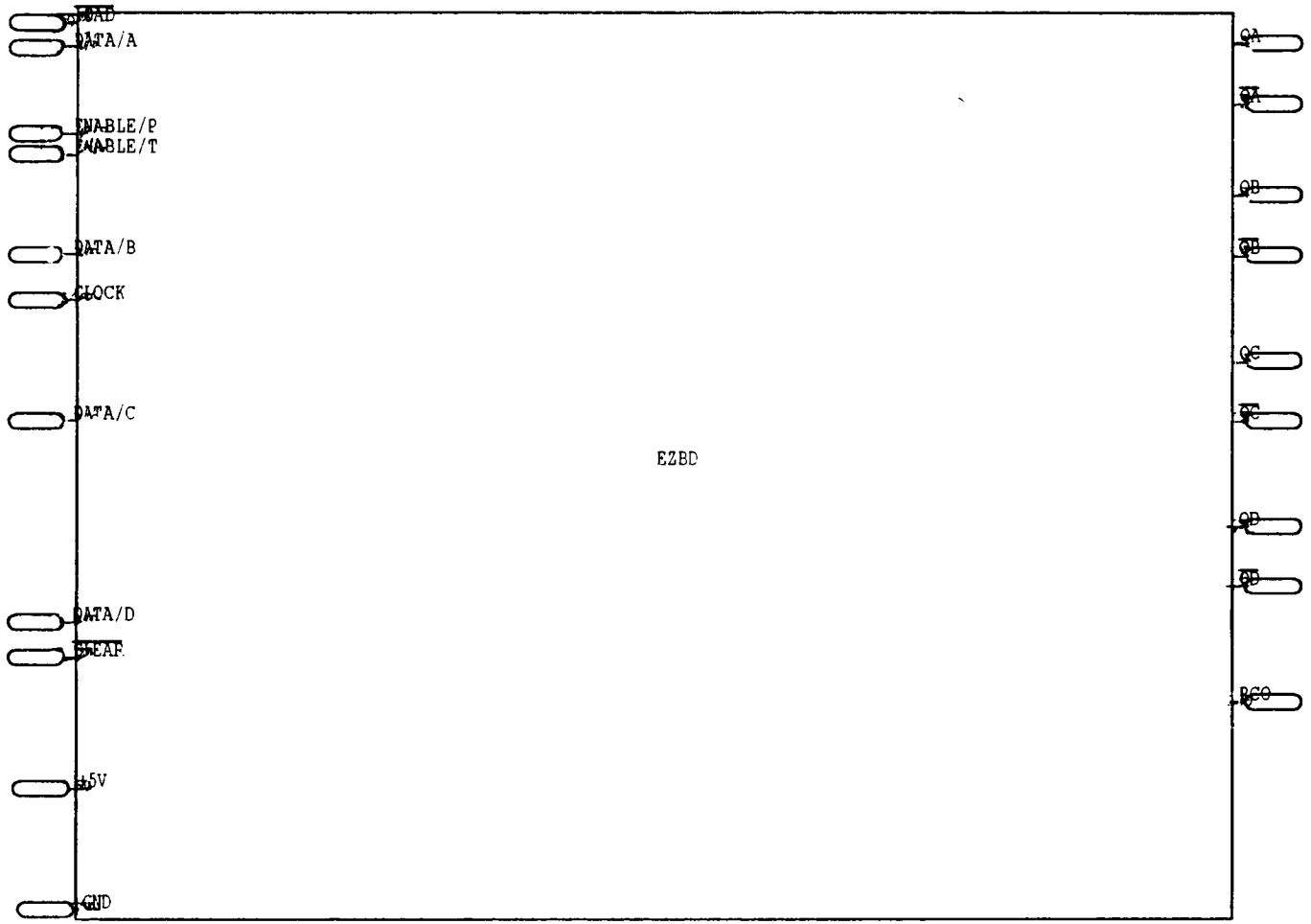

Figure 20. The top level in the hierarchy.

addition of 14 logic symbols is needed before the QP factor is 100.00%. Reducing the number of logic symbols by creating a hierarchy is the logical way to increase the overall QP factor.

Clustering Analysis

Clustering the logic symbols in EZBD tested the capability of the k-means algorithm. Figure 10 shows the logic symbols are aligned in vertical columns. The performance of the k-means algorithm has been shown to decrease [3] as the input prototypes become more aligned. Figures 14 through 18 show, along with the QP factor, the alignment of logic symbols in vertical or horizontal rows is not a degrading factor. Preclustering certain logic symbols and adjusting page boundaries before clustering makes logic symbol alignment not a degrading factor in algorithm performance. Figure 21 shows the results of preclustering and adjusting page boundaries for the schematic EZBD. The filled circles in Figure 21 represent the symbols selected for preclustering, the other components are not shown for clarity.

High Level Information

Payne's criticism [5] of other algorithms due to the lack of high level information not being utilized is not applicable to the hierarchical algorithm. High level information is contained in the schematic capture system text file in the form of previously defined components. Hence, the structure of the schematic capture system text file allows for the use of high level information by the hierarchical algorithm.

Two dimensional schematic representation using circles for the symbols used in the preclustering process for clarity.

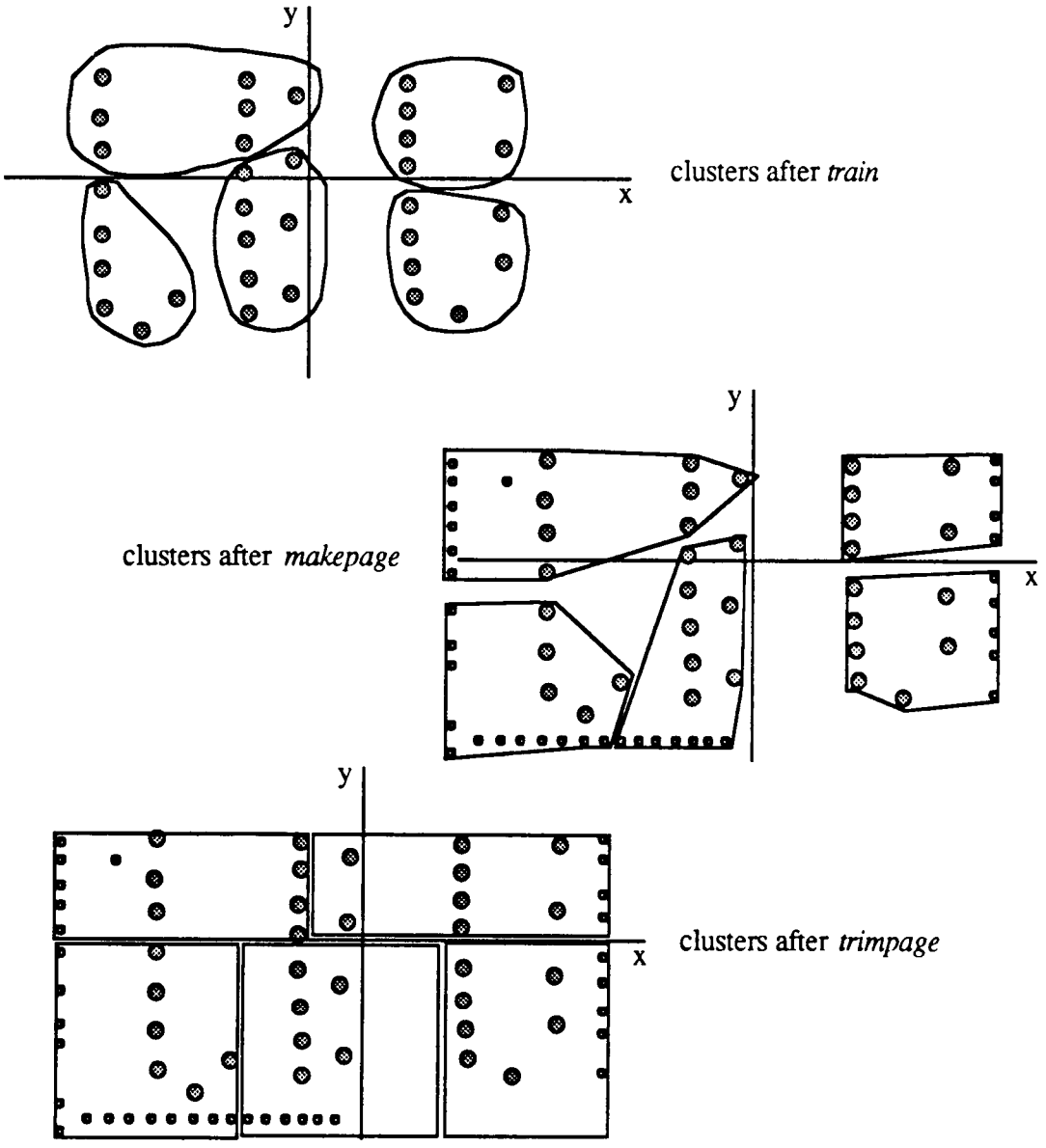

Figure 21. Effects of preclustering and adjusting page boundaries.

CHAPTER V

CONCLUSION

The results obtained by the hierarchical clustering algorithm, presented in this thesis, show the use of clustering in the partitioning process is a viable solution. Choosing a quality measure to grade the partitioning is an attempt to quantify the results obtained with clustering. The hierarchical algorithm can be used with the P-CAD schematic capture package.

The ability to comprehend a schematic by changing the flat schematic into a three dimensional representation is the approach used by the algorithm. Different aspects of the schematic are emphasized by partitioning the schematic into different levels of abstraction. The lowest level emphasizes logic composition, while the middle levels show module communication. The top level of the hierarchy highlights the system inputs and outputs.

I. Future Work

Suggested future work includes developing a quality measure based on human perception. This method involves setting up a statistical experiment with carefully chosen schematic examples. A survey panel grades each schematic sample on clarity and comprehensibility. The data from the survey is statistically analyzed. From this analysis, an equation representing human perception can be obtained. For this method to be valid, careful selection of schematic examples, a large survey population, and careful consideration of the experimental design are required.

Experimentation with other partitioning methods is also suggested for future work. The use of minimum spanning trees (MST), nearest neighbor criteria, farthest neighbor criteria, and graph theory can be explored. These methods can be used in the algorithm presented in this thesis by adjusting the preclustering and clustering functions.

II. Applications

Applications of the hierarchical clustering algorithm include preparing schematic documentation, aiding engineering changes, and preparing the schematic for presentation

purposes. All uses of a schematic capture package are also included as applications for the algorithm. As the complexity of digital circuit schematics increase, so will applications of the hierarchical algorithm.

REFERENCES

REFERENCES

[1] Chun, R.K., K.J. Chang, and L.P. McNamee, "Vision: VHDL Induced Schematic Imaging on Net-Lists," Proc. 24th Annual Design Automation Conference, pp. 436-442, 1987.

[2] Davies, D. L. and D. W. Bouldin, "A Cluster Separation Measure," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. PAMI-1, pp. 224-227, April 1979.

[3] Duda, R. D. and P. E. Hart, Pattern Classification and Scene Analysis, New York: Wiley, 1973.

[4] Miller, G.A., "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," The Psychological Review, vol. 63, pp. 81-97, March 1956.

[5] Payne, T.S., "Automated Partitioning of Hierarchical Specified Digital Systems," Technical Report No. 215 , Stanford, CA: Stanford University, 1981.

[6] Payne, T. S. and W. M. vanCleemput, "Automated Partitioning of Hierarchically Specified Digital Systems," Proc. 19th Annual Design Automation Conference, pp. 182-192, 1982.

[7] Personal CAD Systems Incorporated, PDIF Users Manual, San Jose, CA, Personal CAD Systems Incorporated, February 1986.

[8] Preas, B. T. and P. G. Karger, "Automatic Placement, A Review of Current Techniques," Proc. 23rd Annual Design Automation Conference, pp. 622-629, 1986.

[9] Schuler, D. M. and E. G. Ulrich, "Clustering and Linear Placement," Proc. 9th Annual Design Automation Workshop, pp. 50-56, 1972.

[10] Sha, Lu and R. W. Dutton, "An Analytical Algorithm for Placement of Arbitrarily Sized Rectangular Blocks," Proc. 22nd Annual Design Automation Conference, pp. 602-608, 1985.

[11] Tou, J. T. and R. C. Gonzalez, Pattern Recognition Principles, Reading, MA: Addison-Wesley, 1974.

[12] Venkataraman, V. V. and C. D. Wilcox, "Gems: An Automatic Layout Tool for MIMOLA Schematics," Proc. 23rd Annual Design Automation Conference, pp. 131-137, 1986.

VITA

James Eugene Hilger was born in Wilmington, Delaware, on April 19, 1962. He attended Hurlock Elementary and North Dorchester High School, both on the Eastern Shore of Maryland. In 1980, he entered Delaware Technical and Community College in Georgetown, Delaware and graduated in 1982 with an Associate degree in Electronics. In 1982, he entered Salisbury State College in Salisbury, Maryland and graduated in 1985 with a Bachelor of Science degree in Physics. In August 1985, he entered graduate school at the University of Florida in Gainesville, Florida. He transferred to the University of Tennessee, Knoxville, in January 1986 to complete a Master of Science degree in Electrical Engineering.

During his graduate school experience, the author worked as a guest consultant at Oak Ridge National Laboratories, Oak Ridge, Tennessee and as a teaching assistant at the University of Tennessee. The author is a member of Who's Who Among American High School Students and a recipient of the Excellence in Physics Award from Salisbury State College.