Proc. 15th Design Automation Conference, June 19-21, 1978.

LORES - LOGIC REORGANIZATION SYSTEM

Shunichiro Nakamura, Shinichi Murai and Chiyoji Tanaka
Computer Laboratory, Mitsubishi Electric Corp.
325 Kamimachiya, Kamakura, Japan
Masayuki Terai, Hideo Fujiwara and Kozo Kinoshita
Electronics Engineering Dept., Osaka University
Yamadaue, Osaka, Japan

Abstract

Described is the outline and the experimental results of the system which automatically restructures and partitions a logic circuit consisting of standard SSI's and MSI's so that the gate types and the numbers of input/output terminals of the reorganized circuits are within the restrictions of the specified LSI.

1. Introduction

The utilization of custom LSI's in computers and the related digital systems is becoming widely accepted with the development of semiconductor device technology. The adoption of custom LSI's would give us such advantages as smaller equipment size, better performance, less power consumption, better reliability, and cost reduction possibility due to highly automated batch fabrication process instead of cumbersome assembly procedures. However, custom LSI approach has such serious disadvantages as longer development period, higher development cost, and much more difficult debugging.

In order to overcome these disadvantages and fully utilize above described advantages, it is vital to prepare the CAD system which assists various design activities in the most suitable and efficient way. LORES (LOgic REorganization System) is an experimental CAD system which effectively attacks the problems of logic debugging and development period reduction in the following circumstances.

We quite often encounter the situations where we want to develop a new digital equipment whose function and therefore the logic is basically the same as the field verified existing equipment, but with higher speed and cheaper cost. This is especially the case where the existing equipments use standard SSI's and MSI's, and we want to develop new equipments utilizing custom LSI's.

Now, even though the function of a new equipment is exactly the same as of the existing one, it is usually not possible to implement on LSI's exactly the same logic as the equipment utilizing standard SSI's and MSI's, because the clock inplementation on the selected type of LSI's may not be the same as the original equipment, and the types of elementary logic gates in LSI's are usually restricted according to their device technology especially in case of gate arrays or master-slice type LSI's. Therefore, it is necessary to reorganize the existing logic circuits so that they can be implemented on selected type of LSI's even though the function of the new equipment is exactly the same as the existing equipment.

The objective of the LORES is to automate this type of logic restructuring or conversion and partitioning of the circuit so that the resultant logic circuits fit into individual custom LSI's in the case that the similar clocking system can be implemented on the target LSI's and the circuit does not include any asynchronous portion, thus eliminating the possibility of logic errors, eliminating the necessity of logic simulation, and reducing design period. LORES will be useful even if the circuit includes some asynchronous portion, although a certain amount of logic simulation may be necessary in this case. LORES can also be used in the situation that the exact breadboarding of custom LSI circuits is difficult and the conversion from the breadboard logic to LSI cell logic is necessary.

The functions and the organization of LORES is briefly described in the next section, followed by the description of working storage of the system. Then more precise descriptions of each part of the system are presented, and finally the experimental results are reported.

2. The functions and the organization of LORES

The functions of the LORES are summarized as follows;

1) Extraction of the logic circuit to be processed
   Extract a portion of the logic circuit from the data base of the original equipment or breadboard.

2) Elimination of the unused logic
   Eliminate the unused portion of the extracted circuit (when the circuit is built with standard MSI's and SSI's, all the functions of the devices are not always used, for example, the direct set terminal and the related gates of flip-flops).

3) Logic conversion
   Convert or restructure the logic circuit consisting of standard SSI's and MSI's to the logic circuit with the same function consisting only of logic elements allowed in LSI's.

4) Logic partitioning
   Partition the logic circuit so that the numbers of gates and input-output terminals of each partitioned circuit are within the specified limit.

Fig.1 shows the total CAD system for custom LSI's. Logic simulation is necessary if the circuit includes some asynchronous portion, since the timing consideration in LORES is not sufficient. The organization of LORES is shown in Fig.2. The execution order of logic conversion and partitioning is manually specifiable. The design data base and LSI data base are the integrated engineering data bases based on EDMS.1),2) The manual specification data for LORES are:

1) The execution order of the logic conversion and the logic partitioning.

2) The specification of the portion of the logic circuit to be processed.

   Specified by PCA (printed circuit assembly) names and/or logic element and signal names in PCA of the original equipment.

3) Control data for logic partitioning

   a) Whether or not to partition the logic circuit in a MSI.

   b) Groups of logic elements which are prohibited to be partitioned.

   c) The signal names which should be input/output terminals of the partitioned circuit.

   d) The signal paths which should have the same amount of propagation delay.*

4) Control data for logic conversion

   a) The number of fan-outs of logic elements in the converted circuit.

   b) The number of signals which can be wired-ANDed.

   c) The number of inputs of the elementary logic gates in the converted circuit.

   d) The logic element types which cannot be converted.

* Delay is measured in terms of the number of gates in the paths. Power or wiring length is not considered.

3. Working Storage

Although it is desirable, in general, to set up working storage within main memory area, the working storage of LORES is set up in a random file on disk, in order to process big size problems and to reduce the program complexity required to effectively handle limited amount of main memory. MELCOM UTS/VS EDMS (Extended Data Management System) is used for the maintenance of random file, and it's 20 KB buffer in main memory makes the access time in random file as fast as in main memory as far as the neighboring portion of the circuit is accessed.

Three types of logical connection files are prepared for the three programs of LORES as shown in Fig.3.

The schema of these three types of files are exactly the same, thus enabling to change the execution order of the logic conversion and partitioning (eg. preprocess - conversion, preprocess - conversion - partitioning, preprocess - partitioning - conversion).

4. The Preprocess Program

The preprocess program has two principal functions: one of them is to extract the logic circuit to be processed, and the other is to eliminate the unused portion of the extracted circuit.

(1) Extraction of the logic circuit to be processed

The specific portion of the logic circuit is defined by three types of control cards:

1 Definition by a set of PCA files,

2 Definition by a set of signal lines to be cut off,

3 Definition by a set of logic elements to be excluded (element types or element names).

The preprocess program reads these control cards, retrieves the specified logic area in the PCA files, and forms the corresponding object PCA file. In the course of this extraction procedure, the macro logic expansion are executed for the MSI macros and the repeatedly used PCA's, since the internal logic structures of these macros are stored only once in the data base. Furthermore since the element and signal names are uniquely defined only in PCA file, their uniqueness must be preserved after several number of PCA files are integrated.

(2) Elimination of the unused portion of MSI logic

Since MSIs are general-purpose devices their entire functions are not always utilized in actual logical circuits. In some cases, for instance, only an output, '=', among outputs, '<', '>', and '=', of a comparator may be required, or not all of input/output bits of an MSI encoder may be used. In another case the input ENABLE may be fixed to 1 or 0, so that the circuit is enabled all the time. The preprocess program eliminates such unused portion of MSI logic (Fig.4). The elimination is as follows.

(i) Elimination starting from the external output terminals
The logic elements whose outputs are connected only to the unused external output terminals are eliminated.

(ii) Elimination starting from input terminals
A logically redundant portion caused by some input terminals fixed to "1" or "0" are eliminated.

Algorithm (1)

(a) Trace backward one of the signal lines which are connected to an unused external output terminal.

(b) Return to the preceding node at the point where the line branches.

(c) If the line reaches without branching a gate, which would be regarded as a node, then memorize all the names of the signal lines connected to the node, and restart tracing backward one of them.

(d) In case of (b), trace backward in the same manner the other untraced lines if any. If not, go back to the preceding node and repeat the same operations.

(e) If the process goes back to the starting output terminal, it is over. Then, eliminate the traced logic elements and signal lines from the MSI logic.

Fig. 5 shows an example.

Algorithm (ii)

(a) Trace forward one of the input signal lines fixed to 1 or 0, keeping its logical value.

(b) If a branching point is encountered, which would be a node, then memorize all the succeeding logic elements and trace one of them forward in the same way. Following four cases show the operations when the signal line meets the gates (Fig.6).

1 CASE 1

In case of meeting an inverter it is eliminated, the logical value of the signal line is inverted and the forward tracing goes on

2 CASE 2

In CASE 2 the terminal is eliminated and the tracing goes back.

3 CASE 3

In CASE 3 the NAND gate is eliminated and the forward tracing goes on with the output logical value inverted. At the same time the backward elimination from another input terminal is started by the same algorithm as (i).

4 CASE 4

In case of an exclusive OR gate, as shown in CASE 4, it is converted into an inverter and the tracing goes back. Although CASES 2 and 3 shows the cases of a NAND gate, the same procedure can be applied at NOR gates if the signal values are inverted.

(c) The process is over when the tracing reaches the starting input terminal as in (i).
Tracing of the graph will be easily performed by making use of stacks.

5. Logic Conversion

As previously described, the logic conversion is defined as to convert a logic circuit composed of various types of logic elements such as AND, OR, NAND, NOR, etc. to a logic circuit with the same function but composed only of limited types of logic elements such as NOR gates.

The types of logic elements allowed by LORES in a source circuit are elementary logic gates (AND, NAND, OR, NOR, EOR, ENOR, Wired-AND and Wired-OR gates), J-K flip-flops, D flip-flops and latches. A target circuit can be composed only of k-input NOR gates, 1-input wired-AND gates, J-K flip-flops and D flip-flops.

The outline of the logic conversion procedure is shown in Fig.7. The conversion is first carried out element by element, and then the elimination of the redundant inverters is caried out in a whole circuit.

(1) Decomposition of multi-input gates

Since the converted circuit consists only of K-input NOR gates and 1-input wired-AND gates, the decomposition of the gates having more than k inputs is first executed. As shown in Fig.8 wired-AND gates are used for the decomposition if it's not inappropriate.

(2) Conversion to NOR and AND gates

All the elementary logic gates other than NOR and AND gates are converted to the logically equivalent elementary circuit consisting only of NOR and AND gates (Fig.9). The elimination of the redundant inverters which are generated during the conversion is executed at the same time. The inverter elimination rules are shown in Fig.9.

(3) Conversion to wired-AND gates

The AND gates in the circuit are converted to wired-AND gates if it's possible. The AND gates which cannot be converted to wired-AND gates are converted to inverters and NOR gates (Fig.10). AND gates can be converted to wired-AND gates if and only if

i) no input signals of the AND gates are fed to other gates,

ii) no input signals of the AND gates are the signals external to the target circuits, and

iii) the numbers of inputs of the AND gates are less than or equal to 1.

The redundant inverters generated during this conversion are eliminated at the same time.

(4) Expansion of NOR gates

The conversion shown in Fig.11 is executed by examining the front and the rear of all the inverters in the circuit.

(5) Elimination of inverters in quasi-signals

The elimination of the inverters in quasi-signals is tried. Quasi-signal is a set of signals which would become a same signal, signal (net) if all the inverters in the circuit were eliminated. An example is shown in Fig. 12.

The names of the new elements and signals which were generated during the above conversion procedures are the modified ones of the original elements and signals so that the correspondence of the new elements and signals to the original elements and signals may be easily known. The modification is such that it would show the relationship between the new and the original signal (eg. the new signal is the inverted one of the original).

6. Logic Partitioning

The logic partitioning program accepts the restructured PCA file which has been processed by a set of logic conversion programs and partitions the PCA logic to LSI logics under various restrictions. In general, some objective functions are defined so as to minimize or maximize them in the algorithm of partitioning problem. (3)-(6). For example, given upper limits of the number of external pins or elements included in partitioned circuits, the optimal solution is computed under the following objective functions.

1) Minimize the total number of connections among partitioned sublogic circuits.

2) Minimize the number of partitioned sublogic circuits.

3) Maximize testability and diagnosability.

4) Satisfy a given condition of propagation delay time.

However, the practical algorithm which optimizes above functions for large logic circuits are not founded, therefore, heuristic approaches are used at present so as to locally optimize or suboptimize the functions.

We introduced a practical partitioning algorithm satisfying conditions (2), (3) and (4) whose computing complexity is $0(n)$ or at most $0(n^2)$ where n is the size of the circuit to be partitioned.

The flow chart of this algorithm is shown in figure 13.

The partioning is done as follows.

1) Select elements starting from the externals of a target logic circuit until the limit of the number of elements or terminals are reached.

2) Stop the selection and form a LSI when the limitation is reached.

3) Eliminate the logic corresponding to the newly formed LSI and let the interface signals between the new LSI and the remaining logic be external terminals and return to step 1.
In these steps, the algorithm does not select the elements again which have once been selected so that the computing complexity is $0(n)$ or $0(n^2)$.

In order to satisfy the condition (2) the algorithm selects the locally optimum elements causing the minimum number of terminal increase.

In order to meet the condition (3), the user can specify the test point by control card so as to increase the testability and the resolution of diagnosis.

In order to satisfy the condition (4), the user can specify the equal propagation delay for a certain logic by control cards so that the delay times of the specified paths of the logic are equal after the partitioning.

The following modes are implemented in the program.

(a) Start partitioning eather from an external input terminal or from an external output terminal.

(b) In case of selecting the next element, whether put the priority on the element connected at the input terminal or at the output terminal or none of these (see Fig.14).

(c) In case that the number of the next element to be selected is plural, either select a local optimal element minimizing the increase of interface terminals or select a first encountered element which is connected to the LSI.

The reason why we define mode (a) and (b) is that we can partly control the configuration of a LSI logic such that for example, if we choose the mode of input terminal connected element from input interface the configuration of the LSI is in horizontally long logical chain as shown in figure 15 and if we choose the mode of output terminal connected element from input interface, it is vertically long as shown in figure 16. The numbers in the figures shows the order of selecting elements.

As the connection of elements in a circuit is generally more complicated, this results can not always be expected. However, the user selects either mode and can form the best LSI partitioning from several experiments.

The selection of mode (C) affects the computing time of the program. The locally optimal algorithm is of $0(n^2)$ and the other is of $0(n)$.

The selection of the next element or element group in the partitioning algorithm of Fig.13 should be determined based on the above three modes and the following consideration: Put priority on non partitioning element group.

The user can specify non partition to a set of elements which should belong to a certain LSI. In this case, if these elements are handled in the same manner as the other elements in the locally optimum mode, the computing time is lost, therefore, this non partitioned element group is put into LSI prior to the other elements.

7. Results

The program size is about 16,000 cards in FORTRAN. The execution results of logic conversion is shown in Table 1. The inverter reduction in quasi signal is not executed in all cases in the table. If this reduction is executed, the number of cells of the target circuits would be further reduced.

The major limitation of LORES is that it basically lacks the timing consideration even though it does have the feature of equivalent delay specification upon the arbitrarily selected paths. Therefore, if the circuit includes some asynchronous portion, eather of the following manual assistence is necessary:

1) Check the result of LORES whether or not the timing relations between the signals in the asynchronous portion are preserved, and manually modify the result if necessary.

2) Manually convert the asynchronous portion before the execution of LORES.

In eather case, logic simulation would not be avoidable in order to check the result of manual modification or automatic conversion by LORES, unless the asynchronous portion is small and trivial.

8. Conclusions

LORES is useful for the reorganization of the combinational or synchronous sequential circuits whose clocking system is basically the same as the target LSI's. If the source circuit includes some asynchronous portion, logic simulation and a certain amount of manual modification may be necessary.

Acknowledgements

The authors would like to thank Prof. Ozaki and Mr. Matsumoto for their support and encouragement of this joint work. They would also like to thank the referees of this paper for their valuable comments.

References

1) C. Tanaka et al., "Engineering Data Management System (EDMS) for Computer Aided Design of Digital Computers", Proc. 11th Design Automation Work-shop pp 372-379, June 1974.
2) C. Tanaka et al., "The Application of Data Base for CAD of Digital Computers", Proc. 2nd USA-Japan Computer Conference pp 567-572, August 1975.
3) M. A. Breuer, "Recent Developments in the Automated Design and Analysis of Digital Systems", Proc. IEEE vol.60, No.1, pp 12-27, Jan. 1972.
4) R. L. Russo, "A Computer-Based-Design Approach to Partitioning and Mapping of Computer Logic Graphs", Proc. IEEE vol.60, No.1, pp 28-34, Jan. 1972.
5) S. B. Akers, "Partitioning for Testability", Proc. FTCS-6, pp 121-126, June 1976.
6) A. Goundan, J. P. Hayes, "Partitioning Logic Circuits to Maximize Fault Resolution", Proc. 13th Design Automation Conference, pp 271-277, June 1976.

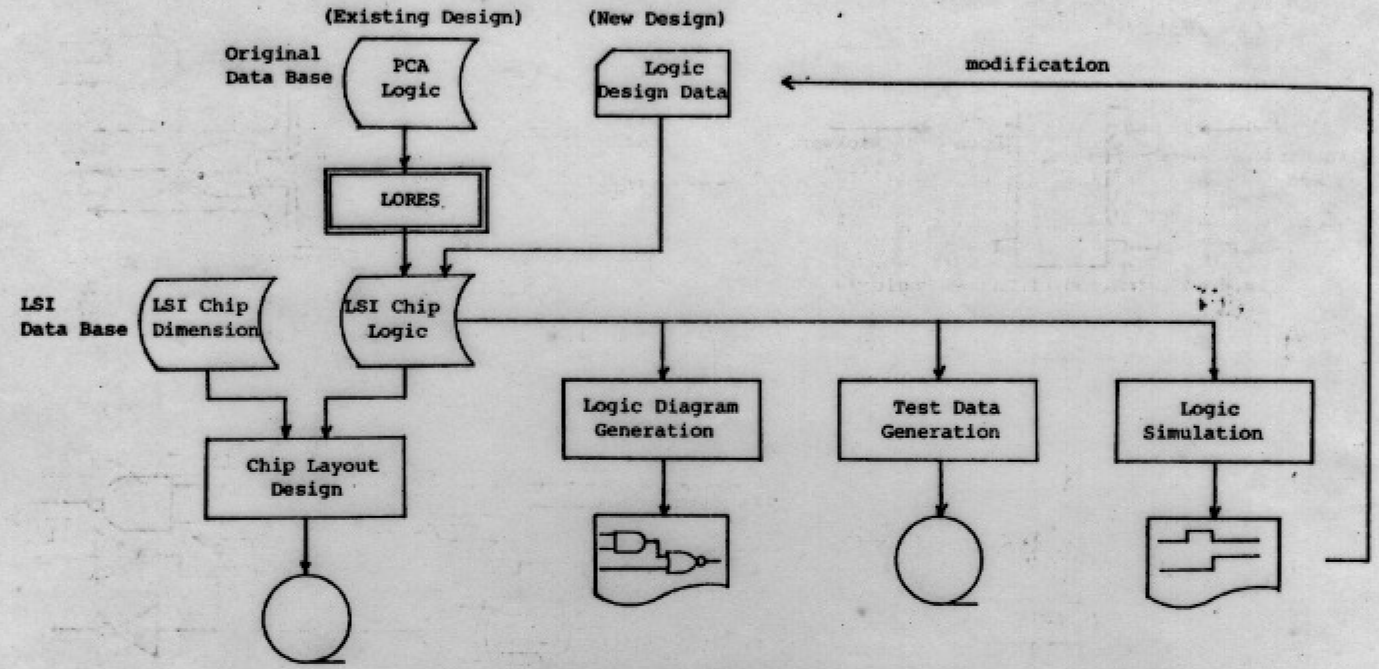

Fig. 1 LORES and the Related CAD Systems for Custom LSI's

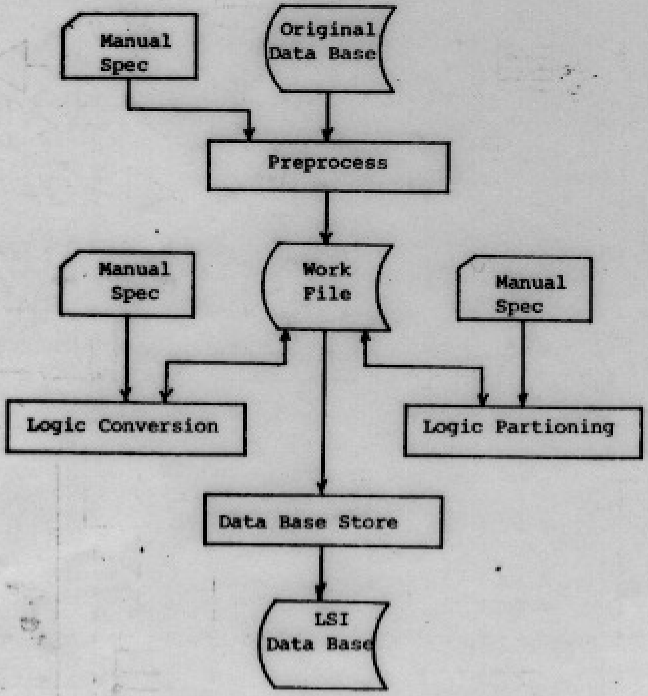

Fig. 2 The Organization of LORES

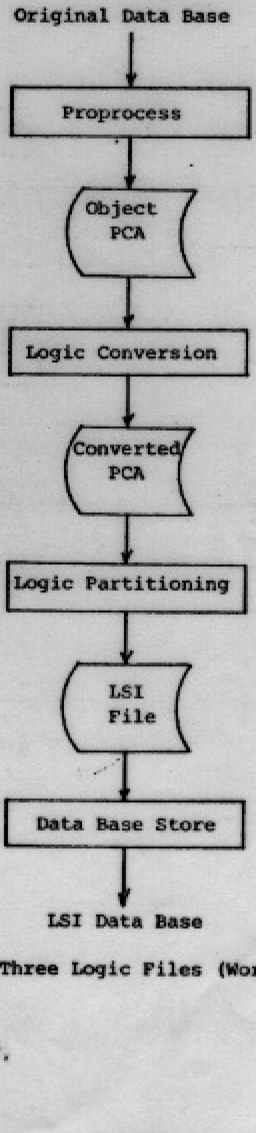

Fig. 3 Three Logic Files (Work File)

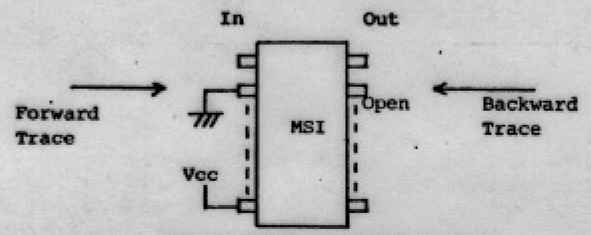

Fig.4 Elimination of Unused Logic

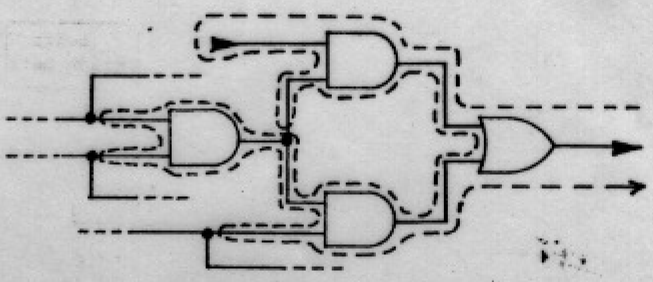

Fig. 5 Output Terminal Elimination

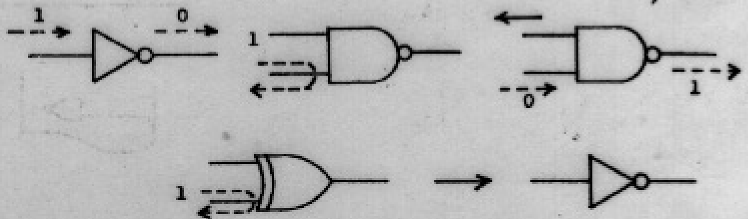

Fig. 6 Input Terminal Elimination

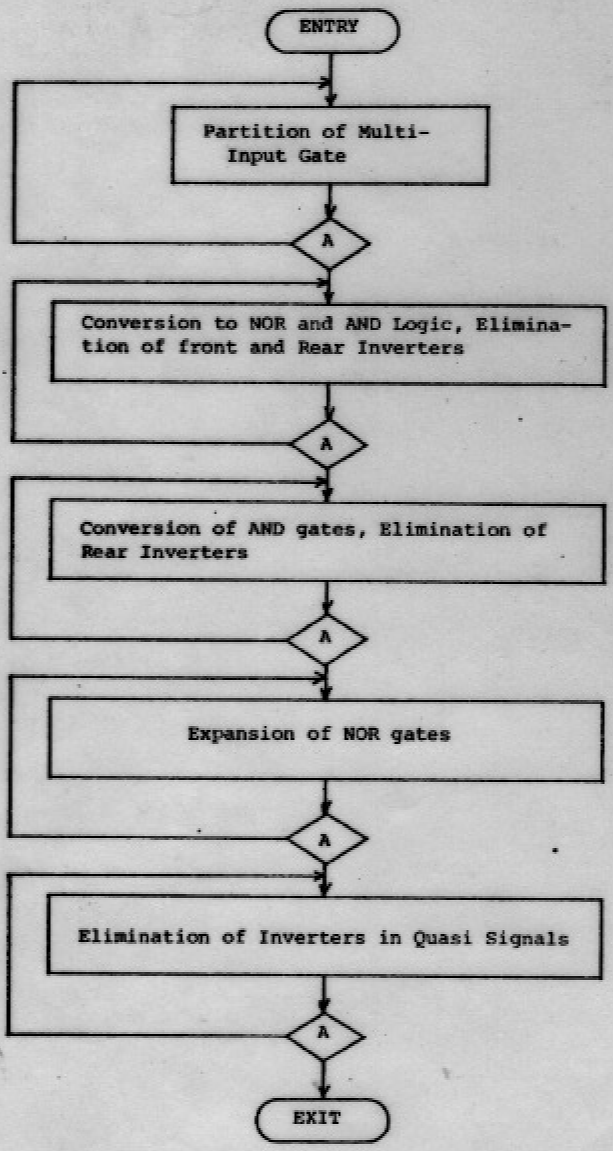
ENTRY
Partition of Multi-Input Gate
A
Conversion to NOR and AND Logic, Elimination of front and Rear Inverters
A
Conversion of AND gates, Elimination of Rear Inverters
A
Expansion of NOR gates
A
Elimination of Inverters in Quasi Signals
A
EXIT

Fig. 7 Flow of Logic Conversion

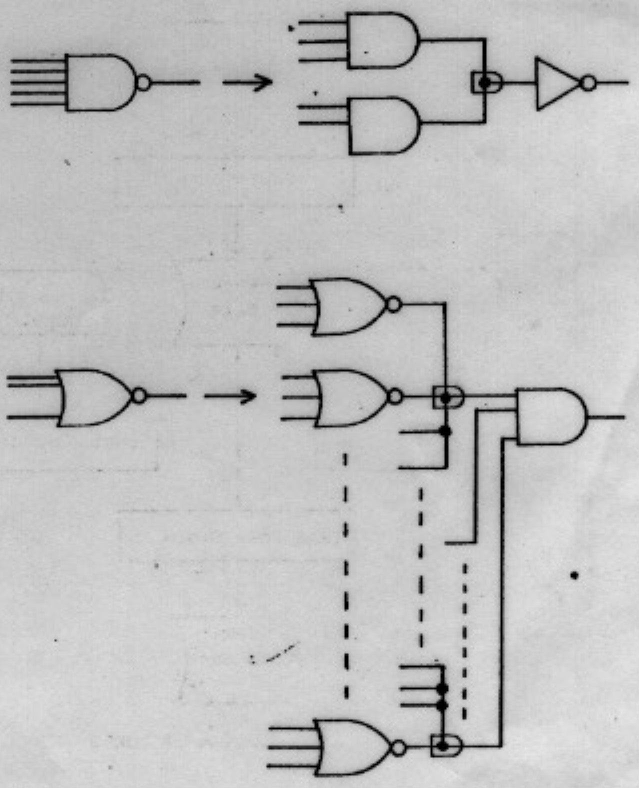

Fig. 8 Partition of Multi-Input Gate

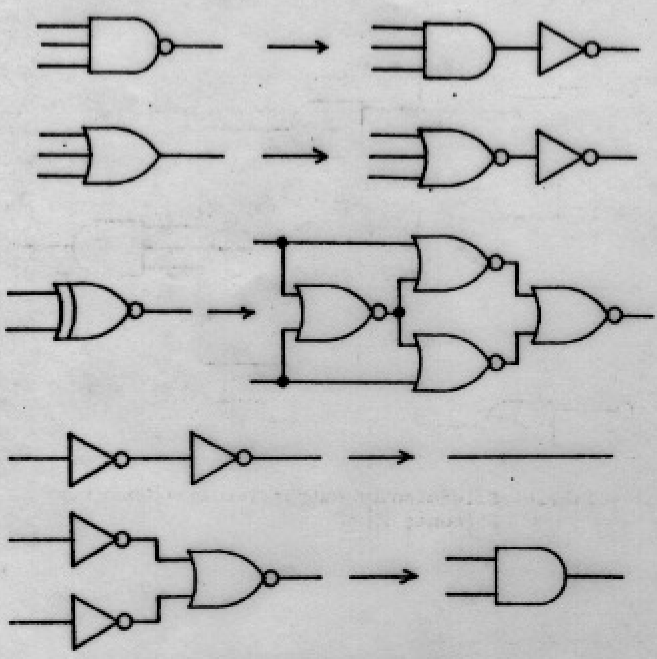

Fig. 9 Conversion to NOR and AND Gates and Elimination of Related Inverters

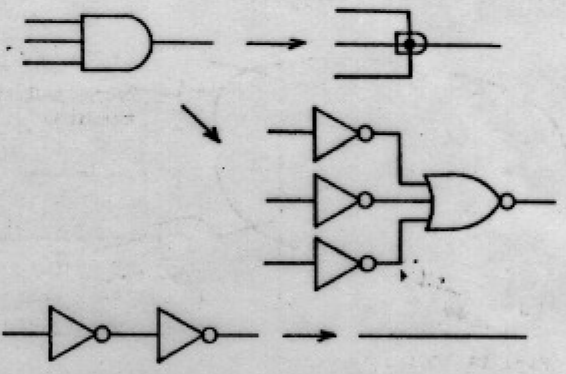

Fig. 10 Conversion of AND Gates and Elimination of Related Inverters

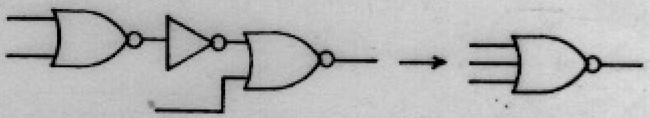

Fig. 11 Expansion of NOR Gates

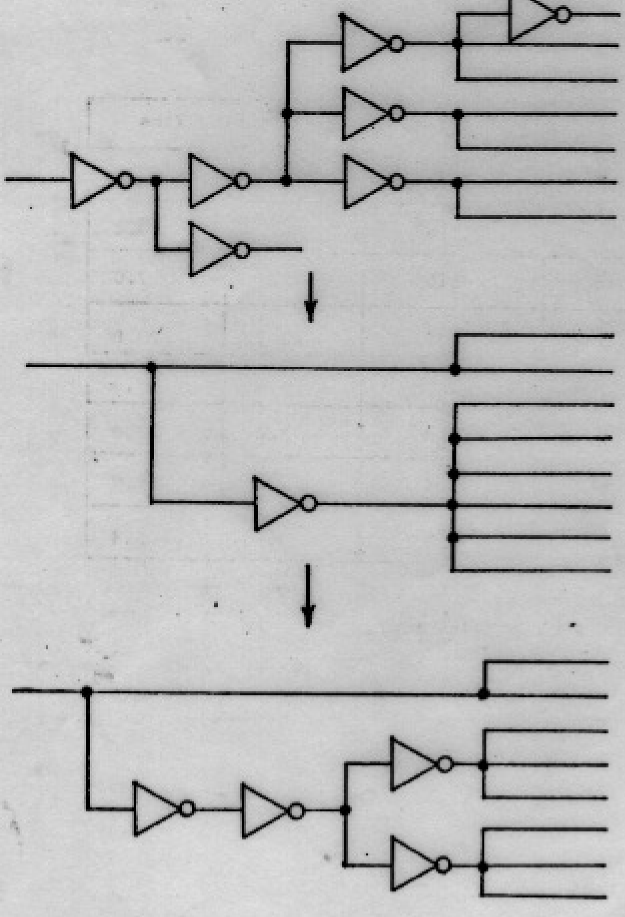

Fig. 12 Elimination of Inverters in a Quasi-Signal

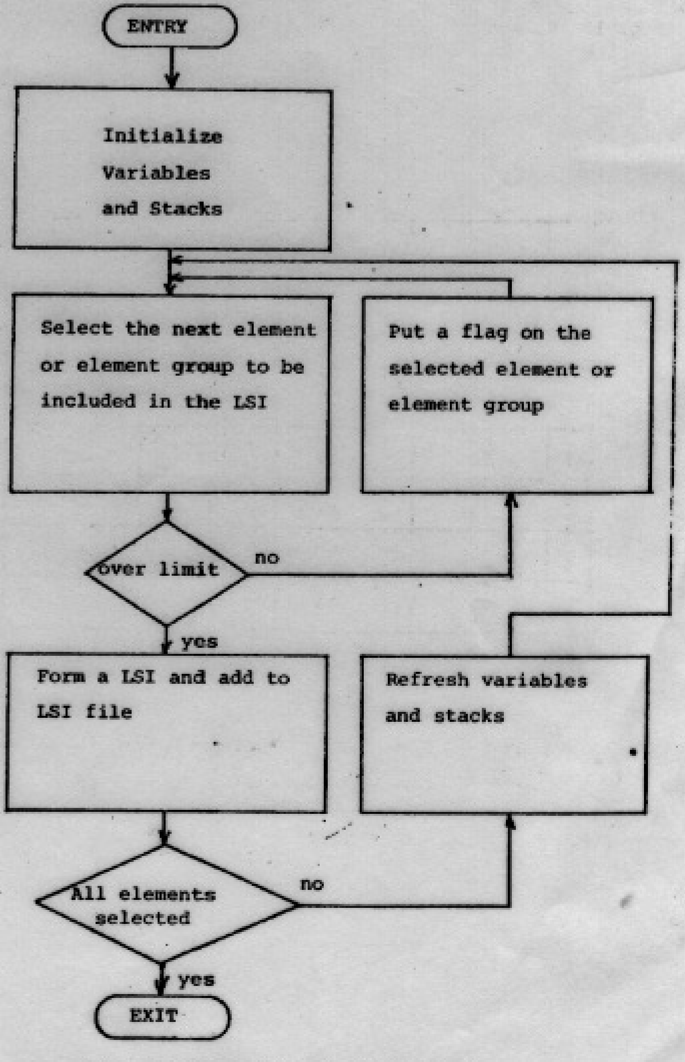
ENTRY
Initialize Variables and Stacks
Select the next element or element group to be included in the LSI
Put a flag on the selected element or element group
over limit
no
yes
Form a LSI and add to LSI file
Refresh variables and stacks
All elements selected
no
yes
EXIT

Fig. 13 Flow of Logic Partitioning

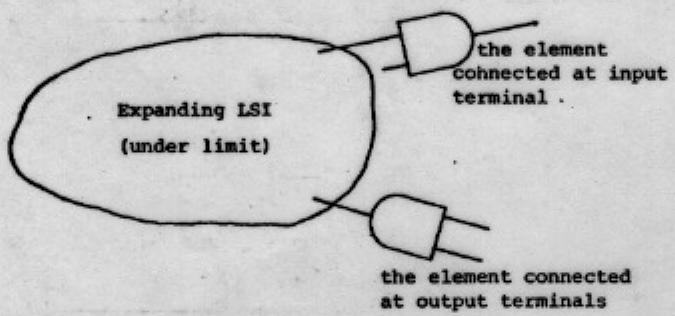
Expanding LSI
(under limit)
the element connected at input terminal .
the element connected at output terminals

Fig. 14 Selection of the Next Element

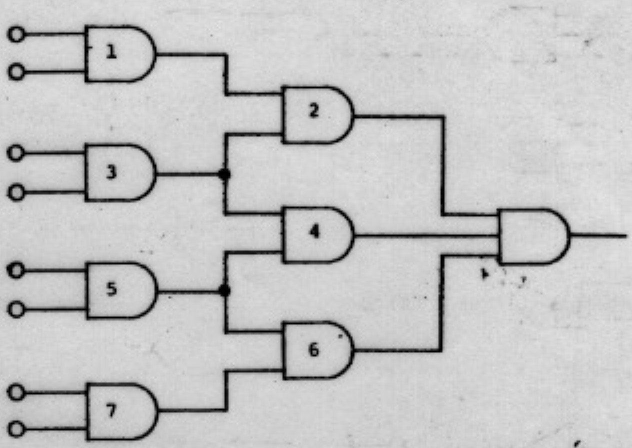

Fig.16 Selection of Output Terminal Connected Elements

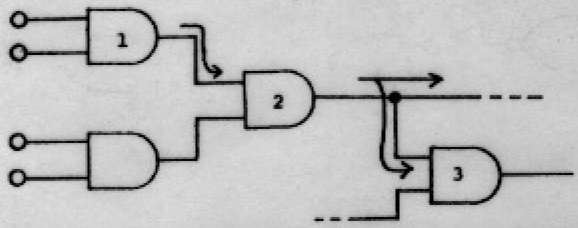

Fig. 15 Selection of Input Trminal Connected Elements

NO.	# elements of source circuit					# cells of terget circ.	Computing Time	
	Inv.	EOR	Gates *	DFF	JKFF		Preprocess	Conv.
1	208	8	340	0	14	1145	5.7	10.1
2	127	14	205	48	16	1159	5.3	7.0
3	318	32	304	20	0	1782	10.3	20.8
4	237	41	299	22	4	1024	5.8	9.7
5	185	34	225	22	2	845	4.8	7.8
6	162	12	255	39	2	1046	5.3	9.0
7	57	24	27	16	0	273	0.86	3.4

inputs of NOR: 3, # inputs of W-AND: 5

fanout of NOR: 10 * elementary gates other than EOR's and Inverters

Table 1 Experimental Results of Logic Conversion

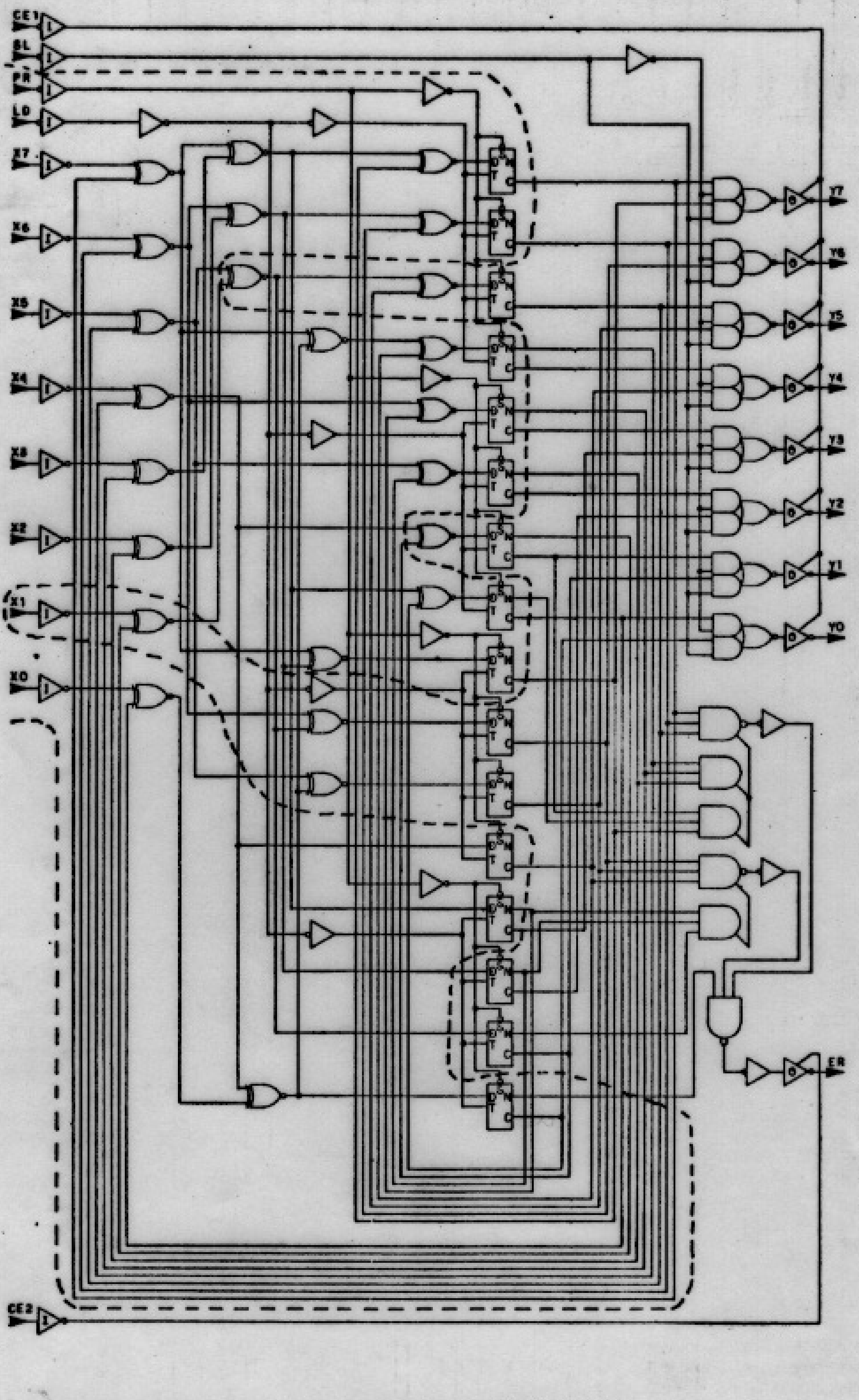

Fig. 17 An example of a source circuit (NO.7 of Table 1)

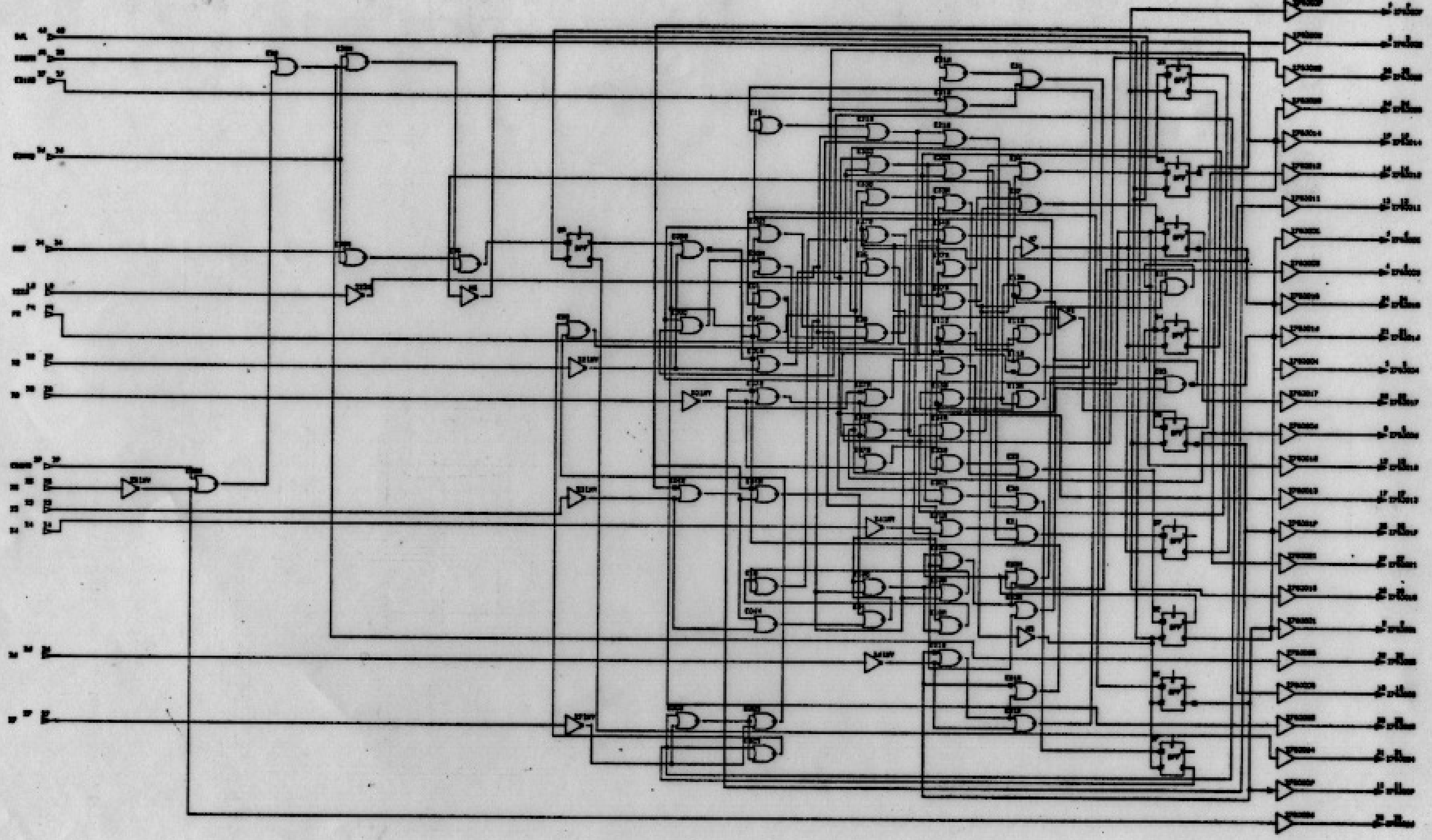

Fig. 18 The result of the reorganization of the circuit in Fig. 17
(The source circuit is converted and partitioned to two parts at the dotted line in Fig. 17.
The left hand side of the dotted line corresponds to Fig. 18)