INCREMENTAL LOGIC SYNTHESIS THROUGH GATE LOGIC STRUCTURE IDENTIFICATION

T. Shinsha*, T. Kubo*, Y. Sakataya**,
J. Koshishita*** and K. Ishihara*

* Systems Development Laboratory, Hitachi, Ltd.
1099 Ohzenji Asao-ku Kawasaki-shi, 215 Japan
** Kanagawa Works, Hitachi, Ltd.
*** Hitachi Software Engineering Co., Ltd.

ABSTRACT

This paper describes incremental logic synthesis for supporting function logic changes in the physical design stage of digital systems. The incremental logic synthesis is distinguished from logic synthesis in the respect that it updates only gate logic components, which must be changed due to the function logic changes, in the physically optimized gate logic structure. For making the incremental logic synthesis feasible, a gate logic structure identification and editing system has been developed with a corresponding gate matrix method as its core. This system has greatly contributed to the increase in design efficiency of the very large computer series M68XH.

1. INTRODUCTION

With the rapid increase in scale and complexity of digital systems, function logic design has been becoming more and more necessary in order to accomplish logic design correctly within a reasonable period of time. The function logic design involves logic synthesis as one of the most important tasks for aiding it. So far, several attempts at logic synthesis[1)-7)] have been made, and most of them will probably continue to be improved with emphasis on the optimization aspect of gate logic to be generated. However, in order to utilize logic synthesis for production, another important aspect involved in function logic changes must be considered sufficiently as well as the optimization aspect.

Some problems caused by function logic changes in logic synthesis are described as follows. In a function logic design environment, following original function logic input, gate logic is generated from the original function logic by a logic synthesis system. Then, if necessary, its logic structure is transformed into a logically optimized logic structure by hand. After that, this logic structure is transformed into a physically optimized logic structure by a layout system and, if necessary, by hand without damaging its logical optimization level. As the design progresses, function logic changes often occur in the original function logic. When these occur, the generation of the updated gate logic by the logic synthesis system is harmful, because it causes the optimized logic structure to be erased completely. That is, this condition causes designers to do extra jobs to recover the lost optimized logic structure. The jobs contain the manual reassignment of optimized design data, the preparation for a rerun of the layout system and the confirmation of its layout result. Moreover, in case the current layout result is largely different from the previous one, a delay check system must be rerun, too. Thus, the disappearance of the optimized logic structure reveals extremely serious problems in both design manpower and design time.

For solving the problems, the preservation aspect of the optimized logic structure must be considered in the generation of the updated gate logic. Such consideration leads a new type of logic synthesis called incremental logic synthesis. It is distinguished from logic synthesis in the respect that only gate logic components which must be changed due to the function logic changes are updated in the optimized logic structure. The incremental logic synthesis involves the preservation of a logically optimized logic structure and that of a physically optimized logic structure. The former is optional depending on the performance of a logic synthesis system. The reason is that if the logic synthesis system can generate a logically optimized logic structure, the preservation can be accomplished automatically. On the other hand, the latter is quite essential in effectively combining a logic synthesis system with a layout system. Therefore, the preservation of a physically optimized logic structure becomes the more important task.

This paper describes incremental logic synthesis, which generates gate logic with a physically optimized logic structure preserved. The incremental logic synthesis must primarily perform the identification of components to be preserved in the physically optimized logic structure. In other words, how structural differences between gate logic structures are identified in detail is the key to an incremental logic synthesis method. From this perspective, functional design verification methods such as the Boolean comparison[8),9)] have little use, because they cannot deal with the gate logic structure. They can, for example, neither detect addition or deletion of any AMP-gate for fan-out adjustment, nor accept addition or deletion of even one input. The desirable method must identify the structural differences at both gate level and pin level. For such a method, the corresponding gate matrix method is devised, which takes the approach characterized below:

1) It treats a pair of logic blocks as a processing unit.
2) It performs gate logic structure identification by dividing the gate logic into three levels of components, i.e., gate, gate element and line.
3) It does so through identifying pairs of corresponding components step by step.
4) It utilizes four kinds of identification indexes based on primary input/output line names of the pair of logic blocks in order to identify pairs of corresponding gates.

0738-100X/86/0000/0391$01.00 ©1986 IEEE

On the basis of the corresponding gate matrix method, a gate logic structure identification and editing system has been developed. It receives both the current gate logic and gate logic generated from the updated function logic, and generates gate logic with the current gate logic structure preserved. This system has been applied to all the logic designs of the very large computer series M68XH together with our logic synthesis system[10].

This paper describes the task of incremental logic synthesis, outlines the gate logic structure identification and editing system and delineates the approach to and the evaluation of the corresponding gate matrix method.

2. INCREMENTAL LOGIC SYNTHESIS

2.1 Objective

In a function logic design environment, once a function logic file is created, this file becomes the design master file for subsequent designs. As the design progresses, function logic changes such as engineering changes, function enhancements and feature additions often occur in the original function logic. When these occur in the physical design stage, the updated gate logic generated from the updated function logic by a logic synthesis system is only an intermediate gate logic. The reason is that the original gate logic structure generated from the original function logic by the logic synthesis system is transformed into the current gate logic structure physically optimized by a layout system. In such a case, an incremental logic synthesis must be performed to update only gate logic components, which must be changed due to the function logic changes, in the current gate logic structure. Therefore, the objective of the incremental logic synthesis is to preserve, as much as possible, physically optimized design data which remain significant despite the function logic changes.

2.2 Gate Logic Structure Identification

For achieving the above objective, gate logic structure identification must be performed. This identification treats as its objects four kinds of gate logics, i.e., original gate logic, current gate logic, intermediate gate logic and updated gate logic. The transition of these gate logics is indicated in Fig. 1. In this figure, the original gate logic and the intermediate gate logic are different in structure. As these differences are derived from functional differences due to function logic changes carried out by designers, they can be regarded as structural differences due to external factors. Similarly, the original gate logic and the current gate logic, while functionally equivalent, are different in structure. As these differences are derived from the optimization of the physical design executed by the layout system, they can be regarded as structural differences due to internal factors.

The structural differences due to external factors are detailed below:

1) Gate level : addition or deletion of gates
2) Pin level : addition, deletion or name change of lines

For example, in Fig. 2 and Fig. 3, gate I03 is deleted and gate K07 added. Also, the primary input line of gate I01, C-N, is deleted, that of gate K02, G-N, added and the common primary input line of gates I04(K03) and I05(K04), D-N, changed into DX-N. Here, I03, K07, etc. represent gate identifications and C-N, *I1, etc. line names. Note that differences in the gate identification and the interconnecting line, with '*' as the first character of its name, can be neglected, because they are automatically named by the logic synthesis system.

On the other hand, the structural differences due to internal factors are detailed below:

1) Gate level : gate replacement
2) Pin level : pin group exchange or pin exchange

In gate array design, a set of technology-dependent cells is used. There are several gates in each cell and several specific gate types, each of which exists in some different types of cells, and more than one specific gate type exists in one type of cell. As the logic synthesis system generates gate logic using a set of predetermined typical gates, the layout system replaces some typical gates of the gate logic with other equivalent gates. The gate type is indicated by the pair of a cell type and a symbol name in the gate logic diagram. Here, it is represented in the form of cell type / symbol name. In Fig. 2 and Fig. 4, LPG101/L1, LPG101/L9 and LPG101/L11 represent 3-input AND gates and LPG101/L1 is predetermined as their typical gate. So, these figures show that gate I02 is replaced with gate J02 and gate I07 with gate J07 respectively.

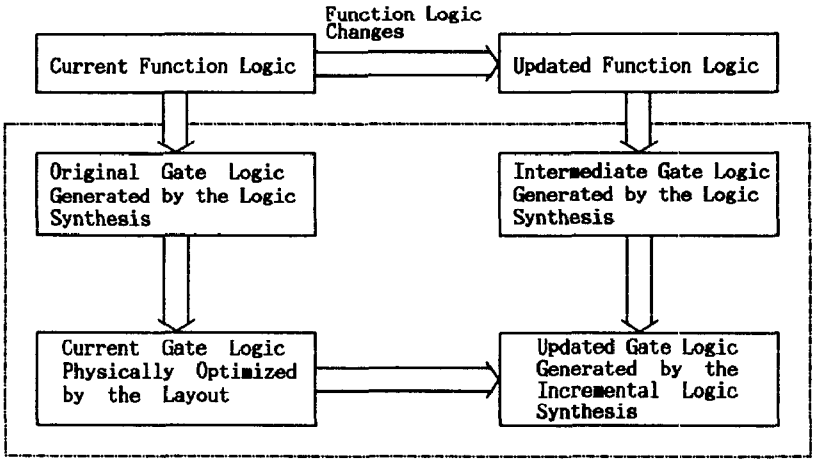

Fig.1 Transition of Gate Logics

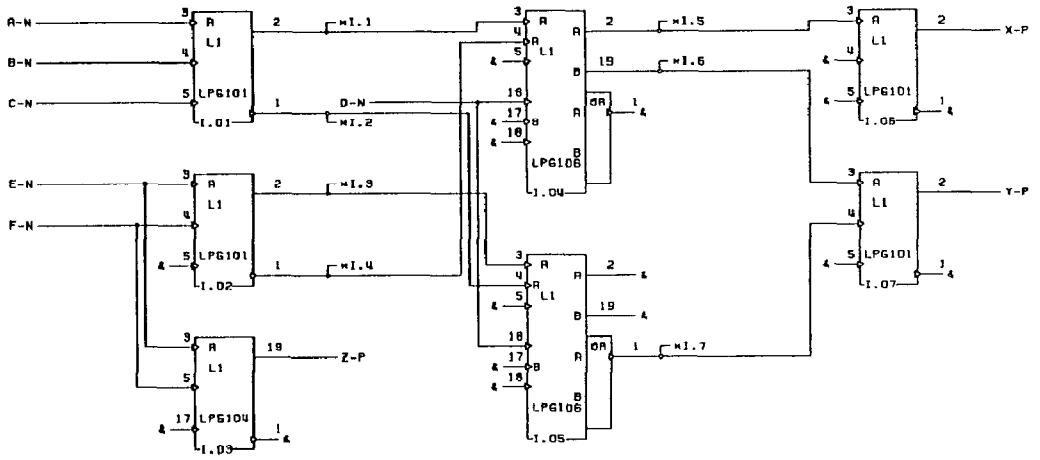

Fig.2 An Example of Original Gate Logic

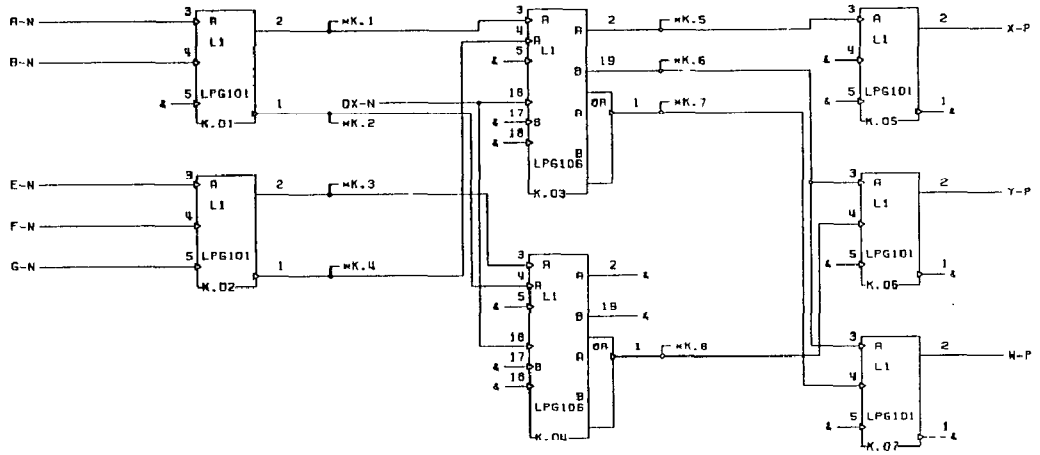

Fig.3 An Example of Intermediate Gate Logic

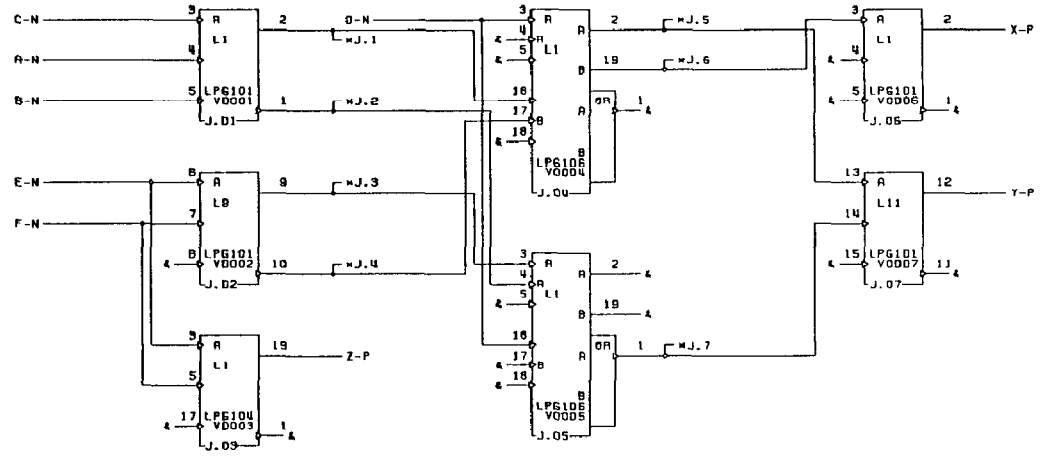

Fig.4 An Example of Current Gate Logic

At the same time as the execution of the replacement, the layout system accomplishes two levels of pin exchanges. This is because the logic synthesis system, if input/output pins of a gate are selectable, selects them according to the order of their number. These can be classified into pin group exchange in a gate and pin exchange in a pin group. The former implies the exchange of gate elements composing a gate and the latter implies that of input pins of a gate element. In a simple gate such as a 3-input AND gate, 6-input OR gate, etc., only the pin exchange is possible because it has only one gate element. On the other hand, in case of a composite gate which is composed of several gate elements, both the pin group exchange and the pin exchange are possible. For example, in Fig. 2, gate I04 has three gate elements, i.e., 3-input AND gate element A with pins 3, 4, 5 and 2, 3-input AND gate element B with pins 16, 17, 18 and 19, and 2-input dotted OR gate element with output pin 1 and the two inputs which are equal to the outputs of gate elements A and B. So, Figures 2 and 4 show that gate element A(B) of gate I04 is exchanged for gate element B(A) of gate J04.

After completing the above gate logic structure identification, the updated gate logic is generated in such a manner that the structural differences due to external factors are rejected and those due to internal factors are accepted. In this gate logic generation, it is also possible to preserve physically optimized design data added to gates and lines themselves such as the gate positions in an LSI, the gate locations on the gate logic diagram and the routing orders. Figure 5 illustrates the updated gate logic to be generated from the gate logics presented in Fig. 3 and Fig. 4. In these figures, V0001, V0002, etc. represent gate positions. Note that gate K07 in Fig. 5 has no gate position, because it is an added gate.

3. GATE LOGIC STRUCTURE IDENTIFICATION AND EDITING SYSTEM

The main stream of the design process is as follows. After initial function logic design, a function logic file is created using the function logic capture system. Then, the logic synthesis system reads the function logic file, performs logic synthesis and writes produced gate logic on the gate logic file. After completing the logic design verification, the physical design is performed, namely, the layout system reads the gate logic file and overwrites the physically optimized logic structure on the input file. Then, the physical design verification is repeated to attain the design goal.

In the above design process, analysis of physical design verification results often requires function logic changes for correcting the current function logic. Figure 6 outlines a design flow for function logic changes in the physical design stage. As seen in this figure, a gate logic structure identification and editing system is introduced to perform an incremental logic synthesis. This system reads both the current gate logic file with the physically optimized logic structure and the intermediate gate logic file generated from the updated function logic file. It then generates the updated gate logic file with the physically optimized logic structure preserved. After executing this process, supplemental physical design is performed upon the added components of the updated gate logic using the layout system. This operation transforms the updated gate logic into the current gate logic corresponding to the updated function logic, which is now the current function logic. In the physical design stage, such design change processes are repeated until the design goal is reached.

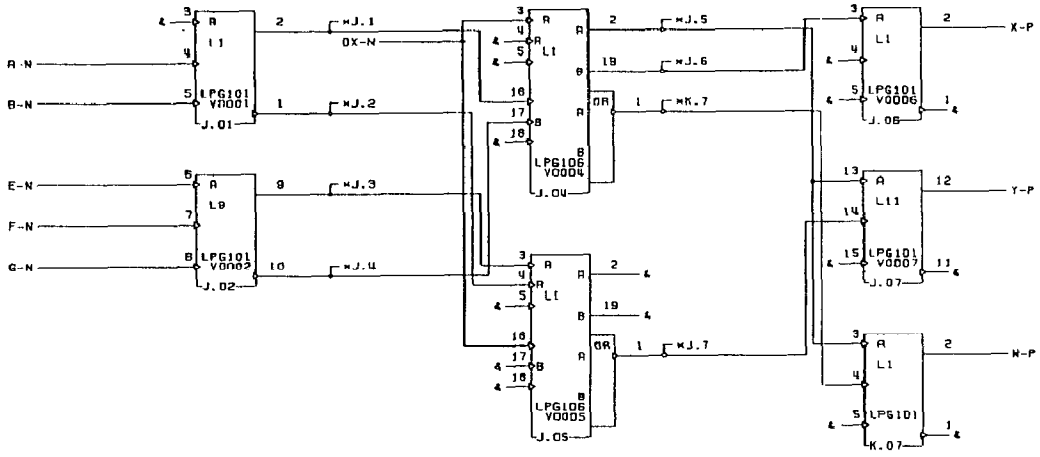

Fig.5 An Example of Updated Gate Logic

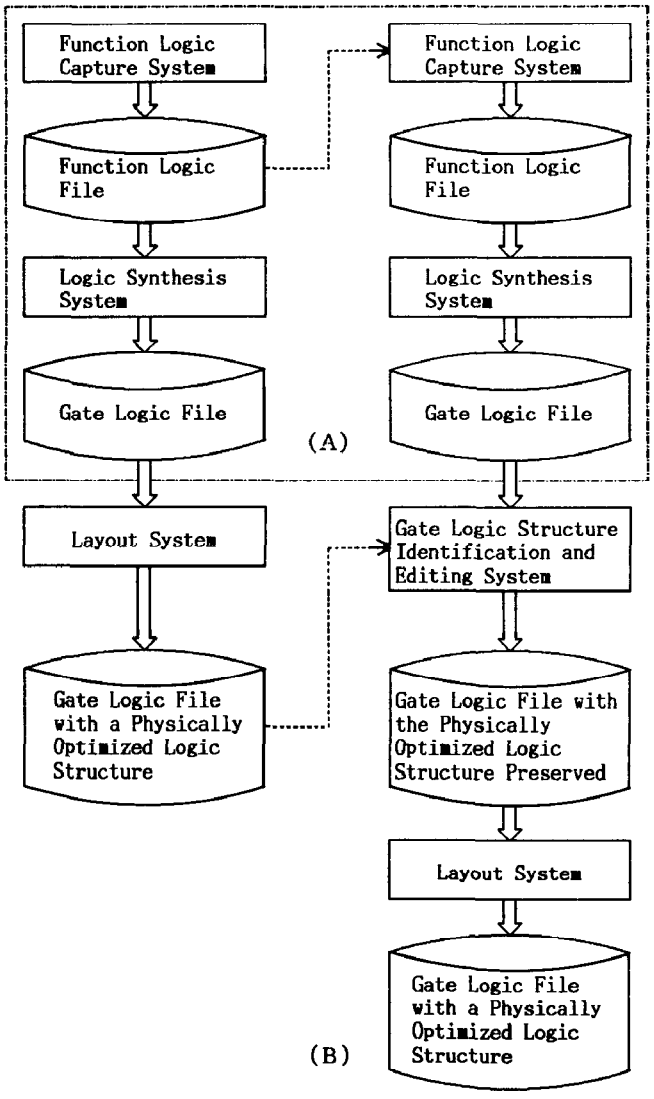

Fig. 6 Design Flows for Function Logic Changes

(A): Design Flow for Function Logic Changes in the Logic Design Stage
(B): Design Flow for Function Logic Changes in the Physical Design Stage

4. CORRESPONDING GATE MATRIX METHOD

4.1 Characteristics of the Method

The gate logic structure identification requires the introduction of some identification indexes. In general, an increase of distinguishable items produces more powerful identification indexes and makes an identification problem easier. However, the fact that most of the gate logic descriptions are automatically generated makes the identification problem treated here very difficult. In the gate logic description, only the primary input/output line name of the pair of logic blocks and the gate type can be selected as the distinguishable items. As the former is uniquely named in the function logic description, it can be regarded as the primary distinguishable item. On the other hand, as the latter can only distinguish a gate from among different type of gates, it can be regarded as the secondary distinguishable item. This consideration suggests that the identification indexes, which are significant to only a pair of the same type of gates, should be derived from information concerning the primary input/output line names.

From this perspective, the following two identification indexes can be assigned to each pair of the same type of gates:

1) Common primary input line ratio : $R_{cil}(G_i,G_j)$

$$R_{cil}(G_i,G_j) = 50\left[\frac{N_{cil}(G_i,G_j)}{N_{il}(G_i)} + \frac{N_{cil}(G_i,G_j)}{N_{il}(G_j)}\right]$$

where G_i denotes a gate composing the current gate logic, G_j one composing the intermediate gate logic, $N_{il}(G_i)$ the number of the primary input lines whose logical values have influence upon the output logical value of G_i and $N_{cil}(G_i,G_j)$ the number of common lines between the primary input lines of G_i and those of G_j respectively.

2) Common primary output line ratio : $R_{col}(G_i,G_j)$

$$R_{col}(G_i,G_j) = 50\left[\frac{N_{col}(G_i,G_j)}{N_{ol}(G_i)} + \frac{N_{col}(G_i,G_j)}{N_{ol}(G_j)}\right]$$

where $N_{ol}(G_i)$ denotes the number of the primary output lines whose logical values are determined by the output logical value of G_i and $N_{col}(G_i,G_j)$ the number of common lines between the primary output lines of G_i and those of G_j respectively.

Moreover, by extending the above concept, the following two identification indexes can also be assigned to each pair of the same type of gates:

3) Common fan-in gate ratio : $R_{cig}(G_i,G_j)$

$$R_{cig}(G_i,G_j) = 50\left[\frac{N_{cig}(G_i,G_j)}{N_{ig}(G_i)} + \frac{N_{cig}(G_i,G_j)}{N_{ig}(G_j)}\right]$$

where $N_{ig}(G_i)$ denotes the number of fan-in gates and the primary input lines directly connected to G_i and $N_{cig}(G_i,G_j)$ the sum of the number of pairs of corresponding fan-in gates and that of common primary input lines respectively.

4) Common fan-out gate ratio : $R_{cog}(G_i,G_j)$

$$R_{cog}(G_i,G_j) = 50\left[\frac{N_{cog}(G_i,G_j)}{N_{og}(G_i)} + \frac{N_{cog}(G_i,G_j)}{N_{og}(G_j)}\right]$$

where $N_{og}(G_i)$ denotes the number of fan-out gates and the primary output lines directly connected to G_i and $N_{cog}(G_i,G_j)$ the sum of the number of pairs of corresponding fan-out gates and that of common primary output lines respectively.

Note that, in $R_{cig}(G_i,G_j)$ and $R_{cog}(G_i,G_j)$, G_i and G_j can also be treated as gate elements instead of gates. With the above identification indexes, $R_{cil}(G_i,G_j)$ and $R_{col}(G_i,G_j)$ have fixed values relative to the overall gate logic structures, but, $R_{cig}(G_i,G_j)$ and $R_{cog}(G_i,G_j)$ have variable values depending on the advance of the identification process. Their values are inclined to become larger because of the increase in corresponding gate pairs.

On the basis of the identification indexes, the corresponding gate matrix method is constructed. This method uses the approach characterized below :

1) It treats a pair of logic blocks as a processing unit.
2) It performs the gate logic structure identification by dividing the gate logic into three levels of components, i.e., gate, gate element and line.
3) It does so through identifying pairs of corresponding components step by step, from the direction of the global viewpoint concerning the overall gate logic structures to the local viewpoint concerning neighborhoods of a pair of components under examination.
4) First, it identifies pairs of corresponding gates through iterating gate matchings based on $R_{cil}(G_i,G_j)$, $R_{col}(G_i,G_j)$, $R_{cig}(G_i,G_j)$ and $R_{cog}(G_i,G_j)$.
5) Second, it identifies pairs of corresponding gate elements through iterating gate element matchings based on $R_{cig}(G_i,G_j)$ and $R_{cog}(G_i,G_j)$.
6) Third, it identifies pairs of corresponding lines in consideration of pin exchangeability.
7) Last, it selects the corresponding components of the current gate logic and the remaining components of the intermediate gate logic, and combines and edits them.

4.2 Outline of the Method

How the corresponding gate pairs are identified is at the core of the corresponding gate matrix method. Here, let us present the gate identification process in which the updated gate logic shown in Fig. 5 is generated from the intermediate gate logic in Fig. 3 and the current gate logic in Fig. 4. The gate identification process is detailed below:

1) The corresponding gate matrix shown in Table 1 (A) is made through calculating $R(G_i,G_j)$. In the table, a column corresponds to a G_j, a row a G_j and a table element a $R(G_i,G_j)$. The suffix of R is either 'cil' or 'col' and the selection of it depends on the dimensional relationship between the number of primary input lines and that of primary output lines. In the intermediate gate logic, as the former is larger than the latter, i.e., $6 > 3$, $R_{cil}(G_i,G_j)$ is selected. A $R_{cil}(G_i,G_j)$ is obtained by the above defined expression. For example, as $N_{il}(J01) = 3$, $N_{il}(K01) = 2$ and $N_{cil}(J01,K01) = 2$,

$$R_{cil}(J01,K01) = 50\left\{\frac{2}{3}+\frac{2}{2}\right\} = 83$$

2) The gate identification is performed according to the large order of a value of a $R_{cil}(G_i,G_j)$. That is, with specific gate pair (G_a,G_b), if only $R_{cil}(G_a,G_b)$ has a unique value to all other $R_{cil}(G_i,G_b)$s and $R_{cil}(G_a,G_j)$s, (G_a,G_b) is regarded as a corresponding gate pair. Otherwise, this implies the condition in which one or more G_is virtually correspond to one or more G_js except for one-to-one correspondence. When this condition occurs, G_is and G_js composing of (G_i,G_b)s and (G_a,G_j)s with the same value as that of $R_{cil}(G_a,G_b)$ are regarded as collision gates, and form a collision gate group. During the repetition of these operations, a G_i or G_j which has already been recognized as a corresponding gate or collision gate is neglected at all times. Consequently, Table 1 (A) indicates an identification result that (J01,K01) and (J02,K02) are corresponding gate pairs and (J04,J05,K03,K04) and (J06,J07,K05,K06,K07) collision gate groups.
3) A collision gate group is selected in order and its corresponding gate matrix is made through calculating $R(G_i,G_j)$. In this procedure, the suffix of R is either 'cig' or 'cog', and the selections of it and the operation order depend on which identification index has been used in the preceding step. In this example, as the identification index is $R_{cil}(G_i,G_j)$, $R_{cog}(G_i,G_j)$ is selected. Also, a collision gate group is selected relative to the small order of its group rank which is the minimum of ranks of G_is in it. Here, the rank of a G_i is determined by tracing gates starting from the primary output lines. With the above identified collision gate groups, as the rank of (J04,J05,K03,K04) is 2 and that of (J06,J07,K05,K06,K07) 1, first the latter and then the former are selected in order and the corresponding gate matrixes shown in Table 1 (B) and (C) are made. These tables indicate identification results that (J06,K05), (J07,K06), (J04,K03) and (J05,K04) are corresponding gate pairs and no collision gate group exist. In Table 1 (C), note that the result of Table 1 (B) is referred to in the calculation of $R_{cog}(G_i,G_j)$, e.g., as $N_{cog}(J04,K03) = 1$ because corresponding gate pair (J06,K05) has already been identified, $N_{og}(J04) = 2$ and $N_{og}(K03) = 3$,

$$R_{cog}(J04,K03) = 50\left\{\frac{1}{2}+\frac{1}{3}\right\} = 43$$

4) Now that there remains no collision gate group to be processed, the gate identification is finished. Here, note that the above identification results also suggest that gate J03 is deleted and gate K07 added, which have not been recognized as corresponding gates in the above gate identification.

5. EVALUATIONS

For evaluating the corresponding gate matrix method, various types of function logic changes on 25 ECL-LSIs were selected. The results of the gate logic structure identification are shown in Table 2. As seen in the table, all gates in each LSI are classified into unchanged gates and changed gates with respect to whether structural differences are detected in them. Moreover, the changed gates are classified into preservable gates, unpreservable gates and added gates through analyzing the acceptance of the structural differences due to internal factors at both gate level and pin level. Here, the added gates are left out of consideration, because the method cannot transform each of their logic structures into any physically optimized logic structure. The table demonstrates that the method has preserved all but 3 gates, namely, it has adequately

identified more than 99 % of the components to be preserved against the function logic changes.

Besides, the method has no performance problems. It can accomplish the gate logic structure identification within a reasonable amount of time.

6. CONCLUSION

The incremental logic synthesis for function logic changes in the physical design stage is presented. It updates only gate logic components, which must be changed due to the function logic changes, in the physically optimized gate logic structure. For making the incremental logic synthesis feasible, the corresponding gate matrix method is also presented. The method identifies both the structural differences due to external factors and those due to internal factors between the gate logic structures in order to recognize components to be preserved. The evaluation results prove that the method is effective enough for production use. The method has been implemented as the core of the gate logic structure identification and editing system. The system has been applied to all the logic designs of the very large computer series M68XH together with the logic synthesis system, and has greatly contributed to the increase of their design efficiency.

ACKNOWLEDGEMENT

The authors acknowledge Dr. Jun Kawasaki of Systems Development Laboratory, Hitachi, Ltd. and Dr. Yasuhiro Ohno and Yooji Tsuchiya of Kanagawa Works, Hitachi, Ltd. for their invaluable guidance and advice.

REFERENCES

1) J. A. Darringer, D. Brand, J. V. Gerbi, W. H. Joyner and L. Trevillyan, "LSS: A System for Production Logic Synthesis", IBM J. Res. Develop., Vol. 28, No. 5, Sept. 1984, pp. 537-544.

2) J. V. Rajan and D. E. Thomas, "Synthesis by Delayed Binding of Decisions", Proc. of the 22nd DAC, June 1985, pp. 367-373.

3) J. Dussault, C. Liaw and M. M. Tong, "A High Level Synthesis Tool for MOS Chip Design", Proc. of the 21st DAC, June 1984, pp. 308-314.

4) O. Karatsu, T. Hoshino, M. Endo and K. Ueda, "An Automatic VLSI Synthesizer", Proc. of ISCAS 85, June 1985, pp. 403-406.

5) N. Kawato, T. Saito and H. Sugimoto, "DDL/SX: A Rule-based Expert System for Logic Circuit Synthesis", Proc. of ISCAS 85, June 1985, pp. 885-888.

6) T. Shinsha, T. Kubo, M. Hikosaka, K. Akiyama and K. Ishihara, "POLARIS: Polarity Propagation Algorithm for Combinational Logic Synthesis", Proc. of the 21th DAC, June 1984, pp. 322-328.

7) T. Shimizu, Y. Takamine, T. Shinsha and T. Kubo, "A Logic Synthesis Algorithm for the Design of a High Performance Processor", Proc. of ISCAS 85, June 1985, pp. 407-410.

8) S. B. Akers, "A Procedure for Functional Design Verification", Proc. of the 10th FTCS, 1980 June, pp. 65-67.

9) G. L. Smith, R. J. Bahnsen and H. Halliwell, "Boolean Comparison of Hardware and Flowcharts", IBM J. Res. Develop., Vol. 26, No. 1, Jan. 1982, pp. 106-116.

10) Y. Tsuchiya, M. Morita, Y. Ikariya, S. Uematsu, E. Tsurumi, T. Mori and T. Yanagida, "Establishment of Higher Level Logic Design for Very Large Scale Computer", Proc. of the 23rd DAC, June-July 1986, this issue.

Table 1 Corresponding Gate Matrixes

(A)

Gj \ Gi	J.01	J.02	J.03	J.04	J.05	J.06	J.07
K.01	83	0				67	67
K.02	0	83				50	50
K.03				67	67		
K.04				67	67		
K.05	50	67				67	67
K.06	50	67				67	67
K.07	50	67				67	67

(B)

Gj \ Gi	J.06	J.07
K.05	100	0
K.06	0	100
K.07	0	0

(C)

Gj \ Gi	J.04	J.05
K.03	42	67
K.04	75	100

Table 2 Evaluation Results

Mean of Total Gates	1288
Mean of Unchanged Gates	1039
Mean of Changed Gates	249
1) Mean of Preservable Gates	190
2) Mean of Unpreservable Gates	3
3) Mean of Added Gates	56

gate : 3-input gate conversion