Portland State University

PDXScholar

Dissertations and Theses

Dissertations and Theses

1989

A new approach to state minimization of finite state machines

William Yue Zhao

Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Part of the [Electrical and Computer Engineering Commons](#)

Let us know how access to this document benefits you.

Recommended Citation

Zhao, William Yue, "A new approach to state minimization of finite state machines" (1989). *Dissertations and Theses*. Paper 3951.

<https://doi.org/10.15760/etd.5835>

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

AN ABSTRACT OF THE THESIS OF William Yue Zhao for the
Master of Science in Electrical Engineering presented February 28, 1989.

Title: A New Approach to State Minimization of Finite State Machines.

APPROVED BY THE MEMBERS OF THE THESIS COMMITTEE:

Marek A. Perkowski

Jack C. Riley

Robert L. Broussard

A complete program to ease the task of large scale Finite State Machine (FSM) minimization presented in this thesis: TDFM (Two Dimensional FSM Minimizer), is a part of the DIADES system. DIADES is an Automatic Design Synthesis System whose development in the Department of Electrical Engineering at Portland State University is supported in part by a research grant from SHARP Microelectronics Technology.

In compliance with the requirement of the DIADES system, the program TDFM accepts two types of FSM formats of input data, .stab (state table) and .kiss (Keep the Internal States Simple). TDFM's minimization procedures include the input based state minimization (column minimization) and internal state based minimization (row minimi-

zation). As the major contribution of the program TDFM and this thesis, it has been proven that this new type of minimization procedure, the column minimization, is possible in the entire state minimization process and is as necessary as the traditional row minimization. Furthermore, both of these two types of minimization can be done basically by the same routines after some modification.

TDFM performs these two minimization processes in series, and iterates these two procedures until there are no more compatible inputs or present states in the final form of the machine. In other words, it generates an equivalent machine M^* , which has the minimal numbers of columns and rows. The optimal machine M^* can also completely replace the initial machine M^0 for the next design stages in the DIADES system. It is the goal of the processes mentioned above to minimize the area of the VLSI circuit, increase the speed, and improve the reliability.

These two minimization procedures basing on the state table are iterately performed through the following steps:

- (1) find all of the compatible pairs of present states or inputs, by detecting corresponding compatible conditions for each minimization process.
- (2) generate all of the compatible groups (CG) for either the input columns (CGI) or the present state rows (CGP) and focus on those maximal compatible groups.
- (3) create the closed and complete covering (CC) sets and then search out one set of the minimal closed and complete covering (MCC) of either the input columns (MCCI) or present state rows (MCCP) from those CGs especially from those maximal CGs after each minimization process.

Finally, after the iteration of above minimization process steps, the last created MCCI and MCCP forming from the most recently created state table are the optimal minimal closed and complete coverings (OMCC) of input columns (OMCCI) and present

state rows (OMCCP).

For the purpose of searching for all the CGs in step 2, and for the MCCs (especially for the OMCC) in step 3 during column minimization and row minimization as well as in the additional procedure of column minimization (the input combinational logic encoding), the program TDFM employs a universal Artificial Intelligence subroutine MULTCOM. This subroutine is used to list all of the CGs (first call for both column and row minimizations described above) and later on to search out the MCCs (second call) as well as to search for the MCC in the input encoding by using different specified cost functions complemented by the respective quality functions and the selected searching strategies.

This thesis discusses the TDFM program in CHAPTER II and its main subroutine MULTCOM in CHAPTER IV in order to explain the entire minimization process.

During the development of the program TDFM, I obtained essential direction and concern from my adviser Dr. Marek Perkowski and the help for understanding the routine MULTCOM from my colleague, Jiuling Liu. Here, I sincerely thank them for their help. Without this help, the program TDFM may have been impossible.

Electrical Engineering Department

Portland State University

William Y. Zhao

**A NEW APPROACH TO STATE MINIMIZATION OF
FINITE STATE MACHINES**

by

WILLIAM YUE ZHAO

**A thesis submitted in partial fulfillment of the
the requirements for the degree of**

**MASTER OF SCIENCE
in
ELECTRICAL ENGINEERING**

Portland State University

1989

TO THE OFFICE OF GRADUATE STUDIES:

The members of the Committee approve the thesis of William Yue Zhao presented
February 28, 1989.

Marek A. Perkowski, Chair
Jack C. Riley
Robert L. Broussard

APPROVED:

Rolf Schaumann, Coordinator, Electrical Engineering
Bernard Ross, Vice Provost for Graduate Studies

TABLE OF CONTENTS

	PAGE
LIST OF TABLES	v
LIST OF FIGURES	vii
MATHEMATICAL SYMBOLS	ix
NOTICES	x
CHAPTER	PAGE
I INTRODUCTION	1
II THE PROGRAM : TWO DIMENSIONAL FSM	
MINIMIZER TDFM	9
2.1. Principle of State Table Minimization	10
2.2. State Minimization Based On the Inputs	
(Column minimization)	25
2.3. State Minimization Based On the Internal States	
(Row Minimization)	31
2.4. Input Signal Combinational Encoding	36
III FORMATS OF FINITE STATE MACHINES	44
3.1. Translation of Input Data Formats from .kiss to .stab	45
3.2. Translation of Input Data Formats from .liss to .stab	50
3.3. Specification of Input Files	57

CHAPTER	PAGE
IV DESCRIPTION OF ROUTINE MULTCOM	61
4.1. Selection of Strategies, Parameters and Size	
Limitation of the Tree	61
4.2. Call to MULTCOM for Finding All Compatible Groups	66
4.3. Call to MULTCOM for Creating MCCs	70
4.4. Call to MULTCOM for Input Combinational Covering	82
V CONCLUSION	87
REFERENCES	90
APPENDIX : USERS MANUAL	92

LIST OF TABLES

TABLE		PAGE
I	An Initial State Table of Mealy Machine M^0	26
II	Merger List ZGOD from State TABLE I for Column Minimization	28
III	List of Compatible Groups TABIMP from State TABLE I	29
IV	New State Table M^1 Created from TABLE I after Column Minimization	30
V	Merger List ZGOD Created from TABLE IV for Row Minimization	33
VI	List of Compatible Groups from State TABLE IV	34
VII	New State Table M^2 Created from TABLE IV after Row Minimization	34
VIII	Optimal State Table M^* after Iteration of Minimizations	35
IX	Process of Input Group Collection in List CGG	37
X	Code Table Created from Accumulation of Input Compatible Groups	38
XI	Process of Tabulation for Encoding Output Function F_m	41
XII	List of Compatible Groups from State Space in Figure 12(a)	42
XIII	Standard .kiss Formatted File	46
XIV	Modified .kiss Formatted Input File from TABLE XIII	49

TABLE	PAGE
XV An Example of State Table Created from Flow Chart in Figure 14.	53
XVI State Table Modified from TABLE XV	54
XVII Data File 'fort.1' for Input Modification	59
XVIII Don't Care Terms of .stab Formatted Data in Percentage	59
XIX Benchmarks of Some FSMs	89

LIST OF FIGURES

FIGURE	PAGE
1. Structure of PLA Based Finite State Machine	3
2. Data Flow Chart of Program TDFM	9
3. Complete Algorithm Flow Chart of Program TDFM	13
4. An Example of Merger Graph G	22
5. Reduced Merger Graph from Figure 4	23
6. Complement Merger Graph from Figure 4	24
7. Colored Complement Merger Graph $\bar{G}$ from Figure 6	25
8. Algorithm of Compatibility Test	27
9. Triangular Merger Table Derived from TABLE I	29
10. Colored Complement Merger Graph for Solving TABLE I	30
11. Multi-Outputs Karnaugh Map and Its Separated Submaps	39
12. State Space for Tree Search of Function F_m and Its Minimized Form	42
13. Algorithm of Decimal Numbers Setting for Inputs	48
14. Flow Chart of Finite State Machine M	51
15. Solution Tree for Creating All CGs	67
16. Verbal Description of Tree Search in Figure 15 for CGs	71
17. Comparison of Merger Graphes for Column and Row Minimizations	73

FIGURE	PAGE
18. State Space for Tree Search of MCC	76
19. Solution Tree for Creating MCC	79
20. Verbal Description of Tree Search in Figure 19 for MCC	81
21. Solution Tree for Creating Covering of Function F_m	85
22. Verbal Description of Tree Search in Figure 21 for MCC	86

MATHEMATICAL SYMBOLS

$\{a\}$: a group of single states.

$\{A\}$: a set of groups. Each group of this set can be represented by several states which belong to one group or be represented by the address of this group.

$\{A\} = \emptyset$: A is a empty set.

$\cup$: union.

$\cap$: intersection.

$\&$: and, conjuncton.

$\vee$: or, disjunction.

$\oplus$: exclusive or.

$a := b$: a and b are compatible.

$a \equiv b$: a and b are consistant.

$\bar{A}$: the complement of A .

$A \subseteq B$: A is included by B .

$a \in B$: a is an element of B .

$A | B$: A conditioned on B .

$a \rightarrow b$: b is implied by a .

$CARD(A)$: the number of elements in A .

$\sum(A)$: the sum of the elements in A .

$\exists$: there exists.

$\forall$: for all.

$m = fix(n)$: m is the integer part of the real number n .

NOTICES

CDA : computer design automation.

FSM : finite state machine.

PS : present states or present state rows.

NS : next states.

CG : compatible group.

CGI : compatible group of input columns.

CGP : compatible group of present state rows.

SCGP : strong compatible group of present state rows.

ICGP : implying compatible group of present state rows.

MCC : minimal closed and complete covering set.

MCCI : minimal complete covering set of input columns.

MCCP : minimal closed and complete covering set of present state rows.

OMCC : optimal closed and complete covering set.

OMCCI : optimal complete covering set of input columns.

OMCCP : optimal closed and complete covering set of present state rows.

COMI : state space of input compatible pairs.

COMP : state space of PS compatible pairs.

COI : state space of input covering.

COP : state space of PS covering.

CLP : the symbol of implying groups.

COV : state space of column and row coverings.

CHAPTER I

INTRODUCTION

The technology level used in circuit design rapidly progresses as the more sophisticated architectures used in VLSI (Very Large Scale Integration) computers and digital circuit controllers are developed. This has especially been the case in the Eighties. One component that plays a very important role in this respect (in the hardware design of computers and other digital circuits) is the Finite State Machine (FSM). The state minimization, state assignment, and Boolean minimization algorithm of FSM produce much better results for the machine that have many don't care terms. Such machines can be described in the initial specifications of control units (CU) in high-level synthesis systems. This is, for instance, the case when the synthesis starts from the timing diagrams (like for the control of the bus interface protocols), or when it uses constrained high-level CU specifications.

The hardware implementation of the FSM presented in this thesis consists of three major components: input encoding circuit, combinational circuit as well as memory components. These can be seen in the structures of the Mealy FSM model as shown in Figure 1(a) as well as in the Moore FSM model as shown in Figure 1(b).

The difference between the Mealy and Moore machines is that the outputs of a Mealy machine are dependent on both the present states and the inputs. This is accomplished in the Mealy machine by using the present states given by the memory components as the functions of the machine states together with the primary inputs to make up its primary outputs. On the other hand, the outputs of a Moore machine are determined by the present states alone. The primary outputs of a Moore machine are created

from the memory components, via a separate output network. In both the Mealy and Moore machines, the secondary states yielded by the combinational circuit form the next states and become the input states to the memory components. The combinational circuit and the memory components form a closed loop so that the internal states are changed in the loop according to the transition functions. The memory components store the representations of the states of the machine at any given time. The inputs to the memory components are the next states and their outputs are the present states. The additional input encoder is created to reduce the overall machine complexity. It is used to convert those primary input signals into the secondary input signals according to the compatibility of the primary input columns. The pertinent reduction procedure which is, of course, not related to the memory components is introduced in § 2.4.

The VLSI implementation of a sequential circuit has to satisfy two major objectives:

- (1) regular and structured design that can be supported by Computer Design Automation (CDA) tools;
- (2) size and performance optimization of the silicon implementation.

A PLA based or ROM based implementation of the FSM combinational circuit and FSM based sequential circuit can be used to realize both of these goals. Since the FSM memory components, as well as the PLAs, can be designed using regular structures, the automation of FSM circuits design are allowable. Moreover, several techniques, such as logic minimization and topological compaction, to design area-efficient PLAs have been practical. Other realization of combinational functions, such as the standard cells, Weinberger layouts, etc. are also applied on FSM design. A PLA-based FSM design can then be optimized with respect to time-efficient performance.

The memory components of a FSM consist of a set of flip-flops. They can be of several types, (Delay (D), Toggle (T), JK). Some functions can be implemented more

(a) Mealy Machine

(b) Moore Machine

Figure 1. Structure of PLA based finite state machine

efficiently with a particular type of flip-flops. For example, counters are usually implemented using T flip-flops and the generic sequential functions using JK flip-flops.

The operation of VLSI systems is often synchronized by the system's clock. The main goal of such a design is to maintain a "race-free" behavior even when the circuit size is large. For this reason, the implementation model of sequential functions used in this thesis is that of a Synchronous Finite State Machine: an FSM whose next states are determined by their present states and inputs at the time of the clock pulse.

The Finite State Machine design theory was conceived in the 1960s and subsequently a few single synthesis systems were implemented, upgraded and improved by

several researchers. These research efforts had very little, if any, impact on the design practice in the industrial environment during that period. The findings of these efforts were not directly applicable to the computer realization. Another shortcoming of the researches done during this period was that the implementation methodologies used in their approaches did not take the exact optimization of designs into consideration.

Almost all research efforts in FSM state minimization begin with a state table because it offers more clarity of the relationship of external and internal states, and so far, the states can easily be translated from one table to the next. An example of such a table is shown in TABLE I. The detailed description concerning state tables is given in § 2.2.

Paul and Unger [1] and Unger [2] have developed a general theory for the minimization of incomplete machines. Their method, however, for obtaining a minimal closed covering involves a great deal of enumeration and inspection. Furthermore, their implication graph only shows the implication between compatible pairs which obscures vital information with respect to larger implied compatible present states containing more than two states.

McCluskey [3], Pager [4], and Enrich [5] have developed ingenious methods for minimizing a restricted class of incomplete FSM. Their techniques, however, are only applicable to a very special class of machines.

Grasselli and Luccio [6] have presented an interesting approach by casting the problem in the form of a linear integer program. Their minimization process which uses the prime compatible present states and a closure covering table, however, is quite complex.

Luccio [7] improved the method from [6] by extending the definition of prime compatible groups. His method, however, is quite lengthy and involves tedious procedures.

As of the 1980s, most approaches to FSM design have begun to concentrate on circuit optimization and the computer design automation. Because of the progress that has taken place in VLSI technology, research began to focus on FSM circuits having a large number of states and inputs (for example, more than 50 present states and/or inputs) as well as many don't care terms (over 50% of possible state transitions, and outputs).

A widely used program for the design of Finite State Machine is the PEG (PLA Equation Generator), implemented by Gordon Hamachi, of UC Berkeley [8]. As an FSM compiler, this program can translate the high level language description (the .peg format input file specified by a flow chart) of a FSM into format .eqn by using the Moore machine model. The FSM is represented by several logic equations in format .eqn. Unfortunately, it deals with only the internal states minimization of the completely specified machines. The result of this minimization, whatsoever, is not even minimal. Moreover, it can be used for neither the state minimization of incompletely specified machines nor for the design related optimization problem of these incompletely specified machines.

In 1985, M. Perkowski and N. Nguyen [9] developed another FSM compiler, SuperPEG: an even more powerful tool for state minimization. SuperPEG can accept Moore and Mealy machine specifications and translate one form of representation into the other. The outstanding contribution that SuperPEG made is that it is able to generate all of the compatible groups of present state rows and select a minimal closed and complete covering for present state rows from the compatible internal state groups. It employs a tree search approach with a cost function in order to cut off those tree branches which are unable to lead to a minimal closed and complete covering of present state rows. The routine MULTCOM, used in this compiler, is a problem-independent Artificial Intelligence (AI) based tree-searching program. It can be personalized for various applications. During the state minimization, MULTCOM has to be called twice. The

first calling to MULTCOM generates all compatible groups of present state rows. In the second calling, a cost function is used to decide if the search on a certain branch of solution tree should be retained or not. When a node is generated in the search process, whose value of cost function exceeds (or equals) the cost of the tentative solution, then a subtree starting from this node is cut-off and backtrack occurs. The other rules are based on the properties of compatible (for instance the necessity to fulfill completeness condition or closure condition for sets of compatible states and the general principles of cut-off sets of nodes of the tree before their creation). The last property utilizes the concept of operators. Application of an operator creates a new node of the tree. The operator anticipates certain properties of descendant node before creating this node in the tree. The tree size can be thus decreased by removing certain descendant nodes from the set of operators to apply in the parent node, before actually opening it.

Considering the advantage of MULTCOM which yields the minimal solution out of all solutions, especially when the state table involves a lot of don't care terms, the subroutine MULTCOM is also acceptable in the approach of this thesis. However, SuperPEG is not able to minimize the input columns of the state table; its solutions are actually not optimal at this point.

In this thesis, the new ideas that the columns of the state table can also be minimized and the optimal solution is always carried out by the iteration of both column and row minimization procedures are developed and presented. Usually, a number of corresponding next states of some columns of the state table are combinable, when all of these corresponding next states and outputs of these columns are consistent. This combination occurs more frequently when there are many don't care terms in internal states and outputs. As a result, the input column minimization should be considered in the entire FSM minimization process. The possibilities of input column minimization should not only be considered in the original state table, but also in the new state tables created by

each process of the present state minimizations mentioned above. This traditional present state minimization is termed state row minimization in this thesis to distinguish it from the input column minimization. After the result of column or row minimizations is created, the program tests whether further minimization on another dimension (rows or columns) is possible. If the answer is true, the result of this process becomes one minimal complete covering for input columns or one minimal closed and complete covering for present state rows, and so far, a new state table in which one dimension has been minimized is created and the next minimization process will be carried out in another dimension. Otherwise, the last created results of column and row minimizations will become the optimal solution. The optimal solution actually includes two parts: the optimal complete covering for input columns and the optimal closed and complete covering. Such an optimal solution is, therefore, obtained only after the ultimately possible minimization has been performed and the minimum state table has been created.

The detailed description of this new approach will be presented in CHAPTER II. The new FSM minimization program TDFM has been implemented in support of the approach presented in this thesis. The subroutine MULTCOM is introduced in CHAPTER IV. Even though there are other methods of performing the above minimizations, such as using the graph coloring method [12] which minimizes only the input columns, these methods discussed in the following chapters can not be applied for minimization programming because of their shortcomings. To shorten the time performance of program TDFM, MULTCOM has been adapted and used in this program for present state row minimization, input column minimization and minimal covering for binary input expressions encoding. The improvements done to the subroutine MULTCOM to achieve these goals are also presented in CHAPTER II and CHAPTER IV. In addition, several machines that have more complicated structures than the machines presented in the previous approaches have been minimized: they have, for example, more internal states, multi-bit inputs and outputs, and a higher percentage of

don't care terms. The specifications for these machines are presented in CHAPTER V.

CHAPTER II

THE PROGRAM : TWO DIMENSIONAL FSM MINIMIZER TDFM

The program, TDFM (Two Dimensional FSM Minimizer) presented in this thesis is an new FSM compiler in FORTRAN 77. It was written by W. Y. Zhao in 1987. The general data flow of this system is shown in Figure 2.

Figure 2. Data flow chart of program TDFM

The interconversion of the input data formats is presented in CHAPTER III. The .kiss or .stab data file represents either a completely or an incompletely specified FSM. The processes of both column and row minimizations are based only on the .stab formatted Mealy state table.

Two types of minimizations, the column minimization and the row minimization, are processed serially and iteratively until no more columns or rows can be minimized. Similarly to SuperPEG, TDFM utilizes the procedure MULTCOM for tree search. The difference is: MULTCOM has been upgraded so that it will not only minimize the PS rows but also minimize the input columns of the state tables and minimize the minimal covering for the binary input encoding. Since these three tasks have different requirements, TDFM can successfully deal with them respectively by using different strategies and parameters. In the following sections, the improvements (such as suiting the input data which have different styles, sizes, or formats, arranging the merger lists for the next column or row minimization, collecting the input compatible groups for the binary input expressions encoding, etc.), will be introduced. Some improvements in MULTCOM (such as the application of particular parameters, the test of the closure condition for PS row minimization, etc.), to match various tasks are also presented to explain the relationship between the main program TDFM and the subroutine MULTCOM.

§ 2.1. PRINCIPLE OF STATE TABLE MINIMIZATIONS

After a .stab formatted input data table is presented, TDFM can start the minimization procedures. As it has been mentioned, the most important improvement to TDFM compared with those previous FSM minimizing methods is that TDFM can deal with two types of the state minimization, the minimization based on the input columns and the minimization based on the PS rows. Although both of them are intended to minimize the structure of the state table, and the methodologies of these two kinds of minimization are

similar, or even the same in some points, they are quite different according to the principle of minimization. The following definitions basing on Mealy state table are necessary for the description of the appropriate problem formulation.

Definition (1):

A group of present states $\{s_i, \dots, s_j\}$ of machine M consists of a state compatible group (CGP), if and only if, under every input column x_r ($1 \leq r \leq mx$) the next states $\{s'_{i,r}, \dots, s'_{j,r}\}$ corresponding to all of the present state rows in the group $\{s_i, \dots, s_j\}$ are compatible (the corresponding next states belong to their parent present states) or are consistent (the corresponding next states of the group either have the same value or are don't-care terms) and the corresponding outputs $\{z'_{i,r}, \dots, z'_{j,r}\}$ are consistent bit by bit (every corresponding bits of corresponding outputs either have the same value or are don't-care terms).

Definition (2):

A group of input states $\{x_i, \dots, x_j\}$ of machine M consist of an input combinable group (CGI) if and only if, under every present state rows s_r ($1 \leq r \leq na$) the next states $\{s'_{r,i}, \dots, s'_{r,j}\}$ corresponding to all of the input columns in this group $\{x_i, \dots, x_j\}$ are consistent and the corresponding outputs $\{z'_{r,i}, \dots, z'_{r,j}\}$ are also consistent bit by bit.

Property (1):

The combinational input state group is also called the input compatible group and the addresses of the input states in this group are indicated by $\{i_i, \dots, i_j\}$ in this thesis.

Definition (3):

For a group of present states $\{s_i, \dots, s_j\}$ if under a certain input column x_r the next states $s'_{i,r}, \dots, s'_{j,r}$ corresponding to all of the present state rows in the present state group $\{s_i, \dots, s_j\}$ belong to another present states group $\{s_p, \dots, s_q\}$, and other next states corresponding to the same present state rows satisfy the compatible condition according to definition (1), then the group $\{s_i, \dots, s_j\}$ is considered to imply the group $\{s'_{i,r}, \dots, s'_{j,r}\}$.

If $\{s_p, \dots, s_q\}$ is compatible, $\{s_i, \dots, s_j\}$ is called the implying compatible group (ICGP).

Definition (4):

A set of compatible groups of machine M $\{S_i, \dots, S_j\}$ is overlap if the same elements appear in different groups of this set.

Definition (5):

A set of compatible groups of present states (CGP) of machine M satisfies the closure condition, if each of its implied compatible groups (ICGP) is also included in a group from this set as well.

Definition (6):

A set of compatible groups of machine M satisfies the completeness condition, if each internal state or input state is contained in at least one group of this set.

Property (2):

A compatible group includes a certain number of elements. The quality of such a CG is defined by Q , the number of its elements. A relatively maximal compatible group has the highest Q value.

Definition (7):

A covering set which satisfies the conditions of both closure and completeness is called a closed and complete covering set (CC) or a solution set.

Definition (8):

A solution set which satisfies the conditions of closure and completeness consists of the minimum number of CGs with the relatively higher Q and less overlap is a minimal solution set (MCC). Such a feasible solution set has the minimum cost value (the number of CGs in this set) defined by CF .

The complete algorithm for describing the behavior of the program TDFM is presented in Figure 3. The procedures performed priorly and posteriorly to this program are KissToStab and StabToKiss, if the input and required output files are .kiss formatted.

Figure 3. Complete algorithm flow chart of program TDFM

The detail of these translations are introduced in CHAPTER III. Here we just concentrate on the program itself. The program consists of two joined minimization procedures. Structurally, the minimizations start from the construction of the compatible groups on every two states, then every three states and so on. After all possible compatible groups are created, the next task is to generate the solution sets by those compatible groups. The feasible solution builds a new state table.

Since the optimal solution can be carried out after the iteration of two minimization procedures, it should be checked whether there exist any more compatible groups in the most recently formed state table except the first time of column minimization. This test had better be inserted between two callings to MULTCOM. If there are still some compatible internal state pairs or compatible input pairs, the program will iterate through another minimization process, as seen in step (5) of Figure 3. This checking can be simply accomplished. When the first calling to MULTCOM is accomplished, all the compatible groups are contained in the array TABIMP (it is assumed that every single present state or input must be compatible with itself. In TABIMP, these single present states or inputs are treated as the compatible groups as well). If the number of compatible groups listed in array TABIMP is the same as the number of the present states or inputs, the further minimization on this dimension is impossible because this minimization procedure does not create even one compatible group which includes at least more than one element. As the result, the iteration of both column and row minimizations should be terminated at this point because the current state table is the result of the minimization on another dimension and therefore it is obvious that the further minimization on that dimension is also impossible.

Some procedures of TDFM for the column minimization and the row minimization processes are similar. For instance, they both call MULTCOM twice. The processes of creating the compatible groups and finding the minimal closed and complete coverings are accomplished by using the same subroutine MULTCOM. The common parts designed for both the column minimization and the row minimization are: the step (1) for testing of the output compatibility, the step (4) for creating CGIs of the input columns or for creating CGPs of the PS rows by calling to MULTCOM, and later on, the step (6) for creating the MCCIs in column minimization or the MCCPs in row minimization by calling to MULTCOM again.

However, there exist also some differences between these two procedures. Particularly, the step (3) for testing the compatibility of input state columns, the step (8) for collecting the addresses of input CGIs in the array CGG, as well as the steps (10) and (11) for input encoding are designed for column minimization alone. On the other hand, the step (3)' for testing the PS compatibility, as well as the step (8)' for arranging the new state symbolic system are designed for row minimization alone.

The compatibility conditions for NSs of these two procedures are entirely different. This is because the nature of these two kinds of compatibilities are different. For the row minimization, the compatibility of the internal state rows in the state table must be tested as follows. For a compatible group of PS rows, every pair of the NSs and outputs at the corresponding intersections of the selected rows and all columns must satisfy the compatibility condition or they must satisfy the implied compatible condition. The compatibility strictly follows the definition (1) for compatible groups (CGP) and also the definition (3) for implied compatible groups (ICGP).

On the other hand, in the column minimization, for a group of input columns, every corresponding NSs and outputs which can be arranged into a compatible group at the intersections of the selected columns and all rows must have the same value or be don't-care terms. The compatibility of input columns (CGI) are decided by definition (2). The implied compatible groups do not exist in this procedure. Comparing the compatible conditions between column and row minimizations, the consistence conditions are the same for the outputs, but the compatibility conditions are different on NSs. By changing the compatible testing conditions, the same facilities, such as the merger list ZGOD and the following of MULTCOM for creating the compatible groups, can be used for both of these two minimizations for making the program shorter and more efficient. In the column minimization, the reason that the term 'compatible' is borrowed from the procedure of row minimization is that the merger list array ZGOD for creating the com-

patible pairs from the state table has been used in both row and column minimization procedures (indicated by the step (1) and the step (3) or the step (3)' of Figure 3). The detailed description of ZGOD and the different usages in these two procedures are introduced in § 2.2 and § 2.3 respectively. This method has been proven very effective, easy to be programmed and easy to be enhanced for dealing with relatively larger scale state tables.

There is also some difference in the test of the closure and completeness conditions of the solution sets during the second MULTCOM call. For instance, some implied compatible groups (ICGP) might be chosen in the solution set in row minimization. The test of closure and completeness must be carried out afterwards. In the second calling to MULTCOM for the column minimization, the solution candidate test of CGIs will not include the closure condition, since all the CGIs are naturally closed.

Even though the column and row minimization utilize the same subroutine MULTCOM, there still exist some differences within the MULTCOM. The analyses of these problems can be found in CHAPTER IV.

Speaking of the nature of these differences, the state minimization based on present state rows is a sequential logic reduction because the relationship of PSs and NSs is actually the transition between the input and output of the sequential memory components. That is, when minimizing the number of PS rows, the NSs will be taken in consideration since they may relate to some other PSs and may affect the compatibility of the PSs. Therefore, in the second calling to MULTCOM, the test of the quality of the internal state compatible groups (CGP) will include the closure, completeness and overlapping of these CGPs with the other selected CGPs. However, in the column minimization, those NSs are not sequentially related to the input constraint conditions. The execution of this minimization is absolutely a purely combinational logic reduction of the corresponding NS/output units among those columns. Strictly speaking, this is not a com-

patibility problem but rather a simplification problem of combinational logic functions. That is why, when the column minimization procedures are accomplished, an input encoder must be inserted between the primary input states and the secondary input states of the machine since the inputs are NOT reduced, but combined. An input encoding procedure must be therefore used for this purpose in MULTCOM correspondingly.

Consequently, the steps that carry out the column and row minimizations are as follows.

- (1) Since these two minimization procedures use the same subroutine MULTCOM, it is necessary to have a flag "rcf" to distinguish the processes of column and row minimizations. The routine will execute the row minimization if this flag equals '0'. Otherwise, it will execute the column minimization if flag 'rcf' equals '1'. The column minimization had better be performed first. The result of the column minimization (a new state table) will become the input data file of the next step, the row minimization. Then, after the row minimization process is executed, the result of the row minimization (another new state table) is also used as the input data file for the next column minimization. All of the state tables created at each stage are stored as the records of the entire minimization process.
- (2) Search every corresponding NS/output units under every two rows or every two columns in the state table to detect whether the two present state rows or two input state columns are compatible according to definitions (1) and (3) or definition (2). Definitions (4) and (5) correspond only to the PS row minimization. The result of this step is to collect all the compatible pairs.
- (3) Generate all the compatible groups of PSs or inputs from all single states

and the compatible pairs created in the last step. Some of these CGs may become the groups of the solution set. All the selected compatible groups of inputs are represented by their addresses shown in the original state table.

- (4) After the compatible groups have been created, check if further minimization is possible after each row or column minimization (notice that this test is not necessary for the first time of row and column minimizations). If true is returned, go to the step (6), and the next stage of iteration must be taken. Otherwise, terminate the iteration. The last created MCCI and MCCP sets are OMCCI and OMCCP (the optimal minimal complete covering for input columns or the optimal minimal closed and complete covering for PS rows).
- (5) Execute the Input Encoding procedure. Call MULTCOM to generate the prime implicants indicated by the collection of CGI in step (3), and stop the program.
- (6) Search CCIs or CCPs from CGIs or CGPs, and subsequently, choose the MCCI and the MCCP sets according to definition (8). The features of MCC set are:
 - (a) It must include the minimum number of groups in the solution set. A non-minimum solution is never acceptable.
 - (b) Each selected address of CGI or CGP must be presented in a certain group of the set (completeness). Closure and completeness are the necessary conditions of a solution. Ideally, each input or PS had better appear in only one group of the set. Such a solution set is a so called non-overlapping solution according to definition (4). For row minimization, this is usually not easy to achieve.

Build a new state table according to the MCCI or MCCP. Then, change the value of the flag 'rcf' and enter the next step of iteration of minimization i.e. go back to step (1).

In the previous approaches, a variety of methods for generating the CGs and, therefore for finding the closed and complete coverings have been investigated aside from using the subroutine MULTCOM. These approaches attempted to use different methods to achieve this goal. A commonly acknowledged method is creating the merger graph G derived from the triangular merger table.

Nripendra Biswas [10] in 1974 used this merger graph G solving the closed and complete covering of PS row minimization. Then, Masatu Yamamoto [11] in 1980 used both of the merger graph G and its complement graph $\bar{G}$ searching the MCCP in row minimization. Later on, M. Perkowski and H. Uong [12] in 1987 introduced $\bar{G}$ searching the MCCI for column minimization. Their method is to "color" the compatible input columns in the same group according to their compatibility.

Essentially, these graph methods are used to create MCC by separating graph G or its complement graph $\bar{G}$ into some constrained complete polygons for finding of the compatible groups or incompatible groups. Such polygons are the so called Maximal Cliques or Maximum Independent Sets of G or $\bar{G}$, respectively.

For the column minimization, this method is very clear and it can easily illustrate the nature of finding the complete coverings for input columns. Since every node in this graph has a certain color, it will not appear in any other group if it has been assigned a certain color. The groups finally make up the minimal complete coverings. Obviously, this complete covering is never overlapped because each node can belong to only one group. If the set has the smallest number of compatible groups and these compatible groups do not overlap, this set is actually the minimal complete coverings in column minimization process.

Basically, the graph method of finding the covering introduced in [12] is unable to prove this covering's closure condition for implied compatible groups. Fortunately, in column minimization, the 'implied' compatible group problem does not exist. According to definition (2), the conditions of column compatibility point out that the NSs at the corresponding intersections must have either the same value or at least one of them must be a don't-care term. In other words, the NSs' compatibilities are not related to any other PS groups not even to their own parent PS groups. Therefore the compatibility of input columns in column minimization actually depends only on the relationship of the corresponding NSs themselves.

Since the closure and completeness must be proven in any stage of row minimization, this method is not able to complete the row minimization without further modification. The modification for the merger graph is introduced in detail in CHAPTER IV of this thesis and Figure 17 will illustrate an example of this.

The processes of the graph coloring and MULTCOM are very different. After the merger graph method creates those CGs it generates only one solution set at each time. The CGs in the solution may not be the maximal. In other words, their quality Q (referring to the Property (2)) may be lower than some CGs which are not chosen to form this solution. Therefore the cost of the solution (referring to the Property (3)) may not be the lowest. If someone wants to compare the costs of all solutions, the process of selecting the solutions has to be applied many times until all of the solutions are created. As a contrast, MULTCOM creates all of the CGs also but then compares the numbers of elements in every CGs (In column minimization, the number of don't-care terms in every binary inputs are also be considered.) and gives the priority to the CGs which have higher Q values in the solution set in the second calling to this subroutine. Therefore, such a solution is minimal. The CGs which have lower Q values will not be used to create the solutions. In other words, these solutions are aborted before they are formed. Even though the

merger graph method can quickly create the solutions, in fact, calculating the qualities of the existing CGs and the costs of the solutions will spend much longer time than creating them. Consequently, the speed of MULTCOM is quite faster than the coloring graph method for finding the MCCs when the size of the FSM is very large and the number of solutions is increased sharply. Therefore, the subroutine MULTCOM has been applied in the program TDFM considering the application of VLSI technology in FSM design. Merger graph method, however, is still of benefit to clarify the compatibility of states or inputs as well as the relationship of the completeness to those compatible groups of the solution set. Therefore, this method is used to analyze the solution set of the machine created by the program TDFM in this thesis.

The method of creating the merger graph is easy to learn. The graph G of the column minimization has to be separated into some complete cliques to generate the compatible groups (CGI). Figure 4 is a simple example of merger graph G . The steps to draw a merger graph are:

- (1) Put all of the input or PS decimal numbers in the nodes.
- (2) Connect all of the compatible pairs of nodes.
- (3) Decide the CGIs in G , and formula (1):

$$W = \frac{n(n-1)}{2} \quad (1)$$

is used. If a group of n nodes belong to a compatible group, the number of the connections among these n nodes should be W . For instance, in Figure 4, a group $\{1,2,4,5\}$ is a good one because $n = 4$ and $W = \frac{4(4-1)}{2} = 6$.

The group $\{1,3,4\}$ is not because $n = 3$ but there are only 2 connections.

It is clear that in graph G the nodes which are connected should belong to the same group. This is shown in Figure 4 by creating a set of all the compatible groups CGI = $\{1, 2, 3, 4, 5, 6, 7, 12, 13, 14, 15, 16, 24, 25, 27, 36, 45, 47, 56, 124, 125, 136, 145,$

Figure 4. An example of merger graph G

156, 245, 247, 1245}. According to formula (1) and the graph G, one closed and complete covering can be derived from the set of CGIs. That is $MCC_1 = \{7, 36, 1245\}$. However, this set may not be a unique solution since there exist other compatible groups which can create other solutions. For instance, if group $\{1,5,6\}$ is removed from G to create the covering, the subgraph as shown in Figure 5 may result. The next compatible group is $\{2,4,7\}$, and the last group is $\{3\}$. Incidentally, $MCC_2 = \{156, 247, 3\}$ is still another solution.

Simply, the solution that includes the maximal compatible groups may be the minimal one but it is not always true. For instance, even though MCC_1 has the same cost as MCC_2 , actually, MCC_2 may be better than MCC_1 evaluated by their Q values. The detail of this discussion will be in CHAPTER IV.

As it has been known, from the set theory, if a set L is separated into two disjoint subsets G and G', the union of these two subsets must be the set L, and the subset G' is the complement of the subset G, proven by formula (2):

$$G + \bar{G} = L \quad (2)$$

Figure 5. Reduced merger graph from Figure 4

As a result, G' must be $\bar{G}$.

For achieving the same task as above, the complement graph $\bar{G}$ can also be applied. $\bar{G}$ has to be separated into some complete cliques of incompatible groups. The disconnected nodes can be colored together for this purpose. The following rules are for creating a complete covering by using the complement graph $\bar{G}$:

- (1) Put all the input or PS decimal numbers into the nodes.
- (2) Connect all the pairs of nodes which are NOT compatible.
- (3) Give any one node a certain color first, supposing that it is 'a'. Then, consider the next adjacent node. If it is connected with the first node, give this node a different color 'b'. Otherwise, color it 'a'. The other nodes should be compared with all the previous colored nodes in the same way.
- (4) The nodes which have no connection with any others will be randomly given the same color with those colored groups respectively.
- (5) Separate the entire graph into various subgraphs according to their colors. The connections between the nodes which belong to different subgraphs will no longer be retained. The number of the colors become

the number of compatible groups in a solution set. Therefore, the solution is better when fewer colors are used.

According to these rules, the number of compatible groups included in a complete covering set is the same as the number of colors used in the merger graph. The nodes which have the same color belong to the same group.

Figure 6. Complement merger graph from Figure 4

As an example, shown in the complement graph $\bar{G}$ of graph G from Figure 6, the nodes which are NOT connected have been arranged in the same group. in this example, step (1) has been executed, and step (2) starts from node '1' (this is not necessary, other nodes can also be a starting node), node '2' would have the same color as '1' but not as '3'. Step (4) is passed because there is not any node which has no connection with any other node. By the way, the same solutions can also be created by coloring the complement merger graph $\bar{G}$ as by graph G . In this way, the complement merger graph is colored as in Figure 7. The same result $MCC_1 = \{7, 36, 1245\}$ as the one from the graph G can also be created.

Figure 7. Colored complement merger graph $\bar{G}$ from Figure 4

§ 2.2. STATE MINIMIZATION BASED ON INPUTS

(COLUMN MINIMIZATION)

The content of this section is tightly related to that of the § 2.3 in some places, so that the reference of the relative topics about the state minimization based on the present states is recommended.

At first, TDFM tests the compatibility conditions. This test is somehow different in column minimization and row minimization. For the outputs $Z'_{i,j}$, the consistence condition is the same for both processes, i.e. two input columns (or PS rows) are compatible under the condition that every two corresponding outputs of two tested columns or rows must be either the same or one or both of two bits of outputs are '-'s. This rule must be applied to every corresponding bits of these two corresponding outputs. For instance, as shown in TABLE I, supposing that the input columns i_2 and i_5 are compatible, every corresponding bits of all corresponding outputs $z'_{2,j}$ and $z'_{5,j}$ ($1 \leq j \leq na$) in these two columns must be either the same or at least one of them is '-'. For an example, $z'_{2,4} = '--'$

and $z'_{5,4} = '-1'$. Obviously, the first bits of these two terms are '-' and '-'. Since the symbol '-' can represent either '1' or '0', these two '-'s can have the same value. The second bits of these two terms are '-' and '1'. Following the description for the first bits, the '-' here could have the value '1' instead of '0'. Therefore it has the same value '1' as the second bit of $z'_{5,4}$. As a result, $z'_{2,4}$ and $z'_{5,4}$ are consistent. This test should be done from $j = 1$ to $j = na$.

TABLE I
AN INITIAL STATE TABLE OF MEALY MACHINE M^0

Present States	NS/ output					
	1	2	3	4	5	6
	000	10-	-11	001	110	010
1	1/--	3/--	1/0-	3/-0	3/--	0/--
2	1/--	3/-0	0/--	1/--	0/--	4/--
3	0/--	0/--	1/-0	1/0-	3/0-	1/--
4	4/-0	3/--	1/--	3/--	3/-1	0/--

To the column minimization, the compatibility condition for NSs is: two input columns are compatible, if all of the corresponding NSs of these two columns are either the same or one or both of them are don't-care terms (indicated by '0').

For suiting the programming, a merger list ZGOD(i, j) is used to substitute the triangular merger table in the program TDFM. The triangular merger table is currently used to confirm the result of searching the compatibility of every pair of inputs or PSs. An example of merger list ZGOD shown in TABLE II is a result of the subroutine shown in Figure 8. The compatibility of every inputs pair must be tested in this subroutine. This subroutine can also be used in PS compatible pair test after a little modification. Figure 9 is an example of the corresponding triangular merger table. Both of these two tables are derived from the .stab formatted state table shown in TABLE I.

A classical merger list ZGOD is initially designed for only PS row minimization.


```

LOOP1: i from 1 to na-1.
      LOOP2: j from i to na.
            k = k+1.

            TEST: outputs compatibility.
            TEST: NSs compatibility.
            SET:  if every two corresponding NSs of two input columns are strongly
                   compatible or of two PS rows are consistent,
                   ZGOD(k,2) = 1;
            SET:  if two two PS rows are implying compatible. The implied compatible
                   NSs are put in the corresponding situations in ZGOD;
                   ZGOD(k,2) = 0;
            SET:  if any two corresponding NSs of two input columns or PS rows are
                   not compatible,
                   ZGOD(k,2) = -1;
      ENDLOOP2.
ENDLOOP1.

```

Figure 8. Algorithm of compatibility test

This list can also be used in input column minimization. It consists of three columns. The decimal numbers at the first column indicate the addresses of the input (or PS) pairs. In the second column of this list, all of the incompatible pairs are indicated by '-1', and all of the compatible pairs are '1'. All the numbers in the third column represent all NSs in the following order. Supposing that a pair of inputs are i_i and i_k , the test of compatibility of the NSs starts from $s'_{i,1}$ and $s'_{k,1}$, and then $s'_{i,2}$ and $s'_{k,2}$ and so on. In column minimization, the third column of ZGOD are all represented by '0's, no matter these pairs of NSs are consistent or not. The compatibility of every input columns of the state table have been decided by the compatibility mark '1' and incompatibility mark '-1'.

In the respective merger table the input (or PS) pairs are arranged at the left side and bottom. The intersections for the incompatible pairs are marked with 'X' and the intersections for the compatible pairs of NSs are left empty, as shown in Figure 9.

The subroutine MULTCOM will be discussed in § 4.1 and § 4.2. The first calling to it will create all the compatible groups (CG) from those single states and compatible

TABLE II
MERGER LIST ZGOD FROM STATE TABLE I
FOR COLUMN MINIMIZATION

1,2	-1	0	0	0	0	0	0	0	0
1,3	-1	0	0	0	0	0	0	0	0
1,4	-1	0	0	0	0	0	0	0	0
1,5	-1	0	0	0	0	0	0	0	0
1,6	-1	0	0	0	0	0	0	0	0
2,3	-1	0	0	0	0	0	0	0	0
2,4	-1	0	0	0	0	0	0	0	0
2,5	1	0	0	0	0	0	0	0	0
2,6	-1	0	0	0	0	0	0	0	0
3,4	-1	0	0	0	0	0	0	0	0
3,5	-1	0	0	0	0	0	0	0	0
3,6	1	0	0	0	0	0	0	0	0
4,5	-1	0	0	0	0	0	0	0	0
4,6	-1	0	0	0	0	0	0	0	0
5,6	-1	0	0	0	0	0	0	0	0

pairs of the states. The second calling to it will create the complete coverings (CC). The data of merger list ZGOD are the input of the first calling to MULTCOM. After being processed by MULTCOM, all resultant compatible groups are put into another array TABIMP. An example of TABIMP is shown in TABLE III. The compatible group set $CGI = \{1, 2, 3, 4, 5, 6, 25, 36\}$. During this call to MULTCOM, eight CGIs are generated. The largest CGIs involve two elements. In other words, through the same transition function, two input columns 2 and 5 can be joined together. Similarly, columns 3 and 6 can be joined together as another group of the set. After this job is done, MULTCOM is called again immediately.

The second calling to MULTCOM creates several CCIs for the machine of TABLE I. The only minimal one is $MCCI = \{1, 4, 25, 36\}$.

Since the merger graph method can also generate a number of CGIs and construct one solution by using these CGIs for column minimization, this method is used here to confirm the result of column minimization. This is illustrated in the complement merger

Figure 9. Triangular merger table derived from TABLE I

TABLE III

LIST OF COMPATIBLE GROUPS TABIMP FROM STATE TABLE I

OP_i	inputs
1	1
2	2
3	3
4	4
5	5
6	6
7	2 5
8	3 6

graph $\bar{G}$ of Figure 10. According to the rules of coloring a merger graph, the nodes '2' and '5' are in the same color of group 'b', and '3' and '6' in color 'c'. Therefore, the $MCCI = \{1, 4, 25, 36\}$ is given out, and the result returned by routine MULTCOM above can therefore be proven.

TDFM offers an array CGG to accumulate the addresses I of compatible input

Figure 10. Colored complement merger graph for solving TABLE I

columns. Every time the second calling to MULTCOM generates the minimal solution, these input column numbers will be rearranged into the new categories of a new CGG. Since the purpose of this accumulation is for the binary input encoding after the state table minimization is completed, the detailed description of this problem is going to be presented in § 2.4.

TABLE IV

NEW STATE TABLE M^1 CREATED FROM TABLE I
AFTER COLUMN MINIMIZATION

Present States	NS/ output units			
	1	4	2,5	3,6
	000	001	10-,110	-11,010
1	1/--	3/-0	3/--	1/0-
2	1/--	1/--	3/-0	4/--
3	0/--	1/0-	3/0-	1/-0
4	4/-0	3/--	3/-1	1/--

Different with the row minimization process, there is an interesting aspect to the new state table created in the column minimization process. Principally, both of the

NS/out units in the corresponding locations of two certain columns are not related to the input addresses. Therefore, every pair of corresponding NSs of a CGI is consistent, i.e. their decimal expressions have the same value or the NS don't-care expressions by '0', and every corresponding bits of the corresponding outputs of a CGI have also either the same symbolic expressions or don't-care expressions represented by '-'. A new symbolic system is not needed to identify the NSs and the outputs but simply the present number expressions can still be used. For instance, in TABLE I, NS/output unit $\{s'/z'\}_{2,2}^0 = 3/-0$ and $\{s'/z'\}_{5,2}^0 = 0/-$. Therefore, the NS/output units in the second column and the fifth column of the old table M^0 become one new column. It is the third column of the new table M^1 of TABLE IV, and the NS/output unit in this respective location is $\{s'/z'\}_{3,2}^1 = 3/-0$.

§ 2.3 STATE MINIMIZATION BASED ON INTERNAL STATES

(ROW MINIMIZATION)

After the column minimization, the new state table M^1 is created. Now the task is to execute the row minimization. Referring to § 2.2, some steps are the same or similar in both of these procedures, so that the description of row minimization will become somewhat simpler. Repetition of the same issues will be avoided, but some of the similar topics will still be described briefly.

The same as the process of the column minimization, the first step for row minimization is also the test of compatibility conditions. The array ZGOD from TABLE V contains all the PS pairs, derived directly from the the state table TABLE IV, the result of the column minimization. If two PS rows satisfy definition (1), that is, if each pair of the corresponding outputs $\{z'_{i,j}, z'_{k,j}\}$ of the pair of PS rows $\{s_i, s_k\}$ are consistent, i.e. $z'_{i,j} \equiv z'_{k,j}$, and every pair of the corresponding next states of these two rows $\{s'_{i,j}, s'_{k,j}\}$ is either consistent, $s'_{i,j} \equiv s'_{k,j}$, or belongs to their parent states s_i or s_k , this pair of two

rows

$\{s_i, s_k\}$ is strongly compatible (non- conditionally compatible). Such a group of PS rows is a strongly compatible group (SCGP). Comparing with the column minimization, the compatible conditions of row minimization are less strict. Otherwise, according to definition (3), if some NSs of these two rows belong to some other pairs of the strongly compatible present states, i.e. $\{s'_{i,j}, s'_{k,j}\} \subseteq \{s_p, s_q\}$, and others satisfy the requirement of strongly compatibility, the pair of these two rows $\{s_i, s_k\}$ which involves $\{s'_{i,j}, s'_{k,j}\}$ is called a implying compatible according to definition (3). Such a group of PS rows is an implying compatible group (ICGP). Implying compatible is also called weakly compatible. Another case is: if some pairs of the next states $\{s'_{i,j}, s'_{k,j}\}$ of a pair of PS $\{s_i, s_k\}$ is included in another incompatible state pair $\{s_p, s_q\}$, the pair of PSs, $\{s_i, s_k\}$ is weakly incompatible. The last case is: if some corresponding bits of the outputs $\{z'_{i,j}, z'_{k,j}\}$ of two PS rows $\{s_i, s_k\}$ are inconsistent (the corresponding bits of these two outputs are '1' and '0') the PS pair $\{s_i, s_k\}$ is strongly incompatible.

The same algorithm as in Figure 8 can be used in the compatibility test. The merger list ZGOD shown in TABLE V illustrates the result of this test. In the second column of TABLE V, the strongly compatible pairs are marked by '1'. The weakly compatible pairs are marked by '0'. Both the weakly and the strongly incompatible pairs are marked by '-1'. In the third column of ZGOD, if there exists any pair of positive numbers in a certain row of this column, they are the NSs that is implied by the PSs of the same row. Therefore, ZGOD has to check the compatibility of another PS pair which this NS pair are included. If that PS pair is not compatible, the implying compatible mark '0' of this implying PS pair should be changed to '-1'.

The implying compatible present state rows can be chosen as the CGPs of a solution in row minimization if the solution that includes such ICGPs satisfy the closure condition. The '0's in the second column of TABLE V are the implying compatible marks

for closure test and such NSs are reserved in the third column of ZGOD for the reference instead of being blocked by '0's.

Another case is that all the NS pairs are not implied by any PS pair. These NSs are not necessary to be kept in the third column, no matter whether they are compatible or not. These sort of NSs are denoted by '0's in the third column of ZGOD.

TABLE V
MERGER LIST ZGOD CREATED FROM TABLE IV
FOR ROW MINIMIZATION

1,2	0	0	0	1	3	0	0	1	4
1,3	1	0	0	0	0	0	0	0	0
1,4	1	0	0	0	0	0	0	0	0
2,3	0	0	0	0	0	0	0	1	4
2,4	-1	1	4	1	3	0	0	1	4
3,4	0	0	0	1	3	0	0	0	0

The following steps are executed for row minimization:

- (1) Call to MULTCOM for creating all the CGPs and ICGPS which are listed in TABLE VI,
- (2) Call to MULTCOM again for creating the closed and complete coverings for PS rows (CCP). The input data of MULTCOM at this time are: aside the array TABIMP, another array SMAX is set for collecting all the closure conditions. One can find the theory and the formulas for the close and complete covering in CHAPTER I, definitions (5), (6) and (7) of CHAPTER II, and also in CHAPTER IV.

Above procedure applied to TABLE IV creates two solutions $CCP_1 = \{134, 2\}$ and $CCP_2 = \{14, 23\}$ after the second calling to MULTCOM. Both of these two solutions are MCCPs. In this particular example, both the solution $MCCP_1$ and $MCCP_2$ have the same cost (the cost value of a solution depends on the number of CGPs and ICGPs

TABLE VI
LIST OF COMPATIBLE GROUPS FROM STATE TABLE IV

OP_i	PSs
1	1
2	2
3	3
4	4
5	1 2
6	1 3
7	1 4
8	2 3
9	3 4
10	1 2 3
11	1 3 4

that a solution set includes. Obviously, the less their number is, the better the solution is). In the tree search of MULTCOM, $MCCP_2$ is created later than $MCCP_1$, and there is no further solution which has lower cost and its compatible groups have higher qualities. To shorten the execution time of searching for the solutions, the routine MULTCOM is limited to generate only the last found minimal solution. Therefore, the last created solution $MCCP_2 = \{14, 23\}$ is acceptable, and the new state table M^2 with new internal states is generated.

TABLE VII
NEW STATE TABLE M^2 CREATED FROM TABLE IV
AFTER ROW MINIMIZATION

Present States	NS/	output	units	
	1	4	2,5	3,6
	000	001	10-,110	-11,010
1	1/-0	2/-0	2/-1	1/0-
2	1/--	1/0-	2/00	1/-0

The way of arranging the new PSs and NSs in a new state table is: since the new

table includes two PS rows that covers those four PS rows in the old table, the S^2 s of the new table are marked $\{1\}^2 = \{1,4\}^0$ and $\{2\}^2 = \{2,3\}^0$. Therefore, each intersection of S^2 s must belong to either $\{1\}^2$ or $\{2\}^2$. Simply, $\{1\}^2 = \{1,4\}^0$ and $\{2\}^2 = \{2,3\}^0$ are illustrated in the respective positions of the new state table M^2 (TABLE VII).

Consequently, the outputs of the new table are formed using the same method as for those in the column minimization. It is performed as follows: the compatibilities of all corresponding outputs are compared bit after bit. If an output bit is '-', it must be subject to the value of its corresponding bit of another output. In the case where all those corresponding bits are '-', they will be retained in the new table.

TABLE VIII
OPTIMAL STATE TABLE M^* AFTER ITERATION
OF MINIMIZATIONS

Present States	NS/	output	units
	4	2,5	1,3,6
	001	10-,110	000,-11,010
1	2/-0	2/-1	1/00
2	1/0-	2/00	1/-0

This result may not be the final one because in the new symbolic expressions of those PSs, NSs and outputs in state table M^2 , some columns may obtain the NS/out units which satisfy the column compatibility condition. If so, the iteration of column minimization and then row minimization is not avoidable. This table should be turned back to the column minimization procedure again, and then to the row minimization and so on. This iteration will be halted only when there is absolutely no any more possibility of minimization, and the most simplified form of FSM M^* can therefore be given out. In the example discussed here, the TABLE VIII is such an optimal machine M^* , which has three input columns and two PS rows. This state table is equivalent to the initial state table M^0 shown in TABLE I.

§ 2.4 INPUT SIGNAL COMBINATIONAL ENCODING

Following the description in § 2.2, the compatible inputs are accumulated in some groups. Functionally, these grouped input addresses indicate the input symbolic min-terms. The example of this accumulation is illustrated in TABLE IX. The inputs indicated by decimal addresses shown in the headings of TABLE I, TABLE IV and TABLE VIII are set in different categories. TABLE IX(a) of CGG lists every original inputs in each category. Every time, when a column minimization process is finished, the input addresses will be distributed in new categories in CGG according to the solution set. For instance, it is found that the binary inputs I_3^0 and I_6^0 are compatible after the first time of column minimization process, so that the binary inputs I_3^0 and I_6^0 are rearranged in a new address I_4^1 shown in TABLE IX(b). The number of the inputs in this group, $\text{CARD}(3, 6) = 2$ is placed in array CFC_4^1 . For the same reason, I_2^0 and I_5^0 are arranged in address I_3^1 . Later on, after the second time execution of column minimization, another new input system is generated and indicated by the address I^* in TABLE IX(c). The input addresses I_1^1 and I_4^1 belong to a CGI. The content of CGG_1^1 and CGG_4^1 are accumulated into another new group in address I_3^* of TABLE IX(c). This example illustrates that this accumulation process is called by the input decimal addresses I instead of the input binary variables X. Generally, after each execution of column minimization process, all the original compatible input columns should be collected into one group. When the iterative machine minimization processes are entirely finished, those groups of inputs will be dealt with in the input signal encoding procedure.

Reviewing the OMCC created by several column minimization processes, a final CGG, after two column minimization processes, is $CGG^* = \{4, 25, 136\}$.

For the combinational logic synthesis, the binary input symbolic expressions are actually the products of literals (cubes).

TABLE IX
PROCESS OF INPUT GROUPS COLLECTION IN LIST CGG

(a)			(b)			(c)		
I^0	CFC^0	CGG^0	I^1	CFC^1	CGG^1	I^*	CFC^*	CGG^*
1	1	1	1	1	1	1	1	4
2	1	2	2	1	4	2	2	2,5
3	1	3	3	2	2,5	3	3	1,3,6
4	1	5	4	2	3,6	-	-	-
5	1	5	-	-	-	-	-	-
6	1	6	-	-	-	-	-	-

$$X_1 = [0 \ 0 \ 1]$$

$$X_2 = \begin{bmatrix} 1 & 0 & - \\ 1 & 1 & 0 \end{bmatrix}$$

$$X_3 = \begin{bmatrix} 0 & 0 & 0 \\ - & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

The minterms of the corresponding 3 variables Boolean function are:

$$X_1 = [0 \ 0 \ 1]$$

$$X_2 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

$$X_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Since the primary inputs X are divided into 3 distinguishable groups in the solution set, the output of the encoder, the secondary inputs of FSM, should have at least 2 bits according to the formula (3). Supposing that B is the number of primary input groups, 'A' denotes the number of the bits of binary code.

$$A = \begin{cases} \text{fix}(\log_2 B) & \text{if } \text{MOD}(\log_2 B) = 0 \\ \text{fix}(\log_2 B) + 1 & \text{if } \text{MOD}(\log_2 B) > 0 \end{cases} \quad (3)$$

This encoding could use any kind of code. Since the binary code yields less '1's which can simplify the structure of Boolean function and the process of Boolean reduction, TDFM has chosen a ten bits of binary standard code and the code table is stored in the file fort.2. For matching the example above, the code table is shown in TABLE X.

TABLE X
CODE TABLE CREATED FROM ACCUMULATION
OF INPUT COMPATIBLE GROUPS

inputs	m	n
X_1	0	0
X_2	0	1
X_3	1	0

A variety of methods have been introduced for solving the Boolean minimization problem. In the approach of [13], the input encoding is done by using a multi-output Karnaugh map. This method is illustrated in Figure 11 for our case.

At first, shown in Figure 11(a), this map has 3 input variables. The encoded symbols 00 are used to represent the group X_1 , 01 for group X_2 , and 10 for X_3 . In every map's cell, mark in the code according to the respective input group. Then, since the codes have 2 bits, the map of Figure 11(a) can be separated into two independent sub-maps shown in (b) and (c) of Figure 11 and these two maps are simply the normal Karnaugh maps. Assuming that the secondary inputs of FSM are F_m and F_n , we have:

$$F_m^* = 0-0 + -11$$

$$F_n^* = 1-0 + 10-$$

The multi-output map method is an easy way to solve encoding problems. Unfor-

tunately, this method merely deals with only 5 or less bits of variables.

Figure 11. Multi-outputs Karnaugh map and its separated submaps

In order to manipulate the inputs encoding functions of larger number of variables, a more systematic procedure which can be carried out by a computer, is necessary. In this thesis, the bit-bound tabulation procedure that follows Quine McCluskey algorithm enhanced with the covering problem solver to create prime implicants has been successfully implemented in program TDFM. Although this is not the best way to minimize Boolean function, it can give out the optimal result for small functions since the inputs have at most ten bits handled by MULTCOM. If encoders were larger, Espresso, PALMINI or other approximate routines can be used.

Similarly to the map method, the variables are separated into two secondary input functions F_m and F_n , depending on the matching of the '1's in the code table. In the column m of TABLE X, the intersection of the third row is '1', so all the minterms of X_3 are loaded in the Boolean function:

$$F_m = 000 + -11 + 010$$

Similar to F_m , for the column n , only the intersection of the second row is 1. Therefore, the function is:

$$F_n = 10- + 110$$

If there were more groups, for instance, X_4 was a member of inputs, one more code 11 would be appended in the code table. All the variables in X_4 would also be used since the column m of this code is also '1'.

The tabulation procedure strictly follows these steps:

- (1) Arrange all minterms in groups, so that all terms which have the same number of '1's are in the same group. Start with the least number of '1's and continue with groups of increasing numbers of '1's. The number of '1's in a term is called the index of that term.
- (2) Compare every term of the lowest index group with each term in the successive group; whenever possible, combine the two terms being compared by means of the combining theorem $Aa + A\bar{a} = A$. Repeat this by comparing each term in a group of index i with every term in the group of index $i+1$, until all possible applications of the combining theorem have been exhausted. Two terms from adjacent groups are combinable if their binary representation differ by just a single digit in the same position; the combined term consists of the original fixed representation, with the different digit replaced by '-'.
- (3) The terms generated in the last step are now compared in the same fashion: a new term is generated by combining two terms which differ by only one '1' and whose dashes are in the same position. The process continues until no further combination is possible. The remaining unchecked terms constitute the set of prime implicants of the function F .

TABLE XI illustrates the process of the tabulation procedure for only the function F_m of the above example. All the selected variables are put in an array DIG shown in TABLE XI(a). If there exist some implicants, such as '-11', they must be decomposed into minterms '011' and '111', as shown in TABLE XI(b). These minterms are rearranged in the order of different groups according to the number of '1's, as shown in TABLE XI(c). The reduced terms, after the first application of step (c) are given in TABLE XI(d). For example, the combination of minterms '000' and '010' is '0-0'. The same rule can also be applied on TABLE XI(d), if there exist any more combinable variables.

TABLE XI
PROCESS OF TABULATION FOR ENCODING
OUTPUT FUNCTION F_m

(a)	(b)	(c)	(d)
1 0 0 0	1 0 0 0	1 0 0 0	1,3 0 - 0
2,4 - 1 1	2 0 1 1	3 0 1 0	3,2 0 1 -
3 0 1 0	3 0 1 0	2 0 1 1	2,4 - 1 1
	4 1 1 1	4 1 1 1	

As the result of above tabulation procedure, the secondary input bit F_m of the prime implicants is:

$$F_m = 0-0 + 01- + -11$$

Comparing with the F_m^* found by Karnaugh map, this Boolean function of all the prime implicants is not minimal. Therefore, the covering minimization procedure must be applied. The routine MULTCOM can also be used for this task after specifying some parameters. The problem of solving the covering table of Figure 12(a) is a pure combinational logic reduction. As the input data of MULTCOM, the implicants of the covering table can be easily transferred to the form of the array TABIMP shown in TABLE XII. The decimal numbers in the second column of this table are the descriptors (or addresses)

	1	2	3	4		1	2	3	4
0-0(1, 3)	X		X		0-0(1, 3)	X		X	
01-(3, 2)		X	X		-11(2, 4)		X		X
-11(2, 4)		X		X					
	(a)					(b)			

Figure 12. State space for tree search of function F_m and its minimized form

of the prime implicants of the primary inputs according to the decimal numbers of the TABLE XI(d) and the numbers of the first column are the operators of them. These prime implicants are like the compatible groups in the state minimization procedure. Therefore, the subroutine MULTCOM can also be employed to select a MCC among them referring to the second calling to this subroutine in state minimization. Since this is a combinational logic design problem essentially, it is similar to the problem of the second calling to MULTCOM in the column minimization.

TABLE XII

LIST OF COMPATIBLE GROUPS FROM STATE SPACE OF COVERING

OP_i	MINTERMS (addresses)
1	1 3
2	2 3
3	2 4

After call to MULTCOM, the simplest form of the minimal expressions of prime implicants for inputs encoding becomes available. The only solution for this particular case is $MCC = \{13, 24\}$. This solution is actually the same as the one which is obtained by using the map method, $F_m = 0-0 + -11$. The solution F_m is illustrated in the final minimal covering table in Figure 12(b). Obviously, this table is the optimal solution equivalent to the initial covering table shown in Figure 12(a).

Some minimizing possibilities could also result from the state assignment process to the minimized state table and .kiss formatted data file, but these problems do not belong to the topic of this thesis.

CHAPTER III

FORMATS OF FINITE STATE MACHINES

The program TDFM can accept two input data formats, .kiss (Keep Internal State Simple) and .stab (State TABLE). Both the .kiss formatted and .stab formatted data are converted from the standard register-transfer flow chart in format .liss. Presented in Figure 2, two possible conversions that precede the two-dimensional minimization procedure are the machine translation from .liss or .kiss into .stab formatted Mealy machine.

The interconversion of the .kiss and .stab formats is a part of the program TDFM. The .kiss or .stab data file represents either a completely or an incompletely specified FSM. The processes of both column and row minimizations are based only on the .stab formatted Mealy state table. The .kiss formatted file has to be converted into .stab format before the two-dimensional minimizations (row minimization and column minimization). On the other hand, the optimal result can also be represented in .kiss or .stab format for the convenience of next design steps. For instance, some assignment programs, such as KISS, that was implemented by De Micheli [13] in 1985, prefer to receive the .kiss formatted data as the input file. Some other programs, such as the programs of CDA system DIADES, use the .stab formatted file for machine type conversion and assignment. Although the flow chart conversion process from .liss to .kiss or to .stab is not included in TDFM, this process is also presented when explaining the forming of a state table. An example of the flow chart .liss format is shown in Figure 14. This format is quite similar to the .peg format by [8].

§ 3.1. TRANSLATION OF INPUT FILE FORMATS FROM .KISS TO .STAB

Simply speaking, .kiss format assembles all 4-tuples of parameters into a list. The parameters in each column are the groups of the external states: the inputs and the outputs, and the groups of the internal states: the present states and the next states. The following notations are presented for .kiss format.

Inputs:

$$X = \{x_1, \dots, x_{mx}\} \quad (4)$$

Present states:

$$S = \{s_1, \dots, s_{na}\} \quad (5)$$

Next states:

$$S' = \{s'_{1,1}, \dots, s'_{na,mx}\} \quad (6)$$

Outputs:

$$Z' = \{z'_{1,1}, \dots, z'_{na,mx}\} \quad (7)$$

Although there exist different types of .kiss format, in this thesis, it is not necessary to discuss too much about this topic. One point has to be mentioned, however, that the reason of using binary symbolic representations for inputs and outputs in the applied .kiss formatted input file in this thesis is because that the De Micheli's encoding method for testing of the consistence of the outputs and performing the input combinational logic encoding has been partly employed as discussed in § 2.4.

A standard .kiss formatted data file is shown in TABLE XIII. For suiting the usage of this format, the input states are represented in their binary expressions x_i shown in the first column. The decimal numbers of the second column are internal states and numbers in the third column are the next state symbols. The binary expressions in the

rightmost column are outputs. An example of a .kiss formatted data list is shown in TABLE XIII.

TABLE XIII
STANDARD .KISS FORMATTED FILE

inputs	PSs	NSs	outputs
101-	1	2	000
1001	1	2	000
1000	1	2	000
011-	1	2	000
0101	1	2	000
0100	1	2	000
101-	2	3	001
111-	2	3	001
0001	2	3	001
0000	2	3	001
0101	2	3	001
0100	2	3	001
101-	3	4	010
1001	3	4	010
1000	3	4	010
001-	3	5	010
0001	3	5	010
0000	3	5	010
111-	4	4	---
1101	4	4	---
1100	4	4	---
1001	5	6	100
1000	5	6	100
1101	5	4	011
1100	5	4	011
001-	5	6	100
011-	5	4	011
101-	6	4	101
1001	6	6	101
1000	6	1	11-
001-	6	4	101
0001	6	6	101
0000	6	1	11-

When this program is started to run, the first thing that TDFM does is to execute the translation subroutine KissToStab if the .kiss formatted input file is applied. The subroutine KissToStab consists of two procedures, the input addressing and the translation of

formats.

In the standard .kiss formatted file, there is no corresponding decimal addresses I for inputs. The program has to arrange the decimal addresses for the input binary expressions when this .kiss file is read in. This procedure is called the input addressing. Since the purpose of this addressing for the .kiss formatted file is to arrange the inputs in the numbers at the headings of state table, the method applied in this process is:

- (1) Enumerate the first term of input with integer 1.
- (2) Increase the number when the present binary expression is different with the expressions already checked above. Otherwise use the same number as the one which has been assigned to this term. The algorithm of this arrangement is presented in Figure 13.

TABLE XIV is the result of applying this subroutine. In this input file, the modified .kiss formatted input file, the decimal numbers have been marked from 1 to 12 inside the parentheses.

If the next state and all bits of output of a NS/output unit are don't-care terms, such a type of don't-care unit is called complete don't-care unit. The complete don't-care units are not presented in .kiss formatted input file. When the .kiss format file is translated into .stab format file, in the corresponding intersections of .stab formatted input file, this type of NS/output units have to be marked by '0' for NS and all '-'s for output. Otherwise, if NS or some bits of the output (not all) of a NS/output unit are don't-cares, it is called incomplete don't-care unit.

There are two reasons why the decimal numbers of the PSs and inputs should start from 1. First, in the incompletely specified FSM, more or less don't-care terms in both of the NSs and outputs are possible. These don't-care terms may be either the complete don't-care units or the incomplete don't-care units. The complete don't-care units do not appear in the .kiss formatted file. During the translation, the complete don't-care

Figure 13. Algorithm of decimal numbers setting for inputs

NS/output units must be specified in the intersections of the new created .stab format. In the input files which TDFM accepts, zero has been defined as the symbol to express the don't-care terms for NSs. Secondly, the decimal numbers of PSs and inputs of .kiss format actually indicate the location of the NSs and outputs in the intersections of corresponding .stab format. Therefore, zero should not be used to represent either the number of PSs or the number of NSs in .stab formatted input file. Meanwhile, as a rule in binary addressing procedure, zero should not be used to address any binary expression.

For translating the data from .kiss format to .stab format, an integer array $AUT(i, j)$ is used to load all of the NSs according to their numbered locations which are indicated

TABLE XIV
MODIFIED .KISS FORMATTED INPUT FILE
FROM TABLE XIII

inputs	PSs	NSs	outputs
101-(1)	1	2	000
1001(2)	1	2	000
1000(3)	1	2	000
011-(4)	1	2	000
0101(5)	1	2	000
0100(6)	1	2	000
101-(1)	2	3	001
111-(7)	2	3	001
0001(8)	2	3	001
0000(9)	2	3	001
0101(5)	2	3	001
0100(6)	2	3	001
101-(1)	3	4	010
1001(2)	3	4	010
1000(3)	3	4	010
001-(10)	3	5	010
0001(8)	3	5	010
0000(9)	3	5	010
111-(7)	4	4	---
1101(11)	4	4	---
1100(12)	4	4	---
1001(2)	5	6	100
1000(3)	5	6	100
1101(11)	5	4	011
1100(12)	5	4	011
001-(10)	5	6	100
011-(4)	5	4	011
101-(1)	6	4	101
1001(2)	6	6	101
1000(3)	6	1	11-
001-(10)	6	4	101
0001(8)	6	6	101
0000(9)	6	1	11-

by the two columns of decimal input and PS expressions of the .kiss formatted file. Another character array, AUTT(i, j, k) is used to load all of the corresponding outputs. In the absence of both NS and output on a certain intersection, the subroutine KissToStab will put a zero in that empty position in AUT and '-' in AUTT, to indicate that there is a

complete don't-care term in that location.

An example of translating the NSs and outputs from the .kiss formatted input file into the corresponding .stab formatted file can be illustrated at row 23 of TABLE XIV. The input number is 3 and the PS's number is 5, so that the NS $s'_{3,5} = 6$ and output $z'_{3,5} = 100$. These two terms consist of a NS/output unit in the corresponding location of the .stab format file shown in TABLE XVI.

The .stab formatted file is a two dimensional matrix and the .kiss formatted file is a list with 4 columns. The decimal numbers marked for inputs of the .kiss formatted file form the numbers of columns in the corresponding .stab formatted file. Meanwhile, the decimal numbers of PSs form the numbers of rows. Therefore, TABLE XVI has 12 columns and 6 rows since the maximal number of addressed inputs is $mx = 12$ and the maximal number of PSs is $na = 6$. According to 'na' and 'mx', the size of .stab input file can be therefore decided.

§ 3.2. TRANSLATION OF INPUT FILE FORMATS FROM .LISS TO .STAB

According to Figure 2, another possible stage that precedes the state minimization procedure is the translation from a high level language description of FSM or a state flow chart (in format .liss) into a .stab formatted Mealy state table directly. An example of such a flow chart is shown in Figure 14. The description of the translation of this example will illustrate how a state table of a FSM is formed in this way. In addition, the following modification of this state table creates a number of complete don't-care terms by using the invariants. These don't-care terms consequently make more columns and rows of the modified state table compatible. Referring this example, the data notation are as follows.

- (1) There are six present states in this flow chart. The symbols s_1 to s_6 in the graph nodes represent these PSs. Therefore, they can be put in the list of PSs in the state table immediately.

- (2) The arrows that connect the states are marked by the parameters with dual elements. The first element expresses the input vectors. All the letters without bars are expressed as '1's and the letters with bars as '0's. The second elements in these parameters describe the outputs. These elements have already been encoded in binary expressions.

Figure 14. Flow chart of finite state machine M

The input branches from the source node to the target node are:

$$F(s_1) = \{-\}$$

$$F(s_2) = \{-\}$$

$$F(s_3) = \{a, \bar{a}\}$$

$$F(s_4) = \{-\}$$

$$F(s_5) = \{b, \bar{b}\}$$

and

$$F(s_6) = \{c, \bar{c}d, \bar{c}\bar{d}\}$$

The input vector transitive function set of machine's product terms is:

$$T(s) = F(s_3) \& F(s_5) \& F(s_6)$$

$$= \{a, \bar{a}\} \& \{b, \bar{b}\} \& \{c, \bar{c}d, \bar{c}\bar{d}\}$$

$$= \{\bar{a}\bar{b}c, \bar{a}\bar{b}\bar{c}d, \bar{a}\bar{b}\bar{c}\bar{d}, \bar{a}bc, \bar{a}b\bar{c}d, \bar{a}b\bar{c}\bar{d}, \bar{a}\bar{b}c, \bar{a}\bar{b}\bar{c}d, \bar{a}\bar{b}\bar{c}\bar{d}, \bar{a}bc, \bar{a}b\bar{c}d, \bar{a}b\bar{c}\bar{d}\}$$

and,

$$CARD(T(s)) = 12.$$

The state table presented in TABLE XV is created from the flow chart. Each product term represents a binary input state x_i . Meanwhile, its address is presented as a decimal number i_i . For the performance of input column minimization, the letters describing the product terms of the transitive function $T(S)$ are translated into their decimal addresses i_1 to i_{12} shown in the heading of the state table. Since $CARD(T(s))$ is 12, and this heading has twelve input columns. s_1 to s_6 form six present state rows shown in the leftmost column of this state table. The NSs in the intersections are arranged according to the states in the target nodes and the outputs according to the second elements near the arrows from a source node to a target node in Figure 14.

By taking some constraint conditions for machine M (so called assertions or invariants) into consideration, this state table can be modified. Let us assume that the invariants determined by the designer are:

- for present state s_1 : $a \oplus b$
- for present state s_2 : $a = c$
- for present state s_3 : $b = 0$

TABLE XV
AN EXAMPLE OF STATE TABLE CREATED FROM
FLOW CHART IN FIGURE 14

Present States	NS/ output units											
	1	2	3	4	5	6	7	8	9	10	11	12
	101-	1001	1000	011-	0101	0100	111-	0001	0000	001-	1101	1100
1	2/000	2/000	2/000	2/000	2/000	2/000	2/000	2/000	2/000	2/000	2/000	2/000
2	3/001	3/001	3/001	3/001	3/001	3/001	3/001	3/001	3/001	3/001	3/001	3/001
3	4/010	4/010	4/010	5/010	5/010	5/010	4/010	5/010	5/010	5/010	4/010	4/010
4	4/---	4/---	4/---	4/---	4/---	4/---	4/---	4/---	4/---	4/---	4/---	4/---
5	6/100	6/100	6/100	4/011	4/011	4/011	/011	6/100	6/100	6/100	4/011	4/011
6	4/101	6/101	1/11-	4/101	6/101	1/110	4/101	6/101	1/110	4/101	6/101	1/11-

- for present state s_4 : $a = b = 1$
- for present state s_5 : $a \oplus b$
- for present state s_6 : $b = 0$

The state table TABLE XV can be modified into TABLE XVI after applying these invariants on every NS/output units. This modification means that the NS/output units which satisfy those invariant conditions for each column will be bound to their own values and others will be replaced with the don't-care terms. For instance, with reference to the decimal input addresses in the headings of TABLE XV, if the input product terms satisfy the invariant for s_1 , the values of NS/output units at the intersections of the corresponding columns and the first row will be retained. This is the case at the intersections of input columns $i_1, i_2, i_3, i_4, i_5, i_6$ and the present state row s_1 . Otherwise, they would be replaced with don't-care terms, such as the NS/output units at the intersections of input columns $i_7, i_8, i_9, i_{10}, i_{11}, i_{12}$ and the present state row s_1 .

The transitive function set $T(s)$ for the corresponding PSs can be specified for every PS rows as follows:

$$T(s_1) = \{\bar{a}\bar{b}c, \bar{a}\bar{b}\bar{c}d, \bar{a}\bar{b}c\bar{d}, \bar{a}\bar{b}c, \bar{a}\bar{b}\bar{c}d, \bar{a}\bar{b}c\bar{d}\}$$

$$T(s_2) = \{\bar{a}\bar{b}c, abc, \bar{a}\bar{b}\bar{c}\bar{d}, \bar{a}\bar{b}c\bar{d}, \bar{a}\bar{b}\bar{c}d, \bar{a}\bar{b}c\bar{d}\}$$

$$T(s_3) = \{\bar{a}\bar{b}c, \bar{a}\bar{b}c\bar{d}, \bar{a}\bar{b}\bar{c}\bar{d}, \bar{a}\bar{b}c, \bar{a}\bar{b}c\bar{d}, \bar{a}\bar{b}\bar{c}\bar{d}\}$$

$$T(s_4) = \{abc, \bar{a}\bar{b}c\bar{d}, \bar{a}\bar{b}\bar{c}\bar{d}\}$$

$$T(s_5) = \{\bar{a}\bar{b}\bar{c}\bar{d}, \bar{a}\bar{b}c\bar{d}, \bar{a}\bar{b}c\bar{d}, \bar{a}\bar{b}\bar{c}\bar{d}, \bar{a}\bar{b}c, \bar{a}\bar{b}c\}$$

$$T(s_6) = \{\bar{a}\bar{b}c, \bar{a}\bar{b}c\bar{d}, \bar{a}\bar{b}\bar{c}\bar{d}, \bar{a}\bar{b}c, \bar{a}\bar{b}c\bar{d}, \bar{a}\bar{b}\bar{c}\bar{d}\}$$

TABLE XVI

STATE TABLE MODIFIED FROM TABLE XV

Present States	NS/ output units											
	1	2	3	4	5	6	7	8	9	10	11	12
	101-	1001	1000	011-	0101	0100	111-	0001	0000	001-	1101	1100
1	2/000	2/000	2/000	2/000	2/000	2/000	0/---	0/---	0/---	0/---	0/---	0/---
2	3/001	0/---	0/---	0/---	3/001	3/001	3/001	3/001	3/001	0/---	0/---	0/---
3	4/010	4/010	4/010	0/---	0/---	0/---	0/---	5/010	5/010	5/010	0/---	0/---
4	0/---	0/---	0/---	0/---	0/---	0/---	4/---	0/---	0/---	0/---	4/---	4/---
5	0/---	6/100	6/100	4/011	0/---	0/---	0/---	0/---	0/---	6/100	4/011	4/011
6	4/101	6/101	1/11-	0/---	0/---	0/---	0/---	6/101	1/11-	4/101	0/---	0/---

A Mealy state table of a Finite State Machine M is defined as an ordered 5-tuples:

$$M = \{X, Z, S, \delta, \lambda\} \quad (8)$$

Where X is a set of all binary input states x_i .

$$X = \{x_1, \dots, x_{mx}\} \quad \text{CARD}(x_1, \dots, x_{mx}) = mx \quad (9)$$

S is a set of all present states s_i .

$$S = \{s_1, \dots, s_{na}\} \quad \text{CARD}(s_1, \dots, s_{na}) = na \quad (10)$$

Z is a set of all output states z_i .

$$Z = \{z_1, \dots, z_k\} \quad \text{CARD}(z_1, \dots, z_k) = k \quad (11)$$

X , S and Z are the basic sets of states of machine M . These three sets of states determine the size of the machine. δ is a two dimensional matrix used to map the successors of PSs, the next states s' related to the inputs. It is the so called next state transition function $X * S \rightarrow S'$.

$$S' = \delta(x_i, s_j) = \{s'_{i,j}\} \quad 1 \leq i \leq mx, \quad 1 \leq j \leq na \quad \text{CARD}(s'_{i,j}) = mx * na \quad (12)$$

λ is another two dimensional matrix used to map the outputs related to the present states and input states. It is the so called output function $X * S \rightarrow Z'$.

$$Z' = \lambda(x_i, s_j) = \{z'_{i,j}\} \quad 1 \leq i \leq mx, \quad 1 \leq j \leq na \quad \text{CARD}(z'_{i,j}) = mx * na \quad (13)$$

According to the structure of Mealy machine M , the PSs and NSs are internal states and the inputs and outputs are external states. In the form of the Mealy state table, these 5-tuple elements are arranged in their individual locations. The PSs s_j ($1 \leq j \leq na$) are in the leftmost column. The inputs x_i ($1 \leq i \leq mx$) are in the heading. Each unit of NS/output $\{s'/z'\}_{i,j}$ is located in an intersection of the table. Their addresses j and i are indicated by PSs s_j and input decimal expressions i_i . Obviously, 'na', the number of PSs also indicates the number of the NS/output units under each input column. So does 'mx', the number of input states also indicates the number of the NS/output units under each PS row.

The state table input format is the most popular format for representing FSMs, because it clearly gives out the relationship of the inputs (represented in the columns of the state table array) and the internal present and next states (in the rows). State table is presented in Mealy or Moore model. In this program, the Mealy machine is recommended for all minimization procedures. All the definitions presented in § 2.1 are also based on the state table of Mealy model because it permits to generate the compatible groups column by column or row by row. The format for representing state tables is

called the format .stab. An example of state table of a Mealy machine M is shown in TABLE XVI. It has twelve columns, which mean the machine has twelve distinguishable input states and six rows which mean six internal present states $s_1, \dots, s_6$. The internal present states shown in the leftmost column of the state table do not need binary symbolic expressions because there is no encoding problem at this stage. Each time when the row minimization is accomplished, a new state numbering system of the state table will be used to replace the old one. Meanwhile, for performing the state minimization based on input columns introduced in § 2.2, they are represented by decimal numbers shown in the input columns at the heading of state table as well as the addresses $i_1, \dots, i_{12}$. For performing the input combinational logic encoding, inputs are marked by binary symbols $x_1, \dots, x_{12}$.

TDFM handles multi-bit input and output machines, so that the outputs are also represented in binary symbols. This is not only for the performance of testing the output consistence in both column and row minimizations but also for the performance of symbolic FSM assignment.

Program TDFM can deal with either completely specified sequential machines or incompletely specified sequential machines. This means it is possible that some binary representations or some bits of these binary representations have 'uncertain' values in the inputs and outputs. In other words, the don't-care terms mean that these bits of the binary representation may have double values '1' and '0'. In the program TDFM, character '-' can represent the don't-care term for both inputs and outputs and binary number '1' or '0' will represent one certain value. To the next states, '0' will represent the don't-care terms and the positive integers will represent the states that have certain specified values.

§ 3.3 SPECIFICATION OF INPUT FILES

The TDFM programming utilizes a group of arrays. Some arrays are quite huge, especially those arrays that have two or three dimensions. Meanwhile, the computers can offer different sizes of memory spaces for running this program. As a result, the size of the FSM must be limited. Running on VAX 11 system, the memory space is limited to 2.5mb. The arrays in TDFM for the inputs and present states are both limited to at most fifty distinguishable expressions if the percentage of don't-care terms is not higher than about 30% by estimation in both of NSs and outputs. These limits are based on the experiment of machine M_b shown in TABLE XIX. Obviously, the present states are represented from 1 to 50 in order (note this: there is no '0'). As the successors of PSs, the next states (NS) are limited in the same domain as PSs. The same limits of fifty different expressions for binary inputs and outputs are also mentioned. Since both of inputs and outputs are represented by binary symbols, according to the formula (14), 'B' is the number of binary bits and 'A' will be the numbers of different expressions.

$$B = \begin{cases} \lceil \log_2 A \rceil & \text{if } MOD(\log_2 A) = 0 \\ \lceil \log_2 A \rceil + 1 & \text{if } MOD(\log_2 A) > 0 \end{cases} \quad (14)$$

Assuming that $A = 50$ and there is no don't-care expression '-' in any bit among those inputs or outputs, at most $B = 6$ bits of binary expressions are permissible.

The binary expressions which have more than six bits are still possible if there exist some '-'s in some bits of inputs and outputs, but the total number of them should not be more than fifty distinguishable expressions. Concentrating on the binary inputs, if there are some don't-care bits in some binary expressions, any two inputs should not be contrary. For instance, the input of the fourth column of TABLE XVI, 011- is actually the combination of the terms 0110 and 0111. Therefore, if this combined term exists in the list, it is illegal to have its implied term included in anywhere else of the input list.

The TDFM program can receive both the .kiss and the .stab input formats. It is, however, not permissible to change anything in the source file of TDFM system since the program TDFM is written in FORTRAN. Usually, the FSMs which users want to minimize may have different formats or sizes. Moreover, different strategies for the application of subroutines of TDFM are possible in practice. In TDFM system, a group of data files 'fort' are offered to suit different cases. Fort.1 is for input data specification. For instance, if the inputs or outputs of several machines have different sizes or different bits, users can simply change some parameters in these fort files for TDFM's process.

In the example of fort.1 shown in TABLE XVII, the input file has been .kiss formatted. The character following the term 'formt' denotes the format. The term 'formt' is modified for dealing with a .kiss formatted input file if the following character is 'k'. The numbers following the terms 'inbit' (means the bit of input) and 'oubit' (means the bits of output) must be settled up for .kiss formatted file. The first and the last numbers of the characters from 'forma' to 'formc' are required to be marked directly corresponding to the 'inbit' and 'oubit'. For example, the 'inbit' of this table is 4 and the 'oubit' is 3 for the .kiss formatted file TABLE XIV. The numbers following the terms 'innum' and 'psnum' should be ignored.

To the .stab formatted input file, the file fort.1 can also be modified to deal with a .stab formatted input file if the character following the term 'formt' is 's'. The numbers following the terms 'oubit', 'innum' (means the number of inputs) and 'psnum' (means the number of present states) must also be settled up according to the size of the state table. The characters from 'formd' to 'forme' are required to be marked directly corresponding to the 'oubit', 'innum' and 'psnum'. Other terms could be ignored in this setting. The example of this specification for .stab formatted file TABLE XVI is: 'oubit' equals '3', 'innum' is '12' and 'psnum' is '6'.

As an extra feature, TDFM gives out the percentage of don't-care terms of .stab

TABLE XVII
DATA FILE 'FORT.1' FOR INPUT MODIFICATION

inbit	03
oubit	03
innum	06
psnum	04
formt	k
forma	(3a1,3x,i3,3x,i3,1x,3a1)
formb	(3a1,1h(i3,1h),3x,i3,3x,3a1)
formc	(1a1,1x,3h = ,3a1,12(1h+,3a1))
formd	(4(i2,1h/,3a1,1h))
forme	(20/4(i3,1h/3a1,1h))

formatted file basing on the products of the number of PSs (na) and the number of inputs (mx). The formula for calculation of this percentage is:

$$DC = \frac{\# \text{ of } \textit{don't care}}{mx * na} * 100\% \quad (15)$$

TABLE XVIII
DON'T CARE TERMS OF .STAB FORMATTED DATA IN PERCENTAGE

Don't care in states: 54.2%
Don't care in outputs: 58.3%
Don't care in bit 1 of outputs: 58.3%
Don't care in bit 2 of outputs: 58.3%
Don't care in bit 3 of outputs: 61.1%

In the example of TABLE XVI, $mx * na = 6 * 12 = 72$. There are 39 zeros out of those NSs in the table so that the percentage of don't-care terms in the next states is: $39 / (12 * 6) * 100\% = 54.2\%$. The percentage of output don't-care terms can be calculated

as well. The number of the outputs which have '-'s on all of 3 bits are 42. As a result, the percentage of output don't-care terms is $42/(12*6) * 100\% = 58.3\%$. The detailed information of output don't-care percentage are also reported for each output bit separately. All the don't-care terms in bits 1 and 2 of the outputs are 42 and the don't-care terms for only bit 3 are 44 so that these percentages are $42/(12*6) * 100\% = 58.3\%$ and $44/(12*6) * 100\% = 61.1\%$ respectively. In TDFM, KissToStab will calculate these percentages immediately after the input file is read in. The example of this process printout is shown in TABLE XVIII.

CHAPTER IV

DESCRIPTION OF ROUTINE MULTCOM

The routine MULTCOM, first introduced by M. Perkowski in 1977 [14] is based on the theory of Artificial Intelligence. It is used for solving problems which have many solution sets, and among them, one or more minimal solutions exist. MULTCOM performs a tree search synthesis to arrange the groups of the elements into sets (or the elements into groups) according to the pre-decided conditions and the priority of these groups (or elements). These conditions are variant in different tasks. For instance, in state table minimization problems, this subroutine is applied twice. The condition of the first application is that all the groups must be compatible. The conditions of the second application are that the compatible groups in a solution set must be closed and complete. The quality of a compatible group is decided by the number of elements that this group has covered and also, particularly, the degree of implying internal states of the CGP in PS row minimization or the number of the binary don't-care terms of the CGI in input column minimization. The set must be minimal under the necessary conditions of the closure and completeness. Such a solution set has a minimum cost. A set which has the minimum cost and all of its CGs have the higher qualities will be the minimal solution.

§ 4.1. SELECTION OF STRATEGIES, PARAMETERS AND SIZE LIMITATION OF TREE

The subroutine MULTCOM's experience comes from the learning and anticipation abilities. Learning means that every time the MULTCOM creates a new node, the routine will compare this node with all its parent nodes. If this node has a higher quality value, it will be kept for the further extension; otherwise, it will not be kept. Anticipation

means that every time before MULTCOM creates a new node, it will look ahead to determine if this new created node will lead to a minimal solution or not. If it does, this node will be created; otherwise, this creation will be aborted. Generally, these learning and anticipation abilities are based on the calculation and comparison of the quality functions of the current applied node and its parent nodes. A number of heuristic, in other words, trial-and-error processes is actually unavoidable; and so far, for speeding up the algorithm, the routine offers a number of search strategies for different tasks:

- (1) **Breadth First:** with this strategy, each newly created node is appended to the end of the 'open-list' which contains all the nodes waiting for the extension. Each time the first node in the list is selected for this extension, it is removed from this list. After all the available operators for this node have been applied, the extension backtracks to another branch in the open-list.
- (2) **Depth First:** this is a strategy to extend the most recently generated node first. Until it reaches a certain specified depth limit (defined by users), the search backtracks to extend another node which is the deepest but does not exceed this depth limit. The newly created node is put at the beginning of the list so that the first node in this list is always the deepest one.
- (3) **Branch and Bound:** with this strategy, a cost function $CF(N)$ calculated by routine $FOC(N)$ is defined for each candidate node, where N is the node's number of a solution set. A desired cost value F has been set before the tree searching started and it holds the minimum cost of the solution already found. Whenever a new node is generated, the cost value of the solution set is compared with the value F . All the solutions whose costs are larger than F will be cut-off from the tree.
- (4) **Ordering:** This strategy can be selected in cooperation with the branch and

bound strategy. For this strategy, a quality function $Q(op)$ is defined to evaluate the cost function $CF(N)$ for all operators (op) of the node under extension. The order of application of these operators is in accordance with their evaluated qualities. Actually, the node which has the highest value Q will be arranged to have the priority to be chosen for the present extension.

Meanwhile, the successive application of this subroutine also depends upon the variant properties of a particular problem. For instance, the satisfaction of the completeness condition and the closure condition for the set of compatible states must be tested in some cases. The time of generating solutions depends on the selection of the strategies because the behavior of the algorithm is strongly parameter-dependent. For instance, the Breadth First strategy can quickly create all solutions without checking their cost values and qualities of the compatible groups (represented by their operators). In contrast, the Branch and Bound strategy cooperating with Ordering will create the first minimal solution without searching all nodes. Generally, creating a solution needs much less time than proving its feasibility. Before the routine is called, the suitable setting of different strategies for different tasks is proposed.

The parameters for MULTCOM are:

- (1) GENAS: This parameter causes the calling of local search procedure GEN for the indispensable nodes. Lack of it causes only the global search.
- (2) PRINT: This parameter causes the printing of descriptions of all states during the search.
- (3) EQUI: This parameter causes the calling to subroutine EQUI for discovering the 'included' compatible groups of a solution set. It is used only for finding the MCC sets.

- (4) **ORD**: This parameter causes the calling to subroutine **ORD** for the Ordering search strategy.
- (5) **MUST**: This parameter causes the calling to subroutine **MUST0** for finding all indispensable operators for nodes extension in the tree.
- (6) **SPRAW**: This parameter causes that each new node of the tree (also the solution) is compared with all previously created nodes. If this node has the same quality as others, it will not be stored in the record of nodes in a tree (the record array **SMAX** stores the operators which have the higher qualities). This parameter is used only for the problem in which the same states are unavoidable in the search.
- (7) **MEMO**: This parameter causes that the number of all newly created nodes be appended at the beginning of the list **OPEN**. Otherwise, they are appended at the end of the list. Hence, the value of this parameter is important only in the case of the first element in the list **OPEN** being selected (this corresponds to the case when the parameter **SELECT** equals 0). In practice, when **SELECT** = 0, **MEMO** = 0 indicates the Breadth First search strategy, and **MEMO** = 1 indicates the Depth First search.
- (8) **SELECT** (values 0,1,2): Value 0 means that the first element in list **OPEN** is selected for tree extension. Value 1 means that the "random" node is selected and a procedure of probability distribution is called for this selection. Value 2 means that the node with maximum value of quality function (an attempt to find the minimal solution) is selected.

Some data are set for the specification of the tree size:

- (1) **NCMAX**: Maximal number of the nodes extended. Standard value is 1000.

- (2) SDMAX: Maximal depth of nodes in the tree for Depth First strategy.
Standard value is 20.

The parameters and the data listed above are set only for solving the problems described in this paper. We will not discuss those unnecessary parameters and data. For the same reason, in this section, all the discussions are also limited to state minimization and minimal covering problem (the input encoding) which are related to just the content of this thesis.

The routine MULTCOM is called by TDFM for completing three different tasks, the column minimization, the row minimization and the input encoding. In both column and row minimization, MULTCOM is called twice. The first calling in both column and row minimizations is for the creation of all compatible groups. The input of this calling is the merger list ZGOD and the output is the compatible groups list TABIMP. The second calling to MULTCOM is for the creation of the minimal closed and complete coverings. The input of this calling is the compatible groups list TABIMP and the output is the newly created state table. There is no difference in setting the parameters and data for both column and row minimization procedures in the first calling to MULTCOM. In the second calling, the closed and complete conditions must be checked in the second calling in row minimization. On the other hand, the checking of closure can be waived in the second calling in column minimization.

Before running MULTCOM, some preparations must be mentioned. Besides the array ZGOD or TABIMP, some other arrays and declarators are needed for these processes. Even though the purpose of these arrays and a variety of the declarators may vary in different tasks, basically, they have a uniform usage. The notation will help us to understand the verbal description of the tree search.

The arrays:

- (1) RS : The auxiliary set of the operators that are NOT applied in the tree.
- (2) AS : The set of closure conditions for every compatible pair in group QS.
- (3) QS : The set of states in a compatible group corresponding to the respective node.
- (4) GS : Set of all operators which can be applied in node NC.

and the declarators:

- (1) NC : Nodes number.
- (2) SD : The depth of the nodes in the tree.
- (3) PC : The number of the direct predecessor.
- (4) OP : The number of the operators from PC to NC.
- (5) CF : The cost function value.

§ 4.2. CALLING TO MULTCOM FOR FINDING ALL COMPATIBLE GROUPS

This calling to MULTCOM is to create all compatible groups. The strategy for defining those parameters is relatively simpler because MULTCOM is needed only to list all the compatible groups (implicants). There is not the feasibility problem for creating the compatible groups. In other words, in this case, there is no cost function for the node and no quality function for the operator. The results must satisfy only the compatible conditions which have been indicated in the list ZGOD.

The strategy of Breadth First is chosen for this task because all the nodes on this tree have to be appended and then the compatibilities among the compatible pairs have to be checked. In particular, parameters MEMO = 0 and SELECT = 0 are necessary. In order to speed up this extension, subroutine GEN is used to check the compatibility of the operators appended on a certain node. Therefore, the parameter GENAS = 1 must be established. These parameters are set in the data file 'fort.3'.

After the strategy has been decided, the respective search process is executed by MULTCOM. At the beginning of the process, the list GS^0 holds all the operators (the numbers of the inputs or PSs). For an example of § 2.2, the set of inputs is $GS^0 = \{1, 2, 3, 4, 5, 6\}$.

Figure 15. Solution tree for creating all CGs

The respective description of the first calling to MULTCOM is as follows. First, give the initial condition of the set of input addresses $\{i_1, \dots, i_{na}\}$ (or states $\{s_1, \dots, s_{mx}\}$), and then store the compatible pairs of elements in the above set in array ZGOD by the subroutine from Figure 8 for forming the merger table introduced in TABLE II. The compatibility condition of every pair of inputs or states has been checked and this information has also been stored in the merger list ZGOD before the tree search procedure starts running. The process of this arrangement can be found in § 2.2 and § 2.3. The first node '0' of the tree has already been at the top of the tree automatically, and the sets GS and QS have the initial form:

$$GS^0 = \{I_1, \dots, I_{mx}\} \quad (16a)$$

$$QS^0 = \emptyset \quad (16b)$$

or

$$GS^0 = \{S_1, \dots, S_{na}\} \quad (16a')$$

$$QS^0 = \emptyset \quad (16b')$$

Formula (16) is for column minimization and formula (16') for row minimization. All the input decimal addresses or PSs which satisfy their individual compatibility conditions are represented by the following formulas. In column minimization, if some input columns are compatible, the formula (17) is satisfied. The input column compatibility relation is:

$$COMI(I_i, I_j) = 1 \quad \text{if } (I_i, I_j) \subseteq GS^0 \times GS^0 \quad (17)$$

Formula (17) is a state space referring directly to the compatibility list ZGOD. In row minimization, it is similar that CPP_j is a compatible pair of two PS rows. If some PS rows are compatible, the formula (17') is satisfied. The PS row compatibility relation is:

$$COMP(S_i, S_j) = 1 \quad \text{if } (S_i, S_j) \subseteq GS^0 \times GS^0 \quad (17')$$

Supposing that the extended node M is the descendant node of the node N, for column minimization, the descendant node M of the node N has the properties:

$$GS^M = (I_i \mid I_i \in GS^N - \{e\}, COMI(I_i, e) = 1) \quad (18a)$$

$$QS^M = QS^N \cup e \quad (18b)$$

For row minimization, the descendant node M of the node N has the properties:

$$GS^M = (S_i \mid S_i \in GS^N - \{e\}, COMP(S_i, e) = 1) \quad (18a')$$

$$QS^M = QS^N \cup e \quad (18b')$$

The variable e is the node to indicate the currently appended operators. The result of the execution of column minimization is a set of all compatible input groups:

$$g = \{CGI_1, \dots, CGI_m\}, \text{ where } CGI_k \in GS^0, k=1, \dots, m \quad (19)$$

The formula (19) fulfills the condition:

$$(\forall CGI_k \in g) (\forall I_i, I_j \in CGI_k) [COMI(I_i, I_j) = 1] \quad (20)$$

Similar to the PS rows minimization, the set of compatible group of PSs can results:

$$g = \{CGP_1, \dots, CGP_m\}, \text{ where } CGP_k \in GS^0, k=1, \dots, m \quad (19')$$

And the formula (19') also fulfills the condition:

$$(\forall CGP_k \in g) (\forall S_i, S_j \in CGP_k) [COMP(S_i, S_j) = 1] \quad (20')$$

The complete process of the extension for inputs is shown in Figure 16; and a clear view of the respective solution tree is shown in Figure 15. The extension for PSs is similar, because in the state space COMP, all compatible pairs are indicated by '1's no matter they are strongly or weakly compatible pairs. MULTCOM always creates the node '0' first, and then starts the global extension. MULTCOM calls its subroutine GENER. In GENER, nodes '1' to '6' are created correspondingly to the operators in GS^0 since the search strategy is Breadth First and all single states are actually compatible with themselves so that all nodes of this level should be created in MULTCOM. Then, the subroutine GEN is called immediately, and the process turns to the local extension. The subroutine GEN adjusts the list GS. For instance, the row 1 to row 5 of TABLE II are node '1' related in this tree. GEN checks the compatibility marks in the second column of the list ZGOD. In this particular case, it can not find any compatibility mark '1' in these positions of the second column (in row minimization, if GEN find the compatibility mark '0' in any positions of this column of ZGOD, it also checks their compatible conditions because these PS pairs may be implied compatible); therefore, the further extension will be aborted, i.e. the operator '1' is removed out of the list GS. If this operator indicates that the inputs of this group are compatible (by checking the compatibility

condition), this operator will be added to a list MUST1. Then, the array QS collects the contents of MUST1 and appends the content of the operator '1'. Otherwise, it is just deleted from GS. After this step, GEN turns to the node '2'. So far, it is found that from the row 6 to row 9 of the first column of ZGOD, there is only one '1' at the 8th row and the first column of this table indicates '2' and '5'. Therefore, the extension is executed, That is, the operator '5' is put on the AS list and the new node '7' is created. After this, the further local extension becomes impossible because there is no operator compatible with both '2' and '5' simultaneously. As a result, GEN creates the new node '7' which presents the $CGI = \{2, 5\}$ and then gives up this branch of the tree and backtracks to node '3'.

The process creates all the compatible groups. These CGs are transferred from QS into a two dimensional array TABIMP one after one and the numbers of the elements of these groups are contained in another array FC. Meantime, it lists all the nodes created by subroutine GEN as the new operators in array GS. All this information will be given out when the subroutine GEN completes its work.

§ 4.3. CALLING TO MULTCOM FOR CREATING MCCS

The second calling to MULTCOM is to create all the minimal closed and complete coverings. The operators in this calling are the addresses of the CGs created by the last calling (in column minimization they are the numbers of the CGIs; and in row minimization they are the numbers of the CGPs) indicated by the array GS.

For explaining the difference between the complete coverings of input columns and the closed and complete coverings of PS rows, the merger graph can be used again. Referring to the discussion about this graph from previous sections, each node of this graph represents an input or a PS. If two nodes are compatible, one connection must exist between them. If the connections among several nodes are complete, which means these

solution tree

nc sd pc op f qs

0 0 * * 0

initial operators list : 1 2 3 4 5 6
 as list after first call to gener:

operators for current node: 1 2 3 4 5 6
 the operator applied to node 0 : 1
 gs list generated by gen :
 output= 0
 as list after extension :

1 1 0 1 0 1

the operator applied to node 0 : 2
 gs list generated by gen : 5
 output= 0
 as list after extension :

2 1 0 2 0 2

the operator applied to node 0 : 3
 gs list generated by gen : 6
 output= 0
 as list after extension :

3 1 0 3 0 3

the operator applied to node 0 : 4
 gs list generated by gen :
 output= 0
 as list after extension :

4 1 0 4 0 4

the operator applied to node 0 : 5
 gs list generated by gen :
 output= 0
 as list after extension :

5 1 0 5 0 5

the operator applied to node 0 : 6
 gs list generated by gen :
 output= 0
 as list after extension :

6 1 0 6 0 6

operators for current node: 5
 the operator applied to node 2 : 5
 gs list generated by gen :
 output= 0
 as list after extension :

7 2 2 5 0 2 5

operators for current node: 6
 the operator applied to node 3 : 6
 gs list generated by gen :
 output= 0
 as list after extension :

8 2 3 6 0 3 6

bc= 0 sc= 0 nc= 8 sd= 2

Figure 16. Verbal description of tree search in Figure 15 for CGs

nodes and connections satisfy the formula (10), the nodes within this complete clique satisfy the closed and complete conditions.

Supposing that the nodes '1' and '2' from Figure 17(a) are in such a CGI, for column minimization, just remove this compatible group from the graph and disconnect the connections between the nodes in this group and the nodes left in the remaining graph. After this step is done, the connections between the nodes '1' and '4', '1' and '3' as well as '2' and '3' are no longer retained.

All compatible groups have been created in the first calling to MULTCOM (includes the maximal compatible groups, i.e. the maximal complete cliques shown in merger graph) and their compatibilities have been proven. Therefore, it is not necessary to check again for column minimization in the second calling to MULTCOM. However, in row minimization, this problem is relatively more complicated. The difficulty is that in the row minimization, using the same graph coloring method as in the column minimization, may cause some CGPs to lose their credit of compatibility. Moreover, it will result in the creation of a wrong solution.

Supposing that a merger graph of Figure 17(a) is employed to minimize the rows of TABLE IV, one solution $\{12, 34\}$ may be created. This solution is derived by the same version of the merger graph as the one used in column minimization or by the second calling to MULTCOM without checking the compatibility. On the surface, this solution indeed satisfies the definitions (1) and (6), and also satisfies the formula (1) because both of the groups $\{1, 2\}$ and $\{3, 4\}$ are compatible. Unfortunately, this set is not a correct solution. Reviewing definitions (3) and (5), it is found that choosing the MCCP must consider the relation of implication between $\{S_i\}$ and $\{S'_{h,i}\}$. The merger graph as the one of Figure 17(a), however, does not illustrate this relationship.

Another sort of merger graph is shown in Figure 17(b) for solving the row minimization problem. Supposing that there are some implied NS group $\{S'_m\}$ in a certain CGP

Figure 17. Comparison of merger graphs for column and row minimizations

$\{S_l\}$ which has been chosen as a candidate subset of a MCCP set, and these NSs are included in another CGP $\{S_h\}$ (these NSs are shown in the square shape blocks on the connections). If the CGP $\{S_h\}$ or the NS group $\{S'_m\}$ is not chosen as another subset of the same set, even when this CGP $\{S_h\}$ is compatible, this set will not be a solution and we have to discard it. To overcome this problem, the following definition must be introduced.

Definition (5)':

Supposing that a group of states $\{S_m\}$ is implied by another group of states $\{S_l\}$, a strongly compatible group $\{S_h\}$ that includes the group $\{S_m\}$, must be selected to be a group of this solution set, if and only if $\{S_l\}$ has been chosen as the other candidate group of the solution set.

According to definition (5)', before a closed and complete covering is chosen, it must be considered whether this MCCP includes some implied NSs. For instance, in Figure 17(b), if $\{2, 3\}$ has been chosen as a subset of the covering set, can the remaining states $\{1, 4\}$ form the other subset? The answer is positive because in the square shape block between the node '2' and '3', the NSs 1 and 4 are included in the group $\{1, 4\}$.

Therefore, the result is $MCCP = \{23, 14\}$.

Usually, this problem can not be solved so easily. For instance, the implied compatible group $\{1, 2\}$ has been chosen. In this particular case, node '1' must be left in the remaining graph. Another subset of this CCP has to be $\{1, 3, 4\}$. As a result, the solution $CCP = \{12, 134\}$ is returned. This overlapping solution, however, can still be a MCCP only in cases when the non-overlapping solution does not exist. This idea will also be used to improve the routine MULTCOM.

Since the routine MULTCOM is called to select a minimal solution among more than one solution, the Branch and Bound strategy is chosen, and the strategy Ordering is added on this task for speeding up the procedure. The parameters used for this goal are stored in the data file 'fort.4':

- GENAS = 1: cause the subroutine GENER to link the subroutine GEN for creating the descendant nodes.
- EQUI = 1: add subroutine EQUI for dealing with equivalent operators (this parameter is not used in column minimization because the compatible input columns are never overlapped).
- MUST = 1: add subroutine MUST0 for appending the indispensable operators to the list MUST1.
- SPRAW = 1: check the newly created node if this expansion is necessary.
- ORD = 1: add subroutine ORD for ordering the operators in decreasing quality values. The operator which has the highest Q value will be dealt first.
- SELECT = 2: select the node which has the highest value of quality function.
- NCMAX = 1000: limit the number of selected nodes.

A special parameter which is created by the algorithm itself is OUTPUT. If OUTPUT = 0, the created node is a branch. If OUTPUT = 1, the created node is an end of the tree. If OUTPUT = 2, the created nodes of this branch create a minimal solution.

Starting from the initial node '0', the contents of the arrays for this procedure are as follows:

$$GS^0 = \{CGI_1, \dots, CGI_n\} \quad (22a)$$

is a set of operators.

$$AS^0 = \{i_1, \dots, i_{mx}\} \quad (22b)$$

is a set of inputs.

$$QS^0 = \emptyset \quad (22c)$$

and the relation of covering states by compatible groups is:

$$COI(i_i, CGI_j) = 1 \text{ if } (i_i \in CGI_j, (i_i, CGI_j) \subseteq AS^0 \times GS^0) \quad (23)$$

For row minimization, the following formulas are available.

$$GS^0 = \{CGP_1, \dots, CGP_n\} \quad (22a')$$

is a set of operators.

$$AS^0 = \{s_1, \dots, s_{na}\} \quad (22b')$$

is a set of PSs.

$$QS^0 = \emptyset \quad (22c')$$

and the relation of covering states by compatible groups is:

$$COP(s_i, CGP_j) = 1 \text{ if } (s_i \in CGP_j, (s_i, CGP_j) \subseteq AS^0 \times GS^0) \quad (23')$$

and the closure relation:

$$CLP(CGP_i, CGP_j) = 1 \text{ if } (CGP_i \rightarrow CGP_j, (CGP_i, CGP_j) \subseteq GS^0 \times GS^0) \quad (23'')$$

which means that CGP_i implies CGP_j .

The tree search bases on the above given conditions. Formula (22) gives out a state space in order to illustrate how the tree search appends the nodes to the tree, and finally builds a complete covering obtained all the states by using a set of compatible groups. Figure 18 is the example of such a state space for solving the column minimization problem.

		states					
		1	2	3	4	5	6
operators	1	1					
	2		1				
	3			1			
	4				1		
	5					1	
	6						1
	7		1			1	
	8			1			1

Figure 18. State space for tree search of MCC

If the node M is the descendant node of the node N, we have:

$$GS^M = GS^N - e \quad (24a)$$

$$AS^M = AS^N - (i_i \mid COI(i_i, e) = 1) \quad (24b)$$

$$QS^M = QS^N \cup e \quad (24c)$$

For row minimization, the description of descendant node are:

$$GS^M = GS^N - e \quad (24a')$$

$$AS^M = AS^N - (s_i \mid (COP(s_i, e) = 1) \vee (s_i \in \{e\}, CLP(e, CGP_j) = 1)) \quad (24b')$$

$$QS^M = QS^N \cup e \quad (24c')$$

In above formulas, the array COI or COP is built in accordance with the file TABIMP. The columns of the state space are states (inputs or PSs) and rows are operators of the compatible groups. e is the operator of a compatible group which is currently applied on the extension (a row of COI or COP).

The solution set of column minimization is established by this:

$$p = \{CGI_1, \dots, CGI_m\} \subseteq g \quad (25)$$

which fulfills the condition:

$$(\forall i_i \in AS^0) (\exists CGI_k \in p) [COI(i_i, CGI_k) = 1] \quad (26)$$

The solution set of row minimization is established by this:

$$p = \{CGP_1, \dots, CGP_m\} \subseteq g \quad (25')$$

which fulfills the condition:

$$(\forall s_i \in AS^0) (\exists CGP_k \in p) [COP(s_i, CGP_k) = 1] \quad (26')$$

and

$$(\forall CGP_i \in p) [CLP(CGP_i, CGP_j) = 1 \Rightarrow CGP_j \in p] \quad (26'')$$

The cost function specified for this task:

$$CF(M) = CARD(QS^M) \quad (27)$$

is the number of the states which have been selected in a group of the solution.

The quality function for current appended operator e is:

$$Q(e) = f_2(e) * \frac{f_2(e)}{1 + f_1(e)} \quad (28)$$

where

$$f_1(e) = \sum_{CGI_i \in GS^M} d_i$$

or

$$f_1(e) = \sum_{CGP_i \in GS^M} d_i$$

In row minimization, d_i is the number of the internal states which satisfy the implied compatible condition in the currently appended ICGP. On the other hand, d_i is the number of non-don't-care terms in the input states of the currently appended CGI in column minimization. Since all the input states should be in a solution set, we always prefer to give the binary inputs which have more don't-care terms to those CGIs which can cover more input states and therefore let the numbers of the products of literals in every input binary categories be relatively similar.

$$f_2(e) = CARD(i_i \in AS^M \mid COI(i_i, e) = 1)$$

or

$$f_2(e) = CARD(s_i \in AS^M \mid COP(s_i, e) = 1)$$

In the above formulas of quality calculation, $f_2(e)$ is a number of candidate states that were contained in the currently selected operator. Obviously, the larger its value is, the better quality the operator has. In the array $COI(i_i, e)$ or $COP(s_i, e)$, i_i or s_i is a group of inputs or PSs in a certain CGI_k or CGP_k indicated by operator e .

Comparing the procedure of the first calling to MULTCOM introduced in § 4.2, this calling is relatively complicated and tedious. This is because the strategy used in this calling requires checking the quality values of CGs, the closure/completeness conditions,

as well as the cost of the solution sets.

Figure 19. Solution tree for creating MCC

The complete process of this calling to MULTCOM is as follows. MULTCOM calls GENER. GENER is the procedure of finding indispensable operators and applying them to extend the related node. The first cycle for global extension starts from the subroutine GENER. At first, GENER calls EQUI for deleting the equivalent node. Secondly, it calls MUST0 for searching the indispensable operators and applying them in list MUST1. Thirdly, it calls GEN for checking the closure and completeness conditions. In the modified version of GEN in this thesis, if the procedure is for handling the row minimization, the array SMAX which contains the compatible conditions is checked for the closure of these CGs. This checking involves only those implied compatible groups (ICGP) and the result of the first calling to MULTCOM is used directly. As a contrast, this procedure is simply avoided in column minimization. In subroutine MUST0, the list MUST1 becomes empty if there is no more indispensable operator in the candidate list GS; then the routine GENER terminates.

Here, MULTCOM puts the data generated by GENER in a long term list and then orders the operators by calling to subroutine ORD. The second cycle starts from GENER again. It performs in the same way as the first cycle discussed above, but the goal of this

cycle is the local extension.

The meaning of Branch and Bound is: Every time a search comes into a branch of the tree, this search turns to a local extension from the global extension. If a node in this local extension will be able to lead to a successive tree search, i.e., the extended nodes tend to have better values of the quality functions and less overlappings, this extension will continue; otherwise, this extension will be cut off and the backtrack occurs.

Generally, in both global and local extensions, if the tested operators satisfy the condition described in the last paragraph, this extension is successive, and a new node is created. If the compatible groups added to this node satisfy the closed and complete conditions, these CGs may consist of a minimal solution set. When a solution is created, all of the indispensable operators which are involved in the extended nodes in a path of the tree have been stored in the array QS. The completeness checking is to compare all of the PS rows indicated by these operators with the list of PS rows in the array GS. If all of the PS rows have been presented, this solution is complete. On the other hand, for column minimization, this process is relatively simple because the closure checking is not necessary. If all of the operators can pass the above checking, finally this procedure calculates these operators' value of cost function by calling to the subroutine FOC(f).

Figure 19 shows the example of this process for searching the minimal solution. Its verbal expression is shown in Figure 20. The input of this process is the list of compatible groups stored in array TABIMP shown in TABLE III. In global search, the indispensable operators '1' and '4' have been chosen as the subsets of any minimal solution. In the second step (the description of global extension), these two operators are put in list MULT1. Therefore, they will no longer appear in the local search. After calling to the ordering subroutine ORD, the operators '7' and then '8' are chosen for local extension, because they have higher quality values. In the following steps, the elements represented by operators '2', '3', '5' and '6' are checked. The result is that they have

solution tree

nc sd pc op f qs

```

initial operators list : 1 2 3 4 5 6 7 8
after processed by equi : 1 2 3 4 5 6 7 8
must1 list: 1 4
gs list generated by gen : 2 3 5 6 7 8
after processed by equi : 2 3 5 6 7 8
must1 list:
as list after first call to gener: 2 3 5 6
the operator applied to node 0 : 7
gs list generated by gen : 2 3 5 6 8
after processed by equi : 3 6 8
must1 list:
output= 0
as list after extension : 3 6

1 1 1 7 4 1 4 7

the operator applied to node 1 : 8
gs list generated by gen : 2 3 5 6
after processed by equi : 2 5
must1 list: 2 5
gs list generated by gen :

* 0 0 8 5 solution
  1 4 8 2 5
  b = 5   sc = 1   nc = 1
  output= 1
  as list after extension :
  the operator applied to node 0 : 2
  gs list generated by gen : 3 5 6
  after processed by equi : 3 5 6
  must1 list: 3 5 6
  gs list generated by gen :
  output= 1
  as list after extension :
  the operator applied to node 0 : 3
  gs list generated by gen : 5 6
  after processed by equi : 5 6
  output= 1
  as list after extension : 2 5 6
  the operator applied to node 1 : 5
  gs list generated by gen : 6
  after processed by equi : 6
  output= 1
  as list after extension : 2 3 6
  the operator applied to node 1 : 6
  gs list generated by gen :
  output= 0

```

Figure 20. Verbal description of tree search in Figure 19 for MCC

```

as list after extension : 2 3 5
2 1 1 6 4 1 4 6
operators for current node: 3 6 8
the operator applied to node 1 : 8
gs list generated by gen : 3 6 * 1 0 8 4 solution
1 4 7 8
b = 4 sc = 2 nc = 2
output= 1
as list after extension :
the operator applied to node 0 : 3
gs list generated by gen : 6
output= 1
as list after extension : 6
the operator applied to node 1 : 6
gs list generated by gen :
output= 1
as list after extension : 3
the last solution is optimal
bc= 7 sc= 2 nc= 2 sd= 1 The Final Solution List:
1 4 7 8

```

Figure 20. Verbal description of tree search in Figure 19 for MCC (continue)

already been involved in the compatible groups represented by operators '7' and '8' respectively. As a result, they are deleted from the list GS by subroutine EQUI before the nodes t_1 , t_2 , t_3 and t_4 are formed. Finally, the procedure creates the minimal solution $MCCI = \{1, 4, 25, 36\}$.

§ 4.4. CALLING TO MULTCOM FOR INPUT COMBINATIONAL COVERING

The relative content of this process has been discussed at the end of § 2.4. The minimization of the covering table of Figure 12(a) is actually such a problem. For reference to this problem, the approach of M. Perkowski and J. Liu [15] is available.

Since this problem can be solved by using a similar method to the second calling to MULTCOM for solving the state table minimization problem introduced in § 4.3, the Branch and Bound strategy is recommended. The parameters and data used in this problem are the same as in the procedure for creating the MCCs in the second calling to MULTCOM. Some differences are in the setting of the initial conditions, in the process

methods for finding the equivalent groups in subroutine EQUI, and for picking up the indispensable subsets in subroutine MUST0. In this input logic encoding problem, MULTCOM is called only once since the minterms have been found by the tabulation procedure (they can be considered the same as CGs in state table minimization). This only calling to MULTCOM is for finding the minimal complete covering.

Comparing with the state table minimization algorithms, the features of this algorithm are:

- (1) In MUST0, the indispensable operators are checked by the numbers indicated in array FC. For instance, if a certain number is 1, this row has only one X.
- (2) In EQUI, comparing any two rows of the covering table, if row i has the same or less numbers of Xs and all these Xs are in the same columns with row j or included in some columns of row j , row i is called equivalent with row j ; therefore, this row i should be deleted from list GS.
- (3) The same as in the second calling to MULTCOM for column minimization, the checking of the compatibility is not necessary; therefore, the algorithm in subroutine GEN becomes relatively simpler.
- (4) Since the closure condition is not necessary to be checked (the same as in the second calling to MULTCOM for column minimization), but the overlap is sometimes still not avoidable.

$$GS^0 = \{R_1, \dots, R_m\} \quad (29a)$$

is the set of operators of binary prime implicants shown at the left side of the covering table in Figure 12(a).

$$AS^0 = \{C_1, \dots, C_n\} \quad (29b)$$

is the set of covering indicators shown at the heading of Figure 12(a).

$$QS^0 = \emptyset \quad (29c)$$

The relation of covering columns by rows is:

$$COV(C_j, R_i) = 1 \text{ if } ((C_j, R_i) \subseteq AS^0 \times GS^0) \quad (30)$$

The solution can therefore be found:

$$p = \{R_1, \dots, R_m\} \subseteq GS^0 \quad (31)$$

which fulfills the condition:

$$(\forall C_j \in AS^0) (\exists R_i \in p) [COV(C_j, R_i) = 1]$$

If the node M is the descendant of the node N, then

$$GS^M = (e \mid e \in GS^0, COV(C_j, e) = 1) \quad (32a)$$

$$AS^M = AS^N - (C_j \mid C_j \in AS^0, COV(C_j, e) = 1) \quad (32b)$$

$$QS^M = QS^N \cup e \quad (32c)$$

and the cost function:

$$CF(M) = c(M) + h(M) \quad (33)$$

where:

$$c(M) = \frac{1}{\sum_{R_i \in QS^M} d_i}$$

and

$$h(M) = CARD(AS^M) * CARD(GS^M) * \frac{\sum_{R_i \in GS^M} d_i * CARD(C_j \in AS^M \mid COV(C_j, R_i) = 1 \& R_i \in GS^M)}{(\sum_{R_i \in GS^M} CARD(C_j \in AS^M \mid COV(C_j, R_i) = 1 \& R_i \in GS^M))^2}$$

The array $COV(C_j, R_i) = 1$ means there is a X in the intersection of row i and column j . The operator e indicates a row which is currently added on the extension.

The quality function is:

$$Q(e) = c_1 * f_1(e) + c_2 * f_2(e) + c_3 * f_3(e) \quad (34)$$

where:

$$f_1(e) = \sum_{R_i \in GS^N} d_i$$

$$f_2(e) = \text{CARD}(C_j \in AS^M \mid \text{COV}(C_j, R_i) = 1)$$

and

$$f_3(e) = \frac{1}{f_2(e)} * \sum_{j=1}^n \text{CARD}(R_i \mid C_j \in AS^M \& \text{COV}(C_j, R_i) = 1 \& R_i \in GS^M \& \text{COV}(C_j, R_i) = 1)$$

In the above formula, n is the number of columns in the array COV.

Figure 21. Solution tree for creating covering of function F_m

The values of variable d_i stored in array FCEN are decided by the number of don't-care terms of prime implicants. The values of parameters c_1 , c_2 and c_3 are user evaluated. In the particular case of Figure 12(a), the values of these parameters are $c_1 = -0.6$, $c_2 = 0.2$ and $c_3 = -0.3$.

As the result of the covering table Figure 12(a), the covering table Figure 12(b) forms the most minimal number of implicants the same as the result shown in the multi-output Karnaugh map.

solution tree

nc sd pc op f qs

initial operators list : 1 2 3
 after processed by equi : 1 2 3
 must1 list: 1 3
 gs list generated by gen : 2

* 0 0 2 4 solution

1 3

b = 4 sc = 1 nc = 0

as list after first call of gener:

the last solution is optimal

bc= 1 sc= 1 nc= 0 sd= 0

Figure 22. Verbal description of tree search in Figure 21 for MCC

The search for the minimal covering can be done by MULTCOM. The operators of the input data from the left side of Figure 12(a) and the indicators of the covering from the heading of the same table are put in the array TABIMP shown in TABLE XII in this tree search. The process of the tree search is shown in Figure 21 and its verbal description is in Figure 22. This process is similar to the search for MCCs in the second calling to MULTCOM. In the global search, MULTCOM finds that the operators '1' and '3' are indispensable. Therefore, these compatible groups of two operators are chosen as the parts of solution and nodes t_1 and t_2 are deleted before they are formed. In the next step, MULTCOM finds that the operator 2 has been covered by operators '1' and '3'. Therefore, the node t_3 is also deleted and the compatible group of operator '2' will not be put in the solution. The minimal closed and complete covering includes only the compatible groups of operators '1' and '3'.

CHAPTER V

CONCLUSION

The complete program Two Dimensional FSM Minimizer TDFM discussed in this thesis has been tried out on several Finite State Machines. Some of those machines contain as many as 50 present states, some contain 20 inputs. From the results of these trials, it appears that TDFM is able to deal with large scale machines and generates the optimal solutions more quickly, if the percentage of don't care terms in these machines is lower. As the percentage of don't care terms increases, the time needed in generating the MCCPs will obviously expand. This is because that even though the generation of compatible groups is a fast process, the identification of the compatibility, especially the identification of feasibility for the closed and complete covering is usually a tedious process in row minimization problem. This identification includes not only the testing of compatibility among the corresponding NSs of those CGPs but also the implied compatibility among the ICGPs and other related compatible PS groups.

Another cause of the complication is related to the strategy setting. In the second calling to MULTCOM for generating the MCCP of row minimization, the parameter SPRAW, for instance, is used to perform the branch and bound strategy. It orders the algorithm to compare every newly generated node with all its parent nodes for deciding each new node's quality. Since the large scale machines always produce a giant tree structure, this comparison will unavoidably involve all the nodes in the established part of the tree. The process will be considerably slowed down.

The last reason for this complication is that since the program deals with the column and row minimizations iteratively, the more don't care terms exist in the initial

machine M^0 , the more times of the iterations will be applied to create both of OMCCI and OMCCP of a FSM M^* .

The questions discussed above do not mean that the program TDFM is not efficient or even not practical. In fact, TDFM was the only program aimed at the large scale machines with a relative high percentage of don't care terms. Therefore, the time spent on the procedure is reasonably worthwhile since as it is known that there is not any other time saving program or even the theory that can construct such a program for state table minimization yet. The AI program MULTCOM is still based on the trial-and-error theory. Up to now, this program has been able to compare the present node with its parent nodes to decide its quality; but the cost of the solution and qualities of the established nodes are still not anticipatable. Presently, the methodology to save time is the suitable selection of the strategies, as well as the suitable selection of the cost function and quality function.

The theory about AI technology applied in MULTCOM can be found in the reference book "Problem Solving Methods in Artificial Intelligence" [16]. Additionally, a study done by E. Lawler, 1966 [17] has discussed the method of branch and bound.

The algorithm of tabulation and then the selection of the minimal binary input covering will not spend a lot of time, since the size of the machine has already been reduced in previous minimization procedures. Even for those slightly reduced large scale machines, time is still not a problem because it is a combinational logic question. In addition, the testing of compatibility in MULTCOM is simpler as has been discussed in § 4.4. For the same reason, the column minimization hires a similar form of MULTCOM in the second calling for solving the closed and complete covering problem. Therefore, the behavior of this problem can also be similar to the binary input covering problem.

Some machines have been practised on the program TDFM. The statistical results of these machines are listed in TABLE XIX.

TABLE XIX
BENCHMARKS OF SOME FSMS

	M_a	M_b	M_c	M_d	M_e	M_f
data style	.kiss	.stab	.kiss	.kiss	.stab	.stab
input bits	3	-	5	3	-	-
output bits	2	2	3	2	4	2
columns of M^0	6	4	24	6	20	8
rows of M^0	4	50	5	5	20	4
% of don't care NSs	25	21.2	40	43.33	74.7	62.5
% of don't care outputs	66.67	34.24	40	60	85	84.8
iteration	3	2	2	4	3	3
columns of M^*	3	4	19	3	11	3
rows of M^*	2	46	5	4	10	3
time of execution(sec.)	1.3	7.4	8.0	1.5	10.3	1.2

REFERENCES

- [1] Paul M.C. and Unger, S.H. : "Minimizing the Number of States in Incompletely Specified Sequential Switching Functions", IRE Trans. Electron. Computers, Vol. EC-8, pp. 356-367, Sept. 1959.
- [2] Unger, S.H. : "Flow Table Simplification - Some Useful Aids", IEEE Transaction on Electron. Computers (short notes), Vol. EC-14, pp. 472-475, June 1965.
- [3] McCluskey, E.J. : "Minimum-State Sequential Circuits for a Restricted Class of Incompletely Specified Flow Tables", Bell System Tech. J., pp. 1759- 1768, Nov. 1962.
- [4] Pager, D. : "Conditions for the Existence of Minimal Closed Covers Composed of Maximal Compatibles", IEEE Transaction on Computers. (short notes), Vol. C-20, pp. 450-452, April 1971.
- [5] Enrich, H. : "A Note on State Minimization of a Special Class of Incomplete Sequential Machines", IEEE Transaction on Computers (short notes), Vol. C-21, pp. 500-502, May 1972.
- [6] Grasselli, A. and Luccio, F. : "A Method for Minimizing the Number of Internal States in Incompletely Specified Sequential Networks", IEEE Transaction on Electron. Computers, Vol. EC-14, pp. 350-359, June 1965.
- [7] Luccio, F. : "Extending the Definition of Prime Compatibility Classes of States in Incomplete Sequential Machine Reduction", IEEE Transaction on Computers, Vol. C-18, pp. 537-540, June 1969.
- [8] Hamachi, G. : "Peg Tutorial", VLSI Tools, Univ. of California, Berkeley, 1984.
- [9] Perkowski, M. and Nguyen, N. : "Minimization of Finite State Machines in SuperPeg", Proceeding of the Midwest Symposium on Circuits and Systems. Louisville, Kentucky, 22-24 .pp 139-147, August 1985.
- [10] Biswas, N. N. : "State Minimization of Incompletely Specified Sequential Machines", IEEE Transaction on Computers, Vol. C-23, pp. 80-84, 1974.
- [11] Yamamoto, M. : "A Method for Minimizing Incompletely Specified Sequential Machines" , IEEE Transactions on Computers, Vol. C-29, No. 8, pp. 732-736, August 1980.

- [12] Perkowski, M. and Uong, H. : "Automatic Design of Finite State Machines with Electronically Programmable Devices", Record of Northcon 87, Portland, 1987, paper 13/4.
 - [13] De Micheli, G., Brayton, R. and Sangiovanni-Vincentelli, A.L. : "Optimal State Assignment for Finite State Machines", IEEE Transactions on Computer-Aided Design, Vol. CAD-4, No. 3, pp. 269-285, July 1985.
 - [14] Perkowski, M. : "An application of General Problem Solving Methods in Computer Aided Design", The Multicomp System And Its Problem Oriented Source Language, IV Intern. Symp. Methods of Heuresis (Warsaw 1977), Vol. 3, pp. 23-90.
 - [15] Perkowski, M., Liu, J. and Brown J. : "A System for Fast Prototyping of Logic Design Programs", Proceedings of 1987 Midwest Conference on Circuits and Systems, Syracuse, New York.
 - [16] Nilsson, Nils J. : "Problems Solving Methods in Artificial Intelligence", McGraw-Hill, 1971.
 - [17] Lawler, E. L. and Wood, D. E. : "Branch-And-Bound Methods, A Survey", Operations Res. 14, pp. 699-719, 1966.
-

APPENDIX

USERS MANUAL

NAME

TDFM - the FSM state table minimization program.

SYNOPSIS

tdfm < input file > output file

DESCRIPTION

TDFM is the state table minimization program designed for Finite State Machine. This program can minimize the state table of Mealy model in two dimensions, i.e. the input columns and the present state rows. Two types of formats, the .kiss and the .stab are acceptable. TDFM offer five data files for suiting every individual procedures. To the users, only the file 'fort.1' is needed to be adjusted. An example of such a file is presented in this:

```
inbit    03
oubit    02
innum    04
psnum    50

formt    k
forma    (3a1,3x,i3,3x,i3,1x,2a1)
formb    (3a1,1h,(i3,1h),3x,i3,3x,i3,3x,2a1)
formc    (1a1,1x,3h = ,3a1,120(1h+,3a1))
formd    (4(i2,1h/,2a1,1h ))
forme    (/20(i3,1h/,2a1,1x))
```

In this file, the term 'formt' following the letter 'k' means that your input file is .kiss formatted. Thereafter, if your input file is .stab formatted, you have to change this letter to 's'. The numbers following the terms 'inbit' and 'oubit' are the bit numbers of

binary inputs and outputs. These numbers respect to the numbers of the characters that follow the formats 'forma' to 'forme' in parentheses. For instance, all of the expressions '3a1' corresponding to the number of input bits, and all of the expressions '2a1' corresponding to the number of output bits. Moreover, the expressions 'i3' are the bits of internal states. All of these expressions should be adjusted in the .kiss formatted input file depending to the bit numbers of your file. On the other hand, for suiting the .stab formatted input file, aside above adjustments, the numbers following the terms 'inum' and 'psnum' are the numbers of the input columns and the present state rows. These two numbers should be adjusted particularly according to the size of your .stab formatted input file. The same as the adjustments for above formats, the number '4' following the format 'formd' is the number of input columns whatsoever.

AUTHOR

William. Y. Zhao

FILES

/yuezhao/507ee/store fort.1
 /yuezhao/507ee/store fort.2
 /yuezhao/507ee/store fort.3
 /yuezhao/507ee/store fort.4
 /yuezhao/507ee/store fort.5
 /yuezhao/507ee/pro tdfm

SEE ALSO

William. Y. Zhao, A New Approach of Finite State Machine Minimization