TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

DASM : a tool for decomposition and analysis of sequential machines

Citation for published version (APA):
Hou, Y. (1987). *DASM : a tool for decomposition and analysis of sequential machines*. (EUT report. E, Fac. of Electrical Engineering; Vol. 87-E-170). Technische Universiteit Eindhoven.

Document status and date:
Published: 01/01/1987

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 22. Aug. 2022

TH
Research Report

Eindhoven
University of Technology
Netherlands

Department of Electrical Engineering

DASM:
A Tool for Decomposition and Analysis of Sequential Machines

by
HOU Yibin

EUT Report 87-E-170
ISBN 90-6144-170-6
ISSN 0167-9708

March 1987

Eindhoven University of Technology Research Reports
EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Eindhoven The Netherlands

DASM:
A TOOL FOR DECOMPOSITION AND ANALYSIS OF SEQUENTIAL MACHINES

by

HOU Yibin

EUT Report 87-E-170
ISBN 90-6144-170-6
ISSN 0167-9708
Coden: TEUEDE

Eindhoven
March 1987

DASM－時序机的計算机辅助分解和分析

侯　义　斌

地址：

中国，西安，西安交通大学

計算机科学与工程系

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Hou Yibin

DASM: a tool for decomposition and analysis of sequential machines /
by Hou Yibin. - Eindhoven: University of Technology. - Fig. -
(Eindhoven University of Technology research reports / Department of
Electrical Engineering, ISSN 0167-9708; 87-E-170)
Met lit. opg., reg.
ISBN 90-6144-170-6
SISO 664　UDC 681.325.65:519.6　NUGI 832
Trefw.: automatentheorie.

ABSTRACT

In this paper we introduce a program package which is a tool for decompositions and analysis of sequential machines. A sequential machine is a mathematical model for discrete, deterministic information processing devices and systems, such as digital computers, digital control units, electronic circuits with sychronized delay elements, as well as other fields like biology, psychology and biochemistry. The package can handle most of functions and topics in research and applications of sequential machines.

Hou Yibin
DASM: A tool for decomposition and analysis of sequential machines.
Department of Electrical Engineering, Eindhoven University of Technology, 1987.
EUT Report 87-E-170

ACKNOWLEGEMENTS

The author is greatly indebted to prof. ir. A. Heetman (*) and prof. ir. M.P.J. Stevens for their support and encouragement to this work. Thanks are also expressed to ir. A. Geurts for his advice with respect to program writing and all members of EB group who gave their help to this work in any way. Dr. L. Jozwiak has made useful discussions and contribution to the new version of DASM for Apollo system and the author would like to record his gratitude here.

(*) Group Digital Systems, Department of Electrical Engineering, Eindhoven University of Technology, The Netherlands.

Address of the author:

Dr. Hou Yibin,
Department of Computer Science and Engineering,
Xi'an Jiaotong University,
Xi'an, China

CONTENTS

0. Introduction.. 1
I. System Structure and Operations.................. 2
1.1 DASM Organization............................ 2
1.2 Menu Driver and Help......................... 4
II. Machine Functions................................. 6
III. Data Structure...................................10
3.1 Description of a Machine.....................10
3.2 Enter a Machine Table........................11
3.3 Representation of Partitions.................13
IV. The Substitution Property Partitions.............14
4.1 Principle....................................14
4.2 Algorithm....................................16
4.2.1 The Smallest SP's.......................17
4.3 Example......................................19
V. Partition Pairs...................................21
5.1 Principle....................................21
5.2 Algorithm....................................22
5.3 Example......................................23
VI. Partition Trinities..............................25
6.1 Principle....................................25
6.2 Algorithm....................................26
6.3 Example......................................28
VII. Equivalent States................................29
7.1 Principles...................................29
7.2 Algorithm....................................30
7.3 Example......................................31
VIII. Minimizing Machines..............................33
8.1 Principle....................................33
8.2 Algorithm....................................33
8.3 Example......................................34
IX. Decompositions on States.........................36
9.1 Principle....................................36
9.2 Algorithms...................................37
9.3 Example......................................40
X. Full-decompositions...............................42
10.1 Principle...................................42
10.2 Algorithm...................................43
10.3 Example.....................................44
XI. ISSM...50
11.1 Principle...................................50
11.2 Algorithm...................................50
11.3 Example.....................................51

References...53

D A S M
A TOOL FOR DECOMPOSITION AND ANALYSIS OF SEQUENTIAL MACHINES

0. INTRODUCTION

A sequential machine is a mathematical model for discrete, deterministic information processing devices and systems, such as digital computers, digital control units, electronic circuits with synchronized delay elements, and so on. The term, sequential machine, machine, finite-state machine, finite-state automaton, and automaton are synonyms. Nowadays, sequential machine also plays an important role in designing complex systems by using VLSI devices [3].

In this paper, we introduce a program package - DASM which is a tool for decompositions and analysis of sequential machines. It can deal with most of functions and topics in the research and applications of machines. The emphasizes for discussing each function of DASM are on three aspects: principle, algorithm and example. Therefore, it is a useful tool for teaching, researching and industry applications related to machine theory. Due to the facts that DASM works on the abstract mathematical level and that the theory of machines has a considerable influence, not only on the development of computer systems and their associated languages and software, but also in biology, psychology, biochemistry [19], etc., DASM can naturally be also used in any other area where machine theory serves as a mathematical model.

I. SYSTEM STRUCTURE AND OPERATIONS

1.1 *DASM Organization*

DASM runs on the following system enviroment:

Hardware : IBM XT/AT;
Operating system : DOS;
Source file language : Pascal.

With the latest version DASM offers the following functions:

a) SP partitions;
b) Partition pairs;
c) Partition trinity;
d) Equivalence States;
e) Minimize a machine;
f) Decompositions on states;
g) Parallel full-decompositions.

In addition to those, some functions are also available for file or data managements:

h) List directories;
i) Specify a data file;
j) Edit a text file.

In the following diagram we list all commands in DASM with different levels. A notation is used there as

A
| C
B

to denote that DASM goes to menu or function *B* from menu *A* under the command C.

```
DOS
 | DASM
DASM
 |     A
 |-------- assignment of states *
 |     E
 |-------- equivalence states
 |     D
 |-------- decompositions on states
 |             |    P
 |             |-------- parallel decomposition
 |             |    S
 |             |-------- serial decomposition
 |             |    Q
 |             |-------- quit current level
 |             |    X
 |             |-------- specify a new data
 |     F
 |-------- full-decompositions
 |             |    P
 |             |-------- parallel decomposition
 |             |    S
 |             |-------- S-type serial decomposition *
 |             |    O
 |             |-------- O-type serial decomposition *
 |             |    Q
 |             |-------- quit current level
 |             |    X
 |             |-------- specify a new data
 |     M
 |-------- minimize machines
 |     I
 |-------- decompositions and analysis on ISSM's
 |             |    P
 |             |-------- weak pairs
 |             |    D
 |             |-------- decompositions *
 |             |    Q
 |             |-------- quit current level
 |             |    X
 |             |-------- specify a new data
 |     O
 |-------- O.C.S.P. partitions
 |     W
 |-------- Edit a text file
```

```
|      P
|--------- partition pairs
|                |    A
|                |-------- S-S pairs
|                |    B
|                |-------- S-O pairs
|                |    C
|                |-------- I-S pairs
|                |    D
|                |-------- I-O pairs
|                |    Q
|                |-------- quit current level
|                |    X
|                |-------- specify a new data
|      T
|--------- partition trinities
|                |    A
|                |-------- all partition trinities
|                |    B
|                |-------- basic partition trinities
|                |    Q
|                |-------- quit current level
|                |    X
|                |-------- specify a new data
|      S
|--------- S.P. partitions
|      L
|--------- list a directory
|      Q
|--------- quit to DOS
|      X
|--------- specify a new data
```

* The function is on development.

1.2 Menu Driver and Help

DASM runs with different functions by the menu driver. The top menu is like

```
*DASM(1986): D(ecomp. S(P's P(airs T(rinities -->[HYB V.2.1]
```

The sign "-->" indicates there are other commands with this level and they can be seen by typing "-->" key. This mechanism is in loop. A submenu can be called by typing a correspondent letter and the menu display is changed to the function. For example, in DASM level, typing "P" we get "Pair function" and the menu shows:

```
Pair> A(s-s B(s-o C(i-s D(i-o Q(uit [HYB V.2.1]
```

Now, we can calculate the pairs by the four commands.

A help routine is in DASM which can be called in DASM level. Typing "H" DASM displays the help menu and some explainations about DASM. Typing a command in help menu some further explaination is given for some functions.

In DASM there is a program to list a directory. Any subdirectories or files can be listed by this command. It displays each file name in three parts:

name.extension filesize

where *name* could be a file name or a subdirectory name and *filesize* is given in byte. When a directory is larger than one page it just lists a full screen and remains a user to see next parts by typing "space-bar".

A nice pull down menu editor is embeded in DASM. It can be called by typing 'W' in DASM level. With the editor a machine table can be entered or updated very easily. In fact we can use the editor to do any editorial affairs which are related or not related to DASM.

II. Machine Functions

By the definition of machines in [18], generally speaking, we shall present the machine $M = (I, S, O, \delta, \lambda)$ with an input symbol $x \in I$ while it is in some state, say $s \in S$. The machine then outputs $\lambda(s,x)$ while it moves to state $\delta(s,x)$. This notion is somewhat cumbersome and we shall introduce the idea of mappings (or functions) induced by the input.

From the viewpoint of inputs, the machine functions, δ and λ, can be considered as sets of functions induced by all inputs :

$$\delta = \{\delta_x \mid \delta_x : S \to S \text{ and } x \in I\} \quad \text{and} \quad \lambda = \{\lambda_x \mid \lambda_x : S \to O \text{ and } x \in I\}$$

where δ_x and λ_x are defined by

$$\forall s \in S \quad \forall x \in I : \delta_x(s) = \delta(s,x) \quad \text{and} \quad \lambda_x(s) = \lambda(s,x).$$

The δ_x and λ_x are called the *next-state function* and *output function*, respectively, *with respect to input* x.

Finally, we make notation as follows:

$$s\delta_x = \delta_x(s) = \delta(s,x) \tag{2.1}$$

$$s\lambda_x = \lambda_x(s) = \lambda(s,x) \tag{2.2}$$

for all $s \in S$ and $x \in I$.

Based on (2.1) and (2.2) we can derive some other functions.

Let A be a set. The power set of A is defined as set $\{a \mid a \subseteq A\}$ and is denoted by 2^A because it has an interesting property: $|2^A| = 2^{|A|}$. Therefore, in other words, 2^A is the set of all subsets of A. Let S and O be sets of states and outputs of a machine. For power sets 2^S and 2^O, we have the following block functions:

$$\bar{\delta}_x : 2^S \to 2^S \quad \text{and} \quad \bar{\lambda}_x : 2^S \to 2^O$$

are defined by $Q\bar{\delta}_x = \{q\delta_x \mid q \in Q \subseteq S\}$ (2.3)

$$Q\bar{\lambda}_x = \{q\lambda_x \mid q \in Q \subseteq S\} \tag{2.4}$$

where $x \in I$.

If $x \subseteq I$, then $\bar{\delta}_x$ and $\bar{\lambda}_x$ are defined by

$$Q\bar{\delta}_x = \{q\bar{\delta}_{x_i} \mid q \in Q \wedge x_i \in x\} \qquad (2.5)$$

$$Q\bar{\lambda}_x = \{q\bar{\lambda}_{x_i} \mid q \in Q \wedge x_i \in x\} \qquad (2.6)$$

A block function maps a subset of S into a subset of S or O.

Let A be a collection of n-arrangements of the state set [26], and let B be a collection of n-arrangements of the output set, and $x \in I$. Then vector functions,

$$\vec{\delta}_x : A \to A \qquad \vec{\lambda}_x : A \to B$$

are defined for any arrangement a in A

$$a = (a_1 a_2 \ldots a_n)$$

$$a\vec{\delta}_x = (a_1\delta_x)(a_2\delta_x)\ldots(a_n\delta_x) \qquad (2.7)$$

$$a\vec{\lambda}_x = (a_1\lambda_x)(a_2\lambda_x)\ldots(a_n\lambda_x) \qquad (2.8)$$

It is obvious that $\vec{\delta}$ keeps n endpoints of n tracks of a machine under input x. When n=1 and $x \in I^*$ it keeps the endpoint under the input sequence

Let $x = x_1 x_2 \ldots x_k \in I^*$, $s \in S$. Then, track functions

$$\tilde{\delta}_x : S \to S^* \quad \text{and} \quad \tilde{\lambda}_x : S \to O^*$$

are defined by

$$s\tilde{\delta}_x = s\tilde{\delta}_{x_1 \ldots x_k}$$
$$= (s\vec{\delta}_{x_1})(s\vec{\delta}_{x_1 x_2})\ldots(s\vec{\delta}_x) \qquad (2.9)$$

and

$$s\tilde{\lambda}_x = s\tilde{\lambda}_{x_1 \ldots x_k}$$
$$= (s\lambda_{x_1})(s\vec{\delta}_{x_1}\lambda_{x_2})\ldots(s\vec{\delta}_{x_1 \ldots x_{k-1}}\lambda_{x_k}). \qquad (2.10)$$

Obviously, $s\tilde{\delta}_x$ and $s\tilde{\lambda}_x$ record the tracks of a machine under input sequence x.

From the definition we can derive many interesting properties of the functions. The detailed proofs can be found in [20].

PROPERTIES:

1) $\forall x,y \epsilon I,\ \forall s \epsilon S$

$$s\delta_{xy} = (s\delta_x)\delta_y = s\delta_x\delta_y; \tag{2.11}$$

$$s\lambda_{xy} = (s\delta_x)\lambda_y = s\delta_x\lambda_y; \tag{2.12}$$

2) $\forall\ x = x_1x_2\ldots x_k\ \epsilon\ I^*,\ x_i \epsilon I,\ 1 \le i \le k,$

$$s\delta_x = s\delta_{x_1x_2\ldots x_k} = s\delta_{x_1}\delta_{x_2}\ldots\delta_{x_k} \tag{2.13}$$

$$s\lambda_x = s\lambda_{x_1x_2\ldots x_k} = (s\delta_{x_1}\ldots\delta_{x_{k-1}})\lambda_{x_k} \tag{2.14}$$

3) if $x = \varepsilon \epsilon I^*$, then $\forall s \epsilon S$: $s\delta_x = s$ and $s\lambda_x = \varepsilon \epsilon O^*$ (2.15)

4) $\forall\ Q_1,Q_2 \subseteq S$ and $\forall\ x_1,x_2 \subseteq I$:

i) $Q_1 \subseteq Q_2 \Rightarrow Q_1\overline{\delta}_{x_1} \subseteq Q_2\overline{\delta}_{x_1}\ \wedge\ Q_1\overline{\lambda}_{x_1} \subseteq Q_2\overline{\lambda}_{x_1}$ (2.16)

ii) $x_1 \subseteq x_2 \Rightarrow Q_1\overline{\delta}_{x_1} \subseteq Q_1\overline{\delta}_{x_2}\ \wedge\ Q_1\overline{\lambda}_{x_1} \subseteq Q_1\overline{\lambda}_{x_2}$ (2.17)

iii) $x_1 \subseteq x_2\ \wedge\ Q_1 \subseteq Q_2$
$\Rightarrow Q_1\overline{\delta}_{x_1} \subseteq Q_2\overline{\delta}_{x_2}\ \wedge\ Q_1\overline{\lambda}_{x_1} \subseteq Q_2\overline{\lambda}_{x_2}$ (2.18)

5) $\forall\ Q_1,Q_2 \subseteq S,\ \forall\ x \epsilon I$:

$$Q_1\overline{\delta}_x \cup Q_2\overline{\delta}_x = (Q_1 \cup Q_2)\overline{\delta}_x \tag{2.19}$$

$$Q_1\overline{\lambda}_x \cup Q_2\overline{\lambda}_x = (Q_1 \cup Q_2)\overline{\lambda}_x \tag{2.20}$$

6) $\forall\ Q \subseteq S\ \forall\ x_1,x_2 \subseteq I$:

$$Q\overline{\delta}_{x_1} \cup Q\overline{\delta}_{x_2} = Q\overline{\delta}_{(x_1 \cup x_2)}, \tag{2.21}$$

$$Q\overline{\lambda}_{x_1} \cup Q\overline{\delta}_{x_2} = Q\overline{\lambda}_{(x_1 \cup x_2)}. \tag{2.22}$$

7) $\forall\ Q_1,Q_2 \subseteq S,\ \forall x \epsilon I, \forall x_1,x_2 \subseteq I$:

$$(Q_1 \cap Q_2)\overline{\delta}_x \subseteq Q_1\overline{\delta}_x \cap Q_2\overline{\delta}_x, \tag{2.23}$$

$$Q_1\overline{\delta}_{(x_1 \cap x_2)} \subseteq Q_1\overline{\delta}_{x_1} \cap Q_1\overline{\delta}_{x_2}; \tag{2.24}$$

$$(Q_1 \cap Q_2)\overline{\lambda}_x \subseteq Q_1\overline{\lambda}_x \cap Q_2\overline{\lambda}_x, \tag{2.25}$$

$$Q_1\overline{\lambda}_{(x_1 \cap x_2)} \subseteq Q_1\overline{\lambda}_{x_1} \cap Q_1\overline{\lambda}_{x_2}. \tag{2.26}$$

8) $\forall\ Q_1, Q_2 \subseteq S, \forall\ x_1, x_2 \subseteq I$:

$$(Q_1 \cup Q_2)\overline{\delta}_{(x_1 \cup x_2)} = \bigcup_{i,j=1,2} Q_i \overline{\delta}_{x_j}, \tag{2.27}$$

$$(Q_1 \cup Q_2)\overline{\lambda}_{(x_1 \cup x_2)} = \bigcup_{i,j=1,2} Q_i \overline{\lambda}_{x_j}. \tag{2.28}$$

9) $\forall\ x_1, x_2 \epsilon I,\ \forall\ s \epsilon S$:

$$s\tilde{\delta}_{x_1 x_2} = (s\delta_{x_1})(s\delta_{x_1 x_2}) \tag{2.29}$$

$$s\tilde{\lambda}_{x_1 x_2} = (s\lambda_{x_1})(s\lambda_{x_1 x_2}) \tag{2.30}$$

10) $\forall\ x_1, x_2 \epsilon I^*,\ \forall\ s \epsilon S$:

$$s\tilde{\delta}_{x_1 x_2} = (s\tilde{\delta}_{x_1})(s\delta_{x_1}\tilde{\delta}_{x_2}) \tag{2.31}$$

$$s\tilde{\lambda}_{x_1 x_2} = (s\tilde{\lambda}_{x_1})(s\delta_{x_1}\tilde{\lambda}_{x_2}) \tag{2.32}$$

11) $\forall\ x, y \epsilon I$ and $a \epsilon A$:

$$a\vec{\delta}_{xy} = (a\vec{\delta}_x)\vec{\delta}_y, \tag{2.33}$$

$$a\vec{\lambda}_{xy} = (a\vec{\delta}_x)\vec{\lambda}_y. \tag{2.34}$$

12) $\forall\ x = x_1 \ldots x_n \epsilon I^*$ and $a \epsilon A$:

$$a\vec{\delta}_x = (\ldots((a\vec{\delta}_{x_1})\vec{\delta}_{x_2})\ldots\vec{\delta}_{x_n}) \tag{2.35}$$

$$= a\vec{\delta}_{x_1}\vec{\delta}_{x_2}\ldots\vec{\delta}_{x_n}$$

$$a\vec{\lambda}_x = a\vec{\delta}_{x_1 \ldots x_{n-1}}\vec{\lambda}_{x_n}. \tag{2.36}$$

If $x = \varepsilon \epsilon I^*$, then $a\vec{\delta}_x = a \quad a\vec{\lambda}_x = \varepsilon$. (2.37)

III. DATA STRUCTURE

In DASM, data structure is quite simple. The data we need for DASM is from two aspects: declairation of a machine, and representation of partitions. Here we explain them in details.

3.1 Description of a Machine

Considering two types of machines, Mealy and Moore Machines, we apply arrays for the storages of them because an array exactly corresponds to a machine table.

We take

NumStates, NumInputs and NumOutputs

as global variables to represent the number of states, number of inputs and number of outputs of a machine, respectively. The variables are used throughout all of the programs in the package. That is,

```
NumStates  = Number of States,
NumInputs  = Number of Inputs,                          (3.1)
NumOutputs = Number of Outputs.
```

With above global variables, we have

```
type Machtable
     = array[1..NumStates, 0..NumInputs] of integer       (3.2)
```

where the two upbounds of the array vary with the size of a machine. In the array, a column just is an input column in the machine table and a row a state row.

The definition of a machine table array is suitable to both Mealy and Moore machines. In case of Mealy machines, we need two arrays, one for transition table and one for output table. And in the case of Moore machine one array is enough both for state transition and outputs where outputs column is the column 0 of machine table array. The

entries of the arrays are integers which represent states and outputs. Considering the situation of incompletely specified machines, we use integer 0 to indicate a don't care entry.

SUMMARY:

We have made

```
NumStates     = number of states
NumInputs     = number of inputs
NumOutputs    = number of outputs
Input set     = {1,2,..., NumInputs}
State set     = {0,1,..., NumStates}
Output set    = {0,1,..., NumOutputs}
don't care    = 0
Moore outputs = Machtable[-,0]
```

3.2 Enter a Machine Table

A machine table array is the internal form of a machine in DASM. We have to enter a practical machine table into an array. The way DASM applied is to use an DOS Edit program to edit a machine table in text form. The reason for this way is that one can easily enter or update a machine table by a familiar edit routine. DASM will read this text file and transform it into a machine table array. There are some parameters we should tell DASM, such as machine type, Mealy or Moore machine, and the size of a machine. They are put in the first two lines of a machine table text. The first one is 1 or 0. Digital 1 indicates the Mealy machine and 0 the Moore machine. The second line consists of three numbers which represent the numbers of state, input and output in order. The rest of text are the entries: next state, and/or outputs which are in the exactly same form as a machine table. The state rows and input columns are implicitely recognized by DASM.

EXAMPLE:

A machine table: data form of the machine:

```
------------------------
state  input                         1
         1   2   3   4               4 4 3
........................
   1    1/2 3/3 2/2 4/3      ==>     1 2  3 3  2 2  4 3
   2    2/1 2/2 3/1 0/0              2 1  2 2  3 1  0 0
   3    4/3 0/0 3/2 1/3              4 3  0 0  3 2  1 3
   4    3/1 3/1 2/2 2/2              3 1  3 1  2 2  2 2
------------------------
   * entry = next-state/output
   * entry 0 = don't care
```

SUMMARY:

Mealy machine data structure:

```
1
NumStates   NumInputs   NumOutputs
next-state/output ....   next-state/output
          .
          .
          .
next-state/output ....   next-state/output
```

Moore machine data structure:

```
0
NumStates   NumInputs   NumOutputs
next-state ... next-state    output
    .              .           .
    .              .           .
    .              .           .
next-state ... next-state    output
```

3.3 Representation of Partitions.

A basic data involved in DASM running is a partition. Some other forms of data can be generated by partitions easily. For example, a partition pair consists of two partitions and a trinity of three partitions. Thus, we make a definition of data representation of a partition to cover all data structures in DASM. As we know, a partition is constructed by blocks. So, we can use an array for a partition like

type partition = array [1..Num] of block (3.3)

where Num could be one of NumStates, NumInputs or NumOutputs depending on the partition type. Furthermore, a block just is a set of elements. In particular, a block is a set of some states, of some inputs or of some outputs. In case of a state partition,

type block = set of state. (3.4)

and state is an integer type bounbed in

type state = 0..NumStates. (3.5)

DASM uses an array for the storage of partitions to meet the different purposes in the package,

PIarray = array [0..MaxPartition] of partition (3.6)

where MaxPartition is depended on the memory size of a system. The array can be used not only for partitions, but also for pairs and trinities.

SUMMARY:

```
state = 0..NumStates
block = set of state
partition = array [1..Num] of block
PIarray = array [0..MaxPartition] of partition.
```

IV. THE SUBSTITUTION PROPERTY PARTITIONS

In this chapter we discuss an important concept that will be used throughout this paper, which is the substitution property (SP) partition.

4.1 Principle

First of all, we should explain what is a partition.

Let Q be any set with n finite elements,

$$Q = \{ q_1, q_2, \ldots, q_n \}.$$

Furthermore, let π be a set of some subsets of Q, that is,

$$\pi = \{B_1, \ldots, B_k\}$$

Set π is a partition on Q iff

$$\forall B_i, B_j \in \pi: \quad B_i \cap B_j = \phi \qquad i \neq j$$

$$\text{and} \qquad \bigcup_{i=1}^{k} B_i = Q \qquad (4.1)$$

In other words, a partition is a set of some disjointive subsets which cover all elements of Q. Subset B_i is called a block of partition π on Q.

Usually, we use $[q]_\pi$ to denote the block of π which contains the element q without explicitly indicating the name of a block. For any two partitions π, τ on Q, there are two operations defined by,

$$\pi = \{B_1, B_2, \ldots, B_k\}$$
$$\tau = \{\beta_1, \beta_2, \ldots, \beta_l\}$$

$$\pi \cdot \tau = \{A \mid \exists B_i, \beta_j: A = B_i \cap \beta_j\} \quad (4.2)$$

$$\pi + \tau = \{A \mid \forall A_0, A_1, \ldots, A_m \in \pi \cup \tau \quad A_i \cap A_{i+1} \neq \phi: \\ A = \cup\{A_0, A_1, \ldots, A_m\} \quad 0 \leq i < m\} \quad (4.3)$$

(4.2) means that the meet of a block in π and a block in τ forms one block of $\pi \cdot \tau$, if it is not an empty set; (4.3) says that continually joint any two blocks in $\pi \cup \tau$ which have at least one element in common, finally get a block of $\pi + \tau$.

EXAMPLE:

Let $S = \{1, 2, 3, 4, 5, 6, 7\}$,

$\pi = \{\overline{1,3}, \overline{2,5}, \overline{4,6,7}\}$

$\tau = \{\overline{1,2}, \overline{3,5}, \overline{4,6}, \overline{7}\}$

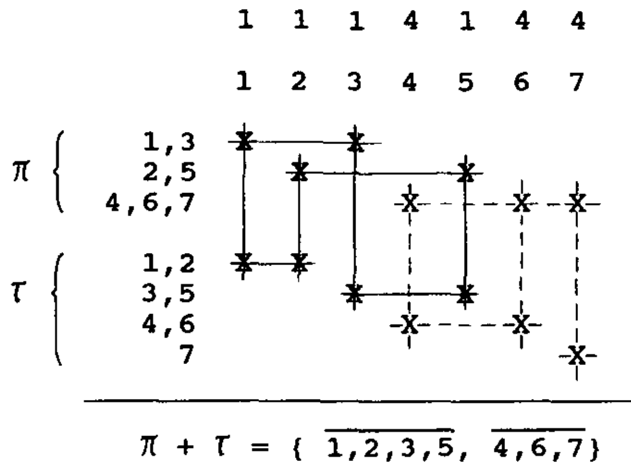
$$\pi \cdot \tau = \{\overline{1,3} \cap \overline{1,2} = \overline{1}, \overline{2,5} \cap \overline{1,2} = \overline{2}, \overline{1,3} \cap \overline{3,5} = \overline{3}, \\ \overline{2,5} \cap \overline{3,5} = \overline{5}, \overline{4,6,7} \cap \overline{4,6} = \overline{4,6}, \overline{4,6,7} \cap \overline{7} = \overline{7}\} \\ = \{\overline{1}, \overline{2}, \overline{3}, \overline{4,6}, \overline{5}, \overline{7}\}$$

$$\pi + \tau = \{\overline{1,2,3,5}, \overline{4,6,7}\}$$

where, $\overline{1,2,3,5} = \cup\{\overline{1,2}, \overline{1,3}, \overline{3,5}, \overline{2,5}\}$

$\overline{4,6,7} = \cup\{\overline{4,6}, \overline{4,6,7}, \overline{7}\}$

In [20] a calculation way of $\pi + \tau$ by hand was introduced which gave a strightforward method and guarantees the correct result obtained. For the above example, it is illustrated as follows:



For a machine M=(I, S, O, δ, λ), there are three different types of partitions, respectively on input set, state set or output set.

A *substitution property* (SP) partition is a partiton π on S of M such that

$$\forall B \in \pi \ \ \forall x \in I \ \ \exists B' \in \pi : B\bar{\delta}_x \subseteq B' \qquad (4.3)$$

Thus, a partition π has SP iff for each x the next-state function maps all blocks of π into blocks of π. On the set of all SP partitions of a machine, partition operations, + and •, are closed. That is, the sum (or product) of any two SP partitions is a SP partition. SP partition plays an important role in the algebraic theory of machines. It is the hard-core of most topics on machines which will be shown in the following sections.

4.2 Algorithm

There are several ways to calculate SP partitions of a machine. In DASM we applied the following steps:

step 1. For all pair of s,t in S, calculate the smallest SP partitions $\pi_{s,t}$.

step 2. Do partition additions on set of $\pi_{s,t}$. Any partition sum of two or more partitions is an SP partition.

4.2.1 The Smallest SP's

To calculate a smallest SP we take several procedures:

Function EXIST is used to reduce the calculations of same pairs of states. For any s,t, if EXIST(s,t) is true, this means that s,t are in the same block of an π, and $\pi_{s,t}$ has been already done.

Procedure SMALLSP is a recursive procedure which calculates the smallest SP partitions for a pair of states. Statement

PI1 := PI1 + PI1

order PI1 to meet the definition of a partition because it is not in partition form during the calls of SMALLSP.

```
FUNCTION EXIST(S1,S2: integer): boolean;
input:  two states, s1 and s2;
output: if s1 and s2 are in a block of PI1[I], EXISTS := true;
procedure:
begin var i: integer;
      glovar NumStates: integer;
      extvar PI1: partition;
      i :=1; EXIST := false;
      do i ≤ NumStates →
              if s1∈PI1[i] ∧ s2∈PI1[i] →
                      EXIST := true; i:=NumStates+1
               ▯ not(s1∈PI1[i] ∧ s2∈PI1[i]) →
                      i:=i+1
              fi
      od
end
```

```
PROCEDURE SMALLSP(S1,S2: integer, var PI1: partition)
input: two states, s1 and s2, machine table array;
output: the smallest SP partition PI1 of s1 and s2
procedure:
begin var i: integer; glovar NumInputs: integer;
      i := 1;
      do i ≤ NumInprts →
              if not(EXIST(δ[s1,i],δ[s2,i])) →
                   PI1[δ[s1,i]]:=PI1[δ[s1,i]]+[δ[s2,i]];
                   SMALLSP(δ[s1,i],δ[s2,i],PI1); i=i+1
              | EXIST(δ[s1,i],δ[s2,i]) → i:=i+1
              fi
      od
      PI1:=PI1+PI1
end

Algorithm SP
input: machine table array;
output: SP set PIarray;
procedure:
begin var i,j,p = integer; PIW : partition;
        { smallest sp set }
        i:=1; p:=0
        do i ≤ NumStates-1 →
                j:=i+1;
                do j ≤ NumStates →
                        SMALLSP(i,j,PIW);
                        if PIW not exists in PIarray →
                                p:=p+1; PIarray[p]:=PIW; j:=j+1
                        | PIW exists in PIarray → j:=j+1
                        fi
                od
        od
        { SP set }
        ALLP(1,P,P);
end
```

In above routins as well as follows, δ[] means the transition array and λ[] the output array of the machine in question. Procedure ALLP(1,P,P) is a universal procedure to do partition additions on the basic partition set PIarray[1..P] and gives the results in PIarray again by variable P which indicates the last partition position in the array. ALLP is recursive and terminates when there is no new partition arised by addition of any two partitions in the array.

4.3 Example

Machine table: a:MO124.dat shown in Fig. 4.1;
from ir. v. d. Eynden;

Procedure:

step 1. Run DASM by typing ***DASM*** under DOS prompt;

step 2. Specify the data file by typing ***a:MO124.DAT;***

step 3. Select the function of SP partitions by typing ***S;***

step 4. Specify the output file by typing ***a:MO124.SP;***

The result is in file a:MO124.sp and shown in Fig. 4.2.

Machine : a:mo124.dat

STATE	INPUT 1	2	3	4	OUTPUT
1	12	2	1	1	0
2	2	2	3	1	0
3	4	5	3	3	0
4	4	4	3	1	0
5	5	5	6	3	0
6	7	8	6	6	0
7	7	7	6	3	0
8	8	8	9	6	0
9	10	11	9	9	0
10	10	10	9	6	0
11	11	11	1	9	0
12	12	12	1	1	0

* entry=0 for don't care

Fig. 4.1 Machine a:MO124.dat

```
SP PARTITION OF a:mo124.dat
===========================
Partition  1.   {(1,3,6,9);(4,7,10,12);(2,5,8,11)}
Partition  2.   {(2,4);(1);(3);(5);(6);(7);(8);(9);(10);(11);(12)}
Partition  3.   {(2,4,5,7,8,10,11,12);(1,3,6,9)}
Partition  4.   {(5,7);(1);(2);(3);(4);(6);(8);(9);(10);(11);(12)}
Partition  5.   {(8,10);(1);(2);(3);(4);(5);(6);(7);(9);(11);(12)}
Partition  6.   {(2,4);(1);(3);(5,7);(6);(8);(9);(10);(11);(12)}
Partition  7.   {(2,4);(1);(3);(5);(6);(7);(8,10);(9);(11);(12)}
Partition  8.   {(5,7);(1);(2);(3);(4);(6);(8,10);(9);(11);(12)}
Partition  9.   {(2,4);(1);(3);(5,7);(6);(8,10);(9);(11);(12)}
------------END------------
```

Fig. 4.2 The SP partitions of a:MO124.dat

In the result file a partition is represented in a little different way. The elements in the same block are bounded by a pair of parentheses and separated by commas. Blocks in a partition are bounded by a pair of braces and separated by semicolons. For example, partition

$$\{\overline{1,2},\overline{3,4},\overline{5,6,7}\}$$

is written as

{(1,2);(3,4);(5,6,7)}

in the result file.

V. PARTITION PAIRS

5.1 Principle

A partition pair is a pair of partitions with some restrictions. Due to the three sets, input, state and output, there are four different types of pairs in machine theory.

An S-S pair consists of two state partitions. One is related to present states and another to nest-states. The two state partitions satisfy the follwing condition:

Let π,τ be two partition on set S of M.

$$\pi = \{A_1, A_2, \ldots, A_m\}$$
$$\tau = \{B_1, B_2, \ldots, B_n\}$$

(π,τ) is an S-S pair

$$\Longleftrightarrow \forall A\in\pi \quad \forall x\in I \quad \exists B\in\tau: \quad A\bar{\delta}_x \subseteq B \qquad (5.1)$$

Thus, if every block of π can be mapped into a block of τ by every input of the machine, π and τ form an S-S pair. That (π,τ) is an S-S pair indicates that for the states in the same block of π, their next-states still in the same block of τ. An S-S pair gives the information dependence between present states and next states of a machine.

By the same way we can have the definitions of other three pairs. Let π be a partition on state set S, and τ on input set I, Ω on output set O.

Then,

(π,Ω) is an S-O pair

$$\Longleftrightarrow \forall A\in\pi \quad \forall x\in I \quad \exists C\in\Omega : \quad A\bar{\lambda}_i \subseteq C \qquad (5.2)$$

(τ,π) is an I-S pair

$$\Longleftrightarrow \forall B\in\tau \quad \forall s\in S \quad \exists A\in\pi : \quad s\bar{\delta}_B \subseteq A \qquad (5.3)$$

(τ,Ω) is an I-O pair

$$\Longleftrightarrow \forall B\in\tau \quad \forall s\in S \quad \exists C\in\Omega : \quad s\bar{\lambda}_B \subseteq C \qquad (5.4)$$

Again, the three pairs depict the information flow dependences between two of inputs, states and outputs. Pairs are important concepts in state assignment, simplification etc. which will be discussed later. There are two operations on the set of some pairs. One is pair addition and another pair multiplication with following definitions.

Let (π_1,τ_1) and (π_2,τ_2) be pairs with same attribution

$$(\pi_1,\tau_1) + (\pi_2,\tau_2) = (\pi_1+\pi_2,\ \tau_1+\tau_2) \qquad (5,5)$$

$$(\pi_1,\tau_1) * (\pi_2,\tau_2) = (\pi_1\cdot\pi_2,\ \tau_1\cdot\tau_2) \qquad (5,6)$$

The operations are closed on the set of pairs. That is, the pair sum and pair procuct of any two pairs are pairs. We should mention it that the conclusion is hold only fo the pairs with same attibution. For example, the pair sum or product of two S-S pairs is an S-S, and so on.

For the two components of a pair we have special names for them. The first one is M-partition and the second m-partition. Let (π,τ) be a pair. Then, π is partition of τ and τ is m-partition of π. Pair (π,τ) can be called an Mm pair.

5.2 Algorithm

In DASM, a pair set was found by the algorithm below.

```
Algorithm PAIR (PairType: CHAR);
input: PairType = A for S-S pair
                = B for S-O pair
                = C for I-S pair
                = D for I-O pair;
       Machine table arrays;
output: pair set of PairType
procedure:
begin var i: integer; glovar CurrentPI: integer;
        BmPIS2(PairType, 1, CurrentPI);
        ALLP2(PairType, 1, CurrentPI, CurrentPI);
        do i ≤ CurrntPI →
```

```
            MPI(PairType, PIarray[i], PIW);
            writepair( PIW, PIarray[i]);
            i := i+1
        od
end
```

In the algorithm, procedure BmPIS2 calculates all basic m-partitions; ALLP2 does partition additions on the m-partions to get more m-partitions, and MPI gives a M-partition for one m-partition in the set. Writepair writes the pair on a file, a disk or a printer.

5.3 Example

Machine table: a:me69fo.dat;
shown in Fig. 5.1

Procedure:

step 1. Run DASM by typing ***DASM*** under DOS prompt;

step 2. Select the function of partition pairs by typing ***P;***

step 3. Specify the data file by typing ***a:me69fo.dat;***

step 4. Select the function of pairs by typing ***A,B,C,D;***

step 5. Specify the output file by typing ***a:me69fo.pp;***

type b:me69fo.pio
Machine: a:me69fo.dat

STATE	INPUT 1	2	3	4	5	6
1	2/3	3/2	5/8	4/9	1/4	3/4
2	3/3	1/2	4/7	5/8	3/4	2/6
3	1/3	2/2	6/9	6/7	2/4	1/5
4	2/8	1/9	1/4	3/4	5/3	6/2
5	1/7	2/8	3/4	2/6	6/3	4/2
6	3/9	3/7	2/4	1/5	4/3	5/2

* entry=(state/ourput)
* entry=0 for don't care

Fig.5.1 Machine me69fo.dat

The result is in file a:me69fo.pp and shown in Figs. 5.2-5.5.

```
Mm S-S partition pair:
======================
Pair  1.   [{(1,2);(3);(4);(5);(6)},
            {(1,2,3);(4,5);(6)}]
Pair  2. * [{(1,2,3);(4,5,6)},
            {(1,2,3);(4,5,6)}]
Pair  3.   [{(1,4);(2);(3);(5);(6)},
            {(1,3,4,5,6);(2)}]
Pair  4.   [{(1,6);(2);(3);(4);(5)},
            {(1,4);(2,3,5);(6)}]
Pair  5.   [{(1,6);(2,4);(3,5)},
            {(1,4);(2,3,5,6)}]
Pair  6.   [{(1,2,6);(3);(4);(5)},
            {(1,2,3,4,5);(6)}]
Pair  7.   [{(1);(2);(3,4);(5);(6)},
            {(1,2,3,5,6);(4)}]
Pair  8.   [{(1);(2);(3,5);(4);(6)},
            {(1,4);(2,3,6);(5)}]
( "*" : pair with SP )
# This machine totally has 1 SP partitions
----------END----------
```

Fig. 5.2 S-S pairs

```
Mm I-S partition pair:
----------------------
Pair  1.   [{(1,2);(3);(4);(5);(6)},
            {(1,2,3);(4);(5);(6)}]
Pair  2.   [{(1,5);(2);(3);(4);(6)},
            {(1,2,5,6);(3,4)}]
Pair  3.   [{(1,6);(2,3);(4,5)},
            {(1,4);(2,3,5,6)}]
Pair  4.   [{(1);(2,4,5);(3);(6)},
            {(1,3,4,5);(2,6)}]
Pair  5.   [{(1);(2,6);(3);(4,5)},
            {(1,2,4,6);(3,5)}]
Pair  6.   [{(1,2);(3,4);(5);(6)},
            {(1,2,3);(4,5);(6)}]
Pair  7.   [{(1);(2);(3,5);(4);(6)},
            {(1,5);(2,3,4,6)}]
Pair  8.   [{(1);(2);(3,6);(4);(5)},
            {(1,6);(2,3,4,5)}]
Pair  9.   [{(1);(2);(3);(4,5);(6)},
            {(1,4);(2,6);(3,5)}]
Pair 10.   [{(1,2);(3,4);(5,6)},
            {(1,2,3);(4,5,6)}]
----------END----------
```

Fig. 5.3 I-S pairs

```
Mm S-O partition pair:
----------------------
Pair  1.   [{(1,2);(3);(4,5);(6)},
            {(1);(2);(3);(4,6);(5);(7,8,9)}]
Pair  2.   [{(1,3);(2);(4,6);(5)},
            {(1);(2);(3);(4,5);(6);(7,8,9)}]
Pair  3.   [{(1,4);(2);(3);(5);(6)},
            {(1);(2,3,4,8,9);(5);(6);(7)}]
Pair  4.   [{(1,5);(2);(3);(4);(6)},
            {(1);(2,3,4,7,8);(5);(6,9)}]
Pair  5.   [{(1,3,4,6);(2);(5)},
            {(1);(2,3,4,5,7,8,9);(6)}]
Pair  6.   [{(1);(2,3);(4);(5,6)},
            {(1);(2);(3);(4);(5,6);(7,8,9)}]
Pair  7.   [{(1);(2,4);(3);(5);(6)},
            {(1);(2,6,9);(3,4,7,8);(5)}]
Pair  8.   [{(1);(2,5);(3);(4);(6)},
            {(1);(2,6,8);(3,4,7);(5);(9)}]
Pair  9.   [{(1);(2,6);(3);(4);(5)},
            {(1);(2,3,4,6,7,9);(5,8)}]
Pair 10.   [{(1);(2);(3,5);(4);(6)},
            {(1);(2,5,8);(3,4,6,7,9)}]
Pair 11.   [{(1);(2);(3,6);(4);(5)},
            {(1);(2,5,7);(3,4,9);(6);(8)}]
Pair 12.   [{(1,2,3);(4,5,6)},
            {(1);(2);(3);(4,5,6);(7,8,9)}]
Pair 13.   [{(1,2,4,5);(3);(6)},
            {(1);(2,3,4,6,7,8,9);(5)}]
Pair 14.   [{(1,4);(2,3);(5,6)},
            {(1);(2,3,4,7,8,9);(5,6)}]
```

Fig. 5.4 S-O pairs

```
Mm I-O partition pair:
----------------------
Pair  1.   [{(1,2);(3);(4);(5);(6)},
            {(1);(2,3);(4);(5);(6);(7,8,9)}]
Pair  2.   [{(1,3,5);(2);(4);(6)},
            {(1);(2);(3,4,7,8,9);(5);(6)}]
Pair  3.   [{(1,3,4,5);(2);(6)},
            {(1);(2);(3,4,5,6,7,8,9)}]
Pair  4.   [{(1,6);(2);(3,4);(5)},
            {(1);(2,7,8,9);(3,4,5,6)}]
Pair  5.   [{(1);(2,3);(4);(5);(6)},
            {(1);(2,4,7,8,9);(3);(5);(6)}]
Pair  6.   [{(1);(2,3,4,6);(5)},
            {(1);(2,4,5,6,7,8,9);(3)}]
Pair  7.   [{(1);(2,5);(3);(4);(6)},
            {(1);(2,4);(3,7,8,9);(5);(6)}]
Pair  8.   [{(1);(2);(3,4);(5);(6)},
            {(1);(2);(3);(4,5,6);(7,8,9)}]
Pair  9.   [{(1);(2);(3,6);(4);(5)},
            {(1);(2,4,8);(3);(5,9);(6,7)}]
Pair 10.   [{(1);(2);(3);(4);(5,6)},
            {(1);(2,3);(4,5,6);(7);(8);(9)}]
Pair 11.   [{(1,2,3,5);(4);(6)},
            {(1);(2,3,4,7,8,9);(5);(6)}]
Pair 12.   [{(1,2);(3,4);(5,6)},
            {(1);(2,3);(4,5,6);(7,8,9)}]
Pair 13.   [{(1);(2,5);(3,4);(6)},
            {(1);(2,4,5,6);(3,7,8,9)}]
----------END----------
```

Fig. 5.5 I-O pairs

VI. PARTITION TRINITIES

6.1 Principle

A partition trinity (PT) is a triple of three partitions on input set, state set and ouprt set respectively. The three partitions are restricted by some conditions defined as follows.

Let π be a partition on S, τ on input set I and Ω on output set O of a machine M. Then,

$$
\begin{aligned}
&(\tau,\pi,\Omega) \text{ is a partition trinity} \\
\Longleftrightarrow\ & \forall A\in\pi \ \ \forall B\in\tau \ \ \exists A'\in\pi \ \ \exists C\in\Omega: \\
& A\overline{\delta}_B \subseteq A' \ \wedge\ A\overline{\lambda}_B \subseteq C. \qquad (6.1)
\end{aligned}
$$

(6.1) is a general definition on any machines, completely or incompletely specified machines. For a completely specified machine, we can prove that (6.2) is equivalence to (6.1).

$$
\begin{aligned}
& (\tau,\pi,\Omega) \text{ is an PT} \\
\Longleftrightarrow\ & (\tau,\ \pi) \text{ is an I-S pair} \\
\wedge\ & (\tau,\ \Omega) \text{ is an I-O pair} \\
\wedge\ & (\pi,\ \pi) \text{ is an S-S pair} \\
\wedge\ & (\pi,\ \Omega) \text{ is an S-O pair} \qquad (6.2)
\end{aligned}
$$

Therefore, (6.2) also provides another way to determine an PT based on the calculations of pairs. The proof of (6.2) is omitted and can be found the details in [20]

On the set of partition trinities there exist two operations, partition addition $\oplus$ and partition multiplication $\odot$. Let (τ_1,π_1,Ω_1) and (τ_2,π_2,Ω_2) be any two PT's on a machine M. Then,

$$
\begin{aligned}
& (\tau_1,\pi_1,\Omega_1) \oplus (\tau_2,\pi_2,\Omega_2) \\
=\ & (\tau_1+\tau_2,\ \pi_1+\pi_2,\ \Omega_1+\Omega_2) \\
\\
& (\tau_1,\pi_1,\Omega_1) \odot (\tau_2,\pi_2,\Omega_2) \\
=\ & (\tau_1*\tau_2,\ \pi_1*\tau_2,\ \Omega_1*\Omega_2) \qquad (6.3)
\end{aligned}
$$

It was proved that two operations are closed on the set of all PT's of M [20]. And there are some interesting properties on the set of all PT's. Let T be the set of all PT's of M. Then,

$$<T; \oplus, \odot, T_I, T_0>$$

is an algebraic system called trinity algebra. The two elements, T_I and T_0, are identity and zero trinities defined by

$$T_I=(\tau(I),\ \pi(I),\ \Omega(I))$$

and $$T_0=(\tau(O),\ \pi(O),\ \Omega(O)) \quad (6.4)$$

respectively. It says if the three partitions of an PT are identity (or zero) partitions, the PT is an identity (or zero) trinity. An PT gives the information flow dependence between inputs, states and outputs simultaneously. The concept of partition trinities is core of full-decompositions to be disscused late.

6.2 Algorithm

The caculation for PT's can be done by (6.1) or (6.2). We noted that in (6.2) an S-S pair (π, π) means that π has SP. Then, that calculate SP partition first and further get other partitions is an easy way for the purpose. A complete description of this way is in the following algorithm.

Algorithm PT;
input: machine table arrays;
output: PT set;
procedure:

```
begin var i: integer;
      get all N SP partitions;
      do i≤N →
              MPI(3, PIarray[i], PISO1);
              MPI(2, PIarray[i], PIi);
              RMPI(4,PIi,PIO2);
```

```
            if PIO1=PIO2 →
                    (PIi, PIarray[i], PIO1) is an PT;
                    i := i+1
             ▯ PIO1≠PIO2 → i := i+1
            fi
      od
end
```

In above algorithm,

```
get all N SP partitions;
```

is a procdure discussed before, which produce all SP partitions kept in PIarray[1..N]. After getting the SP partitions, we search them one by one with the following step:

```
RMPI(3, PIarray[i], PIO1);
```

finds the m-partition of PIarray[i]. First parameter instructs the procedure to work on the I-S pair.

```
RMPI(4,PIi,PIO2);
```

produce the m-partition for PIi and first parameter means this is an I-O pair. Finally, we have to test whether PIO1 and PIO2 are the same partition. If they are identical it means (PIi, PIarray[i], PIO1) is an PT. Otherwize not.

An alternative way for last procedure call to find the m-partition PIO2 of PIi is to judge if (PIi, PIO1) is an I-S pair which can be easily implemented by making a new procedure.

In the algorithm, in order to simplify the description, we ignore parameters which are used to speed up the calculation and can be easily understood from source programs.

6.3 Example

Machine table: a:me8886.dat shown in Fig. 6.1.

Procedure:

step 1. Run DASM by typing ***DASM*** uunder DOS prompt;

step 2. Select the function of trinity by typing ***T***;

step 3. Specify the data file by typing ***a:me8886.dat***;

step 4. Select the function of basic PT's by typing ***B***;

step 5. Specify the output file by typing ***a:me8886.bpt***.

```
type b:me8886.bpt

Machine:a:me8886.dat
```

STATE	INPUT 1	2	3	4	5	6	7	8
1	6/1	5/2	8/1	7/2	2/5	1/6	4/5	3/6
2	5/2	6/2	7/2	8/2	1/6	2/6	3/6	4/6
3	8/1	7/2	6/3	5/4	4/5	3/6	2/7	1/8
4	7/2	8/2	5/4	6/4	3/6	4/6	1/8	2/8
5	2/5	1/6	4/5	3/6	6/5	5/6	8/5	7/6
6	1/6	2/6	3/6	4/6	5/6	6/6	7/6	8/6
7	4/5	3/6	2/7	1/8	8/5	7/6	6/7	5/8
8	3/6	4/6	1/8	2/8	7/6	8/6	5/8	6/8

```
* entry=(state/output)
* entry=0 for don't care
```

Fig. 6.1 Machine a:me8886.dat

```
THE BASIC NONTRIVIAL PARTITION TRINITY OF MACHINE M:
----------------------------------------------------
PT 1.  [{(1,2);(5,6);(7,8);(3,4)},
        {(1,2);(3,4);(5,6);(7,8)},
        {(1,2);(3,4);(5,6);(7,8)}]
PT 2.  [{(1,3);(6,8);(5,7);(2,4)},
        {(1,3);(2,4);(5,7);(6,8)},
        {(1,3);(2,4);(5,7);(6,8)}]
PT 3.  [{(1,4);(6,7);(5,8);(2,3)},
        {(1,4);(2,3);(5,8);(6,7)},
        {(1,2,3,4);(5,6,7,8)}]
PT 4.  [{(1,5);(2,6);(4,8);(3,7)},
        {(1,5);(2,6);(3,7);(4,8)},
        {(1,5);(2,6);(3,7);(4,8)}]
PT 5.  [{(1,6);(2,5);(3,8);(4,7)},
        {(1,6);(2,5);(3,8);(4,7)},
        {(1,2,5,6);(3,4,7,8)}]
PT 6.  [{(1,7);(4,6);(3,5);(2,8)},
        {(1,7);(2,8);(3,5);(4,6)},
        {(1,3,5,7);(2,4,6,8)}]
PT 7.  [{(1,2,3,4);(5,6,7,8)},
        {(1,2,3,4);(5,6,7,8)},
        {(1,2,3,4);(5,6,7,8)}]
PT 8.  [{(1,2,5,6);(3,4,7,8)},
        {(1,2,5,6);(3,4,7,8)},
        {(1,2,5,6);(3,4,7,8)}]
PT 9.  [{(1,3,5,7);(2,4,6,8)},
        {(1,3,5,7);(2,4,6,8)},
        {(1,3,5,7);(2,4,6,8)}]
```

Fig.6.2 PT's of a:me8886.dat

The result is in file a:me8886.bpt and shown in Fig. 6.2.

VII. EQUIVALENT STATES

7.1 Principle

Equivalent states are states that for any input sequence applied to the machine, the output sequences are identical. Therefore, equivalent states are undistinguishable states in the behaviour of machines.

Symbolically, we have

$$s \equiv t \iff \forall\, x \in I^*:\quad s\tilde{\lambda}_x = t\tilde{\lambda}_x \tag{7.1}$$

The determination of two states, of course, can be carried out by (7.1). But, to search for all input sequence is time-consuming and, sometimes, probably impossible. We should try other way for this. We noted that there is the concept of output consistent SP (O.C.S.P) partitions defined as follows.

Let π be an SP partition. If for all x in I and all pair of s and t in S,

$$[s] = [t] \implies s\lambda_x = t\lambda_x \tag{7.2}$$

then π is an O.C.S.P. In other words, an O.C.S.P partition is an SP partition in which all elements in the same block have identical outputs.

In (7.1), from the definition of $\tilde{\lambda}$, we know, for any input sequence

$$X = x_1x_2....x_n \quad \text{and} \quad s,t \in S$$

$$\begin{aligned} s\tilde{\lambda}_x &= s\tilde{\lambda}_{x_1x_2\ldots x_n} \\ &= (s\lambda_{x_1})(s\tilde{\delta}_{x_1}\lambda_{x_2})\ldots(s\tilde{\delta}_{x_1\ldots x_{n-1}}\lambda_{x_n}) \\ t\tilde{\lambda}_x &= (t\lambda_{x_1})(t\tilde{\delta}_{x_1}\lambda_{x_2})\ldots(t\tilde{\delta}_{x_1\ldots x_{n-1}}\lambda_{x_n}) \end{aligned} \tag{7.3}$$

Since π has SP, for any input sequence X in I^*

$$[s\delta_X]_\pi = [t\delta_X]_\pi \tag{7.4}$$

and from (7.2),

$$s\vec{\delta}_X\lambda_x = t\vec{\delta}_X\lambda_x. \qquad (7.5)$$

Repeatedly applying (7.4) and (7.5) to (7.3) we obtain

$$s\tilde{\lambda}_x = t\tilde{\lambda}_x$$

which means that s and t are equivalent states from (7.1). Conclusively, we have proved that

> If π is an O.C.S.P, then the blocks of π are sets of states which are equivalent to each other. (7.6)

The conclusion shows a way to us for finding equivalent states: find an S.P partition and check if it is output consistent. If it is then the partition gives the equivalent state classes.

7.2 Algorithm.

An algorithm based on (7.6) is shown as follows.

```
Algorithm EQStates;
input: machine table arrays;
output: O.C.S.P partition set and equivalent state classes;
procedure:
begin var i,j: integer;
        get all N SP partitions;
        {check O.C. property}
        i:=1; j := 0;
        do i ≤ N  →
                if PIarraay[i] has O.C.S.P →
                        j:=j+1; PIarray[j] := PIarray[i].
                 | PIarray[i] has no O.C.S.P → skip
                fi
                i:=i+1
        od
        if j≥1 → PIarray[1] gives equivalent state classes.
         | j=0 → no equivalent states.
        fi
end
```

In the above algorithm, "get all N SP partitions" is realized by the early procedure of SP partitions. The partitions are put in PIarray in the order of increasing block numbers of partitions. This lets us get the largest class of equivalent states.

The step "PIarray[i] has O.C.S.P" is done by a small procedure by which the outputs of states in every block for all inputs are checked to see if they are identical. Once failure for some block, the procedure stops and returns false of O.C.S.P for the SP partition.

7.3 Example

Machine table: a:me1242.dat shown in Fig. 7.1.

Procedure:

step 1. Run DASM by typing ***DASM*** under DOS prompt;

step 2. Specify the data file by typing ***a:me1242.dat;***

step 3. Select the function of O.C.S.P by typing ***O;***

step 4. Specify the output file by typing ***a:me1242.osp.***

step 5. Select the function of equivalent states by typing ***E;***

step 6. Specify the output file by typing ***a:me1242.eqs.***

Machine: a:me1242.dat

STATE	INPUT 1	2	3	4
1	3/1	4/1	2/1	4/1
2	7/1	5/1	8/1	5/1
3	7/1	5/1	8/1	5/1
4	5/1	5/1	5/1	5/1
5	6/1	6/1	6/1	6/1
6	1/1	1/1	1/1	1/1
7	9/1	6/1	10/1	6/1
8	11/1	6/1	12/1	6/1
9	1/2	1/1	1/2	1/1
10	1/2	1/1	1/2	1/1
11	1/2	1/1	1/2	1/1
12	1/2	1/1	1/2	1/1

* entry=(state/output)
* entry=0 for don't care

Fig. 7.1 a:me1242.dat

The result is in file a:me1242.osp and a:me1242.eqs shown in Figs. 7.2 and 7.3.

This example was from [21].

```
OCSP PARTITION OF a:mel242.dat
==============================
Partition  1.   {(2,3);(1);(4);(5);(6);(7);(8);(9);(10);(11);(12)}
Partition  2.   {(7,8);(9,11);(10,12);(1);(2);(3);(4);(5);(6)}
Partition  3.   {(9,10);(1);(2);(3);(4);(5);(6);(7);(8);(11);(12)}
Partition  4.   {(9,11);(1);(2);(3);(4);(5);(6);(7);(8);(10);(12)}
Partition  5.   {(9,12);(1);(2);(3);(4);(5);(6);(7);(8);(10);(11)}
Partition  6.   {(10,11);(1);(2);(3);(4);(5);(6);(7);(8);(9);(12)}
Partition  7.   {(10,12);(1);(2);(3);(4);(5);(6);(7);(8);(9);(11)}
Partition  8.   {(11,12);(1);(2);(3);(4);(5);(6);(7);(8);(9);(10)}
Partition  9.   {(2,3);(1);(4);(5);(6);(7,8);(9,11);(10,12)}
Partition 10.   {(2,3);(1);(4);(5);(6);(7);(8);(9,10);(11);(12)}
Partition 11.   {(2,3);(1);(4);(5);(6);(7);(8);(9,11);(10);(12)}
Partition 12.   {(2,3);(1);(4);(5);(6);(7);(8);(9,12);(10);(11)}
Partition 13.   {(2,3);(1);(4);(5);(6);(7);(8);(9);(10,11);(12)}
Partition 14.   {(2,3);(1);(4);(5);(6);(7);(8);(9);(10,12);(11)}
Partition 15.   {(2,3);(1);(4);(5);(6);(7);(8);(9);(10);(11,12)}
Partition 16.   {(7,8);(9,10,11,12);(1);(2);(3);(4);(5);(6)}
Partition 17.   {(9,10,11);(1);(2);(3);(4);(5);(6);(7);(8);(12)}
Partition 18.   {(9,10,12);(1);(2);(3);(4);(5);(6);(7);(8);(11)}
Partition 19.   {(9,10);(1);(2);(3);(4);(5);(6);(7);(8);(11,12)}
Partition 20.   {(9,11,12);(1);(2);(3);(4);(5);(6);(7);(8);(10)}
Partition 21.   {(9,11);(1);(2);(3);(4);(5);(6);(7);(8);(10,12)}
Partition 22.   {(9,12);(1);(2);(3);(4);(5);(6);(7);(8);(10,11)}
Partition 23.   {(10,11,12);(1);(2);(3);(4);(5);(6);(7);(8);(9)}
Partition 24.   {(2,3);(1);(4);(5);(6);(7,8);(9,10,11,12)}
Partition 25.   {(2,3);(1);(4);(5);(6);(7);(8);(9,10,11);(12)}
Partition 26.   {(2,3);(1);(4);(5);(6);(7);(8);(9,10,12);(11)}
Partition 27.   {(2,3);(1);(4);(5);(6);(7);(8);(9,10);(11,12)}
Partition 28.   {(2,3);(1);(4);(5);(6);(7);(8);(9,11,12);(10)}
Partition 29.   {(2,3);(1);(4);(5);(6);(7);(8);(9,11);(10,12)}
Partition 30.   {(2,3);(1);(4);(5);(6);(7);(8);(9,12);(10,11)}
Partition 31.   {(2,3);(1);(4);(5);(6);(7);(8);(9);(10,11,12)}
Partition 32.   {(9,10,11,12);(1);(2);(3);(4);(5);(6);(7);(8)}
Partition 33.   {(2,3);(1);(4);(5);(6);(7);(8);(9,10,11,12)}
-------------END-------------
```

Fig. 7.2 O.C.S.P partitions of a:mel242.dat

```
The equivalence states of a:mel242.dat
======================================
State equivalence class 1: {1}
State equivalence class 2: {2,3}
State equivalence class 3: {4}
State equivalence class 4: {5}
State equivalence class 5: {6}
State equivalence class 6: {7,8}
State equivalence class 7: {9,10,11,12}
-----------------END-----------------
```

Fig. 7.3 The equivalent state classes of a:mel242.dat

VIII. MINIMIZING MACHINES

8.1 Principle

For a completely specified machine, to minimize it is easily done by an O.C.S.P partition. Firstly, find the maximal equivalent state classes. Secondly, take each equivalent state class as a new state for the reduced machine. Finally, the output of a new state comes from one state in the correspondent block.

Let $\pi = \{A_1, A_2, \ldots A_m\}$ be the maximal O.C.S.P partition of machine M and M' be the reduced machine of M. So,

$$M = (I, S, O, \delta, \lambda)$$

and $M' = (I', S', O', \delta', \lambda')$

where $I' = I;\ O' = O;\ S' = \pi;$

$$\forall A \epsilon \pi\ \forall x \epsilon I\ \exists s \epsilon A :\quad A\delta'_x = [A\bar{\delta}_x]_\pi \tag{8.1}$$

and

$$A\lambda'_x = s\lambda_x. \tag{8.2}$$

8.2 Algorithm

Algorithm MinMach;
input: machine table arrays;
output: minimized machine M;
procedure:

```
begin var i,j, nextstate, output: integer; PIW: partition;
        get a maximal O.C.S.P partition PIW;
        write inputs;
        i := 1;
        do i ≤ |PIW| →
                get a new line;
                j := 1;
                do j≤NumInputs →
                        write(i);
                        FindaState(PIW[i], s);
```

```
                    FindaBlock(δ[s,j], PIW, B);
                    nextstate := B;
                    output := λ[s,j];
                    write(nextstate, "/" , output);
            od
      od
end
```

The procedure "get a maximal O.C.S.P partition" is basicly like the algorithm EQStates; "write inputs" makes the headers of all inputs; FindaState takes a state s from PIW[i]; FindaBlock puts the block B in PIW, which contains state s; the reduced machine is put on a printer or on a disk file.

8.3 Example

Machine table: a:me1222.dat shown in Fig. 8.1.

Procedure:

step 1. Run DASM by typing ***DASM*** under DOS prompt;

step 2. Select the function of minilizing machines by typing ***M***;

step 3. Specify the data file by typing ***a:me1222.DAT;***

step 4. Specify the output file by typing ***a:me1222.min***

Machine: a:me1222.dat

STATE	INPUT 1	2
1	2/1	3/1
2	4/1	5/1
3	6/1	7/1
4	8/1	9/1
5	10/1	11/1
6	4/1	12/1
7	10/1	11/1
8	8/1	1/1
9	10/2	1/1
10	4/1	1/1
11	2/1	1/1
12	2/1	1/1

* entry=(state/output)
* entry=0 for don't care

Fig. 8.1 a:me1222.dat

The result is in file a:me1222.min and shown in Figs. 8.2 - 8.4. This example was from [25].

```
OCSP PARTITION OF a:me1222.dat
------------------------------
Partition  1.   ((1,3,5,7,11,12);(2,6,10);(4);(8);(9))
Partition  2.   ((5,7);(1);(2);(3);(4);(6);(8);(9);(10);(11);(12))
Partition  3.   ((11,12);(1);(2);(3);(4);(5);(6);(7);(8);(9);(10))
Partition  4.   ((5,7);(1);(2);(3);(4);(6);(8);(9);(10);(11,12))
--------------END-------------
```

Fig. 8.2 O.C.S.P partitions of a:me1222.dat

```
The equivalence states of a:me1222.dat
---------------------------------------
State equivalence class 1: (1,3,5,7,11,12)
State equivalence class 2: (2,6,10)
State equivalence class 3: (4)
State equivalence class 4: (8)
State equivalence class 5: (9)
------------------END------------------
```

Fig. 8.3 The equivalent state classes

```
----------------
STATE  INPUT
        1    2
................
  1    2/1  1/1
  2    3/1  1/1
  3    4/1  5/1
  4    4/1  1/1
  5    2/2  1/1
----------------
* entry=(state/output)
* entry=0 for don't care
```

Fig. 8.4 Minimized a:me1222.dat

IX. DECOMPOSITIONS ON STATES

9.1 Principle

Decompositions of a machine were proposed in 1960's. During that time to reduce the internal components was mainly reason due to the techniques of integrated circuits. A reduction of internal component largely depends on the reduction of states of a machine. Decomposition theory was such a theory by which we could make two or more machines each of them has fewer states and works togerther to realize the behaviour of the original machine.

Decompositions on states can be classed in two types. One is parallel decomposition. That is, the two component machines work independently with common inputs and outputs. Another is serial decomposition. On the connection of two component machines, one of them takes inputs from another one's states or outputs. Thus, the second machine (we call it a tail machine) works not only on the common inputs but also on the results of the first machine.

Decomposition theory shows that for a parallel decomposition, we must find two orthogonal SP partitions. The SP partitions can be used for forming the component machines of the paralle decomposition. If there do not exist such two SP partitions, the machine is said not to be parallely decomposable. Similarly, for a serial decomposition we need an SP partition and a partition that they are orthogonal.

SUMMARY:

Let M be a machine, M_1 and M_2 be two component machins.

$M \lhd M_1 \parallel M_2 \iff \exists\ \pi, \tau$ on M:

$$(\pi \text{ and } \tau \text{ have SP}) \land \pi \cdot \tau = \pi(0) \quad (9.1)$$

where $M_1 = (I, \pi, \delta')$; $M_2 = (I, \tau, \delta'')$;

$$\forall A \in \pi \quad \forall x \in I: \quad A\delta'_x = [A\bar{\delta}_x]_\pi \quad (9.2)$$

$$\forall B \in \tau \quad \forall x \in I: \quad B\delta''_x = [B\bar{\delta}_x]_\pi \quad (9.3)$$

$M \lhd M_1 \rightarrow M_2 \iff \exists\ \pi, \tau$ on M:

$$(\pi \text{ has SP}) \quad \wedge \quad (\pi \cdot \tau = \pi(0)) \qquad (9.4)$$

where $M_1 = (I, \pi, \delta')$; $M_2 = (\pi \times I, \tau, \delta'')$;

$$\forall A \in \pi \quad \forall x \in I: \quad A\delta'_x = [A\bar{\delta}_x]_\pi \qquad (9.6)$$

$\forall B \in \tau \quad \forall x \in I \quad \forall A \in \pi$:

$$B\delta''_{(A,x)} = [(A \cap B)\bar{\delta}_x]_\pi \qquad (9.7)$$

and $\forall s_1 \in \pi, s_2 \in \tau: \lambda'(s_1, s_2, x) = (s_1 \cap s_2)\lambda_x$ (9.8)
where λ' is the mapping for the output of connections.

9.2 Algorithms

The algorithms for decompositions on states largely rely on SP partitions. Once getting some SP partitions, we need find two SP partitions their product is zero-partition. In the case of serial decomposition we should calculate a partition from one SP partitions. We take the largest one to get an optimum decomposition, that is, the component machines have the fewest states.

Algorithm PDECOM;
input: machine table;
output: two component machines M_1 and M_2;
procedure:

```
begin var, i, j, i1, j1: integer; find: boolean;
        get all N SP partitions;
        if N=0 → no paralled decomposition; exit
         ▯ N≠0 →
            find := false; i := 0;
            do (not find) or i ≤ N →
                 i := i+1; j:=i+2;
                 do (not find) or j≤N →
                      if PIarray[i]·PIarray[j] = π(0) →
                           find :=true;
                       ▯ PIarray[i]·PIarray[j] ≠ π(0) →
                           j=j+1;
                      fi
                 od
```

```
                if not find → not parallel decomposable; exit
                 ▯ find → {build the component machines}
                    i1 := 1;
                    do i1 ≤ |PIarray[i]| →
                        j1:=1;
                        do j1 ≤ NumInputs →
                            findastate(PIarray[i,i1], s);
                            findablock(PIarray[i],δ[s,j],B);
                            M1[i1,j1]:=B;
                            j1:=j1+1
                        od;
                        i1=i1+1
                    od
                    i1 := 1
                    do i1: ≤ |PIarray[j]| →
                        j1:=1;
                        do j1 ≤ NumInputs →
                            findastate(PIarray[j1,i1],s);
                            findablock(PIarray[i], δ[s,j], B);
                            M1[i1,j1]:=B;
                            j1:=j1+1
                        od;
                        i1=i1+1
                    od
                fi
           od
       fi
end
```

Algorithm SDECOMP;

input: machine table arrays;

output: component machines M_1 and M_2 of a serial decomposition;

procedure:

```
begin var i1,j1,j2 : integer;
        get alll N SP partition;
        if N=0 → no serial decomposition; exit
         ▯ N≠0 → i1 := 1;
```

```
        do i1 ≤ |PIarray[1]| →
           j1:=1;
           do j1≤NumInputs →
                findastate(PIarray[1,i1],s);
                findablock(PIarray[1], δ[s,j1],B);
                M1[i1,j1]:=B;
                j1:=j1+1
           od;
           i1=i1+1
        od
        ORTHPART(PIarray[1], PIW);
        i1 := 1;
        do i1 ≤ |PIW| →
           j1:=1; j2:=1
           do j1≤|PIarray[1]| →
               do j2≤NumInputs →
                   s1 := PIW[i1]•PIarray[1,i1];
                   findablock(PIW, δ[s1,j2], B);
                   M2[i1,(j1,j2)]:=B;
                   j2:=j2+1
               od;
               j1=j1+1
            od
            i1:=i1+1
        od
    fi
end
```

The procedure ORTHPART calculates a partition PIW which is orthogonal to PIarray[1]. FindaState and FindaBlock are the same as used before. M_1 and M_2 are two machine table arrays that keep the entries for component machines in the decompositions.

9.3 Example

Machine tables: a:me622.dat shown in Fig. 9.1.from [21];
a:me722.dat shown in Fig. 9.2.from [3];

Procedure:

step 1. Run DASM by typing ***DASM*** under DOS prompt;

step 2. Select the function of decompositions on states by typing ***D***;

step 3. Specify the data file by typing ***a:me622.DAT***;

step 4. Select the function of parallel decomposition by typing ***P***;

step 5. Specify the output file by typing ***a:me622.pde***;

step 6. Specify the data file by typing ***a:me722.DAT***;

step 7. Select the function of serial decomposition by typing ***S***;

step 8. Specify the output file by typing ***a:me722.sde***.

The result is in file a:me622.pde and a:me722.sde and shown in Figs. 9.3 and 9.4

***** MACHINE M: a:me622.dat *****

STATE	INPUT 1	INPUT 2	OUTPUT
1	2	3	0
2	3	4	0
3	4	5	0
4	5	6	0
5	6	1	0
6	1	2	0

Fig. 9.1 Machine a:me622.dat

***** MACHINE a:me722.dat *****

STATE	INPUT 1	INPUT 2
1	2/1	3/1
2	4/1	5/1
3	5/1	4/1
4	6/1	7/1
5	7/1	6/1
6	1/1	1/1
7	1/2	1/2

* entry=(state/output)
* entry=0 for don't care

Fig. 9.2 Machine a:me722.dat

```
THE PARALLEL DECOMPOSITION OF a:me622.dat
=========================================


***** MACHINE M1 *****

Partition 1 - {(1,3,5);(2,4,6)}
-----------------------
STATE     INPUT OUTPUT
          1  2
.......................
  1       2  1    0
  2       1  2    0
-----------------------


***** MACHINE M2 *****

Partition 2 - {(1,4);(2,5);(3,6)}
-----------------------
STATE     INPUT OUTPUT
          1  2
.......................
  1       2  3    0
  2       3  1    0
  3       1  2    0
-----------------------

THE MAPPING:( M ==> M1//M2 )
(STATE,OUTPUT) ==> ((S1xS2),(O1*O2))

    (1,0) ==> ((1,1),(0*0))
    (2,0) ==> ((2,2),(0*0))
    (3,0) ==> ((1,3),(0*0))
    (4,0) ==> ((2,1),(0*0))
    (5,0) ==> ((1,2),(0*0))
    (6,0) ==> ((2,3),(0*0))
```

Fig. 9.3 Parallel decomposition of a:me622.dat

```
THE SERIAL DECOMPOSITION OF MACHINE a:me722.dat

***** MACHINE M1 *****

Partition 1 - {(1,4,5);(2,3,6,7)}
----------------
STATE  INPUT
        1    2
................
  1    2/1  2/1
  2    1/1  1/1
----------------
* entry=(state/output)
* entry=0 for don't care


***** MACHINE M2 *****

Partition 2 - {(1,2);(3,4);(5,6);(7)}
----------------------------------
STATE  INPUT(S1xI)            OUTPUT
       (1,1)(1,2)(2,1)(2,2)
..................................
  1      2    2    3    3      0
  2      4    4    2    2      0
  3      3    3    1    1      0
  4      0    0    1    1      0
----------------------------------
* The next state 0 : Don't care.

THE MAPPING:( M ==> (M1-->M2) )
STATE ==> (S1xS2)
  1   ==> (1,1)
  2   ==> (2,1)
  3   ==> (2,2)
  4   ==> (1,2)
  5   ==> (1,3)
  6   ==> (2,3)
  7   ==> (2,4)
```

Fig. 9.4 Serial decomposition of a:me722.dat

X. FULL-DECOMPOSITION OF MACHINES

10.1 Principle

In last chapter we have known how a machine can be decomposed on its states. That is, the component machines have few states. Usually, the component machines keep the same sizes of input set and output set as the original machine. The decomposition results in a partition only on the internal elements of a circuit. In this chapter, we deal with the decomposition both on state set and on input and output sets. This is so-called full-decomposition of sequential machines. An important charateristic of a full-decomposition is that we can make decomposition on the pins and connections of systems.

A full-decomposition is defined as follows.

Let $M=(I,S,O,\delta,\lambda)$ be a Mealy machine. From the study in [20], it is proved that if there are two partition trinities which are orthogonal, then M can be realized by the parallel connection of two smaller machines which are formed by the two partition trinities. That is,

$$T_1 \odot T_2 = T(0)$$
$$\Longrightarrow M \lhd M_1 \parallel M_2 \qquad (10.1)$$

where T_1 and T_2 are two orthogonal PT's and

$$T_1=(\Gamma 1,\pi 1,\Omega 1)$$
and
$$T_2=(\Gamma 2,\pi 2,\Omega 2)$$
$$M_1= (\Gamma 1,\pi 1,\Omega 1,\delta',\lambda')$$
$$M_2= (\Gamma 2,\pi 2,\Omega 2,\delta'',\lambda'')$$

The transition functions and output functions are defined by

$$\forall\ A1\in\pi 1 \quad \forall\ B1\in\Gamma 1:$$

$$A1\delta'_{B1} = [A1\bar{\delta}_{B1}]_{\pi 1} \qquad (10.2)$$

$$A1\lambda'_{B1} = [A1\bar{\lambda}_{B1}]_{\Omega 1} \qquad (10.3)$$

and $\forall\ A2 \in \Pi_2 \quad \forall\ B2 \in T_2$:

$$A2\delta''_{B2} = [A2\bar{\delta}_{B2}]_{\pi 2} \tag{10.4}$$

$$A2\lambda''_{B2} = [A2\bar{\lambda}_{B2}]_{\Omega 2} \tag{10.5}$$

The partition trinities keep that for any A1 and B1 , $A1\bar{\delta}_{B1}$ is certainly contained in and only in a block of $\Pi 1$, and that $A1\bar{\lambda}_{B1}$ is certainly contained in and only in a block of $\Omega 1$. It is the same for (10.4) and (10.5). Also, the orthogonal property of two PT's guarantees each element in original machine sets of state, input and output is mapped into one element in M_1 and one element in M_2, e.g.

$$\forall\ s \in S: \quad s \rightarrow ([s]_{\pi 1}, [s]_{\pi 2}) \tag{10.6}$$

$$\forall\ x \in I: \quad x \rightarrow ([x]_{\tau 1}, [x]_{\tau 2}) \tag{10.7}$$

$$\forall\ y \in O: \quad y \rightarrow ([y]_{\Omega 1}, [y]_{\Omega 2}) \tag{10.8}$$

10.2 Algorithm

Using the algorithm for PT's in chapter 6 we can find the orthogonal PT's. If there exist no such PT's, the given machine is not full-decomposable. For the orthogonal PT's, using (10.2) through (10.5), the component machines M_1 and M_2 can be formed. The mapping can be easily obtained based on (10.6), (10.7) and (10.8).

Algorithm PFdecomp;
input: machine table arrays;
output: two component machine tables;
procedure:

```
begin var N: integer; MS1,MO1,MS2,MO2: MachType;
      get all N PT's;
      if N<6 → "it is not full-decomposible", exit;
       [] N≥6 →
          do there are two largest orthogonal trinites:
             PT1 and PT2 →
               SETSUBM(PT1, MS1, MO1);
               SETSUBM(PT2, MS2, MO2);
               set mappings;
```

```
        od
        write machine tables;
    fi
end
```

In the algorithm, we judge N>6 because an PT is located in three partitions. If N<6, it means there are no two PT's which implies there is no parallel full-decomposition. The do loop finds all pairs of orthogonal partition trinities and set up the corrospondent component machines based on the PT's. SETSUBM is a procedure which calculates the transition and output table. The results of SETSUBM are two machine table arrays. "Write machine tables" writes the component machine table arrays on a file or on a printer.

10.3 Example

Machine table: a:me8886.dat shown in Fig. 6.1;
Procedure:

- *step 1.* Run DASM by typing ***DASM*** under DOS prompt;
- *step 2.* Select the function of full-decompositions by typing ***F***;
- *step 3.* Specify the data file by typing ***a:me8886.dat;***
- *step 4.* Select the function of parallel full-decomposition by typing ***P***;
- *step 5.* Specify the output file by typing ***a:me8886.pfd.***

The result is in file a:me8886.pfd and shown in Figs. 10.1-10.9.

THE FULL-DECOMPOSITION 1 OF MACHINE M:

TRINITY 1. [{(1,2);(5,6);(7,8);(3,4)},
{(1,2);(3,4);(5,6);(7,8)},
{(1,2);(3,4);(5,6);(7,8)}]

The Submachine M1 Based on Trinity 1:

STATE	INPUT 1	2	3	4
1	2/1	3/1	1/3	4/3
2	1/3	4/3	2/3	3/3
3	4/3	1/4	3/3	2/4
4	3/1	2/2	4/3	1/4

* entry=(state/output)

TRINITY 2. [{(1,3);(6,8);(5,7);(2,4)},
{(1,3);(2,4);(5,7);(6,8)},
{(1,3);(2,4);(5,7);(6,8)}]

The Submachine M2 Based on Trinity 2:

STATE	INPUT 1	2	3	4
1	2/1	3/2	4/3	1/4
2	1/4	4/4	3/4	2/4
3	4/3	1/4	2/3	3/4
4	3/2	2/2	1/4	4/4

* entry=(state/output)

MAPPING: (M --> M1//M2)

STATE:	INPUT:	OUTPUT:
(S=(S1xS2))	(I=(I1xI2))	(O=(O1xO2))
1=(1,1)	1=(1,1)	1=(1,1)
2=(1,4)	2=(1,2)	2=(1,2)
3=(4,1)	3=(2,1)	3=(2,1)
4=(4,4)	4=(2,2)	4=(2,2)
5=(2,3)	5=(3,3)	5=(3,3)
6=(2,2)	6=(3,4)	6=(3,4)
7=(3,3)	7=(4,3)	7=(4,3)
8=(3,2)	8=(4,4)	8=(4,4)

Fig. 10.1 Full-decomposition 1

THE FULL-DECOMPOSITION 2 OF MACHINE M:

TRINITY 1. [{(1,2);(5,6);(7,8);(3,4)},
{(1,2);(3,4);(5,6);(7,8)},
{(1,2);(3,4);(5,6);(7,8)}]

The Submachine M1 Based on Trinity 1:

STATE	INPUT 1	2	3	4
1	2/1	3/1	1/3	4/3
2	1/3	4/3	2/3	3/3
3	4/3	1/4	3/3	2/4
4	3/1	2/2	4/3	1/4

* entry=(state/output)

TRINITY 2. [{(1,5);(2,6);(4,8);(3,7)},
{(1,5);(2,6);(3,7);(4,8)},
{(1,5);(2,6);(3,7);(4,8)}]

The Submachine M2 Based on Trinity 2:

STATE	INPUT 1	2	3	4
1	2/1	1/2	3/1	4/2
2	1/2	2/2	4/2	3/2
3	4/2	3/2	1/4	2/4
4	3/1	4/2	2/3	1/4

* entry=(state/output)

MAPPING: (M --> M1//M2)

STATE:	INPUT:	OUTPUT:
(S=(S1xS2))	(I=(I1xI2))	(O=(O1xO2))
1=(1,1)	1=(1,1)	1=(1,1)
2=(1,2)	2=(1,2)	2=(1,2)
3=(4,4)	3=(2,3)	3=(2,3)
4=(4,3)	4=(2,4)	4=(2,4)
5=(2,1)	5=(3,1)	5=(3,1)
6=(2,2)	6=(3,2)	6=(3,2)
7=(3,4)	7=(4,3)	7=(4,3)
8=(3,3)	8=(4,4)	8=(4,4)

Fig. 10.2 Full-decomposition 2

THE FULL-DECOMPOSITION 3 OF MACHINE M:

TRINITY 1. [{(1,2);(5,6);(7,8);(3,4)},
{(1,2);(3,4);(5,6);(7,8)},
{(1,2);(3,4);(5,6);(7,8)}]

The Submachine M1 Based on Trinity 1:

STATE	INPUT 1	2	3	4
1	2/1	3/1	1/3	4/3
2	1/3	4/3	2/3	3/3
3	4/3	1/4	3/3	2/4
4	3/1	2/2	4/3	1/4

* entry=(state/output)

TRINITY 2. [{(1,7);(4,6);(3,5);(2,8)},
{(1,7);(2,8);(3,5);(4,6)},
{(1,3,5,7);(2,4,6,8)}]

The Submachine M2 Based on Trinity 2:

STATE	INPUT 1	2	3	4
1	2/1	3/2	4/1	1/2
2	1/2	4/2	3/2	2/2
3	4/1	1/2	2/1	3/2
4	3/2	2/2	1/2	4/2

* entry=(state/output)

MAPPING: (M --> M1//M2)

STATE: (S=(S1xS2))	INPUT: (I=(I1xI2))	OUTPUT: (O=(O1xO2))
1=(1,1)	1=(1,1)	1=(1,1)
2=(1,4)	2=(1,2)	2=(1,2)
3=(4,3)	3=(2,3)	3=(2,1)
4=(4,2)	4=(2,4)	4=(2,2)
5=(2,3)	5=(3,3)	5=(3,1)
6=(2,2)	6=(3,4)	6=(3,2)
7=(3,1)	7=(4,1)	7=(4,1)
8=(3,4)	8=(4,2)	8=(4,2)

Fig. 10.3 Full-decomposition 3

THE FULL-DECOMPOSITION 4 OF MACHINE M:

TRINITY 1. [{(1,2);(5,6);(7,8);(3,4)},
{(1,2);(3,4);(5,6);(7,8)},
{(1,2);(3,4);(5,6);(7,8)}]

The Submachine M1 Based on Trinity 1:

STATE	INPUT 1	2	3	4
1	2/1	3/1	1/3	4/3
2	1/3	4/3	2/3	3/3
3	4/3	1/4	3/3	2/4
4	3/1	2/2	4/3	1/4

* entry=(state/output)

TRINITY 2. [{(1,3,5,7);(2,4,6,8)},
{(1,3,5,7);(2,4,6,8)},
{(1,3,5,7);(2,4,6,8)}]

The Submachine M2 Based on Trinity 2:

STATE	INPUT 1	2
1	2/1	1/2
2	1/2	2/2

* entry=(state/output)

MAPPING: (M --> M1//M2)

STATE: (S=(S1xS2))	INPUT: (I=(I1xI2))	OUTPUT: (O=(O1xO2))
1=(1,1)	1=(1,1)	1=(1,1)
2=(1,2)	2=(1,2)	2=(1,2)
3=(4,1)	3=(2,1)	3=(2,1)
4=(4,2)	4=(2,2)	4=(2,2)
5=(2,1)	5=(3,1)	5=(3,1)
6=(2,2)	6=(3,2)	6=(3,2)
7=(3,1)	7=(4,1)	7=(4,1)
8=(3,2)	8=(4,2)	8=(4,2)

Fig. 10.4 Full-decomposition 4

THE FULL-DECOMPOSITION 5 OF MACHINE M:

TRINITY 1. [{(1,3);(6,8);(5,7);(2,4)},
{(1,3);(2,4);(5,7);(6,8)},
{(1,3);(2,4);(5,7);(6,8)}]

The Submachine M1 Based on Trinity 1:

STATE	INPUT 1	2	3	4
1	2/1	3/2	4/3	1/4
2	1/4	4/4	3/4	2/4
3	4/3	1/4	2/3	3/4
4	3/2	2/2	1/4	4/4

* entry=(state/output)

TRINITY 2. [{(1,5);(2,6);(4,8);(3,7)},
{(1,5);(2,6);(3,7);(4,8)},
{(1,5);(2,6);(3,7);(4,8)}]

The Submachine M2 Based on Trinity 2:

STATE	INPUT 1	2	3	4
1	2/1	1/2	3/1	4/2
2	1/2	2/2	4/2	3/2
3	4/2	3/2	1/4	2/4
4	3/1	4/2	2/3	1/4

* entry=(state/output)

MAPPING: (M ==> M1//M2)

STATE:	INPUT:	OUTPUT:
(S=(S1xS2))	(I=(I1xI2))	(O=(O1xO2))
1=(1,1)	1=(1,1)	1=(1,1)
2=(4,2)	2=(2,2)	2=(2,2)
3=(1,4)	3=(1,3)	3=(1,3)
4=(4,3)	4=(2,4)	4=(2,4)
5=(3,1)	5=(3,1)	5=(3,1)
6=(2,2)	6=(4,2)	6=(4,2)
7=(3,4)	7=(3,3)	7=(3,3)
8=(2,3)	8=(4,4)	8=(4,4)

Fig. 10.5 Full-decomposition 5

THE FULL-DECOMPOSITION 6 OF MACHINE M:

TRINITY 1. [{(1,3);(6,8);(5,7);(2,4)},
{(1,3);(2,4);(5,7);(6,8)},
{(1,3);(2,4);(5,7);(6,8)}]

The Submachine M1 Based on Trinity 1:

STATE	INPUT 1	2	3	4
1	2/1	3/2	4/3	1/4
2	1/4	4/4	3/4	2/4
3	4/3	1/4	2/3	3/4
4	3/2	2/2	1/4	4/4

* entry=(state/output)

TRINITY 2. [{(1,6);(2,5);(3,8);(4,7)},
{(1,6);(2,5);(3,8);(4,7)},
{(1,2,5,6);(3,4,7,8)}]

The Submachine M2 Based on Trinity 2:

STATE	INPUT 1	2	3	4
1	1/1	2/1	3/1	4/1
2	2/1	1/1	4/1	3/1
3	3/1	4/1	1/2	2/2
4	4/1	3/1	2/2	1/2

* entry=(state/output)

MAPPING: (M --> M1//M2)

STATE:	INPUT:	OUTPUT:
(S=(S1xS2))	(I=(I1xI2))	(O=(O1xO2))
1=(1,1)	1=(1,1)	1=(1,1)
2=(4,2)	2=(2,2)	2=(2,1)
3=(1,3)	3=(1,3)	3=(1,2)
4=(4,4)	4=(2,4)	4=(2,2)
5=(3,2)	5=(3,2)	5=(3,1)
6=(2,1)	6=(4,1)	6=(4,1)
7=(3,4)	7=(3,4)	7=(3,2)
8=(2,3)	8=(4,3)	8=(4,2)

Fig. 10.6 Full-decomposition 6

THE FULL-DECOMPOSITION 7 OF MACHINE M:

TRINITY 1. [{(1,3);(6,8);(5,7);(2,4)},
{(1,3);(2,4);(5,7);(6,8)},
{(1,3);(2,4);(5,7);(6,8)}]

The Submachine M1 Based on Trinity 1:

STATE	INPUT 1	2	3	4
1	2/1	3/2	4/3	1/4
2	1/4	4/4	3/4	2/4
3	4/3	1/4	2/3	3/4
4	3/2	2/2	1/4	4/4

* entry=(state/output)

TRINITY 2. [{(1,2,5,6);(3,4,7,8)},
{(1,2,5,6);(3,4,7,8)},
{(1,2,5,6);(3,4,7,8)}]

The Submachine M2 Based on Trinity 2:

STATE	INPUT 1	2
1	1/1	2/1
2	2/1	1/2

* entry=(state/output)

MAPPING: (M --> M1//M2)

STATE:	INPUT:	OUTPUT:
(S=(S1xS2))	(I=(I1xI2))	(O=(O1xO2))
1=(1,1)	1=(1,1)	1=(1,1)
2=(4,1)	2=(2,1)	2=(2,1)
3=(1,2)	3=(1,2)	3=(1,2)
4=(4,2)	4=(2,2)	4=(2,2)
5=(3,1)	5=(3,1)	5=(3,1)
6=(2,1)	6=(4,1)	6=(4,1)
7=(3,2)	7=(3,2)	7=(3,2)
8=(2,2)	8=(4,2)	8=(4,2)

Fig. 10.7 Full-decomposition 7

THE FULL-DECOMPOSITION 8 OF MACHINE M:

TRINITY 1. [{(1,4);(6,7);(5,8);(2,3)},
{(1,4);(2,3);(5,8);(6,7)},
{(1,2,3,4);(5,6,7,8)}]

The Submachine M1 Based on Trinity 1:

STATE	INPUT 1	2	3	4
1	2/1	3/1	4/2	1/2
2	1/2	4/2	3/2	2/2
3	4/2	1/2	2/2	3/2
4	3/1	2/1	1/2	4/2

* entry=(state/output)

TRINITY 2. [{(1,5);(2,6);(4,8);(3,7)},
{(1,5);(2,6);(3,7);(4,8)},
{(1,5);(2,6);(3,7);(4,8)}]

The Submachine M2 Based on Trinity 2:

STATE	INPUT 1	2	3	4
1	2/1	1/2	3/1	4/2
2	1/2	2/2	4/2	3/2
3	4/2	3/2	1/4	2/4
4	3/1	4/2	2/3	1/4

* entry=(state/output)

MAPPING: (M --> M1//M2)

STATE:	INPUT:	OUTPUT:
(S=(S1xS2))	(I=(I1xI2))	(O=(O1xO2))
1=(1,1)	1=(1,1)	1=(1,1)
2=(4,2)	2=(2,2)	2=(1,2)
3=(4,4)	3=(2,3)	3=(1,3)
4=(1,3)	4=(1,4)	4=(1,4)
5=(3,1)	5=(3,1)	5=(2,1)
6=(2,2)	6=(4,2)	6=(2,2)
7=(2,4)	7=(4,3)	7=(2,3)
8=(3,3)	8=(3,4)	8=(2,4)

Fig. 10.8 Full-decomposition 8

THE FULL-DECOMPOSITION 9 OF MACHINE M:

TRINITY 1. [{(1,5);(2,6);(4,8);(3,7)},
{(1,5);(2,6);(3,7);(4,8)},
{(1,5);(2,6);(3,7);(4,8)}]

The Submachine M1 Based on Trinity 1:

STATE	INPUT 1	2	3	4
1	2/1	1/2	3/1	4/2
2	1/2	2/2	4/2	3/2
3	4/2	3/2	1/4	2/4
4	3/1	4/2	2/3	1/4

* entry=(state/output)

TRINITY 2. [{(1,2,3,4);(5,6,7,8)},
{(1,2,3,4);(5,6,7,8)},
{(1,2,3,4);(5,6,7,8)}]

The Submachine M2 Based on Trinity 2:

STATE	INPUT 1	2
1	2/1	1/2
2	1/2	2/2

* entry=(state/output)

MAPPING: (M --> M1//M2)

STATE:	INPUT:	OUTPUT:
(S=(S1xS2))	(I=(I1xI2))	(O=(O1xO2))
1=(1,1)	1=(1,1)	1=(1,1)
2=(2,1)	2=(2,1)	2=(2,1)
3=(4,1)	3=(3,1)	3=(3,1)
4=(3,1)	4=(4,1)	4=(4,1)
5=(1,2)	5=(1,2)	5=(1,2)
6=(2,2)	6=(2,2)	6=(2,2)
7=(4,2)	7=(3,2)	7=(3,2)
8=(3,2)	8=(4,2)	8=(4,2)

Fig. 10.9 Full-decomposition 9

XI. ISSM.

In practical design,there are some machines which contain don't care entries. In this case, we have to make some modifications on above algorithms to calculate some functions on incompletely specified sequential machines (ISSM). In this chapter we explain the computation for weak partition pairs (WPP).

11.1 Principle

The calculation for WPP is basicly the same as that in Chapter 5 except we have to make sure that

$$\pi = \{A_1, A_2, \ldots, A_m\}$$
$$\tau = \{B_1, B_2, \ldots, B_n\}$$

(π, τ) is an weak S-S pair iff

$$\forall\ s,t \in S \quad \forall\ x \in I: \quad [s]_\pi = [t]_\pi \ \wedge\ s\delta_x \neq 0 \ \wedge\ t\delta_x \neq 0 \implies [s\delta_x]_\tau = [t\delta_x]_\tau \qquad (11.1)$$

The difference is only that we have to check if $s\delta_x$ and $t\delta_x$ are don't cares before we go further. It is the same for other weak pairs. We omit the discussion here.

11.2 Algorithm

Because the transitive property rules out in partition operations on an ISSM, we can not use the way that calculates basic partition pairs. In DASM, we use a procedure to generate a partition. With this partition we obtain a weak m-partition. Continuing the search for all partitions on the set of first component of the pair, we can get all weak partition pairs.

```
Algorithm COMPPW(PT);
input: machine table arrays; partition type PT;
output: weak partition pairs;
procedure:
begin var BR: boolean;
          PIW1,PIW2: partition;
          L1,J,M1,E,F,NS1: integer;
      NEXTP(L1,J,M1,E,F,BR,PIW);
      do not BR →
        NEXTP(L1,J,M1,E,F,BR,PIW1);
        RmPI-W(PT,PIW1,PIW2,J,NS1);
             if NS1>0 and NS1<Num→
                PrintPI(PIW,PIW2);
              ▯ NS1≤0 or NS1≥Num → skip
             fi
      od
end
```

Within the algorithm, procedure NEXTP offers a partition PIW1; RmPI-W calculate out the weak m-partition PIW2 of PIW1. Num is NumStates or NumOutputs depended on pairtype PT.

11.3 Example

Machine table: **a:me654w.dat** shown in Fig.11.1;

Procedure:

step 1. Run DASM by typing ***DASM*** under DOS prompt;

step 2. Select the function of ISSM by typing *I;*

step 3. Specify the data file by typing ***a:me654w.dat;***

step 4. Select the function of weak pairs by typing *P;*

step 5. Specify the output file by typing ***a:me654w.wpp.***

Machine: a:me654w.dat

STATE	INPUT 1	2	3	4	5
1	2/1	4/4	0/3	5/4	3/0
2	1/1	4/3	3/1	0/3	6/3
3	5/2	0/2	2/0	3/3	2/2
4	0/3	1/2	6/2	2/1	0/2
5	4/4	0/1	5/3	1/2	6/1
6	6/0	5/1	0/4	4/0	3/3

* entry=(state/output)

* entry=0 for don't care

Fig. 11.1 a:me654w.dat

The result is in file a:me654w.wpp and shown in Fig. 11.2.

```
WEAK S-S PARTITION PAIR:
------------------------
WPP   1 [{(1,2,3);(4,5);(6)} , {(1,2,3,5,6);(4)}]
WPP   2 [{(1,2,3);(4);(5);(6)} , {(1,2,3,5,6);(4)}]
WPP   3 [{(1,2,4);(3);(5);(6)} , {(1,2,4,5);(3,6)}]
WPP   4 [{(1,2,6);(3,4);(5)} , {(1,2,3,6);(4,5)}]
WPP   5 [{(1,2);(3,4);(5,6)} , {(1,2,3,4,6);(5)}]
WPP   6 [{(1,2);(3,4);(5);(6)} , {(1,2,3,6);(4);(5)}]
WPP   7 [{(1,2);(3);(4,5);(6)} , {(1,2);(3,5,6);(4)}]
WPP   8 [{(1,2,6);(3);(4);(5)} , {(1,2,3,6);(4,5)}]
WPP   9 [{(1,2);(3);(4,6);(5)} , {(1,2,4,5);(3,6)}]
WPP  10 [{(1,2);(3);(4);(5,6)} , {(1,2,3,4,6);(5)}]
WPP  11 [{(1,2);(3);(4);(5);(6)} , {(1,2);(3,6);(4);(5)}]
WPP  12 [{(1,3,4);(2,5);(6)} , {(1,4);(2,3,5,6)}]
WPP  13 [{(1,3,4);(2);(5);(6)} , {(1,4);(2,3,5,6)}]
WPP  14 [{(1,3);(2,4);(5);(6)} , {(1,4);(2,3,5,6)}]
WPP  15 [{(1,3);(2,5);(4,6)} , {(1,2,3,4,5);(6)}]
WPP  16 [{(1,3);(2,5);(4);(6)} , {(1,4);(2,3,5);(6)}]
WPP  17 [{(1,3);(2);(4,5);(6)} , {(1,2,3,5,6);(4)}]
WPP  18 [{(1,3,6);(2);(4);(5)} , {(1);(2,3,4,5,6)}]
WPP  19 [{(1,3);(2);(4,6);(5)} , {(1,2,3,4,5);(6)}]
WPP  20 [{(1,3);(2);(4);(5);(6)} , {(1);(2,3,5);(4);(6)}]
WPP  21 [{(1,6);(2,3,4);(5)} , {(1,4,5);(2,3,6)}]
WPP  22 [{(1);(2,3,4);(5);(6)} , {(1,4,5);(2,3,6)}]
WPP  23 [{(1,5);(2,3);(4,6)} , {(1,5);(2,3,4,6)}]
WPP  24 [{(1,5);(2,3);(4);(6)} , {(1,5);(2,3,4,6)}]
WPP  25 [{(1);(2,3);(4,5);(6)} , {(1,2,3,5,6);(4)}]
WPP  26 [{(1,6);(2,3);(4);(5)} , {(1,4,5);(2,3,6)}]
WPP  27 [{(1);(2,3);(4,6);(5)} , {(1,5);(2,3,4,6)}]
WPP  28 [{(1);(2,3);(4);(5);(6)} , {(1,5);(2,3,6);(4)}]
WPP  29 [{(1,4);(2,5);(3);(6)} , {(1,4);(2,3,5);(6)}]
WPP  30 [{(1,4,6);(2);(3);(5)} , {(1,2,4,5,6);(3)}]
WPP  31 [{(1,4);(2);(3);(5,6)} , {(1,3,4,6);(2,5)}]
WPP  32 [{(1,4);(2);(3);(5);(6)} , {(1,4);(2,5);(3);(6)}]
WPP  33 [{(1,5);(2,4);(3);(6)} , {(1,2,4,5);(3,6)}]
WPP  34 [{(1);(2,4,5);(3);(6)} , {(1,2,4);(3,5,6)}]
WPP  35 [{(1,6);(2,4);(3);(5)} , {(1,4,5);(2,3,6)}]
WPP  36 [{(1);(2,4);(3);(5,6)} , {(1,3,4,6);(2);(5)}]
WPP  37 [{(1);(2,4);(3);(5);(6)} , {(1,4);(2);(3,6);(5)}]
WPP  38 [{(1,5);(2);(3,4);(6)} , {(1,5);(2,3,4,6)}]
WPP  39 [{(1);(2,5);(3,4);(6)} , {(1,4);(2,3,5,6)}]
WPP  40 [{(1,6);(2);(3,4);(5)} , {(1);(2,3,6);(4,5)}]
WPP  41 [{(1);(2,6);(3,4);(5)} , {(1,2,3,6);(4,5)}]
WPP  42 [{(1);(2);(3,4);(5,6)} , {(1,2,3,4,6);(5)}]
WPP  43 [{(1);(2);(3,4);(5);(6)} , {(1);(2,3,6);(4);(5)}]
WPP  44 [{(1,5);(2);(3);(4,6)} , {(1,5);(2,4);(3,6)}]
WPP  45 [{(1,5);(2);(3);(4);(6)} , {(1,5);(2,4);(3,6)}]
WPP  46 [{(1,6);(2,5);(3);(4)} , {(1,3,4,5);(2,6)}]
WPP  47 [{(1);(2,5,6);(3);(4)} , {(1,3,4,5,6);(2)}]
WPP  48 [{(1);(2,5);(3);(4,6)} , {(1,2,3,4,5);(6)}]
WPP  49 [{(1);(2,5);(3);(4);(6)} , {(1,4);(2);(3,5);(6)}]
WPP  50 [{(1,6);(2);(3,5);(4)} , {(1,3);(2,4,5,6)}]
WPP  51 [{(1);(2);(3,5);(4);(6)} , {(1,3);(2,4,5,6)}]
WPP  52 [{(1,6);(2);(3);(4,5)} , {(1,2,4,5,6);(3)}]
WPP  53 [{(1);(2);(3,6);(4,5)} , {(1,2,3,4);(5,6)}]
WPP  54 [{(1);(2);(3);(4,5);(6)} , {(1,2);(3);(4);(5,6)}]
WPP  55 [{(1,6);(2);(3);(4);(5)} , {(1);(2,6);(3);(4,5)}]
WPP  56 [{(1);(2,6);(3);(4);(5)} , {(1,3,6);(2);(4,5)}]
WPP  57 [{(1);(2);(3,6);(4);(5)} , {(1);(2,3,4);(5,6)}]
WPP  58 [{(1);(2);(3);(4,6);(5)} , {(1,5);(2,4);(3);(6)}]
WPP  59 [{(1);(2);(3);(4);(5,6)} , {(1,3,4,6);(2);(5)}]
----------END----------

WEAK I-S PARTITION PAIR:
------------------------
WPP   1 [{(1,2);(3,5);(4)} , {(1,2,4);(3,5,6)}]
WPP   2 [{(1,2);(3);(4);(5)} , {(1,2,4);(3);(5,6)}]
WPP   3 [{(1,3);(2,4);(5)} , {(1,2,3,4,5);(6)}]
WPP   4 [{(1,3);(2);(4);(5)} , {(1,3);(2,4,5);(6)}]
WPP   5 [{(1);(2,3,5);(4)} , {(1,3,4,5,6);(2)}]
WPP   6 [{(1);(2,3);(4,5)} , {(1,6);(2,3,4,5)}]
WPP   7 [{(1);(2,3);(4);(5)} , {(1,6);(2);(3,4);(5)}]
WPP   8 [{(1,4);(2);(3);(5)} , {(1,4,6);(2,3,5)}]
WPP   9 [{(1);(2,4);(3,5)} , {(1,2);(3,4,5,6)}]
WPP  10 [{(1);(2,4);(3);(5)} , {(1,2);(3);(4,5);(6)}]
WPP  11 [{(1);(2);(3,4);(5)} , {(1,5);(2,3,6);(4)}]
WPP  12 [{(1);(2,5);(3);(4)} , {(1);(2);(3,4,5,6)}]
WPP  13 [{(1);(2);(3,5);(4)} , {(1);(2);(3,5,6);(4)}]
WPP  14 [{(1);(2);(3);(4,5)} , {(1,6);(2,3,4,5)}]
----------END----------

WEAK S-O PARTITION PAIR:
------------------------
WPP   1 [{(1,2,6);(3);(4);(5)} , {(1,3,4);(2)}]
WPP   2 [{(1,2);(3);(4);(5,6)} , {(1,3,4);(2)}]
WPP   3 [{(1,2);(3);(4);(5);(6)} , {(1,3,4);(2)}]
WPP   4 [{(1);(2,3,4);(5);(6)} , {(1,2,3);(4)}]
WPP   5 [{(1);(2,3);(4);(5);(6)} , {(1,2,3);(4)}]
WPP   6 [{(1);(2,4);(3,6);(5)} , {(1,2,3);(4)}]
WPP   7 [{(1);(2,4);(3);(5);(6)} , {(1,2,3);(4)}]
WPP   8 [{(1);(2);(3,4);(5);(6)} , {(1,2,3);(4)}]
WPP   9 [{(1,5);(2);(3);(4);(6)} , {(1,2,4);(3)}]
WPP  10 [{(1,6);(2);(3);(4);(5)} , {(1,3,4);(2)}]
WPP  11 [{(1);(2,6);(3);(4);(5)} , {(1,3,4);(2)}]
WPP  12 [{(1);(2);(3,6);(4);(5)} , {(1,2,3);(4)}]
WPP  13 [{(1);(2);(3);(4);(5,6)} , {(1,3,4);(2)}]
----------END----------

WEAK I-O PARTITION PAIR:
------------------------
WPP   1 [{(1);(2,3,5);(4)} , {(1,3,4);(2)}]
WPP   2 [{(1);(2,3);(4);(5)} , {(1,3,4);(2)}]
WPP   3 [{(1);(2,4,5);(3)} , {(1,2,3);(4)}]
WPP   4 [{(1);(2,4);(3);(5)} , {(1,2,3);(4)}]
WPP   5 [{(1);(2,5);(3);(4)} , {(1,3);(2);(4)}]
WPP   6 [{(1);(2);(3,5);(4)} , {(1,3,4);(2)}]
WPP   7 [{(1);(2);(3);(4,5)} , {(1,2,3);(4)}]
----------END----------
A>
```

Fig. 11.2 Weak pairs of a:me654w.dat

REFERENCES

[1] ALGEBRAIC THEORY OF MACHINES, LANGUAGES, AND SEMIGROUPS. Ed. by M.A. Arbib.
New York: Academic Press, 1968.

[2] Arbib, M.A.
THEORIES OF ABSTRACT AUTOMATA.
Englewood Cliffs, N.J.: Prentice-Hall, 1969.
Prentice-Hall series in automatic computation

[3] Davio, M. and J.-P. Deschamps, A. Thayse
DIGITAL SYSTEMS WITH ALGORITHM IMPLEMENTATION.
Chichester: Wiley, 1983.

[4] Dijkstra, E.W.
A DISCIPLINE OF PROGRAMMING.
Englewood Cliffs, N.J.: Prentice-Hall, 1976.
Prentice-Hall series in automatic computation

[5] Eilenberg, S.
AUTOMATA, LANGUAGES, AND MACHINES. Volume A.
New York: Academic Press, 1974.
Pure and applied mathematics: A series of monographs and textbooks

[6] Eilenberg, S.
AUTOMATA, LANGUAGES, AND MACHINES. Volume B.
New York: Academic Press, 1976.
Pure and applied mathematics: A series of monographs and textbooks

[7] Friedman, A.D. and P.R. Menon
THEORY & DESIGN OF SWITCHING CIRCUITS.
Woodland Hills, Cal.: Computer Science Press, 1975. London: Pitman, 1977.
Digital system design series

[8] Ginzburg, A.
ALGEBRAIC THEORY OF AUTOMATA.
New York: Academic Press, 1968.
ACM monograph series

[9] Cioffi, G. and S. De Julio, M. Lucertini
OPTIMAL DECOMPOSITION OF SEQUENTIAL MACHINES VIA INTEGER NON-LINEAR PROGRAMMING: A computational algorithm.
Digital processes, Vol. 5(1979), p. 27-41.

[10] Mealy, G.H.
A METHOD FOR SYNTHESIZING SEQUENTIAL CIRCUITS.
Bell Syst. Tech. J., Vol. 34(1955), p. 1045-1079.

[11] Moore, E.F.
GEDANKEN-EXPERIMENTS ON SEQUENTIAL MACHINES.
In: Automata Studies. Ed. by C.E. Shannon and J. McCarthy.
Princeton University Press, 1956.
Annals of mathematical studies, Vol. 34. P. 129-153.

[12] Hartmanis, J.
ON THE STATE ASSIGNMENT PROBLEM FOR SEQUENTIAL MACHINES I.
IRE Trans. Electron. Comput., Vol. EC-10(1961), p. 157-165.

[13] Hartmanis, J.
LOOP-FREE STRUCTURE OF SEQUENTIAL MACHINES.
Inf. & Control, Vol. 5(1962), p. 25-43.

[14] Hartmanis, J.
FURTHER RESULTS ON THE STRUCTURE OF SEQUENTIAL MACHINES.
J. Assoc. Comput. Mach., Vol. 10(1963), p. 78-88.

[15] Stearns, R.E. and J. Hartmanis
ON THE STATE ASSIGNMENT PROBLEM FOR SEQUENTIAL MACHINES II.
IRE Trans. Electron. Comput., Vol. EC-10(1961), p. 593-603.

[16] Hartmanis, J. and R.E. Stearns
SOME DANGERS IN STATE REDUCTION OF SEQUENTIAL MACHINES.
Inf. & Control, Vol. 5(1962), p. 252-260.

[17] Hartmanis, J. and R.E.Stearns
PAIR ALGEBRA AND ITS APPLICATION TO AUTOMATA THEORY.
Inf. & Control, Vol. 7(1964), p. 485-507.

[18] Hartmanis, J. and R.E. Stearns
ALGEBRAIC STRUCTURE THEORY OF SEQUENTIAL MACHINES.
Englewood Cliffs, N.J.: Prentice-Hall, 1966.
Prentice-Hall international series on applied mathematics

[19] Holcombe, W.M.L.
ALGEBRAIC AUTOMATA THEORY.
Cambridge University Press, 1982.
Cambridge studies in advanced mathematics, Vol. 1.

[20] Hou Yibin
TRINITY ALGEBRA AND FULL-DECOMPOSITIONS OF SEQUENTIAL MACHINES.
Ph.D. Thesis. Eindhoven University of Technology, 1986.

[21] Lewin, D.
DESIGN OF LOGIC SYSTEMS.
Wokingham, Berks.: Van Nostrand Reinhold, 1985.

[22] SIGNETICS INTEGRATED FUSE LOGIC (IFL).
Eindhoven: Philips Elcoma, May 1983.
Philips data handbook, Part IC 10.

[23] Yoeli, M.
THE CASCADE DECOMPOSITION OF SEQUENTIAL MACHINES.
IRE Trans. Electron. Comput., Vol. EC-10(1961), p. 587-592.

[24] Yoeli, M.
CASCADE-PARALLEL DECOMPOSITIONS OF SEQUENTIAL MACHINES.
IEEE Trans. Electron. Comput., Vol. EC-12(1963), p. 322-324.

[25] SEMI CUSTOM DESIGN TRAINING COURSE.
Lecture Notes. Digital Systems Group, Faculty of Electrical Engineering,
Eindhoven University of Technology, 1987.

[26] Lew, A.
COMPUTER SCIENCE: A mathematical introduction.
Englewood Cliffs, N.J.: Prentice-Hall, 1985.
Prentice-Hall international series in computer science

ISSN 0167-9708
Coden: TEUEDE

(149) Beeckman, P.A.
MILLIMETER-WAVE ANTENNA MEASUREMENTS WITH THE HP8510 NETWORK ANALYZER.
EUT Report 85-E-149. 1985. ISBN 90-6144-149-8

(150) Meer, A.C.P. van
EXAMENRESULTATEN IN CONTEXT MBA.
EUT Report 85-E-150. 1985. ISBN 90-6144-150-1

(151) Ramakrishnan, S. and W.M.C. van den Heuvel
SHORT-CIRCUIT CURRENT INTERRUPTION IN A LOW-VOLTAGE FUSE WITH ABLATING WALLS.
EUT Report 85-E-151. 1985. ISBN 90-6144-151-X

(152) Stefanov, B. and L. Zarkova, A. Veefkind
DEVIATION FROM LOCAL THERMODYNAMIC EQUILIBRIUM IN A CESIUM-SEEDED ARGON PLASMA.
EUT Report 85-E-152. 1985. ISBN 90-6144-152-8

(153) Hof, P.M.J. Van den and P.H.M. Janssen
SOME ASYMPTOTIC PROPERTIES OF MULTIVARIABLE MODELS IDENTIFIED BY EQUATION ERROR TECHNIQUES.
EUT Report 85-E-153. 1985. ISBN 90-6144-153-6

(154) Geerlings, J.H.T.
LIMIT CYCLES IN DIGITAL FILTERS: A bibliography 1975-1984.
EUT Report 85-E-154. 1985. ISBN 90-6144-154-4

(155) Groot, J.F.G. de
THE INFLUENCE OF A HIGH-INDEX MICRO-LENS IN A LASER-TAPER COUPLING.
EUT Report 85-E-155. 1985. ISBN 90-6144-155-2

(156) Amelsfort, A.M.J. van and Th. Scharten
A THEORETICAL STUDY OF THE ELECTROMAGNETIC FIELD IN A LIMB, EXCITED BY ARTIFICIAL SOURCES.
EUT Report 86-E-156. 1986. ISBN 90-6144-156-0

(157) Lodder, A. and M.T. van Stiphout, J.T.J. van Eijndhoven
ESCHER: Eindhoven SCHematic EditoR reference manual.
EUT Report 86-E-157. 1986. ISBN 90-6144-157-9

(158) Arnbak, J.C.
DEVELOPMENT OF TRANSMISSION FACILITIES FOR ELECTRONIC MEDIA IN THE NETHERLANDS.
EUT Report 86-E-158. 1986. ISBN 90-6144-158-7

(159) Wang Jingshan
HARMONIC AND RECTANGULAR PULSE REPRODUCTION THROUGH CURRENT TRANSFORMERS.
EUT Report 86-E-159. 1986. ISBN 90-6144-159-5

(160) Wolzak, G.G. and A.M.F.J. van de Laar, E.F. Steennis
PARTIAL DISCHARGES AND THE ELECTRICAL AGING OF XLPE CABLE INSULATION.
EUT Report 86-E-160. 1986. ISBN 90-6144-160-9

(161) Veenstra, P.K.
RANDOM ACCESS MEMORY TESTING: Theory and practice. The gains of fault modelling.
EUT Report 86-E-161. 1986. ISBN 90-6144-161-7

(162) Meer, A.C.P. van
TMS32010 EVALUATION MODULE CONTROLLER.
EUT Report 86-E-162. 1986. ISBN 90-6144-162-5

(163) Stok, L. and R. van den Born, G.L.J.M. Janssen
HIGHER LEVELS OF A SILICON COMPILER.
EUT Report 86-E-163. 1986. ISBN 90-6144-163-3

(164) Engelshoven, R.J. van and J.F.M. Theeuwen
GENERATING LAYOUTS FOR RANDOM LOGIC: Cell generation schemes.
EUT Report 86-E-164. 1986. ISBN 90-6144-164-1

(165) Lippens, P.E.R. and A.G.J. Slenter
GADL: A Gate Array Description Language.
EUT Report 87-E-165. 1987. ISBN 90-6144-165-X

(166) Dielen, M. and J.F.M. Theeuwen
AN OPTIMAL CMOS STRUCTURE FOR THE DESIGN OF A CELL LIBRARY.
EUT Report 87-E-166. 1987. ISBN 90-6144-166-8

(167) Oerlemans, C.A.M. and J.F.M. Theeuwen
ESKISS: A program for optimal state assignment.
EUT Report 87-E-167. 1987. ISBN 90-6144-167-6

(168) Linnartz, J.P.M.G.
SPATIAL DISTRIBUTION OF TRAFFIC IN A CELLULAR MOBILE DATA NETWORK.
EUT Report 87-E-168. 1987. ISBN 90-6144-168-4

(169) Vinck, A.J. and J. Pineda de Gyvez, K.A. Post
IMPLEMENTATION AND EVALUATION OF A COMBINED TEST-ERROR CORRECTION PROCEDURE FOR MEMORIES WITH DEFECTS.
EUT Report 87-E-169. 1987. ISBN 90-6144-169-2