Theoretical and Design Techniques for Stored Program Implementations of Sequential Systems

This item was submitted to Loughborough University's Institutional Repository by the/an author.

Additional Information:

- A Doctoral Thesis. Submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of Loughborough University.

Metadata Record: https://dspace.lboro.ac.uk/2134/12170

Publisher: © A.M.A. Ahsan

Please cite the published version.
This item was submitted to Loughborough University as a PhD thesis by the author and is made available in the Institutional Repository (https://dspace.lboro.ac.uk/) under the following Creative Commons Licence conditions.

You are free:

- to copy, distribute, display, and perform the work

Under the following conditions:

**Attribution.** You must attribute the work in the manner specified by the author or licensor.

**Noncommercial.** You may not use this work for commercial purposes.

**No Derivative Works.** You may not alter, transform, or build upon this work.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the Legal Code (the full license).

For the full text of this licence, please go to: http://creativecommons.org/licenses/by-nc-nd/2.5/
<table>
<thead>
<tr>
<th>AUTHOR/FILING TITLE</th>
<th>HUSSAIN, A</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESSION/COPY NO.</td>
<td>085039/01</td>
</tr>
<tr>
<td>VOL. NO.</td>
<td>CLASS MARK</td>
</tr>
<tr>
<td>ARCHIVES COPY</td>
<td></td>
</tr>
<tr>
<td>FOR REFERENCE ONLY</td>
<td></td>
</tr>
</tbody>
</table>
THEORY AND DESIGN TECHNIQUES FOR STORED PROGRAM IMPLEMENTATIONS OF SEQUENTIAL SYSTEMS

by

Ahsan Mohammed Ali Hussain, B.Sc., M.Sc.

A Doctoral Thesis submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of the Loughborough University of Technology. August 1976

Supervisor: Dr. M. E. Woodward,
Department of Electronics
and Electrical Engineering

© by A. M. A. Hussain
ACKNOWLEDGEMENTS

The author gratefully acknowledges the help and advice of Dr. I. Dempster, Dr. P. Moran and Mr. D. Hoare. The author is indebted to Dr. M. E. Woodward for supervising this work and to the Electrical and Electronic Department for making this project available.

In particular the author wishes to acknowledge his deep gratitude to all members of his family, without whose financial support, encouragement and patience this project would not have been completed. Thanks and gratitude are also due to my wife whose understanding and encouragement helped no end.

Finally thanks are due to Mrs. A. Beaven for typing this work.
SUMMARY

The basic principles of sequential switching theory were first developed by Huffman and later generalised by Moore and Mealy. These techniques and subsequent ones based on them were mainly concerned with minimizing the amount of logical hardware in the form of discrete gate components. One result of this early work was the development, by Hartmanis, of an algebraic structure theory for sequential systems.

In recent years, however, the advent of MSI/LSI has changed the fundamental design requirements, and new design criteria were thus created and many of the conventional minimization methods were rendered obsolete. In particular, no systematic techniques exist for designing systems at the sub-system or system level which the MSI and LSI technology requires.

In this thesis, using the criterion of minimal total storage requirements of a given sequential switching system, the applicability of the structure theory due to Hartmanis, in conjunction with MSI/LSI modules is examined, and the different possible resulting structures are also examined for their suitability to LSI/MSI realisations. Also, the interpartition relationships that lead to these structures are studied and best possible component sizes within the different possible structures are determined. In this connection, a procedure has been developed which systematically leads to either least storage or most uniform component machines.

However, since a large section of sequential switching systems either do not decompose into convenient sizes and structures or that a large amount of redundancy has to be introduced in order to make them decompose, alternative realisation techniques which can be used to realise such systems have been developed.

These are the State and Input techniques, which resemble in some aspects the Ashenhurst-Curtis type of disjunctive decompositions, and are general and result in uniform components. The size and structure of the components can be varied so as to suit available modules. Above all, these systems offer a simple and effective method of realising asynchronous systems requiring no special state assignment. This is done through the use of inertial delays in conjunction with a decoder in the feedback loops of the system.
CONTENTS

CHAPTER ONE - INTRODUCTION

1.1 - Purpose of Project ............................ 1
1.2 - Existing Methods of Implementing Sequential Systems 4
1.3 - Design Criteria for Sequential Systems at the Sub-System Level 7
1.4 - Use of MSI/LSI Memory Modules 9

CHAPTER TWO - SURVEY OF PREVIOUS WORK

2.1 - Historical Background .......................... 11
2.2 - Synchronous Machines and their State Assignments 14
2.3 - Asynchronous Machines: their Decomposition and State Assignment 16
2.4 - Modular Decomposition .......................... 18
2.5 - ISI and MSI Techniques ......................... 19

CHAPTER THREE - THE ALGEBRAIC STRUCTURE OF SEQUENTIAL MACHINES AND THEIR STATE ASSIGNMENT

3.1 - Algebra and Finite-Automata .......................... 21
3.2 - Closed Partitions and their Properties .................. 22
3.3 - Ordering of Closed Partitions .......................... 23
3.4 - Types of Decompositions .............................. 24
3.5 - State Assignment .................................. 29
3.6 - Partition Pairs and Mm-Pairs .......................... 32
3.7 - State-Assignment using p.p. and Mm-Pairs .................. 33

CHAPTER FOUR - STRUCTURAL ANALYSIS OF STORED PROGRAM IMPLEMENTATION

4.1 - Introduction .................................. 36
4.2 - Design Criteria Using SPMs .......................... 37
4.3 - Choice of Closed Partitions ......................... 39
4.4 - Algebraic Derivations of Best Structures ................. 44
4.5 - Applications of Derived Formulae .......................... 49
4.6 - Maximum Justifiable Added Redundancy .................. 52
4.7 - Discussion and Conclusions .......................... 54
## List of Symbols and Abbreviations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSI</td>
<td>Medium Scale Integration</td>
</tr>
<tr>
<td>ISI</td>
<td>Large Scale Integration</td>
</tr>
<tr>
<td>IC</td>
<td>Integrated Circuit</td>
</tr>
<tr>
<td>X (or x)</td>
<td>Number of external inputs to a sequential system.</td>
</tr>
<tr>
<td>S (or s)</td>
<td>Number of feedback loops, or state-variables, of a sequential system.</td>
</tr>
<tr>
<td>(</td>
<td>x</td>
</tr>
<tr>
<td>N</td>
<td>Number of internal states of a sequential system.</td>
</tr>
<tr>
<td>(\sum)</td>
<td>Sum of</td>
</tr>
<tr>
<td>Z</td>
<td>Number of outputs of a system.</td>
</tr>
<tr>
<td>P</td>
<td>Number of address lines to a module.</td>
</tr>
<tr>
<td>n</td>
<td>Number of components in a system.</td>
</tr>
<tr>
<td>ROM</td>
<td>Read-Only-Memory</td>
</tr>
<tr>
<td>RAM</td>
<td>Random-Access-Memory</td>
</tr>
<tr>
<td>SIM</td>
<td>Stored-Program-Module</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>(\leq)</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>(\geq)</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>(\tau_i)</td>
<td>Propagation time of a particular component machine.</td>
</tr>
<tr>
<td>M</td>
<td>Any finite automata.</td>
</tr>
<tr>
<td>SI</td>
<td>Set of internal states.</td>
</tr>
<tr>
<td>I</td>
<td>Set of external inputs.</td>
</tr>
<tr>
<td>O</td>
<td>Set of outputs; (O_i) a particular output function.</td>
</tr>
<tr>
<td>(\delta)</td>
<td>Next-State function.</td>
</tr>
<tr>
<td>(\lambda_i)</td>
<td>Output function; in Chapter 4 - Lagrange multiplier.</td>
</tr>
<tr>
<td>(</td>
<td>\pi</td>
</tr>
<tr>
<td>(\Pi)</td>
<td>A closed partition on (S^i).</td>
</tr>
<tr>
<td>(s_i \equiv s_j(\pi))</td>
<td>(s_i) and (s_j) are in a common block of (\Pi).</td>
</tr>
<tr>
<td>(\cap)</td>
<td>Intersection of</td>
</tr>
<tr>
<td>(\cup)</td>
<td>Union of</td>
</tr>
<tr>
<td>(\subseteq)</td>
<td>Within</td>
</tr>
<tr>
<td>(\lambda_i)</td>
<td>Input-consistent partition.</td>
</tr>
<tr>
<td>(\lambda_o)</td>
<td>Output-consistent partition.</td>
</tr>
<tr>
<td>iff</td>
<td>If and only if</td>
</tr>
<tr>
<td>g.l.b.</td>
<td>Greatest lower bound</td>
</tr>
</tbody>
</table>
l.u.b - Least upper bound
\# - Number of blocks
\varnothing - Null
\prod - Product of
p.p. = (r', r) - Partition pair
M(\tau) - Front partition of a p.p.
m(\tau) - Back partition of a p.p.
M_o - Storage requirement of undecomposed system
M_D - Storage requirement of decomposed system
\varepsilon_i - Redundancy factor of ith component
S_i (or s_i) - Number of state variables of ith component
S_D - Number of state variables of decomposed structure
\gamma - Inter-partition dispersion ratio
K_i - Number of states in the largest block of \pi_i
D_i - Number of variables needed to encode the blocks of \pi_i
d_i - Number of variables needed to encode the states in
the largest block of \pi_i
\alpha_D - Maximum spread in the D-values of a particular set of partitions
\ne - Not equal to
\neq or \approx - Approximately
\chi - Next-state variable
\varphi - Present state variable; or set of present state variables
\Theta and \Phi - Algebraic functions
t_c - Cycle time - in n sec.
t_e - Access time of a SPM - in n sec.
t_{ex} - External delay - in n sec.
t_{id} - Propagation delay through decoder - in n sec.
t_{i} - Inertial delay - in n sec.
t_s - Steady state delay - in n sec.
f_{\text{max.}} - Maximum frequency
M/E - Memory Enable pin of SPM
C - Number of Columns in a state-table.
\rightarrow - NAND gate.
\rightarrow - Inverter.
\rightarrow - OR gate.
\rightarrow - NOR gate
CHAPTER ONE

INTRODUCTION

1.1 Purpose of Project

Existing methods of implementing and minimizing the size of sequential switching systems have largely been superseded by technology; the majority of these methods were developed for the minimization and implementation of these systems using discrete logic gates.

The partition theory of decomposition due to Hartmanis and Stearns \(^1-^8\) was a powerful tool when used in conjunction with discrete gate implementations and resulted in large reductions in the size of the combinational logic required. However, when using MSI/LSI memory modules for implementing switching systems, this theory loses some of its relevance. The Quine-McCluskey minimization method for reducing logical functions \(^9-^{11}\) has likewise lost its significance when applied to MSI/LSI realisations.

The logic designer nowadays has at his disposal a large variety of ICs, on each of which a large number of gates, latches, etc. are available. Above all, the logic designer has the MSI/LSI memory matrices (which have rendered the minimization procedures totally redundant). The designer is no longer concerned with reducing the combinational logic required but with minimizing either the storage requirement of the system or the total number of modules. The designer's objectives are no longer the reduction of the number of logic gates required or the total number of gate-inputs used, but minimizing the total number of input and output variables (this leads to minimum storage and, usually, few modules).
In dealing with LSI* modules, the designer has no logical or systematic method or procedure to follow in designing sequential systems since no such methods or procedures exist. The modules available today are not suited for designing at the discrete gate level since this would normally involve large redundancies. It therefore follows that what is required is a systematic design technique which would enable the logic designer to make efficient use of these modules.

One purpose of this project has been to study the application of Hartmanis's Algebraic Structure Theory\(^1-8\) to the implementation of sequential systems using present-day LSI memory modules; to study the relative merits of the various resulting structures as to the size of their resulting components as well as to their ability to reduce the storage requirement of the systems. The uniformity in size and structure of the component machines resulting from the various structures is also considered.

The concept of closed partitions on which Hartmanis's theory is based is further examined by studying the inter-partition relationships that could exist in sequential systems. The nature and structure of the closed partitions and their relation to the resulting structures of the machines, as well as the profound effect their choice has on the storage requirements, are studied in depth. This thesis develops a procedure for choosing a particular set of closed partitions so as to result in either the most uniform system structure, or in least total storage.

Another purpose of this project has been the development of novel implementation and realisation techniques for sequential systems. This

\* For the remainder of this thesis LSI implies MSI/LSI.
was achieved through the adaptation of methods and techniques used in the implementation of combinational switching systems. Decomposition techniques for combinational logic have been effectively used, with appropriate modifications, to realise sequential systems.

It is shown that these methods lend themselves in a very suitable manner to the realisation of sequential systems using available LSI modules. They lead to realisations of pre-determined component sizes and structures. These can be varied so as to suit the modules available. In the main, the component machines resulting from these techniques are uniform— a property highly desirable when using ICs. The simplicity of applying these techniques is another significant advantage in their favour. The storage requirement of most machines realised using these techniques compares favourably with their closed-partition-implemented counterparts.

The generality of these techniques is proved by applying them to both synchronous and asynchronous machines. In fact, a method of implementing asynchronous systems is proposed which results in uniform components and total hardware requirements similar to those of the synchronous counterparts. This method requires no special state-assignment, and as an immediate result of this, it leads to minimal storage requirements with no risk of malfunctioning due to hazards and races 12. This is achieved through the use of inertial delays12, of suitable duration, which eliminate the effects of races and hazards. The result is a general and simple realisation of asynchronous switching systems which does not require vast amounts of hardware and, at the same time, retains its capability of operating at its maximum possible speed, dependent only on the type of hardware used.
1.2 Existing Methods of Implementing Sequential Systems

Present-day implementation of sequential switching systems is very different from the original relay-type implementations of Huffman\textsuperscript{13-15}. The speed of operation of these systems was very limiting, and due to the size of the individual components (the relays), only small sized systems could be contemplated. The relays were superseded by the discrete logic gate which enabled the size of the systems being implemented to increase greatly. Minimization techniques such as the Quine-McCluskey method\textsuperscript{9-11} were developed to reduce the number of discrete gates required to implement a particular logic function to a minimum. Other criteria for use with other minimization techniques, such as the number of gate-inputs, were also used. Hartmanis and Steams developed their Algebraic Structure Theory in dealing with sequential machines; by decomposing a sequential system in accordance with its internal characteristics they achieved large reductions in the hardware requirements when these systems were implemented using discrete gates.

This theory, when used in conjunction with present-day technology, resulted in some drawbacks. Firstly, the resulting sub-systems were, in general, of differing sizes and structures which rendered it very uneconomical in utilizing the available LSI memory modules; secondly, only a subset of the sequential systems were readily decomposable in their reduced form. By augmenting undecomposable sequential machines, they were rendered decomposable but the chances of obtaining an economical implementation were then greatly reduced for some of the possible resulting structures.

Fig. 1.1 shows a general method of implementing a sequential system. The combinational logic forms the outputs and the next-state variables of the system. The present-state variables are delayed
versions of the next-state variables. These are either fed back through latches (or Flip-Flops) in the synchronous case, or through delays in the asynchronous case. The synchronous case, as its name implies, is governed by an external timing signal (the clock-pulse) which synchronizes the clocking of the next-state variables of the system. The operation of such a system, though restricted by this timing signal, is free from the shortcomings associated with an asynchronous system (hazards and races)\(^\text{12}\) in which no such synchronizing signal exists and thus the changes in the state variables are fed through the delays to the inputs of the combinatorial logic. This greatly complicates the state assignment problem and, in most cases, calls for a much larger number of state-variables than the equivalent synchronous type assignment.

The size of a sequential machine is governed by the number of its external inputs, \(X\), and by the number of feedback loops, \(S\), required to encode the different internal states of the machine, since each logical combination of these feedback loops, in general, indicates a distinct internal state of the sequential machine. Thus, if a machine has \(N\) internal states then:

\[
S = \lfloor \log_2 N \rfloor
\]

A very important point to consider when designing an asynchronous sequential system using combinational logic is that different logic functions may require different numbers of logic levels to implement them, and since a certain amount of propagation delay is associated with each level of logic then, in general, different numbers of levels imply unequal path propagation delays; this results in creating race conditions. Even employing additional external delays for equalisation purposes does not entirely eliminate this shortcoming. This is the main reason why very complicated state-assignments are required to overcome it.\(^\text{12}\)
Fig. 1.1 Circuit representation of a sequential machine using combinational logic.

Fig. 1.2 Sequential machine representation using a Stored Program Module.
The advent of LSI memory modules has meant that the combinational logic of Fig. 1.1 can be replaced by a single semiconductor memory module. This is shown in Fig. 1.2. The outputs of this module can be considered as either system outputs or as next-state variables. In other words, this module exactly replaces the combinational logic in the previous implementation. The advantage of using these memory modules over the conventional implementation is that all the logic paths have near equal propagation delays.

However, only small and medium size sequential machines can be easily implemented using these modules for the simple reason that very large ones are not at present available. Sequential systems implemented using a single memory module usually incorporate large amounts of redundant storage because of the unavailability of the exact size module required. In general, sequential systems include, through their structure, redundancies which when eliminated (through decomposition for example) result in an appreciable reduction in storage requirements.

The partition theory due to Hartmanis 7,8 achieves exactly this when dealing with discrete gates. This theory is applicable in conjunction with LSI logic, though with somewhat reduced effectiveness, since, in general, it leads to submachines of non-uniform sizes and structures which makes it unsuitable for such implementations. Above all, this theory is applicable in its entirety when dealing with synchronous systems only. With asynchronous systems, this theory can only be applied, in a straightforward manner, when the resulting structures happen to be of the parallel type 16. The serial case leads to complex timing problems, the solution to which may require state assignments utilizing a large number of state-variables.
1.3 Design Criteria for Sequential Systems at the Sub-System Level

When using discrete gates for the implementation of sequential systems, it is either the total number of gates used, or the total number of gate-inputs that are taken as the criterion by which the size of the system is judged. Thus, when minimization and decomposition techniques are used to reduce the amount of hardware required for a particular implementation, it is mainly these factors which are used as the bases for comparison.

When using either of these two criteria for judging the size of a sequential system, a reasonably clear picture is formed as to the size and structure of the system under consideration. This is due to the limited spectrum of possible discrete gate sizes available, and thus due to the limited number of possible ways in which a given system can be realised using discrete gates.

The reason why using either of these two criteria are nearly equivalent is that the size of a discrete gate increases, more or less, linearly with the number of its inputs. Thus, for example, the size of a system requiring 4 gates of 8 inputs each and one gate of 4 inputs is, more or less, equivalent to another (in size) requiring 9 gates of 4 inputs each.

However, when dealing with LSI memory modules, two possible criteria by which the size of the system can be judged are now discussed. These are:

1. The total storage capacity of the modules used.

This is given by:

$$\sum_{i=1}^{n} z_i \cdot 2^{P_i}$$

\[ (1,1) \]

(where $P_i$ is the number of address lines to the $i^{th}$ module; $z_i$ is its...
number of outputs, and \( n \) is the number of modules used).

Using binary number representation, with each additional input, the number of storage locations (or memory words) in a module is doubled and with each additional output, the number of memory bits in each word is increased by one. Thus the size of a module increases logarithmically with the number of inputs, and linearly with the number of outputs; i.e. for implementation purposes, the reduction of the number of inputs to the various modules is the item of major concern if storage is to be most efficiently utilized.

2. The total number of module-inputs, and outputs.

Due to the large number of LSI modules available at present, coupled with the widely varying storage capacity of modules with a given number of inputs and outputs, and of the different possible structures, it therefore follows that quoting a figure of total number of module-inputs and outputs forms no clear indication as to either the system structure that is used or to the relative sizes of the different modules. Also, because of the storage capacity of a module being a logarithmic function of the number of inputs, to quote a total figure of module-inputs and outputs leads to a practically endless number of ways in calculating the storage of the system, since each combination of inputs and outputs leads to a totally different storage capacity. For example, when a figure of, say, 20 module-inputs and outputs is quoted, the number of possibilities of satisfying such a figure, from the wide spectrum of available modules, is enormous.

Thus, from this, using the actual number of module inputs and outputs, separately or collectively, as a criterion for judging the sizes of two or more possible structures is somewhat less effective than using the total store size.
Hence from these arguments, the only possible consistent criterion for judging the relative sizes of two or more structures in conjunction with LSI modules, and with a clear indication as to the actual size of the system, is the actual number of storage bits in the structures. In this thesis, this is the criterion which is taken as the basic one for judging the effectiveness of different structures.

However, because of the logarithmic nature of the relationship between the structure of an LSI module and its storage capacity, it follows that this criterion is a non-linear one.

1.4 Use of MSI/LSI Memory Modules

Conventionally, a sequential machine is implemented using combinational logic in conjunction with some memory elements as shown in Fig. 1.1. This combinational logic, as was pointed out in Section 1.2, can be replaced by an MSI/LSI module of the correct size and structure (i.e. of the correct number of memory bits, and the correct number of inputs and outputs respectively).

Memory modules can be of several types, each based on a different manufacturing principle\textsuperscript{17,18}. However, the two major types are:

1. The Read-Only-Memory (ROM): with this type of memory, the information contained in the module is supplied by the logic designer and is programmed during manufacture. The information is thus permanent and can only be addressed or read-out (with some ROMs, the information can be erased using ultra violet light).

2. The Random Access Memory (RAM): these are volatile modules in which the programmed information is lost once the power to the system is turned off. A re-programming is thus necessary.
every time the system is switched on.

However, the theory developed in this thesis is intended to be mainly device-independent and, as such, all storage matrices will subsequently be referred to as Stored-Program-Modules (SPM), irrespective of their type.

The information is programmed into an SPM in accordance with the truth table of all the functions in the sequential machine. A particular location in the memory is programmed by means of addressing it and then inputting the required information on the data inputs of the SPM - the Read and Write Enables, where applicable, being correctly addressed.

When all locations have been programmed in this manner, then the SPM acts as a combinational circuit, giving a particular output for a particular input (to its address lines) and whenever the same address input recurs, the same output is issued from the SPM.

Where the implementations of sequential systems differ from those of combinational systems is in the fact that some of the outputs of the SPMs are fed back to the inputs of the module to give it its sequential property - as shown in Fig. 1.2.
CHAPTER TWO

SURVEY OF PREVIOUS WORK

2.1 Historical Background

Switching systems can be either of the combinational logic type or of the sequential type (the former has its outputs dependent only on the present state of its inputs; whereas, the latter has an in-built memory which allows the system to have its own internal state and thus the outputs of such systems are dependent on the internal state of the system as well as on its inputs). These two types share similar fundamental design, minimization, and implementation principles, and background. The two cases are treated separately below:

2.1.1 Combinational switching systems

The mathematical theory of switching circuits was first postulated by Shannon, and was based on a 2-valued Boolean algebra. The formal statement of a combinational system leads directly to a Boolean canonical function which must, in the majority of cases, be simplified to provide an economical hardware solution. There are many methods of minimizing combinational systems. Two of the most common are:

1. The Karnaugh Map Method:

   This is a graphical technique based on the Venn diagram; was originally proposed by Veitch, and later improved by Karnaugh. However, this method can only be used in conjunction with systems of a small number of variables; i.e. \( s \leq 6 \).

2. The Tabulation Method:

   For problems with a large number of variables a tabular or algorithmic method due to McCluskey, based on an original technique due
to Quine \(^9,10\), is used.

The actual realisation of combinational systems can be carried out in several ways according to the type and size of the system \(^{21,24}\).

Combinational switching functions can be decomposed using the chart method developed by Ashenhurst \(^{25,26}\). The resulting disjunctive decomposition can either be simple or complex, depending on whether the resulting structure has two or more components. All possible decomposition structures can be chosen from decomposition charts showing all possible variable dependencies \(^{26}\).

When a system is functionally decomposed, its hardware requirement is appreciably reduced. However, apart from the fact that not all functions are non-trivially decomposable, there is no guarantee as to how useful the resulting structure would be.

2.1.2 Sequential switching systems

Modern sequential switching network theory was pioneered by Huffman \(^{13}\) who, using an asynchronous-type sequential machine, laid down the basic rules for synthesizing and implementing sequential machines. He also studied the state-assignment problem, detection of race conditions, equivalence, merging, and developed the state-table technique of representing sequential machines. Huffman used relays as basic building blocks for his practical implementation and studied the memory requirements of sequential switching circuits \(^{14}\) and the design of hazard-free switching networks \(^{15}\).

About the same time, and independently of Huffman, Moore \(^{27}\) was developing a similar theory, in a more abstract form, for synthesizing synchronous sequential machines. Moore considered machine equivalence by means of inputting known sequences into a particular machine (treated
as a black box) and observing the output sequences; his model had just one set of outputs associated with each internal state.

It was Mealy \(^{28}\) who developed a general model for sequential machines of which a Moore-type machine may be regarded as a special case. Mealy, like Moore, considered the synchronous type machine but applied his theory to asynchronous relay cases (i.e. to Huffman-type machines). Mealy implemented his machines from discrete logic gates. Merging of internal states and equivalence tests were carried out (using Moore's findings) in order to reduce the size of the state table so as to require the least number of delay elements and discrete components in the practical implementation. Unlike the Moore model, the Mealy machine can have several output sets related to each internal state.

Starting from the Mealy model, Aufenkamp \(^{29-31}\) studied the equivalence concepts of sequential machines and developed a "connection matrix" which defined the complete behaviour of the sequential machine. Aufenkamp then partitioned the states of this matrix for the purpose of synthesizing sequential machines; he also studied the effects of input restrictions on the number of states of a machine. Gill \(^{32}\) followed by applying the concept of the connection matrix to achieve decompositions of finite automata.

However, the methods and techniques of decomposition for the two different types of sequential systems (i.e. the synchronous and the asynchronous) are different, not only in nature but also in complexity; the two different types are discussed separately below:
2.2 Synchronous Machines and their State-Assignments

Hartmanis's algebraic structure theory was a novel step in designing and implementing sequential machines using discrete logic gates. This theory offered a new method of internal state assignment: this is based on the closed partition concept. In his first paper the closed partition property was used to decompose sequential machines into two or more components operating in parallel. Hartmanis showed algebraically that the components operating in parallel were isomorphic to the original machine.

This loop-free structure of machines was further developed by Hartmanis and by Steams and Hartmanis to include cascade and composite decompositions. The concept of closed partitions was also generalised by the two authors into the partition pair and pair algebra concepts. The partial ordering of partitions and partition pairs into lattice structures was also developed.

General procedures for deriving and utilizing closed partitions and partition pairs for the purpose of reducing variable dependencies were laid down; thus, state-assignments with reduced dependencies resulted and, as a consequence, a sizeable reduction in hardware requirements was obtained.

The technique of state-splitting for introducing redundant states into the state-table of a sequential machine in order to make it decompose was first developed by Kohavi. Hartmanis and Steams employed this technique to further develop their machine structure theory.

Curtis generalised Hartmanis's methods to produce efficiently and systematically, for finite-state sequential machines, state-assignments in which state-variables were optimally reduced; partition pairs as well
as closed partitions were used in developing these state-assignments. Using the principles of decomposition theory as developed by Ashenhurst, Curtis further generalised his state-assignment technique so as to produce minimal variable dependencies.

Yeoli and Kohavi studied the structure theory and loop-free decomposition techniques of finite automata on similar lines to Hartmanis and Stearns, though not on such an abstract and algebraic level. Most of their findings were extensions of those of Hartmanis and Stearns.

Zeiger, basing his theorems on the general group theories of Rhodes and Krohn, developed a cascade-decomposition scheme where the resulting component machines either permuted their states or reset them to a particular state, depending on the state of the external inputs.

The methods and procedures of deriving state assignments which result in reduced variable dependencies, whether they be based on closed partition or partition pair concepts, were treated fully by Hartmanis and Stearns and by Kohavi. Karp produced an algebraic treatment for state-assignments which resulted in reduced variable dependence; Karp also developed algorithms for choosing the appropriate partition pairs for the purpose of constructing the most economical state-assignments. Dolotta and McCluskey also studied state-assignments which resulted in the least number of gates, and gate-inputs, required.

Appendix 1 illustrates the exact methods of using the closed partition and partition pair concepts for the state assignments of synchronous machines.
2.3 Asynchronous Machines: their Decomposition and State-Assignment

The synthesis of asynchronous sequential machines was pioneered by Huffman\(^1\) who laid down the fundamental rules for designing and implementing asynchronous machines. In a later paper Huffman\(^3\) treated the design of hazard-free asynchronous machines. Unger\(^4\) followed by treating asynchronous machine design taking into account the presence of hazards (due to unequal delays in the different paths of the circuit). Unger\(^5\) treated the problem of hazards by using additional external delays. When no essential hazards are contained in the flow table of a sequential machine, Unger\(^6\) developed a method of state-assignment which required no delay elements.

The synthesis of asynchronous machines by Unger was the equivalent of Hartmanis's synthesis of synchronous ones. Kinney\(^7\), Chung-Jan Tan\(^8,9\), Armstrong\(^10,11\) and Tracey\(^12\) also treated the synthesis of asynchronous machines and the problem of internal state-assignment, extensively.

Frosini and Gerace\(^13,14\) synthesized a restricted class of asynchronous sequential machines (Pulse-Input) and studied the problem of its state-assignment. Transformation algorithms from any asynchronous machine to a pulse-input type machine were also outlined.

The treatment of incompletely specified asynchronous state-tables was outlined by Unger\(^15,16,17\), and Reed\(^18\). This problem was made much more complex than in the equivalent synchronous case since the unspecified states were incorporated in state-merging, reduction of state tables and, thus, in reducing the number of state variables required for a particular state-assignment.

Hartmanis's algebraic structure theory was also applicable to the asynchronous case except that the timing and the state-assignment
problems were made more complex\textsuperscript{16,58}. The parallel decomposition of asynchronous machines could be carried out on similar lines to the synchronous case but with each component machine being implemented in accordance with a valid asynchronous state-assignment. The serial decomposition was made more complex by the presence of timing problems; this was due to the fact that changes in the state of any of the different cascaded components would have had to propagate through the remaining successor component machines in order to achieve a change in the internal state of the total machine. Tan\textsuperscript{16} treated this problem by considering two separate cases:

1. The internal state of the predecessor component would be made to change first and then it was followed by a change in the state of the successor component. Thus, when a machine was decomposed into \( n \) component machines, the final successor machine arrived at its final steady state after \( nT' \) seconds of propagation delay (where \( T' \) was the time taken for a change in state to propagate through one component machine).

2. The opposite of (1) where the successor component machine changed its internal state first; this was then followed by a change in the state of the immediate predecessor machine and so on ....

It should be noted that in both cases the delay in arriving at the final output of the machine was accumulative.

The flexibility of the design specifications played an important role in selecting a particular state-assignment and thus in the number of state-variables needed for that assignment; e.g. when more than one state-variable was allowed to change simultaneously, then precautions against critical races had to be taken and this, in general, implied that a larger number of state-variables was needed\textsuperscript{12} for a particular
sequential machine than the minimum that would be required for the number of internal states.

A summary of the state-assignments that are most frequently used with asynchronous machines is given in Appendix 2.

2.4 Modular Decomposition

Synchronous, as well as asynchronous, sequential machines can be decomposed into interconnections of component machines of a fixed internal structure. Each of these components is called a "Module" and each has a fixed number of inputs which may be connected to machine variables, to fixed logical constants ("ones" or "zeros"), to the outputs of other modules or to some control inputs which select between the various remaining input lines. Each module has only one output which takes up the same value as the selected input.

In spite of the fact that modular decomposition is suitable for large scale integration purposes, the number of modules required to implement a particular sequential machine makes it a very uneconomical method of implementation. The size and thus the internal structure of the particular module used has a direct bearing on the number of modules required for a particular implementation.

The difference between the Synchronous Modular Decomposition and the Asynchronous Modular Decomposition is that whereas the number of "levels" of modules in the synchronous case is immaterial, since with each clock pulse the information contained in the various modules propagates to the next-level and stops there, awaiting the next clock pulse, the delay in the asynchronous case is accumulative. In general the modules are connected in parallel to limit this effect in asynchronous machines. Also, with such implementations of asynchronous
machines, the usual restriction of only one variable changing at any one time is assumed.

In general the use of "modules" for designing and implementing sequential machines has not proved very useful or economical.

2.5 LSI and MSI Techniques

The uses of MSI and LSI storage techniques in conjunction with digital computer circuitry, automatic telephone switching networks, and, as the main fast access memory stores in digital computers and automatic telephone exchanges are very well known. SPMs are also extensively used in the implementation of combinational and sequential circuits. The speed of operation of these modules makes them very attractive for use in the design of fast digital processing systems. The use of these elements in the design and implementation of the microprogram section of the control unit of digital computers, and in the control and microprogram units of automatic exchanges, are also well known. Grasselli and Montanari studied the minimization of ROM sizes in microprogrammed digital computer applications. Effective encoding techniques for reducing the word lengths in microprograms based on the Maximal Compatibility classes were outlined.

The implementation of sequential machines has been greatly influenced by the advent of LSI and MSI techniques. This has had the effect of rendering the existing design procedures of these machines (which were based mainly on discrete gate implementations) very nearly obsolete. Thus, new methods and design procedures are needed in order to make the best use of these modules. Howard outlined the effects of implementing sequential machines using SPMs.

The speed of operation of such implementations is greatly increased.
since the access time of these storage modules is very short as compared to multi-level gate implementations with all that it entails of unequal path propagation delays. The design principles are also simplified since the designer is no longer concerned with the time-consuming process of reducing the combinational logic required for a particular implementation, but with using the least amount of storage possible.

The amount of storage required to implement a sequential machine depends to a large extent on the number of inputs and on the number of state variables required to encode the internal states of the machine. In fact the relationship between the storage required and the total number of inputs is logarithmic since for each added input to the memory store, the number of words or locations in that store is doubled.

This leads to two demands:

1. To reduce the number of total inputs to the store as far as possible by means of eliminating some of the external inputs to the machine, or by reducing the number of state-variables to an absolute minimum by choosing an appropriate state-assignment \(^7,46\).

2. To decompose large sequential machines in order to be able to implement them using existing LSI modules. The need for decomposition stems from the fact that very large sequential machines require very large (and as yet unavailable) modules, and that implementing large sequential switching systems using a single memory module is very wasteful in terms of storage requirements.
CHAPTER THREE

THE ALGEBRAIC STRUCTURE OF SEQUENTIAL MACHINES AND THEIR
STATE ASSIGNMENT

3.1 Algebra and Finite-Automata

Any finite automata, $M$, may be represented algebraically as a quintuple:

$$M = \left\{ S^i, I, 0, \delta, \lambda \right\}$$

where

- $S^i$ = finite non-empty set of states,
- $I$ = finite non-empty set of inputs,
- $O$ = finite non-empty set of outputs,
- $\delta : S^i \times I \rightarrow S^i$ is the transition or next-state function.

And, for a Moore-type machine:

$$\lambda : S^i \rightarrow O$$ is the output function,

whereas for a Mealy-type machine:

$$\lambda : S^i \times I \rightarrow O$$ is the output function.

A Moore-type machine thus has its output function "$O$" mapped by the set of states "$S^i$" and, whenever a particular state occurs, one output (or a set of outputs) which is associated with that particular state is obtained. The output is then thought to occur when the machine enters the internal state associated with that output.

A Mealy-type sequential machine has its output function "$O$" mapped by the set of states "$S^i$" as well as by the set of external inputs $I$. A particular output is associated with a particular combination of internal states and external inputs. The outputs in this model are then thought of as occurring during the transitions between states (precipitated by a change in the external inputs).
Fig. 3.1 Schematic representation of $M_\epsilon (S,I,O,\delta,\lambda)$.

Fig. 3.2 Parallel decomposition of $M$ into $n$ components.
Fig. 3.1 schematically illustrates the above formulation for the general (Mealy) case. In the Moore case the line from the input source to the output logic is omitted.

3.2 Closed Partitions and their Properties\textsuperscript{7,8}

Definition 3.1: A partition, \( \tau \), on a set \( S' \) is a disjoint number of blocks whose union is the set \( S' \).

Definition 3.2: A closed partition (or a partition with the Substitution Property), \( \pi \), on \( S' \) is a partition such that if \( s_i \) and \( s_j \) are two states in the same block of \( \pi \) then for every \( I_k \) in \( I \), \( \delta(s_i, I_k) \) and \( \delta(s_j, I_k) \) are in a common block of \( \pi \). Put algebraically:

\[
\text{If, } s_i = s_j(\pi) \quad \text{i.e. } s_i \text{ and } s_j \text{ are in a common block of } \pi \quad \text{then,} \\
(s_i, I_k) = (s_j, I_k)(\pi).
\]

Definition 3.3: The zero partition, \( \pi(0) \), is the partition in which each state in \( S' \) constitutes a separate block.

Definition 3.4: The Unity partition, \( \pi(1) \), is the partition in which all the states in \( S' \) constitute one block only.

If \( \pi_1 \) and \( \pi_2 \) are two closed partitions, then so are \( \pi_1 \cdot \pi_2 \) and \( \pi_1 + \pi_2 \), where \( \pi_1 \cdot \pi_2 \) is the intersection \( \bigcap \), and \( \pi_1 + \pi_2 \) is the union \( \bigcup \) of \( \pi_1 \) and \( \pi_2 \).

A partition \( \pi_1 \) is said to be greater than, or equal to, a partition \( \pi_2 \) iff every block of \( \pi_1 \) is greater than, or equal to, every block of \( \pi_2 \).

From the above, it is clear that \( \pi(0) \) is the smallest possible closed partition and, \( \pi(1) \) is the largest possible closed partition on \( S' \).

Definition 3.5: A partition \( \lambda_1 \) on the states of a machine \( M \) is said to be input-consistent iff, for every state \( s_1 \) of \( M \) and all inputs \( I \), the next states \( \delta(s_1, I_k) \), for all \( I_k \leq I \), are in the same block of \( \lambda_1 \).
Thus if a closed partition $\Pi$ also happens to be input consistent then the component machine realised through the use of that particular partition would be independent of all external inputs. Such machines are termed "clocks" since they depend, for their state-sequencing, on just the external clock pulse.

An extension of the above definition is when a closed partition $\Pi$ is input consistent with respect to a subset of the external inputs $\mathcal{I}$. Thus, if a component machine is based on such a partition, then it would be independent of the external inputs with respect to which $\Pi$ is consistent.

Definition 3.6: A partition $\lambda_0$ on the states of a machine $M$ is said to be output-consistent iff, for every block of $\lambda_0$ and every input $I_k \subseteq \mathcal{I}$, all the states contained in the block have the same outputs.

The existence of an output-consistent partition $\lambda_0$ on the states of $M$ implies that there exists an assignment for $M$ such that the outputs depend, at most, on the external inputs and on the variables assigned to the blocks of $\lambda_0$.

### 3.3 Ordering of Closed Partitions

A lattice is a partially ordered set in which every pair of elements has a unique g.l.b. (greatest lower bound) and a unique l.u.b. (least upper bound).

The g.l.b. of two closed partitions $\Pi_1$ and $\Pi_2$ is $\Pi_1 \cap \Pi_2$. The l.u.b. of $\Pi_1$ and $\Pi_2$ is $\Pi_1 + \Pi_2$.

The set of closed partitions on the set of states, $S'$, of a sequential machine, $M$, forms a lattice, $L$, under the natural partition ordering (i.e. under the "$+$" and "." operations). The lattice of closed
partitions contains the trivial partitions of $\Pi(0)$ and $\Pi(1)$. Thus, a lattice, when considered in conjunction with the set of closed partitions shows the relationships between the different partitions and, in a sense, is a graphic representation of the different possible decomposition structures discussed below.

3.4 Types of Decompositions

Based on the theory of partitions, a sequential machine may be decomposed in a number of different structures. Interconnecting the resulting submachines in a predetermined manner results in the realisation of the original machine. Often, this process results in a reduction in hardware requirements. In this section the various decomposition schemes are briefly outlined.

3.4.1 Parallel Decomposition

When a sequential machine possesses two or more closed partitions whose product (or intersection) results in the zero partition $\Pi(0)$ then this machine can be decomposed into two or more components operating in parallel. Each of the submachines would then be based on a distinct closed partition. The number of state-variables of component machine $j$ would then be:

$$S_j = \log_2 \#(\Pi_j)$$

Thus, if a machine possesses $n$ closed partitions $\Pi_1, \ldots, \Pi_n$ such that

$$\Pi_1 \cdot \Pi_2 \cdot \ldots \cdot \Pi_n = \Pi(0)$$

then this machine can be decomposed into $n$ component machines operating in parallel.

Algebraically, if $M_1$ and $M_2$, connected in parallel, realise $M$ then:
\[ S'_1 \times S'_2 \rightarrow S' \quad ; \quad I_1 \times I_2 \rightarrow I \]
\[ \delta_1 \times \delta_2 \rightarrow \delta \quad ; \quad \lambda_1 \times \lambda_2 \rightarrow \lambda \]

and,
\[ 0_1 \times 0_2 \rightarrow 0 \]

where,
\[ M_1 = \left\{ S'_1, I_1, 0_1, \delta_1, \lambda_1 \right\} \]
\[ M_2 = \left\{ S'_2, I_2, 0_2, \delta_2, \lambda_2 \right\} \]

and,
\[ M = \left\{ S, I, 0, \delta, \lambda \right\} . \]

The next-state and output functions are defined as:
\[ \delta \left\{ (s_1, s_2) \ ; \ (x_1, x_2) \right\} = \left\{ \delta_1(s_1, x_1) \ ; \ \delta_2(s_2, x_2) \right\} \]
and,
\[ \lambda \left\{ (s_1, s_2) \ ; \ (x_1, x_2) \right\} = \left\{ \lambda_1(s_1, x_1) \ ; \ \lambda_2(s_2, x_2) \right\} \]

where
\[ x_1 \subseteq I_1 \ ; \ x_2 \subseteq I_2 \ ; \ s_1 \subseteq S'_1 \ ; \ s_2 \subseteq S'_2 . \]

For parallel decomposition, \( S'_1 \) and \( S'_2 \) are disjoint subsets of \( S' \).
i.e. \( S'_1 \cap S'_2 = \emptyset \). \( I_1 \) and \( I_2 \) may or may not be disjoint subsets of \( I \). The outputs of the original machine may be obtained from the separate component machines by means of some output logic circuitry.

Fig. 3.2 shows a schematic representation of parallel decomposition.

3.4.2 Serial Decomposition 7,8

When a sequential machine possesses one closed partition, \( \Pi \), it can be decomposed into two component machines operating in cascade. The predecessor component would be based on \( \Pi \); whereas, the successor component would be based on a non-closed partition, \( \mathcal{T} \), such that:
\[ \Pi \cdot \mathcal{T} = \Pi (0) \]

The predecessor component is only dependent on \( I \) and on the variables assigned to the blocks of \( \Pi \); whereas, the successor component would either be:

(i) dependent on the outputs of the predecessor machine only. In other
Fig. 3.3 Serial decomposition of $M$ into $n$ components.

Fig. 3.4 Serial decomposition of a State/Moore machine into $n$ components.
words, the predecessor machine provides a set of outputs which are fed as inputs to the successor machine, as in the Mealy-model of sequential machines shown in Fig. 3.3.

Algebraically, if machines $M_1$ and $M_2$, when cascaded, realise $M$ then:

$$I_1 = I; \; 0_1 = I_2 \; \text{and} \; 0 = 0_2$$

where,

$$M_1 = \{S_1', I_1, 0_1, \delta_1, \lambda_1\},$$

$$M_2 = \{S_2', I_2, 0_2, \delta_2, \lambda_2\},$$

and,

$$M = \{S', I, 0, \delta, \lambda\}.$$

The next-state of $M_1$ is given by $\delta_1(s_1', x)$, where $x \subseteq I$, and $s_1' \subseteq S_1$.

The next-state of $M_2$ is given by $\delta_2\{s_2', \lambda_1(s_1, x)\}$, where $s_2' \subseteq S_2$; i.e. the next-state of the total machine is given by:

$$\delta\{(s_1', s_2'), x\} = \{\delta_1(s_1, x), \; \delta_2\{s_2', \lambda_1(s_1, x)\}\}$$

In the above formulation, it should be noted that the final output of the total machine is arrived at after a delay equivalent to the sum of the delays of $M_1$ and $M_2$, since $M_1$ has to compute its next-state and outputs before $M_2$ can compute its next-state and outputs. Thus, if there are $n$ component machines connected in this manner then the propagation delay would be $n$ times the delay through one component. This is a serious limitation of serial decomposition.

The output function $\lambda$ is obtained from the final successor component:

$$\lambda\{(s_1', s_2'), x\} = \lambda_2\{s_2', \lambda_1(s_1, x)\}$$

*(ii) dependent on the set of external inputs $I$ as well as the set of present-state variables of the predecessor machine as in the cases of the Moore and State machines, a schematic drawing of which is given in Fig. 3.4.*

* Assuming a 2-component serial structure.
A state machine $M$ is a 3-tuple:

$$M = (S', I, \delta)$$

where the symbols are as defined previously.

$M$ is said to decompose into two components $M_1$ and $M_2$ operating in cascade such that:

$$M_1 = (S'_1, I_1, \delta_1) \text{ and } M_2 = (S'_2, I_2, \delta_2)$$

where, $S'_1 \times S'_2 \rightarrow S$ ; $S'_1 \times I_1 \rightarrow I_2$ and, $I_1 = I$.

This implies that the external inputs $I$ are common to both $M_1$ and $M_2$.

When $M$ has an output function $\lambda$ then:

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

where,

$$\delta \{ (s_1, s_2), x \} = \left\{ \begin{array}{l} \delta_1(s_1, x), \delta_2(s_2, (s_1, x)) \end{array} \right\}$$

Thus, unlike the previous case, the next-state of the successor component is computed separately from the next-state of the predecessor machine since the next-state of any component machine is dependent upon the external inputs as well as the present-state of the immediate predecessor component.

The output function of this type of circuit is obtained through a mapping of:

$$\lambda : S_1 \times S_2 \times I_1 \rightarrow O$$

When a machine possesses $n$ closed partitions such that:

$$\pi_1 > \pi_2 > \pi_3 \ldots \ldots \pi_n$$

where $\pi_n = \pi(0)$,

then this machine is decomposable into $n$ component machines operating in cascade such that the first predecessor machine would be based on $\pi_1$, the second predecessor (i.e. the first intermediate) machine would be based on a non-closed partition $\tau_2$ such that:

$$\pi_1 \cdot \tau_2 = \pi_2$$

The third predecessor machine would be based on $\tau_3$ such that:
\[
\pi_2 \cdot \tau_3 = \pi_3
\]

and, in general, the \( j \)th predecessor machine would be based on \( \tau_j \) such that:
\[
\pi_{j-1} \cdot \tau_j = \pi_j
\]
The final successor machine would be based on \( \tau_n \):
\[
\pi_{n-1} \cdot \tau_n = \pi(0), \text{ since } \pi_n = \pi(0).
\]

3.4.3 Composite Decompositions \(^7,8\)

There are two types of composite decomposition:

(a) When one or more parallel components are followed by a successor machine:

If a machine possesses \( n \) closed partitions such that:
\[
\pi_1 \cdot \pi_2 \cdot \ldots \cdot \pi_n = \pi_p \text{ and, } \pi_p > \pi(0),
\]
then this machine can be decomposed into \( n \) parallel components (with an effective closed partition \( \pi_p \)) followed by a successor component machine based on a non-closed partition \( \tau \) such that:
\[
\pi_p \cdot \tau = \pi(0)
\]
The parallel components would be based on the individual closed partitions \( \pi_1, \ldots, \pi_n \), as in the case of parallel decomposition considered previously.

The algebraic relationships between \( I, S, O, \delta \) and \( \lambda \) for such decompositions follow closely the derivations for the parallel and serial cases.

(b) When a common component is followed by two or more components in parallel:

If a machine possesses \( n \) closed partitions such that:
\[
\pi_1 \cdot \pi_2 \cdot \ldots \cdot \pi_n = \pi(0)
\]
and, another closed partition \( \pi_s \) such that:
then a common component machine can be "factored" out of all the parallel components; thus acting as a predecessor to all of them.

The \( j \)th parallel component machine would be based on a non-closed partition, \( \tau_j \), such that:

\[
\pi_s \supseteq \pi_1 \supseteq \pi_2 \supseteq \cdots \supseteq \pi_s \supseteq \pi_n.
\]

3.5 State Assignment

When a sequential machine is implemented in terms of hardware, all the functions have to be encoded in binary form. The problem (in general) specifies the input and output variables in terms of binary notation. Thus, the logic designer is left with the coding of the internal states; this depends on two factors:

1. The type of sequential machine the designer is dealing with, i.e. synchronous or asynchronous.
2. The resulting structure required, and operation specifications.

The synchronous and asynchronous cases are discussed separately below.

3.5.1 Synchronous State-Assignment

The operation of the synchronous class of sequential systems is governed by an external timing signal (the clock pulse) and, provided a sufficient time interval is allowed between successive pulses (so as to allow the circuit to arrive at its correct next-state), then correct operation of the circuit is assured irrespective of the type of internal state assignment used in the realisation. However, the internal state assignment of this class of switching systems does have a decisive effect on:

1. the amount of hardware required - whether in terms of discrete gates
or total LSI storage locations required; and

ii) the size and structure of LSI modules that would be required for a particular implementation.

The coding of the internal states of this class of sequential systems is thus carried out so as to obtain the maximum possible reduced functional dependencies amongst the state-variables. The concepts of closed partitions and partition pairs are the tools employed for achieving this.

If a sequential machine possesses one or more closed partitions, these can be used to achieve a reduction in the functional dependence of the state-variables and possibly in that of the output functions of the machine. The extent of this reduction is a function of the number and structure of these partitions. The inter-partition relationships and their partial ordering also has a profound effect on the state-assignment used, and thus on the structure and hardware requirement of the machine.

The state-assignment based on the closed partition concept may or may not be a satisfactory one in terms of the resulting structure and/or in terms of the hardware required. This depends entirely on the S.P. partitions, and their properties, the machine possesses; indeed, some machines in their reduced form may not possess any non-trivial closed partitions.

Another type of state assignment is one which is based on the concept of partition pairs. The resulting structures are not detectable as SP partitions, and may result in more reduced functional dependencies. This is discussed in some detail later on in this Chapter.

3.5.2 Asynchronous State-Assignment

Asynchronous sequential systems have no synchronizing signal to govern the changes in their internal states. The free running nature of these systems gives rise to a very complex and difficult problem - the
state assignment. The reason for their complexity is the presence of races and hazards (the races are a result of inaccurate circuit design, whereas the hazards are the consequence of variations in the physical components used in the implementation). Thus when an asynchronous machine is designed, its correct operation must be assured by taking into account all possible races and hazards. This, in the vast majority of asynchronous systems, implies an appreciable increase in the number of variables required for the state assignment which, in its turn, may imply a large increase in the hardware requirement of the system. In particular, when dealing with LSI realisations, this increase in the number of variables may imply an enormous increase in the size of the required module (since the relation between the number of inputs to a module and its size is a logarithmic one).

In assigning codes to the internal states of an asynchronous system, some restrictions are, almost always, placed on the manner in which the state-variables are assigned. In fact, with most state assignments, the number of variables and external-inputs changing simultaneously is usually limited to only one, so as to overcome these difficulties.

Various techniques (see Appendix 2) can be used to reduce the number of state-variables required to an acceptable level but this is usually carried out at the expense of some other parameter, such as reduced maximum speed of operation and the appearance of spurious outputs which accompany multi-transition state assignments.

However, irrespective of the type of assignment used to realise an asynchronous system, the hardware requirement, in all but the very small systems, suffers an appreciable increase over the equivalent (in size) synchronous system.
3.6 Partition Pairs and Mm-Pairs

A partition pair \((\tau, \tau')\) on the set of states \(S'\) of \(M\) is an ordered pair of partitions such that if \(s_i\) and \(s_j\) are in the same block of \(\tau\) then, \(\delta(s_i, x)\) and \(\delta(s_j, x)\), where \(x = I\), are in the same block of \(\tau'\); i.e. the partition \(\tau'\) consists of the blocks of next-state (or successor states) implied by the partition \(\tau\) on the present-states.

For any partition \(\Pi\) on \(S'\) of \(M\), \((\Pi, \Pi(I))\) and \((\Pi(0), \Pi)\) are trivial partition pairs.

If \((\Pi, \Pi')\) and \((\tau, \tau')\) are p.p.s on \(S'\) of \(M\) then \((\Pi, \Pi', \tau, \tau')\) and \((\Pi + \tau, \Pi' + \tau')\) are also p.p.s on \(S'\).

If \((\tau, \tau')\) is a p.p. on \(S'\) of \(M\) then so are:

\((\tau, \tau'_S)\) and \((\tau'_I, \tau')\) where, \(\tau'_S \supseteq \tau'\) and, \(\tau'_I \subseteq \tau\).

If \(\tau'\) is a partition on \(S'\) of \(M\) then, define a partition \(M(\tau')\) such that:

\[ M(\tau') = \sum \tau'_i \]

where the sum is taken over all partitions \(\tau'_i\) such that \((\tau'_i, \tau'_I)\) is a partition pair on \(S'\).

Similarly, define a partition

\[ m(\tau) = \prod \tau'_i \]

where the product is taken over all \(\tau'_i\) such that \((\tau, \tau'_I)\) is a partition pair on \(S'\).

Any p.p. \((\tau, \tau')\), where \(\tau = M(\tau')\), and, \(\tau' = m(\tau)\) is called an Mm pair.

For any given \(\tau\), \(m(\tau)\) represents the largest amount of information which can be computed about the next-state of \(M\) knowing only the block of \(\tau\) in which the present-state of \(M\) is. Similarly, for any given \(\tau'\), \(M(\tau)\) represents the least amount of information which has to be available.
about the present-state of \( M \) in order to compute \( \tau' \) for the next-state.

If \( (\lambda, \lambda') \) is an \( \text{Mm} \) pair on \( S' \) of \( M \), then \( \lambda \) is the largest possible partition whose successor states are contained in \( \lambda' \); equally \( \lambda' \) is the smallest possible partition which contains the successor states (or blocks) implied by \( \lambda \). Thus, by enlarging \( \lambda' \) or by refining \( \lambda \), other \( \text{p.p.s} \) are obtained. It follows that for every \( \text{p.p.} \ (\tau, \tau') \) on \( S' \) there exists an \( \text{Mm} \) pair \( (\lambda, \lambda') \) such that \( \lambda \supseteq \tau \), and, \( \lambda' \subseteq \tau' \).

Since any \( \text{p.p.} \) can thus be generated from a corresponding \( \text{Mm} \) pair, it follows that the set of all the \( \text{Mm} \) pairs completely characterizes the set of all the \( \text{p.p.s} \) on \( S' \) of \( M \).

The closed partitions can be derived in a straightforward manner from the set of \( \text{Mm} \) pairs. This is done as follows:

If \( (\lambda, \lambda') \) is an \( \text{Mm} \) pair then the partition \( \lambda \) is closed iff \( \lambda' \subseteq \lambda \); and if, \( \lambda' \subseteq \lambda \) then \( \lambda' \) is also closed. Thus both \( \lambda \) and \( \lambda' \) are closed if \( \lambda' \subseteq \lambda \).

If, \( \lambda = \lambda' = \pi \), then the blocks of \( \lambda \) are mapped onto themselves and since this is the definition of a closed partition then \( \pi \) is closed. The reverse is also true - if \( \pi \) is a closed partition then \( (\pi, \pi) \) is an \( \text{Mm} \) pair.

### 3.7 State-Placement using p.p. and Mm-Pairs

State-assignments based on S.P. partitions are easily implemented, lead to reduced variable dependencies, and thus result in large savings in terms of hardware - see Appendix 1. But not all reduced variable dependencies are brought out merely by computing the S.P. partitions.

Partition Pairs (and thus Mm pairs) show another type of reduced variable dependence which leads to what is called "cross-decomposition" and which cannot be detected using S.P. partitions. The steps followed
in making a state-assignment according to P.s are outlined below:

(1) The set of all Mm pairs is computed.

If a machine M has S state-variables (S = \lfloor \log_2 N \rfloor, where N is the number of internal states of M) then, the best possible state-assignment obtainable using the Mm pairs is one in which S partitions are detected such that:

\[ \tau_1 \cdot \tau_2 \cdot \ldots \cdot \tau_S = \pi(0), \]

where each partition has only two blocks and thus requires only one state-variable to encode it.

These partitions are chosen such that \( M(\tau_1), M(\tau_2), \ldots, M(\tau_S) \) are each \( \geq \) the product of as small a subset of \( \tau_1, \ldots, \tau_S \) as possible (where \( \{M(\tau_1), \tau_1\}, \{M(\tau_2), \tau_2\}, \ldots, \{M(\tau_S), \tau_S\} \) are all Mm pairs on \( S_i \) of M).

Since when: \( M(\tau_a) \geq \tau_b \cdot \tau_c \)

then, the product \( \tau_b \cdot \tau_c \) provides at least as much information as \( M(\tau_a) \) and since \( \tau_a \) is dependent on \( M(\tau_a) \) only, it then follows that the state-variable assigned to \( \tau_a \) would be dependent on the state-variables associated with \( \tau_b \) and \( \tau_c \) and is independent of all the other state-variables.

When: \( M(\tau_i) \geq \tau_j \)

then the variable assigned to \( \tau_i \) is only dependent on the state-variable assigned to \( \tau_j \).

In the following, \((M, m)\) stands for any Mm pair, m refers to the back partition and M to the front partition.

(2) Choose a 2-block m-partition \( \tau_1 \) (which may be an enlarged version of an m-partition) such that the corresponding M-partition \( \tau_2 \), is \( \geq \) another m-partition \( m' \). The larger the M-partition, the more likely that this will happen.
(3) Form $\tau_1, \tau_2$ and choose a third partition $\tau_3$ such that it has only two blocks and that $\tau_3 \triangleright m''$, where $m''$ is an $m$-partition in yet another $M_m$ pair $(M'', m'')$.

Again $M''$ should be chosen such that:

$M'' \triangleright$ the product of a subset of $\tau_1', \ldots, \tau_s'$.

(4) Step (3) is repeated until the product of all the chosen partitions is $= \pi(0)$. Appendix 1 illustrates the application of $M_m$ and p.p. pairs in the state-assignment of sequential machines.
CHAPTER FOUR

STRUCTURAL ANALYSIS OF STORED PROGRAM IMPLEMENTATION

4.1 Introduction

When a sequential machine is implemented using discrete logic components, the ultimate aim of the designer is to reduce the amount of combinational logic required; state-assignments are thus devised so as to achieve this. However, when a sequential machine is implemented as a stored-program switching network, shown in Fig. 1.2, the emphasis is shifted to reducing the overall store size of the SPM used.

Hartmanis's Algebraic Structure Theory\textsuperscript{1-8} was developed originally for implementations using discrete logic gates and, when applied in this manner, it resulted in considerable savings in the hardware requirements for a large number of machines. This was achieved by decomposing the machines into smaller units.

This theory can still be applied in conjunction with LSI memory modules but with somewhat reduced effectiveness due to the unpredictable resulting structures and the non-uniform sizes of the resulting component machines. Some of these structures are more suited for such implementations than others; e.g., parallel decomposition generally results in larger reductions in storage requirements and is more suited for LSI implementations (since it results in more uniform submachines) than other decomposition structures.

However, a large number of sequential machines are not readily decomposable in their reduced form. Such machines are augmented (by means of state-splitting\textsuperscript{7,8} some, or all, of the states of the machine) in order to make them decompose.
In this chapter some general formulae and guidelines, which can be applied in choosing the best possible structure of a given sequential machine, are presented. The best possible (relative) component machine sizes in a particular resulting structure are discussed. The effects of added redundancy (in order to achieve a decomposition) on the storage requirements of the resulting augmented machine are outlined and upper limits to the added redundancy in relation to the various decomposition structures are also obtained. Finally, the merits and limitations of the various structures of decomposition are discussed.

4.2 Design Criteria Using SPMs

Conventionally a sequential machine is implemented using combinational logic in conjunction with some memory elements. The function of the combinational logic is to obtain the next-state variables and the output functions of the machine as functions of the external inputs and present-state variables.

When designing sequential systems using SPMs, the combinational logic is replaced with an SPM of the appropriate size and structure - see Section 1.2; the size of the SPM implies the total number of memory bits contained in the module, whereas the structure of an SPM implies the number of inputs and outputs of the module.

Using binary representation, a module with \( P \) inputs would give rise to \( 2^P \) addressable locations (or words) each of which has a distinct address combination. Each of these locations when addressed gives rise to \( Z \) outputs (or output-bits). Thus the size of an SPM with \( P \) inputs and \( Z \) outputs is given by:

\[
\text{Size of SPM} = 2^P \text{ words of } Z \text{ bits each}
\]

\[= Z \cdot 2^P \text{ bits.}\]  \( \cdots (4.1) \)
From equation (4.1) it is clear that, with each additional input to the module, the size of the SPM exactly doubles, and with each additional output, the number of bits in each word is increased by one.

When an SPM is used in the implementation of a sequential machine, the inputs to the module are divided up between the external inputs and the internal state-variables of the machine. The outputs of the SPM are similarly divided up between the outputs of the whole machine and its next-state variables—see Fig. 1.2.

The relative effects of increasing the number of external inputs, outputs, and state-variables on the size of the SPM required are discussed below.

(1) Increasing the number of external inputs by one while leaving the other parameters unaffected has the effect of exactly doubling the original store size.

(2) Increasing the number of outputs of a machine by one, while leaving the other parameters constant, increases the number of bits in each memory location by one.

(3) Increasing the number of state-variables by one while leaving the other parameters constant more than doubles the original store size, since it increases both the number of inputs to the store by one (thus doubling its original size, as in (1)), and the number of outputs from the store by one (thus increasing the number of bits in each word by one, as in (2)).

Thus, when designing sequential systems using SPMs, the most decisive factor is the total number of inputs that the store must have; and because of the logarithmic relationship between the store size and the number of inputs, it becomes clear how vitally important it is to reduce the number of inputs to the SPM to a minimum. (This usually implies
the simultaneous reduction of the number of outputs, since the state-variables are common to both the inputs and outputs of the store).

The outputs of the system are obtainable from a separate SPM, the inputs to which depend on the particular machine implemented\(^\text{27,28}\).

The reduction in the number of inputs is usually achieved by decomposing the machine into smaller units. The total storage capacity of the decomposed system, as compared with the original storage requirement of the undecomposed system, is the basic criterion on which the effectiveness of the particular decomposition structure is judged — see Section 1.3.

The loop-free structures resulting from the application of Hartmanis's Structure Theory achieves exactly this for some sequential machines — in that the resulting component machines have a reduced number of inputs and outputs and thus enable some large systems to be realised from relatively small SPMs. However, the resulting structures depend entirely on the algebraic characteristics of the original machine.

In the following two sections, the properties of the closed partitions for the different possible loop-free structures are discussed, and algebraically formulated.

4.3 Choice of Closed Partitions

The structure resulting from decomposing a sequential machine has a profound effect on the total size of the required SPMs, and as the structure of the decomposed machine is directly related to the closed partitions, the choice of these partitions (when choice exists) plays an important role in the storage requirements of these machines. The three different cases of decompositions are considered separately below.
4.3.1 Parallel Decomposition\textsuperscript{7,8}

In a parallel decomposition, each component machine is independent of the state-variables of all other components. The total state of the decomposed machine is formed by the state-variables of all the component machines in conjunction with the external inputs. Each component may have a different number of state-variables (depending on the closed partition on which it is based).

A machine $M$ with $S$ state variables can be decomposed into $S$ components operating in parallel iff it possesses $S$ closed partitions, $\Pi_1, \Pi_2, \ldots, \Pi_S$, such that each partition has only two blocks, and:

$$\Pi_1 \cdot \Pi_2 \cdot \ldots \cdot \Pi_S = \Pi(0).$$

Such an arrangement is shown in Fig. 3.2 and results in components with one state-variable each; thus, resulting in the least possible storage requirement for a machine of this size since each component is independent of the state-variables of all other components, and this is given by:

$$M_D = S \cdot 2^{1+2} = 2 \cdot S \cdot 2^X,$$

whereas, that of the undecomposed machine is given by:

$$M_0 = S \cdot 2^{S+2}$$

Thus, $M_D$ represents a reduction by a factor of $2^{S-1}$ on $M_0$.

If the closed partitions chosen had 4 blocks each, then each component would require 2 state-variables of its own and still be independent of all other components. Each component would now be of size, $2.2^{x+2}$, but in this case there would only be $S/2$ components; thus the total storage requirement would, in this case, be:

$$M_D' = 2.2^{x+2} \cdot \left(\frac{1}{2} S\right) = 4 S.2^X,$$

which is exactly double $M_D$.

From the above, the closed partitions that result in least storage requirement (in the case of parallel structures) should, as nearly as
possible, have the least number of blocks – to the nearest power of 2; be as nearly as possible of equal size, and of uniform structure.

When these properties are satisfied, the resulting component machines are as nearly as possible equal in size, which is the most suited structure for ISI implementations. Also, as will be shown later on in this chapter, such properties lead to the maximum reduction in storage requirement.

4.3.2 Serial decomposition

Fig. 3.4 shows the general case of serial decomposition, where the jth successor machine is dependent on all its predecessors. Thus, if the jth component $M_j$ has $S_j$ state-variables, then:

- $M_1$ would have $S_1 + X$ inputs
- $M_2$ would have $S_1 + S_2 + X$ inputs
- $M_j$ would have $S_1 + \ldots + S_j + X$ inputs

Because the number of inputs to the final successor components is much larger than the number of inputs to the first few predecessor components, and because the size of each SPM doubles with each additional input, it follows that when implementing a sequential machine using serial decomposition the sizes of the final successor components will be the decisive factor in the amount of storage required.

The machine $M$ in the previous section can be decomposed into $S$ component machines connected in series iff it possesses $S$ closed partitions, $\pi_1, \ldots, \pi_s$, such that:

$$\pi_1 > \pi_2 > \pi_3 > \ldots > \pi_s,$$

where, $\pi_s = \pi(0)$.

Each component machine in such a decomposition would then have only one state-variable of its own. The total storage in this case is:

$$M^n_D = 2^{n+1} \cdot (2^S - 1)$$
In memory of my dear Father
This is the least amount of storage that would be required when decomposing a machine with $S$ state-variables using serial decomposition. The sizes of the various submachines in such a structure, generally, more than doubles with each successive component; thus, the size of the first predecessor component would, in general, be very small compared to that of the final successor components. Such structures are particularly unsuitable for LSI realisations.

A more suited serial structure for such realisations would be to decompose the system into a smaller number of components of more uniform sizes. Such structures would have to be based on a relatively small number of closed partitions (as compared to the total number of internal states); thus leading to a small number of component machines. However, with such structures the reduction in the storage requirement that could be achieved as a result of decomposition is very much reduced on the previous case.

Thus the best structure that could be achieved through serial decomposition (from the point of view of LSI realisations) would be to obtain component machines that are, as near as possible, of equal size; their structure is generally different for the obvious reason that the number of inputs to the successor components is usually larger than the numbers of inputs to their predecessor ones. The necessary mathematical conditions for such serial decompositions are formulated later on in this Chapter.

Thus, with serial decomposition, the chosen closed partitions should have the following properties:

1) If the choice of closed partitions is to result in least storage requirement, then these should have the least number of blocks to the nearest power of 2; i.e. they should be as large as possible and thus leading to component machines with as few state-variables
as possible. However, the resulting components would be of vastly differing sizes and structures.

ii) If the choice of closed partitions is to lead to components of uniform, or near uniform, sizes then the structure of the different partitions should be related algebraically - see later on in this chapter.

4.3.3 Composite decomposition

This type of decomposition is a combination of the parallel and serial cases discussed previously - see Chapter 3.

The number of inputs to the successor machine (or machines) is likely to be much larger than the number of inputs to the individual predecessor ones; hence reducing the number of inputs to the successor machine is imperative if a sizeable reduction in storage requirement is to be achieved.

Thus, because the sizes of the serial and parallel parts are so clearly different (in the sense that the total size of the parallel components is very much less than that of the serial ones) a significant reduction in storage can only be achieved through reducing the size (or sizes) of the serial successor components to a minimum. Reducing the size of the parallel components achieves little in this way.

Thus, the treatment of composite structures is reduced to two parts: the parallel one which, in terms of storage, has little significance, and the serial case which calls for reducing the sizes of the final successor components to a minimum.

In the remainder of this chapter, composite structures are only treated where they are different from the parallel and serial cases.
4.4 Algebraic Derivations of Best Structures

In this section, the algebraic formulations that lead to minimal program storage requirement for the various possible loop-free structures, realised using LSI modules, are derived. The tool used for these derivations is the Lagrange Multipliers Method for obtaining the minima of multi-variable functions; the total storage requirement of the particular structure is thus minimized with respect to the number of state-variables of the various component machines in that structure.

Since the variables concerned are all integers, and because the method of Lagrange Multipliers is used in conjunction with continuous functions, it therefore follows that the only meaningful results are those of integer values. Thus, when using the results derived here, these should be interpreted as the nearest integer.

It should also be remembered that in the following derivations, the conditions for minimum storage are derived with respect to the number of component machines assumed in the structure, and not the absolute minimum storage since this is obviously the condition where (for all types of structures) each component machine has only one state-variable.

In the following derivations: let $S$ and $S_D$ represent the number of state-variables of the original undecomposable machine and of its decomposed version respectively; $X$ and $Z$ represent the numbers of external inputs and output-functions respectively. Also, let component $M_i$, in a particular structure, be independent of $E_i$ of the $X$ external inputs, and (in the case of serial decomposition) of the state-variables of the predecessor components.

4.4.1 Parallel decomposition

Fig. 3.2 shows a sequential machine decomposed into $n$ components connected in parallel.
If $M_t$ has $S_i$ state-variables, then the storage requirement of $M_t$ is given by:
\[ m_x = S_t \cdot 2^{s_t + e_1}, \] ...
and the total storage requirement of such a structure is given by:
\[ m_x = 2^x \{ S_1 + S_2 + \ldots + S_n \} \] ...
\[ m_x = 2^x \{ S_1 + S_2 + \ldots + S_n \} \]
where, $S = S_1 + S_2 + \ldots + S_n$ ...

Algebraically, equations (4.5) and (4.6) can be expressed as:
\[ m_x = f \{ S_1, S_2, \ldots, S_n \} \] ...
and, $Q(S_1, S_2, \ldots, S_n) = 0$ ...
where $f$ and $Q$ are two mathematical functions.

Using Lagrange's method of undetermined multipliers, $M_D$ can be minimized under constraint $Q$:
\[ \frac{\partial f}{\partial s_1} + \lambda \frac{\partial Q}{\partial s_1} = 0 \]
\[ \frac{\partial f}{\partial s_2} + \lambda \frac{\partial Q}{\partial s_2} = 0 \]
\[ \ldots \]
\[ \frac{\partial f}{\partial s_n} + \lambda \frac{\partial Q}{\partial s_n} = 0 \]

where, $\lambda$ is the Lagrange multiplier.

Now, \[ \frac{\partial m_D}{\partial s_1} = 2^x \{ 2^{s_1} - e_1, (1 + s_1 \ln 2) \} \] for, $i = 1, 2, \ldots, n$
and, \[ \frac{\partial Q}{\partial s_1} = 1. \]

Applying these results to the set of equations (4.9) results in:
\[ 2^x \{ 2^{s_1} - e_1, (1 + s_1 \ln 2) \} + \lambda = 0 \]
\[ 2^x \{ 2^{s_2} - e_2, (1 + s_2 \ln 2) \} + \lambda = 0 \]
\[ 2^x \{ 2^{s_n} - e_n, (1 + s_n \ln 2) \} + \lambda = 0 \]

Equating any two equations in (4.10), e.g. 1st and 2nd, results in:
\[ 2^{s_1} - e_1, (1 + s_1 \ln 2) = 2^{s_2} - e_2, (1 + s_2 \ln 2) \]
or, in general, for any two parallel components \( r \) and \( k \):

\[
2^s_r - \varepsilon_r (1 + S_r \ln 2) = 2^s_k - \varepsilon_k (1 + S_k \ln 2) \quad \ldots (4.11)
\]

From equation (4.11) it is clearly seen that the best possible parallel structure for a machine with \( S \) state variables is one that comprises components of, as nearly as possible, equal size and of the same structure. This stems from the logarithmic relationship between the number of inputs to an ISI module and its capacity. It also illustrates that, taking into account all possible redundancies, when the two sides of (4.11) are not satisfied by a large margin then the condition of equi-sized components is not satisfied; and it is therefore likely that the condition of least storage is not, either. The applications of (4.11) are discussed more fully later on in this Chapter.

4.4.2 Serial decomposition

Fig. 3.3 shows a sequential machine decomposed into \( n \), serially-connected, components.

If component \( M_i \) has \( S_i \) state-variables, then the total number of inputs, including the external inputs, is given by:

\[
X + S_1 + S_2 + \ldots + S_i - \varepsilon_i,
\]

where \( \varepsilon_i \) is the redundancy factor for that component.

The total storage requirement of this structure is given by:

\[
N_D = 2^X \left\{ S_1 s_1 - \varepsilon_1 + S_2 s_2^{s_1 - \varepsilon_1} + \ldots + S_n s_n^{s_1 - \varepsilon_1} \right\} \ldots (4.12)
\]

where,

\[
S_D = S_1 + S_2 + \ldots + S_n.
\]

Applying Lagrange's method of Undetermined multipliers (as was done with the parallel case) on equation (4.12) results in:

\[
2^X \left\{ 2^{(s_1 - \varepsilon_1)} (1 + S_1 \ln 2) + S_2 s_2^{s_1 + s_2 - \varepsilon_2} \ln 2 + \ldots + S_n s_n^{s_1 + s_2 + \ldots + S_{n-1} - \varepsilon_{n-1}} \ln 2 \right\} + \lambda = 0 \quad (4.13)
\]

continued over
Equating any two consecutive equations from (4.13), e.g. the 1st and 2nd, results in:

\[ 1 + S_1 \cdot \ln 2 = 2^{s_2 + E_1 - E_2} \]  

... (4.14)

or, in general, \[ 1 + S_r \cdot \ln 2 = 2^{s_r + E_r - E_{r+1}} \]  

... (4.15)

Equation (4.15) relates the number of state-variables \( S_r \) of any component machine in a serial structure to that of its immediate successor component.

When \( E_r = E_{r+1} \), then equation (4.15) reduces to:

\[ 1 + S_r \cdot \ln 2 = 2^{s_r + 1} \]  

... (4.16)

From this equation, it is clear that the relationship between any two successive components in a serial structure should be very nearly logarithmic if the storage requirement of the structure is to be reduced to a minimum for the number of components in the structure.

4.4.3 Composite decomposition

With composite decomposition, as with the serial case, the size of the successor machines tend to be the major factor in determining the total size of the required store since their number of inputs (i.e. the number of state-variables of the whole structure as well as the external inputs) is usually much larger than that to the individual parallel components.

*This case is similar to the serial one and is included in order to present a complete algebraic treatment.*
When there is more than one successor component machine, the problem reduces to that of the purely serial case, once the parallel components have been chosen.

When the parallel components are of uniform or near uniform sizes and structures, then their total storage requirement would, generally, form only a fraction of the total. However, when the parallel components are of vastly differing sizes and structures, then the sizes of the larger of these components would be the ones of significance; and provided that the closed partitions used result in optimal or near optimal encoding (discussed in Chapter 5) then, generally, there would only be one component machine amongst the parallel ones of size comparable to that of its successors. In such cases the problem reduces to that of a 2-component serial decompositions with the difference, being that the total number of state variables $S_D$ is that including all state-variables of the parallel components. This case is formulated below:

For 2-parallel components followed by a serial successor, the storage requirement of the whole system is given by:

$$x_f = \sum_{j=1}^{n-2} x_{j+1} \mathcal{E}_j + x_{n-1} \mathcal{E}_{n-1} + x_n \mathcal{E}_{n}$$

where $k = \mathcal{E}_1 + \mathcal{E}_2 + \mathcal{E}_3$; all the symbols are as defined before;

$$S_D = S_1 + S_2 + S_3.$$

Since, one of the parallel components is assumed much larger than the other, it follows that:

$$S_2 = \frac{S_1 - \mathcal{E}_1}{2}. $$

Thus:

$$x_f = \sum_{j=1}^{n-2} x_{j+1} \mathcal{E}_j + x_{n-1} \mathcal{E}_{n-1} + x_n \mathcal{E}_{n}$$

Again, applying the Lagrange multipliers method, the equations:

$$2^x \left\{ 2^{s_1 - \mathcal{E}_1} (1 + s_1 \ln 2) \right\} + \lambda = 0$$

and,

$$2^x \left\{ 2^{s_1 - \mathcal{E}_1} \right\}+ \lambda = 0,$$
are obtained; equating them, results in:
\[ S_1 - \varepsilon_1 (1 + S_1 \ln 2) = 2S_D - k \]
\[ \cdots (4.17) \]

The similarity between equations (4.17) and (4.14) is obvious; in fact, equation (4.14) can be obtained from equation (4.17) merely by replacing \( S_D \) with \( S_2 \), and \( k \) with \( \varepsilon_2 - \varepsilon_1 \).

4.5 Applications of Derived Formulae

The applications of the formulae derived in the previous section are discussed for the different possible structures.

I. The Parallel Case

The main use of the parallel formula (equation 4.11) obtained in the previous section is to show that the best possible parallel structure comprising a given number of components, from the point of view of storage requirement, is when the components are of exactly equal size. It takes into consideration the redundant parameters that a particular structure may possess.

Another use of this formula is in choosing a particular structure out of two or more. This could be done in two ways:

i) Equation (4.11) can be rewritten as:
\[ (1 + S_1 \ln 2) = (1 + S_2 \ln 2) 2^{S_2 - S_1} \]
(assuming that \( \varepsilon_1 = \varepsilon_2 = 0 \)). This formula can be used to compare the sizes of 2 or more components in a particular structure; e.g. a machine with \( S_D = 7 \), and which can be decomposed according to either:

a) \( S_1 = 2, \varepsilon_1 = 0 ; \ S_2 = 4, \varepsilon_2 = 2 ; \ S_3 = 1, \varepsilon_3 = 0, \) or

b) \( S'_1 = 2, \varepsilon'_1 = 0 ; \ S'_2 = 2, \varepsilon'_2 = 1 ; \ S'_3 = 3, \varepsilon'_3 = 0. \)

In both cases, the first component is exactly the same. However, applying the above equation to the second and third components it becomes
clear that choosing \( S_2 \) and \( S_3 \), in conjunction with \( \varepsilon_2 \) and \( \varepsilon_3 \), results in the more economical realisation since they satisfy the above minimal condition more closely.

ii) When the number of state-variables of 2 or more possible parallel structures are different, then least storage is obtained when the structure with

\[
\sum_{i=1}^{n} \left(1 + S_i \ln 2\right) 2^{s_i - \varepsilon_i}, \text{ for } i = 1, 2, \ldots, n,
\]

having the least value is chosen.

However, this last condition is equivalent to comparing \( \sum_{i=1}^{n} S_i 2^{s_i - \varepsilon_i} \) (i.e. the total storage requirement) of all parallel structures under consideration.

The above, formulates mathematically and affirms what might be detected, by intuition, of parallel structures.

II. The Serial Case

In Section 4.3, it was shown that the best possible serial structures, in terms of storage requirement, are obtained when an \( S \)-state-variable machine is decomposed into \( S \) components, each having only one state-variable of its own. However, such structures lead to component machines of vastly differing sizes and structures, which is a very unsuitable property for LSI realisations; e.g. a 6-state-variable machine with \( X = 4 \), would have its first predecessor component of size \( 1.2^{4+1} (= 32 \text{ Bits}) \), whereas, the final successor component in such a structure would be of size \( 1.2^{4+6} (= 1024 \text{ Bits}) \).

The formula derived in Section 4.4 can be used to determine the best possible component size and structure for a particular number of component machines in a particular structure.

A machine with \( S \) state-variables can best be serially decomposed into \( n \) components using equation (4.15) when:
i - n \leq S, if the S state-variables are distributed between the n component machines in accordance with equation (4.15). As an example, considering a machine with S = 10 and n = 2 least storage is obtained when S_1 = 7 and S_2 = 3 - i.e. when the predecessor has 7 state-variables and the successor has 3. Any other pair of values would require a larger storage requirement. The two values 7 and 3 satisfy equation (4.15) more closely than any other values for a 10 state-variable machine being decomposed into 2 components in a serial structure. In Chapter 5 the full implications of this formula are discussed in more detail.

ii - n \leq S if the distribution of state-variables is done in such a way as to make the final successor components have as few state-variables as possible - i.e. to render the structure as close as possible to the ideal case where each component has only one state-variable except the first few predecessor ones, where they are made as large as possible at the expense of reducing the size of the final components in the structure; e.g. S = 10; n = 6, then the best distribution of the 10 state-variables is given by:

S_1 = 4; S_2 = 2; S_3 = 1 = S_4 = S_5 = S_6; the first three components conform to equation (4.15); whereas, the last three successor components have one state-variable each.

III. Composite Decomposition

The formula derived in Section 4.4 for composite structures - i.e. equation (4.17) - is only applicable in cases where one of the parallel components is much larger than all the others; the problem then reduces to that of the serial case. Thus, it could be used in judging the best possible structure in a very similar way to that of the serial case discussed above.
Fig. 4.1 Variation of storage requirement of $M$ with $S_1$,
(M being decomposed into two parallel components).
Fig. 4.2 Variation of storage requirement of M with $S_1$;
(M being decomposed into 2 serial components).
4.6 **Maximum Justifiable Added Redundancy**

Some sequential machines are not readily decomposable in their reduced form and, in such cases, redundant states are introduced into the state-table in order to decompose them. However, the amount of justifiable added redundancy depends entirely on the type of structure that the machine decomposes into. All cases are discussed below:

1 - **Parallel Structures**

Fig. 4.1 shows the variation of the storage requirement of an augmented version of an undecomposable machine M, with the number of state-variables of one of the two parallel components ($S_1$) - the machine M is assumed to have 5 state variables and 3 external inputs and that it decomposes into 2 parallel components. Various degrees of redundancies are considered. The minimum points in these graphs clearly indicate that the optimum size of the components occur when they are, as nearly as possible, of equal size; when unequal redundancies exist, the minimum is obtained when the component machine with the larger number of redundant variables (i.e. with the larger value of $\varepsilon$) is chosen having a large number of state-variables at the expense of the other components.

If the limit of useful added redundancy is taken as the storage requirement of the undecomposed machine then it is clear, from Fig. 4.1, that, with parallel decomposition, provided the component machines are of uniform or near uniform structure, then the added redundancy can be very considerable - see Chapter 5. Indeed the amount of added redundancy is directly proportional to the number of components in the parallel structure and, to the size of individual components.

2 - **Serial Structures**

Fig. 4.2 shows the variation of the storage requirement of the same machine M, as that in the parallel case above, being augmented in order to decompose it into 2 serial components, with $S_1$ (the number of state-
Fig. 4.3 Variation of storage requirement of $M$ with $S_1$; ($M$ being decomposed into a composite structure of two parallel and one serial components).
variables of the predecessor machine) under various degrees of redundancies. It can be verified that the minimum points of all curves satisfy equation (4.15). These curves demonstrate the effect of reducing the number of inputs to the predecessor and successor components. The effect of a redundant variable (i.e., \( \varepsilon_1 = 1 \)) in the predecessor component is demonstrated through curve II and, clearly, only becomes appreciable when the size of the predecessor component becomes comparable with that of the successor machine; whereas, curve III demonstrates the effect of a redundant variable (i.e., \( \varepsilon_2 = 1 \)) in the successor component and is appreciable for all values of \( S_2 \) (the number of state-variables of the successor; \( = S_D - S_1 \)) since, by reducing the number of inputs to this component, its size (which is appreciable for all values of \( S_2 \)) is halved.

These curves also demonstrate that, for this machine, the storage requirement of the decomposed structure with \( S_D = 6 \) and, \( \varepsilon_1 = \varepsilon_2 = 0 \) (i.e. with no redundancies) is greater than that of the undecomposed machine - see curve I in Fig. 4.2. Curve IV demonstrates the effect of \( \varepsilon_2 = 2 \); the storage requirement, as compared with curve I, is reduced by a large factor.

3 - Composite Structure

Fig. 4.3 shows the variation of the storage requirement of the same machine \( M \) (and its augmented version) when it is decomposed into 3 components forming a composite structure, with \( S_1 \) - the number of state-variables of one of the predecessor components - under various degrees of redundancies. Curves I and II show that fixing the size of the successor component results in a minimum at the point:

\[
S_1 = S_2 = \frac{(S_D - S_3)}{2},
\]

which is very similar to the purely parallel case; whereas, curves III and IV show that fixing the size of one of the two parallel components
results in storage requirement that decreases logarithmically with reducing the size of the successor component. This decrease continues until the sizes of the successor component becomes comparable with that of the largest of the parallel components, at which point increasing the size of that same component leads to an increase in its size which, more than offsets the decrease in the size of the successor component; this limit is reached only in cases where the number of state-variables of the other parallel components is very small compared to that of the large component.

4.7 Discussion and Conclusions

All decomposition structures, provided $S_D = S$, result to varying degrees in a reduction in the storage requirements over those of the undecomposed systems. Parallel decomposition, in general, results in a larger reduction than other structures.

When added redundancy results in an increased number of state-variables (i.e. $S_D > S$), then the resulting decomposed structure has a profound effect on the storage requirement; the effect of the additional variables on this, is discussed in the three cases:

1 - Parallel Case: Provided the resulting components happen to be of similar sizes, parallel structures still result in a reduction in the storage requirement for large degrees of added redundancy. Also, the larger the original system is, the greater the factor of added redundancy can be; the storage requirement of the decomposed structure would still be $\leq$ that of the original undecomposed machine.

2 - Serial Case: The final successor component machines in such structures are, to a large extent, the decisive factor in their storage requirements; thus reducing the size of these components to a minimum, normally, leads to appreciable reductions in storage.
If a serial structure is the result of added redundancy, economical structures are obtained for relatively large sequential machines only. Also, due to the nature of such structures, their component machines cannot possibly be of uniform size or structure; this is a severe disadvantage when dealing with LSI realisations.

3 - Composite Case: As in the serial case, the most significant components in such a structure, in terms of storage requirements, are the serial successor components; and, thus minimizing the sizes of these components reduces the storage requirement of the system appreciably.

However, provided the successor machine has a very small number of state-variables, and provided the parallel components are of similar sizes and are as small as possible, then the reduction in storage can be considerable for medium and large systems.

The formulae and guidelines derived in this Chapter, for arriving at the best possible (relative) decomposition structures, however, bear no direct relationships to the closed partitions on which these structures would be based. The redundancy, assumed introduced into the system in order to make it decompose, was not specified or its source exactly defined (whether through state-splitting; usage of non-uniform closed partitions or through the use of over-redundant partitions). The inter-partition relationships and their effects on the resulting structures and on the storage requirements have also not been dealt with.

In the following Chapter, these aspects of loop-free structures are examined; the applicability of the formulae derived in this Chapter are also examined in conjunction with the various cases of closed partitions.
5.1 Introduction

In Chapter 4, formulae and relationships, which could enable the designer to arrive at the best possible decomposition structures (i.e. to those which result in least total storage requirement), were developed. Also, the best possible distribution of \( S \) state-variables between two or more component machines operating in either serial or parallel structures were obtained.

In this Chapter, the inter-partition relationships that lead to economical (or uneconomical) structures are considered. The effect of a machine possessing uniform or non-uniform closed partitions on the total storage requirement of a particular structure are discussed in detail. The effects of the number of internal states \( N \) being an exact power of 2, or otherwise, on the choice of the closed partitions, and hence on the possible decomposition structures, are also discussed.

The effects of input and output consistent partitions on possible choices of closed partitions for the various structures and their resulting influence on the storage requirement of these structures are also considered.

A procedure for applying the results of Chapters 4 and 5 is then formulated. The procedure enables a sequential machine to be synthesized in terms of its decomposability by comparing the possible structures and choosing either the one which results in the least total storage or the one which gives maximum uniformity of the resulting component machines.
Parameters and Definitions

Definition 1: For any two closed partitions \( \Pi_1 \) and \( \Pi_2 \), such that \( \Pi_1 \geq \Pi_2 \), define an inter-partition dispersion ratio \( y_j \) as the maximum number of blocks in \( \Pi_2 \) that form a single block in \( \Pi_1 \). It is assumed that \( \Pi_1 \) and \( \Pi_2 \) are both relevant to a particular decomposition structure (serial or composite), and that there are no intermediate closed partitions relevant to that particular structure.

Example: \( \Pi_1 = 1,2,3,4,5 ; 6,7,8 \), \( \Pi_2 = 1,2 ; 3 ; 4,5 ; 6,7 ; 8 \).

Clearly \( \Pi_1 \geq \Pi_2 \), and since the blocks \( 1,2 ; 3 ; 4,5 \) in \( \Pi_2 \) form a single block \( 1,2,3,4,5 \) in \( \Pi_1 \), it therefore follows that \( y_2 = 3 \).

Definition 2: For synchronous machines, a decomposition is defined as trivial when one or more components require the same number of state-variables to encode its internal states as the original undecomposed machine. It follows that each such component is then capable of performing all the functions of the undecomposed machine. In the asynchronous case this definition of triviality is not valid since the complexity of the state-assignment of each component depends to a large extent on the actual number of states that a particular component has, and thus on the number of blocks that the partition, on which that component is based, possesses.

Examples: For a synchronous machine with 8 internal states the closed partition \( \Pi_1 = 1,2 ; 3 ; 4,5 ; 6,7 ; 8 \) is a trivial one since its encoding would require at least 3 state-variables – the same as the requirement of the original machine; whereas, for an asynchronous machine, \( \Pi_1 \) is a valid closed partition since it leads to a component machine with only 5 states – as compared with 8 in the original machine.
Thus in the analysis that follows in this Chapter, the notion of triviality applies only to the synchronous case. As a consequence, a closed partition is trivial if its number of blocks exceed \(2^{s-1}\); where, \(S = \lfloor \log_2 N \rfloor\), and \(N =\) number of internal states.

Definition 3: A redundant set of closed partitions is one which neither satisfies the conditions for serial nor parallel decompositions without requiring a larger number of state-variables than \(S\). This applies mainly to the case of non-uniform closed partitions; e.g.

\[
\pi_1 = 1,2,3,4,5 \ ; \ 6,7,8 ;
\]
\[
\pi_2 = 1,2,3,6,7 \ ; \ 4,5,8 .
\]

If \(\pi_1\) and \(\pi_2\) are used individually or collectively to decompose a machine, then \(S_D \gg S\).

In the case of uniform closed partitions this definition applies only when the partition set, as a whole, is used for the decomposition; e.g.

\[
\pi_1 = 1,2,3,4,5,6,7,8 ; \pi_2 = 1,2,3,5,4,6,7,8 .
\]

Again, \(S_D \gg S\).

Also define \(D_i\) as the number of binary variables needed to encode the blocks of a closed partition \(\pi_i\); i.e.

\[
D_i = \lceil \log_2 \#(\pi_i) \rceil , \quad \text{where } \#(\pi_i) \text{ is the number of blocks in } \pi_i ;
\]

and define \(d_i\) as the number of binary variables needed to encode the states in the largest block of \(\pi_i\); i.e.

\[
d_i = \lceil \log_2 K_i \rceil , \quad \text{where } K_i \text{ is the number of states in the largest block of } \pi_i .
\]

In the following sections these parameters and definitions are used to analyse the various decomposition structures in conjunction with the inter-partition relationships that exist between the various partitions.
5.3 Parallel Decomposition

If a machine possesses n closed partitions whose product gives the zero partition \( \Pi(0) \) then, this machine is decomposable into n parallel components;

i.e. \( \Pi_1 \cdot \Pi_2 \cdot \ldots \cdot \Pi_n = \Pi(0) \)

and,

\[
S_D = D_1 + D_2 + \ldots + D_n ,
\]

where \( S_D \) is the number of state-variables of the decomposed machine.

In the following, the different possible structures of closed partitions and their effects on the resulting parallel decompositions are discussed:

5.3.1 Closed partitions resulting in minimal encoding

When n uniform closed partitions result in a parallel structure such that:

\[
S_D = S = D_1 + D_2 + \ldots + D_n , \quad \text{where} \quad S = \lfloor \log_2 N \rfloor ,
\]

then the minimum number of state-variables are needed to encode N states.

If N is an exact power of 2 then, for this condition to be satisfied, all the closed partitions involved must be uniform and their binary encoding requires the least number of state-variables. A further proof of the uniformity and non-redundancy of the closed partitions involved, is the satisfaction of the following formula:

\[
d_j = \sum_{j \neq i}^n D_j , \quad \text{for all} \quad j \neq i .
\]

Example:

- \( \Pi_1 = \{1,2,3,4,5,6,7,8\} ; \quad \{9,10,11,12,13,14,15,16\} \)
- \( \Pi_2 = \{1,2,3,4,9,10,11,12\} ; \quad \{5,6,7,8,13,14,15,16\} \)
- \( \Pi_3 = \{1,5,9,13\} ; \quad \{2,6,10,14\} ; \quad \{3,7,11,15\} ; \quad \{4,8,12,16\} \)

Thus \( \Pi_1 \cdot \Pi_2 \cdot \Pi_3 = \Pi(0) \); all are uniform; and,
\( d_1 = 3 \); \( d_2 = 3 \); \( d_3 = 2 \).

\( D_1 = 1 \); \( D_2 = 1 \); \( D_3 = 2 \).

\[ d_1 = \sum_{j=1}^{3} D_j \text{ for } j \neq 1, \text{ thus } d_1 = D_2 + D_3 = 1 + 2 = 3. \text{ etc.} \]

Hence, \( S_D = S = D_1 + D_2 + D_3 = \lceil \log_2 N \rceil = 4. \)

This is the case of minimal encoding where the least number of state-variables are required to encode the blocks of all the partitions involved. Provided the \( D \)-values of the different closed partitions do not vary greatly, the sizes of the resulting component machines would then be similar. When these are exactly equal for all closed partitions involved then, apart from the effect of possible input consistencies, the component machines would be of exactly equal size and of the same structure. In all such cases:

\[ M_D \ll M_o \text{ where:} \]

\[ M_o = S \cdot 2^{S_o x}, \text{ and } M_D = 2^x \left\{ \sum_{i=1}^{n} D_i \cdot 2^{D_i} \right\}. \]

5.3.2 Non-minimal closed partitions

The following synthesis applies to cases of: i - uniform but redundant sets of closed partitions; and, ii - non-uniform partitions. In both cases \( S_D \gg S \). The exact number of state-variables required by the decomposed structure, \( S_D \), depends largely on how redundant and/or how non-uniform the relevant closed partitions are.

Assuming the machine to decompose into \( n \) equi-sized components, then if the total storage requirement of such a structure is not to exceed that of the undecomposed version, then:

\[ M_o \gg M_D \text{ where, } M_o = S \cdot 2^{S_o x} \text{ and, } M_D = 2^x \left\{ n \cdot D_1 \cdot 2^{D_1} \right\}. \]

Thus, \( S \cdot 2^S \gg n \cdot D_1 \cdot 2^{D_1} \); and since, \( D_1 = D_2 = \ldots = D_n = D \).
then,  \( S^{2^n} \leq nD.2^D \)

or,  \( n \leq (S/D).2^{S-D} \)  \( \ldots (5.1) \)

The two extremes are:

i. - when \( D \) is comparable with \( S \), giving a very small value of \( n \) if the storage requirement of the decomposed structure is not to exceed \( M_0 \).  

\( \text{e.g. } S = 8 \text{ and, } D = 6 \text{ then, } n \leq 4. \)

ii. - when \( D \) is small compared with \( S \). The value of \( n \) could be very large and yet the overall storage of the system does not exceed \( M_0 \); \( \text{e.g. } \ S = 8, \ D = 4 \text{ giving } n \leq 32. \)

An approximate, simple relationship governing \( n \) and \( D \), and derived from equation (5.1) is given by:

\( D = S - |\log_2 n| \)  \( \ldots (5.2) \)

Equation (5.2) sets an approximate upper limit to the number of equi-sized parallel components a machine can be decomposed into, without suffering an increase in the storage requirement; \( \text{e.g. if } n = 4, \text{ then, } D \leq S - 2, \text{ etc.} \)

However, if the resulting components are of differing sizes and structures, then:

\( M_0 = \frac{x}{2} \cdot \sum_{i=1}^{n} D_i \cdot 2^{D_i}. \)

If the storage requirement of the smaller components in such a parallel structure is negligible compared to that of the large ones, then if the size of the large components is uniform or near uniform and if there are \( h \) such components, then the upper limit on \( h \) is approximately given by equation (5.2) where, \( n \) is replaced by \( h. \)

When there are two or more sets of closed partitions which could result in parallel structures of the same number of component machines such that some lead to components of more uniform sizes than others, then the least uniform set could still lead to the greater storage requirement.
than the uniform, or near uniform, set of partitions if:

$$S_{Du} \leq S_{In} + \alpha_D - 2^*$$

... (5.3)

where $S_{Du}$ = total number of state-variables of the uniform set.
$S_{In}$ = " " " " " " non-uniform set.
$\alpha_D$ = the maximum spread in the D-values of the closed partitions of the least uniform set.

As an example of $\alpha_D$ consider three closed partitions having $D_1 = 2$,
$D_2 = 4$ and $D_3 = 5$ then, $\alpha_D$, for this set of closed partitions, $= D_3 - D_1 = 3$.

Thus, if inequality (5.3) is satisfied, then:

$$M_{Du} \leq M_{In}$$

where $M_{Du}$ and $M_{In}$ are the storage requirements of the uniform and non-uniform, structures respectively.

Example: $D_1 = 3; \; D_2 = 6; \; D_3 = 9$; thus

$$S_{In} = 3 + 6 + 9 = 18.$$  

Now, $\alpha_D = D_3 - D_1 = 6$

Thus, $S_{Du}$ can be as high as $S_{In} + \alpha_D - 2 = 22$ state-variables.
And, $M_{Du} = 2 \cdot 7 \cdot 2^7 + 8 \cdot 2^8$; whereas, $M_{In} = 3 \cdot 2^3 + 6 \cdot 2^6 + 9 \cdot 2^9$.

Clearly $M_{Du} \ll M_{In}$.

Inequality (5.3) can be applied to any number of parallel components.
It derives the maximum number of state-variables in a uniform machine structure which has the same storage requirement as a non-uniform structure comprising the same number of components.

However, when the two sets of closed partitions do not lead to parallel structures with the same number of component machines then, generally:

* It is clear from this inequality, that the definitions of non-uniformity apply only to cases where the D-values of that set of partitions vary by more than 2.
The smaller the value of $a_D$ for the particular set of closed partitions, the more uniform are the resulting components.

If $\alpha_{D_1} = \alpha_{D_2}$, and $S_{D_1} = S_{D_2}$, then, clearly, least storage is obtained with the structure comprising the larger number of components.

5.3.3 LSI realisations and parallel structures

When dealing with non-uniform closed partitions, the variation in the size of the resulting components is a severe disadvantage when realising such structures using LSI modules.

Some sets of closed partitions may display a choice of parallel structures; some may be based on relatively large closed partitions; whereas, others may be based on relatively small closed partitions. The former case, generally, leads to parallel structures with large numbers of components, and requiring relatively little storage; also, such closed partitions are likely to result in components of uniform, or near uniform, sizes and structures. However, they may require a relatively large number of state-variables $S_D$ due to the likelihood of their being interrelated, or "over-redundant". On the other hand, utilizing the smallest of the closed partitions leads, generally, to a smaller number of components (of relatively large size), but with an increased storage requirement (over the previous case). Such parallel structures also require a large value of $S_D$; it would, however, be generally smaller than its equivalent in the previous case due to the reduced inter-relation between small closed partitions.

Thus, to obtain a parallel structure with reasonable values for $S_D$ and, $M_D$, the more uniform and least redundant of the small partitions are used in conjunction with the more uniform and least redundant of the large partitions. This arrangement is a compromise between those two cases,
and results in a parallel structure comprising, mainly, two sizes of components: one relatively small and, is thus based on the larger of the closed partitions, and the other one relatively large and based on the smaller of the closed partitions.

5.3.4 Parallel decomposition with $N \neq$ a Power of 2

When $N$, the number of internal states of the sequential system under consideration, is not a power of 2 (i.e. 4, 8, 16 etc...), then the probability that a set of closed partitions on that system results in a non-trivial decomposition is increased as compared to the case where $N$ is an exact power of 2, considered previously. The non-uniformity of the various closed partitions may be partly, or totally, compensated for by $N$ being less than $2^S$; e.g.

\[
\pi_1 = 1, 2, 3, 4; \ 5, 6, 7; \ 8; \ 9, 10, 11;
\]

\[
\pi_2 = 1, 5, 8, 10; \ 2, 6, 9; \ 3, 7, 11; \ 4;
\]

i.e. $N = 11; \ S = \left| \log_2 N \right| = 4; \ D_1 = D_2 = 2; \ \pi_1 \pi_2 = \pi(0); \ \text{and,} \ \ S_D = \sum D_1 = D_1 + D_2 = 4 = S. \ \text{Thus } \pi_1 \ \text{and } \pi_2 \ \text{are uniform within the bounds of } N \ (\text{i.e. the non-uniformity of the closed partitions has been off-set by } N \ \text{being less than } 2^S).

The syntheses of such cases are then exactly the same as those of the uniform closed partitions discussed in Sub-section 5.3.1.

However, when the non-uniformity, or redundancy, in the closed partitions is not fully compensated for by the reduced number of internal states, then this case resembles that of Sub-section 5.3.2; only the total effects of the redundant partitions is reduced in proportion to the value of $N$ (i.e. to the difference $2^S - N$).
<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$N_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$(=8-S_1)$</td>
<td>In BITS</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>1794</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>1544</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>1304</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1088</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>928</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>896</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1152</td>
</tr>
</tbody>
</table>

Table 5.1 Total storage requirement of a machine with $S=8$ being decomposed into 2 components in series; the predecessor has $S_1$ state-variables.

<table>
<thead>
<tr>
<th>$S$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>Minimum Storage $N_0$ in BITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>90</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>2</td>
<td>192</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>2</td>
<td>416</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>2</td>
<td>896</td>
</tr>
<tr>
<td>9</td>
<td>7(OR 6)</td>
<td>2(OR 3)</td>
<td>1920</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>3</td>
<td>3968</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
<td>3</td>
<td>8192</td>
</tr>
<tr>
<td>12</td>
<td>9</td>
<td>3</td>
<td>16896</td>
</tr>
<tr>
<td>13</td>
<td>10</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>11</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>12</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>13</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>14</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>15(14)</td>
<td>3(4)</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2 Least storage for various values of $S$, $S_1$ and $S_2$. 
5.4 Serial Decomposition

Serial decomposition is much more complex and diverse than the parallel case discussed in the previous section; the storage requirement and component machine sizes are totally different. The choice of closed partitions has a profound effect on both. The various cases are discussed in detail:

5.4.1 Intuitive serial decomposition

In Chapter 4, it was shown that a sequential machine with $S$ state-variables can best be decomposed into 2 components operating in series when $S_1$ and $S_2$ — the numbers of state-variables of the predecessor and the successor components respectively — are related, as closely as possible, by the formula:

\[ 1 + S_1 \ln 2 = 2^{S_2} \quad \ldots \quad (5.4) \]

Equation (5.4) can be used to choose the most suitable size of any two successive component machines in a general serial structure so that the total storage requirement is a minimum for the number of components in the structure. As an example consider a machine with 8 state-variables which is assumed to decompose into 2 serial components; assuming the predecessor to have $S_1$ state-variables (thus $S_2 = S - S_1$, where $S = \left\lfloor \log_2 N \right\rfloor$), the total storage requirement of the decomposed structure can then be calculated for all possible non-trivial values of $S_1$. This is given in Table 5.1.

From Table 5.1, it is clear that least storage is obtained with $S_1 = 6$ and $S_2 = 2$. It is easily verified that those values satisfy equation (5.4) most closely. In other words, any machine with 8 state-variables, decomposed into 2 components requires least storage when the predecessor is based on a closed partition, $\mathcal{P}_1$, the coding of which requires 6 state-
variables (i.e. \(33 \leq \#(\pi_1) \leq 64\)); whereas, the successor is based on a non-closed partition, \(T_2\), whose blocks require 2 state-variables to encode (i.e. \(3 \leq \#(T_2) \leq 4\)).

Table 5.2 shows the values of \(S_1\) and \(S_2\) of a machine decomposed into 2 serial components which result in least storage for different values of \(S\). From this, the best serial decomposition structures into any number of components (where \(n \ll S\)*) can be easily determined since the relationship between any pair of successive components is the same as that governing the values of \(S_1\) and \(S_2\). For example a machine with \(S = 10\) is best decomposed into 3 components by choosing the predecessor, intermediate and final successor components to have 7, 2 and 1 state variables respectively. In fact, with machines serially decomposed into 3, or more, components, the final successor components that follow the first 2 should not, when possible, have more than one state-variable each at the expense of reducing the sizes, of the first two predecessor components, below the values given in Table 5.2 since, this would lead to a large increase in the size of these successor components at no appreciable reduction in the size of the predecessor components (this applies to machines with \(S \ll 18\) and thus covers most practical systems - see Table 5.2).

### 5.4.2 Uniform closed partitions

The following synthesis applies to serial decompositions for which

\[ S_D = S = \sum_1^n d_i, \text{ and } S = \lfloor \log_2 N \rfloor. \]

A necessary and sufficient condition for serial decomposition into \(n\) component machines is the existence of \(n\) closed partitions such that:

\[ \pi_1 \gg \pi_2 \gg \pi_3 \ldots \gg \pi_n = \pi(0) 7,8. \]

---

* The ideal serial structure, from a storage requirement point of view, is when \(S = n\), and thus each component has only one state-variable.
Now, \( D_1 = \left| \log_2 \#(T_1) \right| \); \( D_2 = \left| \log_2 \#(T_2) \right| \); and, in general, \( D_i = \left| \log_2 \#(T_i) \right| = \left| \log_2 \frac{\#(T_i)}{\#(T_{i-1})} \right| \) for \( i > 1 \); where:

\[ T_{i-1} \cdot T_i = T_i \]

e.g. \( T_1 = 1,2,3,4,5,6,7,8 \); \( 9,10,11,12,13,14,15,16 \),

\( T_2 = 1,2,3,4 \); \( 5,6,7,8 \); \( 9,10,11,12 \); \( 13,14,15,16 \),

\( T_3 = 1,2 \); \( 3,4 \); \( 5,6 \); \( 7,8 \); \( 9,10 \); \( 11,12 \); \( 13,14 \); \( 15,16 \).

First predecessor would have \( D_1 = \left| \log_2 \#(T_1) \right| = 1 \) state-variable; 2nd would have \( D_2 \); 3rd \( D_3 \) and final successor \( D_4 \) where:

\[ D_2 = \left| \log_2 \frac{\#(T_2)}{\#(T_1)} \right| = \left| \log_2 \frac{4}{2} \right| = 1. \] Similarly, \( D_3 = D_4 = 1. \)

The condition of uniformity can be violated without any apparent increase in \( S \) provided the non-uniformity is the result of \( N \) being less than \( 2^3 \); i.e. when:

\[ S = D_1 + \left| \log_2 Y_2 \right| + \left| \log_2 Y_3 \right| + \left| \log_2 Y_4 \right| \]; and, in general,

\[ S = D_1 + \sum_{i=2}^{n} \left| \log_2 Y_i \right| , \] where \( Y \) is the inter-partition dispersion ratio defined in Section 5.2. \( D_1 = \left| \log_2 Y_1 \right| \) only when the closed partitions concerned are uniform, or uniform within the value of \( N \) for the machine under consideration.

The storage requirement in this case is a function of \( S \) and is always less than \( M_0 \) (note: the above presumes that the machine under consideration is in its minimal reduced form).

Thus, \( S_D = S \) and, \( M_D < M_0 \).

However, if the machine has to be augmented before it is made decomposable, then:

i - If \( S_D = S + 1 \), it is possible that \( M_D \ll M_0 \) for machines with \( S \gg 4 \); the lower limit, i.e. when \( S \approx 4 \), is true only for ideal, or near ideal serial structures where each component, and particularly the for the final successor component \( \left| \log_2 Y_4 \right| = d_4 \) since \( T_4 = T(0) \).
Fig 5.1 Storage requirement of two-component serial structures:
I - for $S_D = S$; II - for $S_D = S + 1$ and, III - for $S_D = S + 2$. 

The number of state variables of original $N_H$. 

---

---
final successor ones, has only one state-variable. However, for structures based on equation (5.4) — i.e., where the component machines are of nearly equal size — then the lower limit for which \( M_D \leq M_o \) is \( S \approx 7 \). This can be clearly seen in Fig. 5.1 which shows the storage requirement of a 2-component serial structure plotted for different values of \( S \); curve I shows that of the original undecomposed machine, whereas, curve II shows the case of \( S_D = S + 1 \).

As the number of components in a serial structure is increased, the limit of meaningful decompositions, i.e., the values of \( S_D \) for which \( M_D \leq M_o \), are decreased at the cost of having large variations in the sizes of the components.

- **ii** — If \( S_D = S + 2 \), then if the machine decomposes into only 2 components then, assuming no input and output redundancies, \( M_D \leq M_o \) is only possible for very large sequential systems (i.e. \( S \geq 21 \)). This case is shown in curve III of Fig. 5.1. However, as the number of component machines in a serial structure is increased, the lower limit of \( S \), for economical serial structures, is reduced accordingly; and for the structure comprising only single-state variable-components (i.e. for what has been termed an ideal structure in this thesis) this limit reduces to \( S \geq 8 \).

- **iii** — If \( S_D \geq S + 3 \) then, by extrapolating from the above two cases, 2-component serial-structures cannot lead to economical structures in practically-realisable sequential systems. In fact, any component structure in a serial decomposition, with such a large amount of redundancy introduced into the system, is very unlikely to result in an economical decomposition except either for very large, or very redundant, systems.

Clearly then, with all forms of serial decompositions, minimal storage

* In terms of input and output redundancy.
is obtained when the various components have one state-variable each; this, however, leads to components of vastly different sizes. For more uniform sizes, the number of components is reduced at the expense of increased storage requirement.

5.4.3 Non-uniform closed partitions with $N = a$ power of 2

Since $N$ is an exact power of 2, the utilization of non-uniform closed partitions to decompose a sequential machine, automatically implies the requirement of a larger number of state-variables than $S$, where:

$$ S = \left| \log_2 N \right| = \log_2 N . \quad \text{Thus, } S_D > S . $$

The number of state-variables, $S_D$, needed to encode the various blocks of the closed and, non-closed, partitions involved in a particular structure depends on how non-uniform the closed partitions are.

The relationship between $S_D$ and the storage requirement for different values of $S$ is again shown in Fig. 5.1 for structures based on equation (5.4); the same limits apply as in the previous case.

The effect of non-uniformity in the closed partitions on the storage requirement of the machine is dealt with in an algebraic format as follows:

Using the two characteristic values $D_i$ and $d_i$ of a closed partition $\Pi_i$, in conjunction with its inter-partition dispersion ratio $\gamma_i$, any decomposition structure can be synthesized:

I - When only one closed partition can be chosen out of a set of unrelated closed partitions, then the most suitable one, from the storage point of view is the one that most closely satisfies equation (5.4); namely:

$$ 1 + D_i \ln 2 = 2^{d_i} . \quad ... \ (5.5) $$
The closed partitions compared should have the same values of \( D_1 + d_1 \).
Starting with the partitions of least \( D_1 + d_1 \) values, least storage is obtained in conjunction with the one that most closely satisfies equation (5.5). Only when the least value of \( D_1 + d_1 \) does not satisfy this equation closely, higher values can be tried. In fact when:

\[
D_1 + d_1 \geq D_j + d_j + 2,
\]

for any two closed partitions \( \Pi_i \) and \( \Pi_j \), \( \Pi_j \) always leads to the better decomposition - see Section 5.4.2 - except possibly for extremely large systems - with \( S \geq 21 \) - see curve III in Fig. 5.1.

**Example:** \( \Pi_1 = 1, 2, 7; 3, 9; 5, 11; 6; 7, 8; 13, 14; 4, 10, 12; 15, 16 \),
\[
\Pi_2 = 1, 2, 3, 4, 5, 6; 7, 8, 9, 10; 11, 12; 13, 14, 15, 16,
\]
\[
\Pi_3 = 1, 2, 3; 4, 5; 6, 7, 8, 9, 11; 10, 12; 13, 14; 15; 16.
\]

Thus, \( N = 16 \) giving \( S = 4 \); \( D_1 = D_3 = 3 \); \( D_2 = 2 \); \( d_1 = 2 \);
\[
d_2 = d_3 = 3.
\]

Hence, \( D_1 + d_1 = D_2 + d_2 = 5 \); \( D_3 + d_3 = 6 \).

Thus, only 2-component, serial-decompositions, are possible using any of these partitions.*  Making use of \( \Pi_3 \) requires 6 state-variables whereas, using \( \Pi_1 \) or \( \Pi_2 \) requires only 5. Thus the real choice is between \( \Pi_1 \) and \( \Pi_2 \); and since \( \Pi_1 \) satisfies equation (5.5) more closely than \( \Pi_2 \), it therefore follows that \( \Pi_1 \) is the best choice between the three. However, as was pointed out in Section 5.4.2, 2 component serial decompositions result in storage reduction only for \( S \geq 7 \) when \( S_D = S + 1 \).

II - When more than 2 serial components are possible, the total number of state-variables required for the whole structure is in general given by:

\[
S_D = D_1 + \sum_{i=2}^{n-1} \left\lfloor \log_2 Y_1 \right\rfloor + d_{n-1} \quad \cdots (5.6)
\]

* This example does not take into consideration the closed partitions obtained through the "-" and "+" operations on \( \Pi_1, \Pi_2 \) and \( \Pi_3 \).
for an n-component serial structure where \( \pi_n = \pi(0) \), and where
\[
\left| \log_2 \gamma_1 \right| \geq D_1 - D_{i-1}.
\]

It should be noted that with non-uniform closed partitions, it is the maximum number of blocks that a particular block of a partition breaks into from one partition to the next smaller one that dictates the number of state-variables of the intermediate component machines.

Applying equation (5.6) to the 2-component example given previously:
\[
S_D = D_1 + d_{2-1} = D_1 + d_1.
\]

5.4.4 Non-uniform closed partitions with \( N \) not a power of 2

In such cases using non-uniform closed partitions may or may not result in the utilization of more state-variables than \( S \), i.e.
\[
S_D \geq S \left( = \left| \log_2 N \right| \right).\]

As in the previous case:
\[
S_D = D_1 + \sum_{i=2}^{n-1} \left| \log_2 \gamma_i \right| + d_{n-1}.
\]

However, when \( N \) is \( <2^S \), a degree of non-uniformity in the closed partitions may be allowed without any subsequent increase in the total number of state-variables over \( S \). Two cases are considered below:

I - When only one closed partition may be selected:

As in the last section, the values of \( D \) and \( d \) for each of the closed partitions the machine possesses, are computed. The closed partitions \( \pi_i \), with \( D_i + d_i \) equal to \( S \) thus, results in a 2-component serial structure requiring a total of \( S = \left| \log_2 N \right| \) state-variables; and when \( D_i + d_i = S + \Delta S \) then, the serial structure requires \( S + \Delta S \) state variables.

The effects of this increase in the number of state-variables can be seen in Fig. 5.1 for \( \Delta S = 1 \) and 2.

* \( \Delta S \) is a small integer indicating the relative increase in the value of \( S_D \) over \( S \).
Example:

\[ \pi_1 = \{1, 2, 3, 4, 5 \};  \{6, 7, 8, 9, 10, 11, 12, 13\} \]
\[ \pi_2 = \{1, 2, 3, 4 \}; \{5, 6, 7, 8, 9, 10, 11, 12, 13\} \]

Thus, \( d_1 + d_1 = 1 + 3 = 4 = S \)
and, \( d_2 + d_2 = 1 + 4 = 5 = S + 1 \).

Thus, \( \pi_1 \), would lead to a 2-component decomposition with \( S_D = S = 4 \);
whereas, with \( \pi_2 \), \( S_D = 5 \). Clearly, \( \pi_1 \) is preferable to \( \pi_2 \) since the
successor component with \( \pi_1 \) would have only 3 state-variables \( (= d_1) \);
whereas, with \( \pi_2 \), the successor component would have 4 \( (= d_2) \).

II - When more than one closed partition is incorporated:

When there are \( n \) closed partitions involved, then provided that:

\[ \log_2 \gamma_i = D_i - D_{i-1}, \text{ for } 2 \leq i \leq n-1, \]

where, \( \pi_n = \pi(0) \) then, the various component machines would have the
following numbers of state-variables:

\[ S_1 = D_1 ; \]
\[ S_2 = D_2 - D_1 ; \]

And, in general, \( S_i = D_i - D_{i-1} ; \)

and, \( S_n = d_{n-1} \).

The total number of state-variables in a decomposed structure based
on \( n \) such closed partitions is given by:

\[ S_D = (d_j + d_j)_{\text{max}}, \text{ for } 1 \leq j \leq n-1 ; \]

i.e. \( S_D \) is given by the maximum value of \( d_j + d_j \) for all the closed
partitions involved in the serial structure; in other words, by the most
redundant closed partition utilized in the decomposition.

Example:

\[ \pi_1 = \{1, 2, 3, 4, 5, 6 \};  \{7, 8, 9, 10, 11, 12, 13, 14\} \]
\[ \pi_2 = \{1, 2, 3, 4 \}; \{5 \}; \{6 \}; \{7, 8, 9 \}; \{10 \}; \{11, 12, 13, 14\} \]
\[ \pi_3 = \{1, 2 \}; \{3, 4 \}; \{5 \}; \{6 \}; \{7, 8 \}; \{9 \}; \{10 \}; \{11, 12, 13 \}; \{14 \} \]
Thus, $\pi_1 > \pi_2 > \pi_3 > \pi(0)$; hence a serial structure involving all the three closed partitions is possible:

$$
\begin{align*}
D_1 &= 1; \quad D_2 = 3; \quad D_3 = 4; \\
D_1 &= 3; \quad d_2 = d_3 = 2.
\end{align*}
$$

The number of state-variables in such a decomposed structure is given by:

$$S_D = (D_j + d_j)_{\text{max}} = D_3 + d_3 = 6.$$

1st predecessor component has $D_1$ state-variables $= 1$,
1st intermediate (or 2nd predecessor) has $\left\lfloor \log_2 r_2 \right\rfloor$ state-variables
$= D_2 - D_1 = 2$.
2nd intermediate (3rd predecessor) has $\left\lfloor \log_2 r_3 \right\rfloor$ state-variables
$= D_3 - D_2 = 1$.
Final successor component has $d_3$ state-variables $= 2$.

Thus, $S_D = D_1 + (D_2 - D_1) + (D_3 - D_2) + d_3 = D_3 + d_3 = 6$.

However, when the closed partitions involved are of a more redundant nature than their $D$ and $d$ values suggest, then some, if not all, of the intermediate component machines, $M_i$, would have $\left\lfloor \log_2 r_i \right\rfloor$ state-variables, where:

$$\left\lfloor \log_2 r_i \right\rfloor > D_i - D_{i-1}, \quad \text{for } 2 \leq i \leq n - 1.$$

If for any closed partition $\pi_i$ incorporated in a serial decomposition structure, $D_i + d_i > S$ then, it is clear that there can be no reduction in the storage requirement on that of the undecomposed machine for systems with $S \leq 7$, except when the structure comprises a relatively large number of components, and when the final successor components have only one state-variable each.

Furthermore, if $\left\lfloor \log_2 r_i \right\rfloor > D_i - D_{i-1}$, for any two closed partitions $\pi_i$ and $\pi_{i-1}$, and if, for either of these two partitions, $D + d > S$ then, clearly $S_D > S + 2$. Hence, as has been shown in Section 5.4.2,
there can be an overall storage reduction in only very large sequential systems. The higher the value of $S_D$ as compared with $S$ the more random, and redundant, the closed partitions are.

5.5 Elimination of Redundant Closed Partitions

In order to arrive at the most economical loop-free decomposition structures for a particular system with the least effort, the possible closed partitions should be identified and all redundant (or costly)* ones eliminated before the procedure for actually choosing the best decomposition structure is considered. However, the term "redundant closed partitions" implies different meanings for the serial and parallel cases:

I - Parallel Case: The redundant closed partitions in this case are easily eliminated from the list of all possible partitions simply by comparing the $D$ values of all closed partitions involved in the different possible structures.

Other parameters of importance are $\sum D$ for all closed partitions involved in these structures and $n$, the number of component machines in the different structures.

The variations in the $D$-values show the variation in the sizes of the different components; $S_D - S$, where $S_D = \sum D$, is a measure of the redundancy involved in the different components; and when these parameters are considered in connection with $n$, the best parallel structure, in terms of component uniformity and storage requirements can easily be determined.

II - Serial Case: Three types of redundant closed partitions exist:

If there are two, or more, closed partitions, $\Pi_1, \Pi_2, \ldots, \Pi_k$, such

* i.e. closed partitions that require an unnecessarily large number of state-variables.
that, $\pi_1 \geq \pi_2 \geq \cdots \geq \pi_k$, then:

1. If they all have the same value of $d$ but different values of $D$, then amongst all these partitions the one with the least value of $D$ should be retained in the list of closed partitions and all others eliminated since, these require a larger number of state-variables to encode their blocks without supplying any additional information.

Example: $\pi_1 = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$, $\pi_2 = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$, $\pi_3 = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$.

Thus, $\pi_1 > \pi_2 > \pi_3$; $d_1 = d_2 = d_3 = 3$; whereas, $D_1 = 2$; $D_2 = 3$; $D_3 = 4$.

Hence $\pi_1$ is chosen and $\pi_2$ and $\pi_3$ are eliminated from the list of possible partitions.

2. If they all have the same value of $D$ but different values of $d$, then amongst all these partitions the one with the least value of $d$ should be retained and all others eliminated. The partition with the least value of $d$ is the most uniform of all those involved.

Example: $\pi_1 = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$, $\pi_2 = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$.

$\pi_1 > \pi_2$; $D_1 = D_2 = 3$; $d_1 = 3$; $d_2 = 2$.

Hence, $\pi_2$ is retained and $\pi_1$ eliminated.

3. If they all have the same $D$ and $d$ values and, if the inter-partition dispersion ratio of $\pi_i$ (for $i = 1, 2, \ldots, k$), with respect to the closed partition immediately larger than $\pi_1$ is $\gamma_i$, then the closed partition with the least value of $\gamma$ is retained in the list; whereas, all the remaining partitions amongst $\pi_1, \ldots, \pi_k$ are eliminated since, they supply no information which cannot be obtained from the one retained (in conjunction
with others in the final list).

Example: \( \pi_1 = 1,2,3,4,5,6,7,8 ; 9,10,11,12,13 \),
\( \pi_2 = 1,2,3 ; 4,5,6,7 ; 8 ; 9,10,11 ; 12,13,14 \),
\( \pi_3 = 1,2,3 ; 4 ; 5 ; 6 ; 7 ; 8 ; 9,10,11 ; 12,13,14 \).

Clearly \( \pi_1 > \pi_2 > \pi_3 \); \( D_1 = 1; D_2 = D_3 = 3 \); \( d_1 = 3; d_2 = d_3 = 2 \).

Hence, since \( D_2 = D_3 \), and \( d_2 = d_3 \), it therefore follows that one of them is redundant.

Now \( Y_2 = 2 = D_2 - D_1 \); whereas, \( Y_3 = 3 \neq D_3 - D_1 \). Thus \( \pi_3 \) is more non-uniform and is thus eliminated because \( Y_3 > Y_2 \).

5.6 Effects of Input and Output Consistent Partitions

This section analyses the effects of input, and output, consistent partitions on the possible choices of loop-free decomposition structures. The bases on which the choice of partition is made are either the storage reduction achieved, or the uniformity of the resulting component machines. The various possible cases are dealt with separately:

5.6.1 Effect of input consistency on parallel structures

Considering a machine with the usual parameters \( (x,S,z) \), let it be decomposable into \( n \) components in parallel. This machine must then possess \( n \) closed partitions, \( \pi_1, \ldots, \pi_n \), such that,
\[ \pi_1 \cdot \pi_2 \cdot \ldots \cdot \pi_n = \pi(0) \cdot \]

Let component \( M_i \) be based on \( \pi_i \); thus:
\[ S_i = \log_2 \#(\pi_i) \quad \text{and} \quad S_D = \sum_{i=1}^{n} S_i \cdot \]

Also, \( M_D = z^{\frac{n}{2}} \sum_{i=1}^{n} S_i \cdot 2^{S_i} \); whereas, \( M_o = S \cdot 2^{S+X} \).

If \( \pi_i \) is an input-consistent partition with respect to \( i \) of the
external inputs, the storage requirement of component \( M_i \) is then given by:

\[
m_i = S_i \cdot 2^{s_i + x} - f_i = (\frac{1}{2} f_i) \cdot S_i \cdot 2^{s_i + x}.
\]

Then, the total storage of the decomposed structure becomes:

\[
M^*_D = 2^x \cdot \sum_{i=1}^{n} \left( \frac{1}{2} f_i \right) \cdot S_i \cdot 2^{s_i}.
\]

When all \( n \) components happen to be of equal size, then if one closed partition \( \Pi_i \) is input-consistent with respect to only one external input, then \( M_D \) is reduced by a factor of \( \frac{1}{2n} \). When \( \Pi_i \) is input-consistent with respect to \( f_i \) of the external inputs, then the size of \( M_D \) is reduced by a factor of:

\[
\left( \frac{1}{n} \right) \left( 1 - \frac{1}{2} f_i \right)^* = \left( \frac{1}{n} \right) \left( 1 - \frac{1}{2} f_i \right)
\]

From equation (5.7), when \( n \) is very small (i.e. \( n \leq 3 \)), then if:

i - all \( n \) components are of equal size then, the effect of an input consistency is to reduce the size of the component concerned appreciably; e.g. \( n = 3, f_1 = 2 \) then, \( M_D \) is reduced by a factor \( \frac{1}{4} \); ii - all \( n \) components are of uneven sizes, then the larger the component machine is, the more statistically improbable that the closed partition, on which it could be based, would happen to be input consistent; since, a large size component implies a small closed partition and, the smaller the partition is, the more unlikely that it would be input-consistent. Thus, the smaller of the component machines in a parallel structure, are the more likely ones to be input-consistent; this, however, is likely to lead to an even greater

* When all \( n \) components are of equal size:

\[
M_D = n \cdot \left( S_D/n \right) \cdot 2^{x + S_D/n} = S_D \cdot 2^{x + S_D/n}.
\]

When one component is independent of \( f \) of the \( X \) external inputs, then:

\[
M^*_D = (n-1) \cdot \left( S_D/n \right) \cdot 2^{x + S_D/n} + \left( \frac{1}{2} f \right) \cdot \left( S_D/n \right) \cdot 2^{x + S_D/n}.
\]

\[
= (n - 1) + \frac{1}{2} \left( S_D/n \right) \cdot 2^{x + S_D/n}; \quad \text{which is}
\]

\[
(1 - \frac{1}{2} f) \left( S_D/n \right)^2, \quad \text{less than } M_D; \quad \text{thus, } (5.7).
\]

77
degree of non-uniformity in the sizes of the different components.

However, if \( n \) is \( \geq 4 \), then whether or not the components are uniform, the effects of input-consistencies on the total storage requirement is unlikely to be significant. However, since with a large number of components, the closed partitions on which the components are based are large, it is thus more likely that some of them would be input-consistent. As the formula in (5.7) indicates, the storage reduction due to individual input-consistencies is inversely proportional to the number of components.

5.6.2 Effects of output consistency on parallel structures

The two classes of machines are considered separately:

1 - Moore Machines\textsuperscript{27}: The outputs of such machines can be obtained from a separate matrix, the inputs to which are the \( S_D \) state-variables of the whole structure. However, when some of the closed partitions, on which the various components are based, happen to be output-consistent\textsuperscript{8,40} with respect to one or more of the output functions then, these can be more economically realised from individual SPMs, the inputs to which are the relevant state-variables only.

Thus, if \( \pi_1 \) is output-consistent with respect to \( Z_j \), then, \( Z_j \) is dependent on \( S_i \) state-variables only, where:

\[
S_i = \left\lfloor \log_2 \#(\pi_i) \right\rfloor^*,
\]

With each output-consistent partition, the total store size required would be reduced by:

\[
\frac{S_D}{2} - \frac{S_i}{2} \text{ Bits,}
\]

and when there are \( Z \) output-consistent partitions then, the output store size is reduced by:

\[
* \text{ When } S_i \text{ is } \leq 3, \text{ then the output function concerned would be more economically obtained through some simple combinational logic.}
\]
If the structure comprises $n$, equi-sized, parallel components then:

$$S_i := \frac{S_D}{n},$$

and the total storage reduction is then given by:

$$Z \cdot \left( 2^{S_D} - 2^{S_D/n} \right).$$

Thus, with $n \geq 3$, the store size required to implement the output functions in such cases, is substantially reduced.

II - Mealy Machines: In this case, since any of the $X$ external inputs are unlikely to be redundant in the realisation of the output functions (since this would imply a Moore machine), it therefore follows that this case is similar to that in (I) above, except that a factor of $2^X$ must be taken into account.

5.6.3 Effect of input consistency on serial structures

Assuming a sequential system to decompose into a serial structure of $n$ components, then if:

1. $n \ll S$: the best serial structure, in terms of storage reduction, and in terms of uniformity in the component sizes, is when the distribution of state-variables between the $n$ components is as close as possible to that given in Table 5.2. With such structures, the numbers of state-variables of the successor components are much smaller than those of their predecessor ones; it therefore follows that the partitions on which the successor machines are based are larger, in terms of block sizes, than those of the predecessor ones and, as a consequence, it is more likely for the successor components in such decompositions to be input consistent and thus more likely to result in substantial storage reduction.
Assuming all \( n \) components in such a structure to be of exactly equal size then an input-consistency, with respect to one external input, reduces the total storage requirement by a factor of \( \frac{1}{2n} \).

\( ii \) - \( n \) is comparable with \( S \): In such structures, all components are based on nearly equi-sized partitions, and the larger the value of \( n \) the larger the partitions on which the various components are based. Thus, all components in such structures have nearly equal statistical probability of being input-consistent. However, the effect of an input-consistency in one of the final successor components is much greater (in terms of storage reduction) than an input-consistency in one of the first predecessor ones since, in such structures, the sizes of the components increase in a logarithmic manner from the first predecessor to the final successor.

The presence of input-consistent partitions, in one or more of the possible serial structures, only starts to affect the choice of a particular set of closed partitions when the additional state-variables that are introduced through the augmentation of the system under consideration, are more than offset (in terms of storage requirements) through input consistencies. A simple comparison in both storage requirement and component sizes reveals the best possible structure.

5.6.4 Effect of output consistency on serial structure

This is very similar to the parallel case discussed in Section 5.6.2. The difference is that if \( \pi_i \) is an output-consistent partition, then the output function \( Z_i \), which is consistent with respect to \( \pi_i \), is obtainable from the state-variables of component \( M_i \), and up to a maximum of all those of its predecessors (i.e. all those components which together realise \( \pi_i \)). In the parallel case \( \pi_i \) is realised by \( M_i \) only.
The reduction in storage due to each output-consistent partition is exactly the same as in the parallel case; i.e.,
\[ x_{D}^2 - 2 x_{I} \] in the Moore case and,
\[ 2^x \left( x_{D}^2 - x_{I} \right) \] in the Mealy case.

5.7 Generalised Procedure for Choosing a Decomposition Structure

In this section, a procedure for choosing the most suitable loop-free decomposition structure for a sequential machine so as to lead to either the most uniform component machine sizes and structures or to the least storage requirement for the whole structure is proposed.

Starting from the list of closed partitions, the procedure is as follows:

1 - Form the lattice of the closed partitions 7, 8.

2 - From this lattice, form all possible serial and parallel structures for the machine under consideration.

3 - Compute the values of \( D, d \) and \( \left| \log_2 \eta_i \right| \) for all closed partitions involved in step 2.

A - The Parallel Case:

i - Eliminate all redundant, or costly, closed partitions in accordance with Section 5.5.

ii - Compute the value of \( S_D = \sum_{i=1}^{n} D_i \) for all components in the possible parallel structures.

iii - Compute the spread in the D-values of the closed partitions involved in each of the parallel structures - i.e. \( \alpha_D \) in Section 5.3.2.

iv - If the best parallel structure from component-uniformity point of view is desired, then the structure with the least value of \( \alpha_D \) is chosen.
v - The best overall choice is that which leads to near-uniformity and, at the same time, achieves a sizeable reduction in storage. The possible structures are compared two at a time starting with the least $S_D$ values. The uniformity of the resulting structures as well as their storage requirements are compared. When the non-uniformity is offset by the reduction in storage, then the structure with the least storage is then chosen and compared with the other possible structures.

vi - The effects of input and output consistent partitions on the overall final structures - i.e., on those which share similar properties, in terms of uniformity of components and storage requirements, are examined; the best final choice is then made.

B - The Serial Case:

i - Eliminate all redundant closed partitions in accordance with Section 5.5.

ii - From the computed values of $D$, $d$, and the relevant values of $\gamma$, all possible serial structures are then listed; two cases exist:

a - When the values of $\gamma$ are all given by:

$$\left| \log_2 \gamma_j \right| = D_j - D_i$$

where $\pi_i \gg \pi_j$ in a particular serial structure, and that $\pi_i$ and $\pi_j$ are both relevant to that structure.

The first predecessor component in such a structure has $S_1 = D_1$ state-variables, and a storage requirement $m_1 = D_1 \cdot 2^{D_1}$ Bits.

The second predecessor (or lst. intermediate) component has $S_2 = D_2 - D_1$ and,

$$m_2 = (D_2 - D_1) \cdot 2^{D_2}.$$ 

In general, the jth component has

$$S_j = D_j - D_{j-1} \text{ and, } m_j = (D_j - D_{j-1}) \cdot 2^{D_j}.$$ 

The final successor component has $S_n = d_{n-1}$ and $m_n = d_{n-1} \cdot 2^{S_D}$

where: $S_D = S_1 + S_2 + \ldots + S_j + \ldots + S_n$.

* Ignoring the effects of the X external inputs.
From equation (5.8) it is clear that the first \( n - 2 \) terms are all negative since the values of \( D \) increase progressively from the first predecessor component. Thus:

\[
M_D = m_1 + m_2 + \ldots + m_j + \ldots + m_n.
\]

By comparing \( M_D \) in (5.9) with the storage requirement of the undecomposed machine, the economical aspect of the particular decomposition structure is determined (the effects of input and output consistent partitions having been taken into consideration).

b - When \( |\log_2 \gamma_j| > D_j - D_1 \), the various components would have the following parameters:

The first predecessor component has \( S_1 = D_1 \) state-variables and, \( m_1 = D_1 \cdot 2^k \) Bits of storage requirement.

The 1st intermediate (2nd predecessor) component has \( S_2 = D_2 - D_1 \), or \( |\log_2 \gamma_2| \), whichever is the greater; \( m_2 = \beta_2 \cdot 2^{D_1 + \beta_2} \),

where \( \beta_j = |\log_2 \gamma_j| \).

In general, \( S_j = \beta_j; \) \( m_j = \beta_j \cdot 2^{\beta_j + \ldots + \beta_2 + D_1} \).

The final successor machine would then have \( S_n = d_n \); and,

\[
m_n = d_n \cdot 2^{d_n + \beta_{n-1} + \ldots + \beta_2 + D_1} = d_n \cdot 2^{s_D}.
\]

where:

\[
S_D = S_1 + S_2 + \ldots + S_n \text{ and, } S_n = d_n.
\]
Clearly a particular value of $S$ in case (b), when $S_p \gg S$, can lead to a large storage requirement. A further disadvantage, when a large amount of redundancy is introduced, or when very non-uniform closed partitions are used, is the extremely random sizes of the resulting components due to the unpredictable number of inputs to the various components.

5.8 Comments and Conclusions

Chapter 4 dealt with the best possible, relative, component machine sizes, in terms of total storage requirements and component uniformity, for a particular loop-free structure. The formulae derived there, assumed a choice of closed partitions which rendered the different structures possible. However, these partitions had little or no real relationship to the actual closed partitions that a particular sequential system may possess; and thus represented an incomplete analysis.

Chapter 5, however, developed the relationships between these formulae and the actual closed partitions. The interpartition relationships that may exist in a particular sequential machine and, which give rise to a particular structure were considered in relation to the overall storage requirement of the system and its component sizes and structures:

For the parallel loop-free structure it was shown that best structures are obtained for equi-sized components when least amount of redundancy is introduced into the system. Also the extent to which redundancy, in the form of either non-uniform and redundant closed partitions, or through state-splitting\(^3\), could be incorporated into a system in order to decompose it into a parallel structure and still results in an economical realisation (using LSI memory modules) was determined; this was also considered in relation to the uniformity of the resulting components.

For serial structures, the usefulness of the formula (5.4):

\[
1 + S_1 \cdot \ln 2 = 2^{S_2}
\]
in choosing the best possible serial structures in terms of component sizes, so as to result in least storage requirement (for a given number of components) is illustrated. Also, the effects of introduced redundancy (in the form of, either state-splitting, or through the usage of non-uniform closed partitions) on the storage requirement of the structure were considered. The lower limits of system sizes to which different degrees of redundancy is added and still resulted in economical serial structures were determined: as a general rule, if the system decomposes into only 2-components then most practical-sized systems (i.e. $S \leq 18$) can only tolerate an increase of 2 in the number of their state-variables; however, as the number of components, in a given structure, is increased, the size of the original system, which could be economically decomposed, is gradually reduced (at the cost of increased variation in component sizes and structures).

The effects of input, and output, consistent partitions on the overall storage requirement of the system were considered, for the different structures.

Finally, a useful procedure for choosing the most suitable decomposed structure, in terms of component uniformity and/or storage requirement, was developed (starting from the complete list of S.P. partitions).
CHAPTER SIX
REALISATION TECHNIQUES AT THE SUB-SYSTEM LEVEL

6.1 Introduction

In Chapters 4 and 5, the loop-free structures due to Hartmanis's structure theory were dealt with in detail; the best possible structures, and the inter-partition relationships that best lead to them were also discussed. However, as was clearly implied in these chapters, either not all sequential systems result in structures comprising components of convenient size and structure, or that they have to be augmented to a large degree in order to make them decompose. In the latter case, again the component machines are likely to be of inconvenient sizes and structures, as well as comprising (in some cases) an enlarged storage.

In this chapter, two realisation techniques which can be used to decompose sequential systems, into a predetermined number of components, each having a predetermined size and structure, are presented. The uniformity of the components and their pattern of inter-connection make them very suitable for ISI realisations.

The two techniques discussed in this chapter, termed State and Input decompositions, make use of the principles of functional decompositions first proposed by Ashenhurst\textsuperscript{25} and Curtis\textsuperscript{26}.

In a combinational system, if two sets of variables exist whose effect is clearly separated in the time domain, its functional decomposition is then termed trivial\textsuperscript{25,26}, since such a system can then be regarded as comprising two independent sub-systems. However, sequential systems, in their normal mode of operation, comprise two such sets of variables and thus the principles of functional decomposition can be applied on them in a non-trivial manner.
With a combinational switching system, there is only one set of variables, and through the use of functional decomposition such systems are realised from much smaller components, the inputs to which are determined by the characteristics of the switching function concerned.

With sequential switching systems, two sets of variables exist - the external inputs and the state-variables; however, only one set of variables is, normally, allowed to change at any one time. In other words, the total effect is sequential; i.e. the external inputs change and, as a consequence of this, a change in the state-variables takes place. Thus, it is these two sets of variables which, when separated in the time domain, lead to the input, and state, realisation techniques - on bases analogous to those of functional decomposition.

These two techniques result, for any sequential system, in uniform components and in regular interconnections, and can lead to component sizes compatible with available SPM sizes and structures. They lead to systems whose maximum speed of operation is not affected by such realisations (i.e. dependent only on the switching characteristics of the hardware used) and, as will be shown in Chapter 7, lead to a simplification of the state-assignment problem.

6.2 Theoretical Model of State/Input Realisation Techniques

In a sequential switching system, any next-state variable $Y$, and any output function $O$, can be expressed as:

$$ Y = f \left( I_1, I_2, \ldots, I_x; y_1, y_2, \ldots, y_s \right) , $$

and,

$$ O = g \left( I_1, I_2, \ldots, I_x; y_1, y_2, \ldots, y_s \right) , $$

where: $\{I\}$ is the set of external inputs.

However, the effects of the two sets of variables $I$ and $y'$ are consecutive since a change in the external inputs to the system, i.e. in one or
Fig. 6.1 Schematic representation of disjunctive decomposition.

Fig. 6.2 a- Model for State Decomposition of a sequential machine.

b- Model for Input Decomposition of a sequential machine.
more of the \( X \) variables, leads, after a certain time interval has lapsed, to a change in the state-variables. Thus, assuming the machine to be in its fundamental mode of operation\(^{12}\), the effects of the two sets of variables can be separated in the time domain, and represented as:

\[
Y = \Theta_2 \left[ \Theta_1 (I_1, I_2, \ldots, I_x), y_1, y_2, \ldots, y_s \right], \ldots (6.1)
\]

and,

\[
0 = \Phi_2 \left[ \Phi_1 (I_1, I_2, \ldots, I_x), y_1, y_2, \ldots, y_s \right], \ldots (6.2)
\]

where \( \Theta_1, \Theta_2, \Phi_1 \) and \( \Phi_2 \) are all logical algebraic functions.

Thus, any output function \( 0 \), or any next-state variable \( Y \), can be realised through the combined effects of the external inputs, \( I \), superimposed on that of the present-state variables, \( y \).

Equations (6.1) and (6.2) can be modelled as shown in Fig. 6.1, where the first component computes the effects of \( I \) and issues an appropriate output function \( \Theta_1 \) (or \( \Phi_1 \)) which is then fed into the second component which realises the function \( \Theta_2 \) (or \( \Phi_2 \)).

The difference between the state and input cases is that in the former's, \( \Theta_1 \) (or \( \Phi_1 \)) computes the combined effects of changes in the state-variables \( y \) and, \( \Theta_2 \) (or \( \Phi_2 \)) those in the external inputs \( I \), i.e. the sets of variables in equations (6.1) and (6.2) are interchanged. This model is shown in Fig. 6.2.a; whereas, in the input case, \( \Theta_1 \) (or \( \Phi_1 \)) computes the effects of changes in \( I \) and, \( \Theta_2 \) (or \( \Phi_2 \)) those in \( y \), as given in equations (6.1) and (6.2). The model for the input technique is shown in Fig. 6.2.b.

When considering functional decomposition of combinational switching functions, a function \( f \) may be expressed as a composite function of functions:

\[
f(a_1, a_2, \ldots, a_m) = f_2 \left( f_1(b_1, b_2, \ldots, b_n), c_1 c_2, \ldots, c_r \right)
\]

\( \ldots (6.3) \)
where: \( B = (b_1, b_2, \ldots, b_n) \) and,
\[ C = (c_1, c_2, \ldots, c_r) \]
are two subsets of:
\[ A = (a_1, a_2, \ldots, a_m), \]
such that, \( B \cup C = A \).

When the subsets \( B \) and \( C \) are disjoint then:
\[ f(a_1, a_2, \ldots, a_m) = f_2 \left( f_1(b_1, b_2, \ldots, b_n), c_1, c_2, \ldots, c_{m-r} \right) \]
\[ \ldots (6.4) \]
i.e. \( m = n + r \). Such functional decompositions are called disjunctive decompositions. A schematic representation of equation (6.4) can be taken as Fig. 6.1 merely by replacing \( \varphi_1 \) with \( f_1 \) and \( \varphi_2 \) with \( f_2 \).

Thus, the similarity of the state and input techniques to that of disjunctive decomposition is clear since, each function in a sequential system can be realised in an analogous manner to a disjunctively decomposed combinational logic function. However, when dealing with sequential switching functions, no decomposition charts are necessary to determine the two sets of components in state and input techniques since the two sets of inputs are independent and their effects are separated in the time domain.

6.3 State and Input Decompositions

These two realisation techniques for sequential switching systems are very similar in their outline and, indeed, can be thought of as being each other's complement. They are based on principles which are analogous to those of functional decomposition of combinational switching functions.

By treating a sequential switching system as consisting of two distinct parts, the combinational logic part and the feedback loops, the former can be decomposed in accordance with the state or input techniques. The two cases are discussed below:
Fig. 6.3 General outline of State decomposition.
6.3.1 Outline of state decomposition

Fig. 6.3 shows an outline of state decomposition where the present-state variables, as a subset of the total set of inputs to a sequential system, are fed into a decoder which acts as the $\Theta_1 (S_1)$ function in Fig. 6.1; the decoder computes its output and feeds it to the second combinational network which has the external inputs as well as the output of the decoder as its inputs. This second component, which is equivalent to the function $\Theta_2 (S_2)$ in Fig. 6.1, is further decomposed into as many combinational networks as there are outputs from the first component (the decoder). Thus the only difference between the systems shown in Figs. 6.1 and 6.3 is in the number of functional-type structures contained in the latter.

When dealing with LSI modules, the second component machine is realised from $N$ modules (where $N$ is the number of internal states of the system) the inputs to all of which are the external inputs $I$ only. The outputs of the first component are used to "select" or "enable" the appropriate final component which, in conjunction with the output of the first one, leads to the desired outputs from the system (irrespective of whether these are the actual output functions of the system or its next-state variables which are then fed back as inputs to the first component — see Figs. 6.2.a and 6.3).

In other words, depending on the present-state of the machine, as determined by the output of the first component — the decoder — the second component arrives at its outputs, depending on the state of the external inputs.

The decoder "enables" only one SPM at a time, in accordance with the present-state of the machine. Whenever the machine changes its internal state, the SPM being enabled changes accordingly.
Fig. 6.4 General Outline of Input decomposition.
The external inputs are fed in a parallel fashion to all SPMs, and they constitute the address inputs to the storage locations. When a particular location is addressed, it issues a pre-programmed set of outputs. The outputs of all SPMs are also connected in parallel and they constitute the next-state variables which are then fed back through latches on delays to the inputs of the decoder; they may also constitute the output functions of the system.

6.3.2 Outline of input decomposition

This is the counterpart of the state technique outlined previously; the roles of the external inputs and present-state variables being interchanged. The external inputs are fed directly to the decoder, the outputs of which are then used to enable the different SPMs. Fig. 6.4 shows a schematic diagram of input decomposition.

6.3.3 Theory of state decomposition

A combinational switching network can be thought of as a single-state sequential switching network since such a network can only "remember" its present-state, and its output is therefore dependent only on the current value of its external inputs. Thus, such a network can be realised using a single SPM of appropriate structure (X inputs and Z outputs).

Sequential switching systems can be realised using LSI memory modules, by connecting together N SPMs, each, corresponding to one of the N internal states of the system under consideration. Thus, in effect, each of these N networks becomes a purely combinational one whose outputs depend entirely on the current value of its inputs. When the sequential machine happens to be in state $T_i$, the SPM associated with this internal state is activated and, only this component then responds to the changes
in the external inputs (exactly as in a combinational network); however, the effects of a change in the outputs of any of the SPMS may or may not end there (unlike the case of combinational networks\*), since some, if not all, are fed back through delays or latches to the inputs of the machine. It is obvious that the changes in the outputs of any SPM in this realisation are the combined result of the present-state, the sequential machine happen to be in, and, the external inputs.

In order to ensure that only one SPM, and the appropriate one at that, gets activated (while all other SPMS are not), it is necessary to associate the changes in the internal-states to changes in the SPM being "enabled". The presence of the decoder in the feedback loops of the system can guarantee this since, only one output of the decoder - the one which corresponds to the internal state the system is in - is ever at a logical one. Thus, by inter-connecting the SPMS to the appropriate outputs of the decoder, the sequential character of this realisation is completed.

With SPM realisations, it is only possible to "enable" one SPM, and to simultaneously "disable" another, by applying a logical "1" on to the former's M/E ** and a logical "0" on to the latter's M/E. Thus, the decoder which should have N output lines should "disable" N-1 SPMS and "enable" the one SPM corresponding to the internal state the machine is in.

Thus, referring to Fig. 6.3, at any instant of time, any next-state variable \( Y_j \) is given by:

\[
Y_j = \theta_j \left( \theta^i_j (y); I \right) \quad \ldots \quad (6.5)
\]

where \( \theta \) and \( \theta^i \) denote two algebraic functions; \( y \) is the set of state-variables, and \( I \) is the set of external inputs.

\* Where no output function is fed back to the inputs of the network.

\** M/E is the Memory Enable pin of any SPM.
The similarity between equations (6.5) and (6.1) is obvious, and they can, in fact, be regarded as isomorphic. Thus, any next-state function can be realised, at any instant of time, in an analogous manner to a disjunctively decomposed combinational switching function, where the two sets of variables are \( y \) and \( I \).

Since at any instant of time, the whole sequential system is modelled by a single SPM it therefore follows that for the particular SPM being "activated":

\[
y_j = \Theta_j(I) \quad \text{... (6.6)}
\]

where \( \Theta_j(y) \) in equation (6.5) is either 0 or 1, and is 1 for the SPM being activated - i.e. the SPM issuing the relevant outputs of the machine at any given instant. Clearly \( \Theta_j(y) \) corresponds to the individual internal states of the system.

By feeding the set of state-variables \( y \) into a decoder, the function \( \Theta^r(y) \) in equation (6.5) is realised since the outputs of the decoder then correspond to the individual internal states of the system; thus, the decoder serves as the equivalent of the front part of a disjunctively decomposed combinational function - i.e. the function \( \Theta_1 \) (or \( \Phi_1 \)) in Fig. 6.1. Similarly, the combinational network consisting of \( N \) SPMs in Fig. 6.3 realises the equivalent of the tail part of a disjunctively decomposed combinational function - i.e. \( \Theta_2 \) (or \( \Phi_2 \)) in Fig. 6.1. Each of the \( N \) SPMs issues its outputs in accordance with the \( I \) external inputs in conjunction with the output of the decoder.

Thus, all next-state functions of a sequential system have been realised from two disjoint sections: one taking into account the effects of changes in the state-variables and, the other the effects of the external inputs.
The output functions of the system \( O_k \) can be obtained in a similar manner; i.e.

\[
O_k = \phi_k \left( \phi'_k(y); I \right)
\]  

... (6.7)

where, \( O_k \) denotes any output function, and \( \phi \) and \( \phi' \) are two algebraic functions.

Equation (6.7) can be reduced to

\[
O_k = \phi_k \left( I \right)
\]  

... (6.8)

when considering the individual SPM being activated.

6.3.4 Theory of Input Decomposition

This follows very closely that of the state technique, outlined previously, except that the roles of the external inputs \( I \) and the state-variables \( y \), are interchanged. Thus, the first component (the decoder) computes the function \( \Theta' (I) \) and, the second component the function \( \Theta (y) \), in conjunction with the output of the first component.

Thus:

\[
Y_j = \Theta_j \left( \Theta'_j (I); y \right)
\]  

... (6.9)

and, for the particular SPM being activated:

\[
Y_j = \Theta_j \left( y \right)
\]  

... (6.10)

In equation (6.9), \( \Theta' (I) \) computes the column of the state-table the machine happens to be in and issues an output of 1 or 0, depending on whether the machine is in a particular column or not; equation (6.10) is the form to which equation (6.9) reduces to when a 1 is issued.

Thus, in an input decomposition, \( \Theta' \) is realised by the front component whereas, \( \Theta \) is realised by the tail component which, as in the state case, is realised from \( N \) modules, the inputs to all of which are \( y \).

The equivalent equations for the output functions, \( O \), are given by:

\[
O_k = \phi_k \left( \phi'_k(I); y \right)
\]  

... (6.11)

*The algebraic functions \( \Theta \) and \( \Theta' \) in equations (6.9) and (6.5) are different, but are retained to bring out the complementary nature of the two techniques.
6.3.5 Operation of state decomposition

Denoting the internal states of a sequential system by $T_i$, for $i = 1, 2, \ldots, N$, the operation of a machine decomposed using the state technique is as follows:

I - Synchronous Operation

Let the machine be in present-state $T_i$ under external input $I_j$ - i.e. under input combination, or column, $I_j$. Thus component $M_i$ (see Fig. 6.3) is enabled and issues the next-state $\delta(T_i, I_j)$ as its outputs. Two things can happen at this moment: 1 - a clock pulse arrives and thus feeds the code for $\delta(T_i, I_j)$ to the inputs of the decoder which in turn "disables" $M_i$ and enables $M_x$ (where $\delta(T_i, I_j) = T_x$) simultaneously. Component $M_x$ then issues its next-state $\delta(T_x, I_j)$, etc.

2 - The external inputs change from $I_j$ to $I_y$; thus the address fed to $M_i$ changes and, correspondingly its next-state changes to $\delta(T_i, I_y)$. This new next-state now awaits the arrival of the clock pulse which precipitates a cycle of changes as was witnessed in 1.

It should be remembered that with synchronous machines, the normal mode of operation is for the clock pulse to arrive at a given time interval after a change in the external inputs; the two are prevented from occurring at the same instant of time.

II - Asynchronous Operation

The state assignment problem in asynchronous machines is a very complex one, and is discussed in Appendix 2. In Chapter 7, it is treated using modified versions of the state, and input, techniques.

* Component $M_i$ in the state technique corresponds to internal state $T_i$. 
However, assuming a valid state assignment, an asynchronous machine can be decomposed using the state technique and its operation is as follows:

Assuming the machine to operate in its fundamental mode\(^{12}\), let it be in stable state \( T_i \) under external input \( I_j \); thus, component \( M_i \) is enabled, and since the machine is in a stable-state, it therefore follows that the output of \( M_i = \delta(T_i, I_j) \) is also \( T_i \) — see Fig. 6.3. This state of affairs continues until there is a change in the external inputs — say to \( I_y \) — when the output of \( M_i \) changes to \( \delta(T_i, I_y) \) and, assuming this to be \( T_x \), then the binary code of \( T_x \) is fed back through the external delays to the decoder which, after a short interval, "enables" \( M_x \) and "disables" \( M_i \). However, since the machine is assumed to operate in its fundamental mode, it therefore follows that component \( M_x \), in conjunction with \( I_y \) should issue \( T_x \) as its next-state. Thus, the circuit arrives at stable-state \( T_x \); and, again the circuit continues to be in state \( T_x \) until there is another change in the external inputs when, another cycle of changes takes place.

### 6.3.6 Operation of input decomposition

Again, the synchronous and asynchronous types of sequential systems, when realised using the input technique follow closely the operation of their counterparts in the state technique.

### 6.3.7 Theoretical speed limitations

The speed of operation of an asynchronous sequential system using either the state, or input, techniques is somewhat higher than their synchronous counterparts, when using the same type of hardware for both realisations. Thus, by evaluating the maximum theoretical speed of operation of the asynchronous type, an upper limit can be established.
for the synchronous case.

The state decomposition is used for the following calculation; however, the input technique yields very similar results.

Let the access time of any of the SPMs used (Fig. 6.3) be $t_e$; let the delay through the decoder be $t_d$ and, let the external delay in the feedback loops be $t_{ex}$.

Starting the sequence of changes with a change in the external inputs, then:

i - After an interval of $t_e$, the SPM being enabled changes its outputs to the new next-state.

ii - After a further interval of $t_{ex}$, the new next-state is fed into the decoder.

iii - The decoder responds to the new set of inputs and thus changes its output after an interval $t_d$. The appropriate SPM is then enabled; assuming a fundamental mode type of operation, the machine then arrives at its new stable state.

Thus, the total time interval that elapses, from the instant a change in the external inputs is detected until the circuit arrives at its new stable-state is given by:

$$t_c = t_e + t_{ex} + t_d.$$  

Hence, the maximum frequency of operation is $f_{max} = 1/t_c$.

When multi-transition assignments are used, the effective speed of operation is reduced accordingly - e.g. if $k$ transitions are allowed, then the maximum speed of operation is reduced by a factor of $k$, since it would then take the circuit a period of $(k \cdot t_c)$ to reach some of its intended final stable-states.
6.3.8 Size of individual components and overall storage

I - State Technique

Consider a system with the usual parameters (i.e. \( X, S \) and \( Z \)), then, if such a system is implemented using the state technique, each component would have \( X \) inputs and \( S \) outputs. Hence the size of each component is given by:

\[ m = S \cdot 2^X \text{ Bits.} \]

The \( S \) state-variables are fed into the decoder and, its \( N \) outputs correspond exactly to the number of the internal states of the system, where:

\[ 1 + 2^{S-1} \leq N \leq 2^S. \quad \ldots (6.13) \]

Thus, for a sequential system with \( S \) state-variables, the total amount of storage needed to realise it, using the state technique is given by \( M_D \), where:

\[ (1 + 2^{S-1}) \cdot S \cdot 2^X \leq M_D \leq S \cdot 2^{S+X}. \quad \ldots (6.14) \]

Thus, the upper limit of \( M_D \) in (6.14) is equal to \( M_o \) - the storage requirement of the undecomposed machine.

If the machine under consideration has \( Z \) outputs, then these are obtainable from an SPM of size:

\[ M_Z = Z \cdot 2^{S+X} \text{ Bits.} \quad \ldots (6.15) \]

Equation (6.15) assumes the general Mealy case where, the output functions may be dependent on all the \( X \) external inputs and on all of the \( S \) state-variables of the system.

Thus, the total storage requirement of a sequential system realised using the state technique is less than, or at most equal to, that of the original undecomposed system.

* It should be noted that this machine is assumed to have \( Z = 0 \).
Fig. 6.5 State decomposition of a sequential machine with 4 state variables where only 3 are decoded while the fourth is fed directly to the 8 resulting component machines.
Such a machine can then be realised using at most:

\[ 2^S \text{ SPMs, each of size } S \cdot 2^X \text{ Bits; and, one SPM of size } Z \cdot 2^{S+X} \text{ Bits.} \]

### 6.3.8.2 Input technique

Following similar and equivalent reasoning, a sequential system with the parameters \( X, S \) and \( Z \) can be implemented, using the input technique, with at most:

\[ 2^X \text{ SPMs, each of size } S \cdot 2^S \text{ Bits, and one SPM of size } Z \cdot 2^{S+X} \text{ Bits.} \]

The number of components is given by \( C \) where:

\[ 1 + 2^{X-1} \leq C \leq 2^X \] \( \ldots \) (6.16)

and, \( M_D \) - the total storage requirement of an input-decomposed system with \( Z = 0 \) - is given by:

\[ (1 + 2^{X-1}), S \cdot 2^S \leq M_D \leq S \cdot 2^{S+X} \] \( \ldots \) (6.17)

### 6.3.9 Variation on state/input techniques

Using the state and input techniques to realise large sequential machines, generally lead to large numbers of component machines (i.e. SPMs), each with either \( X \) inputs and \( S \) outputs (with state technique), or \( S \) inputs and \( S \) outputs (with input technique). For reasons of economy in storage and/or availability of exact size SPMs, it might be more convenient to realise such machines from a larger or smaller number of components than has hitherto been suggested.

In the case of the state technique, this is achieved by feeding only a subset of the present-state variables into the decoder; the remaining state-variables are fed into all the SPMs, in parallel with the \( X \) external inputs. Such an arrangement is shown in Fig. 6.5 for a system with \( 4 \) state variables, three of which are fed into the decoder.
whereas, the fourth is fed into all resulting SPMs. Such a realisation leads to only 8 SPMs (i.e. half the number in the basic state technique); the size of each, however, is doubled by the addition of an extra input (the number of outputs S is not affected).

Thus, in effect, when one state-variable is fed directly (or through a delay equivalent to that of the decoder) to the SPMs, then each represents 2 rows, or internal states, in the state-table of the system under consideration. In general, if d state-variables are fed directly to the SPMs, then each represents $2^d$ rows in the state-table.

Clearly, it is also possible to reduce the size of the individual SPMs (at the expense of increasing their number) by feeding a subset of the external inputs, in an appropriate manner, to the decoder.

The same type of variation is possible with the input technique.

6.4 Hybrid Realisations

The state and input realisation techniques can be further developed, through the use of some simple combinational logic modules in conjunction with LSI SPMs, to reduce the total number of, and the size of, some, or all, of the component machines resulting from the basic state and input structures. Some of the component machines resulting from such implementations become so diminished in size that their realisation using combinational logic becomes more economical and less redundant.

The following analyses and methods assume that an output logic 0 on an SPM's output, or on the output of a combinational circuit (connected in parallel with one or more SPM) brings down a logical level 1 output to 0, irrespective of whether the high output results from the SPM or from the combinational circuit. As an example, this can be ensured
through the use of open-collector type logic elements in conjunction with open collector TTL memory modules; this facilitates wire OR type connections.

When other types of logic are used, the same can be ensured either through the use of appropriate interface or through the use of compatible elements (i.e. equivalent to the open-collector one assumed here).

The two cases are dealt with separately below:

6.4.1 The state technique

In order to study algebraically the effects of redundancies, built into the structure of the majority of sequential systems, the following definition is necessary:

Definition 6.1: A row-input-consistent partition $\lambda_i$ is an input-consistent partition $\delta$ taken with respect to a single row $T_i$ in the state-table; in other words, the next-states in that particular row are independent of all the external inputs. Thus for a particular state $T_i$, the next-states $\delta(T_i, I_1), \delta(T_i, I_2), \ldots, \delta(T_i, I_c)$ are all equivalent (Note: $C$ is the number of columns in the state table).

An extension of this definition is when the next-states in a particular row are independent of only a subset of the external inputs—represented as $\lambda_{ij}$ when, $\lambda_i$ is input-consistent with respect to input $I_j$ only.

The procedure of making use of the concept of $\lambda_{ij}$ is best illustrated through the use of an example.

Thus, referring to the machine in Table 6.1, it is clear that:

$$\lambda_1 = \overline{T_1, T_2}; \lambda_2 = \overline{T_3, T_4}; \lambda_3 = \overline{T_1, T_2, T_3, T_4}; \lambda_4 = \overline{T_1, T_3}.$$

Hence, for this machine, it is clear that no row-input-consistent partition
<table>
<thead>
<tr>
<th>$y_1 y_2$</th>
<th>F/S</th>
<th>I_1 I_2 I_3</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>T_1</td>
<td>T_1 T_1 T_2</td>
<td>T_2 T_1 T_2</td>
</tr>
<tr>
<td>0 1</td>
<td>T_2</td>
<td>T_4 T_3 T_3</td>
<td>T_4 T_4 T_3</td>
</tr>
<tr>
<td>1 0</td>
<td>T_3</td>
<td>T_1 T_2 T_3</td>
<td>T_4 T_4 T_3</td>
</tr>
<tr>
<td>1 1</td>
<td>T_4</td>
<td>T_3 T_1 T_1</td>
<td>T_1 T_1 T_3</td>
</tr>
</tbody>
</table>

Table 6.1 Machine A.

Fig. 6.6

(a) State decomposed version of machine in Table 6.1.
(b) State decomposed version of machine A where the compatible components M_2 and M_4 are merged.
(c) The use of state-assignments in reducing the number of component-outputs.
(d) Combinational logic realisation of component M_4.
exists since, none of the above partitions contain only one state.

However, there do exist some row-input-partitions which are independent of one or more of the three external inputs \( I_1, I_2 \) and \( I_3 \). The following is a list of all row-input-partitions with respect to all individual external inputs:

\[
\begin{align*}
\lambda_{11} &= \lambda_{12} = T_1, T_2 ; T_1, T_2 ; T_1, T_2 ; T_1, T_2, \\
\lambda_{13} &= T_1 ; T_2 ; T_1 ; T_2, \\
\lambda_{21} &= \lambda_{22} = T_4 ; T_3 ; T_3 ; T_4, \\
\lambda_{23} &= T_3, T_4 ; T_3, T_4 ; T_3, T_4 ; T_3, T_4, \\
\lambda_{21} &= \lambda_{31} = T_1 ; T_2 ; T_3 ; T_4, \\
\lambda_{32} &= T_1, T_4 ; T_2, T_3 ; T_2, T_3 ; T_1, T_4, \\
\lambda_{33} &= T_1, T_2 ; T_3, T_4 ; T_3, T_4 ; T_1, T_2, \\
\lambda_{41} &= \lambda_{42} = T_1 ; T_1 ; T_1 ; T_3, \\
\lambda_{43} &= T_1, T_3 ; T_1, T_3 ; T_1, T_3 ; T_1, T_3.
\end{align*}
\]

(6.18)

In the above \( \lambda_{i,j} \) is the row-input-partition of row \( T_i \) with respect to input \( I_j \); each block in any of the partitions in (6.18) represents the next-states for that particular row (the external input \( I_d \), for \( d \neq j \), remain constant for all possible binary values of \( I_j \)).

An input independency is confirmed iff each of the blocks in a particular partition has only one state in each block. Thus \( \lambda_{13}, \lambda_{31}, \lambda_{41} \) and \( \lambda_{42} \) all represent input independencies with respect to the external input concerned in that partition. Hence, \( \lambda_{13} \) is independent of \( I_3 \) etc.

Thus, considering the whole list of row-input partitions for the machine in Table 6.1, it is clear that with the state realisation technique of this machine, shown in Fig. 6.6.a, the following redundant variables can be eliminated:

\[ M_1 \text{ is independent of } I_3, \]
\[ M_2 \text{ is independent of } I_1 \text{ and } I_2. \]
$M_3$ is independent of $I_1$, and, $M_4 = \text{""""} I_1 \text{ and } I_2$.

Thus, from this simple procedure of computing the row-input partitions, all redundant input variables are identified and so eliminated from the final state realisation of that particular system.

The second step in the process is to reduce the number of SPMs needed to a minimum. This is done by merging those compatible components in the state realisation technique that result after the elimination of the redundant input variables – i.e. those components that depend on the same subset of external inputs. Thus referring to Fig. 6.6.a, it is clear that $M_2$ and $M_4$ are both dependent on $I_3$ only and thus can be merged into one component through the use of a single 2-input OR-gate as shown in Fig. 6.6.b; the inputs to the OR gate are the two different logic values corresponding to $T_2$ and $T_4$ (as outputs from the decoder) and thus the combined component machine is enabled when either of the internal states $T_2$ or $T_4$ is entered. However, in order to distinguish between these two internal states, one of them is fed as an input to the SPM (which corresponds to $T_2$ and $T_4$) which is then programmed accordingly. Such a realisation utilizes 3 SPMs of exactly equal size and structure.

Another important consideration which may lead to a further reduction in storage requirement is to carry out the state assignment so as to leave the maximum number of next-state variables in a particular row constant:

Consider machine A of Table 6.1; row $T_1$ leads to next-states $T_1$ or $T_2$; row $T_2$ to $T_3$ or $T_4$; $T_3$ to $T_1,T_2,T_3$ or $T_4$; and $T_4$ to $T_1$ or $T_3$. Row $T_3$ has all 4 states as next-states and thus, using only 2 state-
variables, it is impossible that any next-state variable can be left constant; however, all remaining rows, $T_1$, $T_2$, and $T_4$ have only 2 states in their next-state entries and thus, it is possible (by making $T_1$ adjacent to both $T_2$ and $T_3$, and by making $T_3$ adjacent to $T_4$ — i.e. by making the next states in each row, when possible, adjacent, or to differ by the least number of variables) to make: $Y_1$ of components $M_1$ and $M_2$, and $Y_2$ of $M_4$ independent of all inputs.

The result of this is that all three components $M_1$, $M_2$ and $M_4$, have one of their outputs eliminated and thus leading to a substantial reduction in their storage requirements. However, the correct logical values of these "redundant next-state variables" when, their corresponding SPMs are enabled, can be derived as follows:

Assuming the SPMs to be enabled with a logical "0" applied to their M/E, then, as shown in Fig. 6.6.c, all these constant logical values are derived through some simple combinational logic: since $Y_1$ of $M_1$ remains at logical 0 (see Table 6.1 for the state assignment, $y_1$ and $y_2$) whilst $M_1$ is enabled, and is high at logical 1 when $M_1$ is disabled, $Y_1$ is the same as the logical value applied to the M/E of $M_1$ at any instant of time. However, due to the difference in the propagation delays of the different next-state variables that such an implementation may give rise to, the next-state variables obtained in this manner are passed through a delay equivalent to the access time of the SPMs. In Fig. 6.6.c, such a simple delay element consists of two invertors connected in cascade.

Similarly, $Y_1$ of $M_2$ remains at logical 1 whilst $M_2$ is enabled and also, when $M_2$ is disabled; i.e. it remains high permanently. This constant value is obtained merely by leaving the $Y_1$ output from $M_2$ open circuited.

$Y_2$ of $M_4$ is obtained in exactly the same manner as $Y_1$ of $M_1$. 

104
<table>
<thead>
<tr>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_3$</th>
<th>P/S</th>
<th>$I_1$</th>
<th>$I_2$</th>
<th>N/S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A</td>
<td>B</td>
<td>H</td>
<td>G</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>B</td>
<td>D</td>
<td>G</td>
<td>D</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>C</td>
<td>C</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>D</td>
<td>E</td>
<td>H</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>E</td>
<td>E</td>
<td>H</td>
<td>D</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>F</td>
<td>G</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>G</td>
<td>D</td>
<td>G</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>H</td>
<td>B</td>
<td>H</td>
<td>B</td>
</tr>
</tbody>
</table>

Table 6.2 Machine B and its state assignment.

FIG. 6.7 Input realisation of machine B above.
Thus $M_1$, $M_2$ and $M_4$ in Fig. 6.6.c have been appreciably reduced in size.

Some of the component machines resulting from state decomposition and the subsequent elimination of redundant variables may become very diminished in size, compared to the others. In such cases, the reduced components could be much more economically realised using some simple combinational logic in conjunction with the M/E signal to that particular component. As an example of this, component $M_4$ of Fig. 6.6.c could be replaced with the combinational logic shown in Fig. 6.6.d.

6.4.2 The input technique

Sequential systems realised using the input technique can be reduced in size along similar lines to those followed in the case of the state decomposition — the basic tool employed is again discrete logic components.

Since with this realisation technique, each component has only the present-state variables as inputs, and the next-state variables as outputs, it therefore follows that by carefully choosing the internal state assignment, a substantial number of inputs and outputs can be eliminated from the various SPNs and, thus, lead to a corresponding reduction in the storage requirement of the whole machine.

The state assignment which results in most redundant state variables is the one that satisfies all, or as nearly as possible all, least dependency requirements in the various columns of the state table. As an example, consider machine B shown in Table 6.2; the procedure for choosing such a state-assignment is to treat each column in the state-table separately.

Column 00 of Table 6.2 has 4 states in it (B, C, D and E) and thus
requires 2 state-variables to distinguish between them; it therefore follows that only one state-variable may remain constant for those 4 states. Let this variable be $y_1$ - Table 6.2; thus assigning $y_1$ the value 0 for B, C, D and E and, 1 for the remaining 4 states (or vice-versa) satisfies the redundancy requirement of this column.

Column 01 has only 2 states in it (G and H) and, thus, requires only one state variable to distinguish between them; this leaves the other 2 variables constant for both G and H. Since, $y_1$ is already fixed at 1 for both G and H, allowing $y_2$ to have the value 1 for both would satisfy the redundancy requirement of this column.

In the same way, column 11 is assigned leaving one variable to remain constant - in fact this could be satisfied with $y_2$ only. The unfilled states of the 3-variables are then filled so as to distinguish between the different internal states of the machine. One possible state assignment, that satisfies these conditions, is the one given in Table 6.2.

Thus, the only essential dependencies are the following:

- $M_1$ is dependent on $y_2$ and $y_3$,
- $M_2$ is dependent on $y_2$,
- $M_3$ is dependent on $y_1$ and $y_3$,

and, $M_4$ is dependent on $y_1$, $y_2$ and $y_3$.

Thus, an input-decomposed version of machine B in Table 6.2, with all redundancies eliminated from the inputs and outputs of its 4 SPMs is shown in Fig. 6.7. The redundant next-state variables are obtained through some simple combinational logic, in exactly the same way as was done with the state technique.

Again, as with the state technique, the components which become
diminished greatly in size may be realised using some simple combinational logic in conjunction with that component's M/E signal.

6.5 Incorporation of Realisation Techniques in the Loop-Free Structures

When a sequential system is realised using the loop-free structure theory of Hartmanis, the resulting component machines are, generally, of various sizes and structures. This is a serious shortcoming when realising a system using SPMs.

One or both of the realisation techniques outlined in this chapter can be used to further decompose some, or all, of the component machines resulting from the loop-free technique - i.e. by decomposing only those components which are of inconvenient size and structure and thus leading to more uniform components.

The method of applying those techniques is to treat each component machine resulting from the loop-free structure as individual sequential machines* and since, in general, each component machine has a subset of the external inputs and state-variables as inputs, it therefore follows that either of these two subsets of variables can be decoded (as with the basic state and input realisation techniques) and the corresponding outputs from the decoder are then used to enable the appropriate number of sub-components. Interconnecting the resulting components, as in the loop-free structure, leads to a decomposed version of the original machine; however, it is here made up of component machines whose sizes and structures can be varied so as to suit the available SPMs.

To best illustrate this feature of the state and input realisation techniques, a sequential machine with \( X = 4; \ S = 6 \) and \( Z = 6 \), is considered. For the purposes of the example, it is assumed that the

* The output matrix can also be decomposed using a similar procedure.
Fig. 6.8

a- Assumed loop-free structure of a particular machine.

b- Structure in (a) is further decomposed using only 4x2 SPH
machine is decomposable into a loop-free parallel structure with 2 components such that \( S_1 = 2 \) and \( S_2 = 4 \), as shown in Fig. 6.8.a, and its realisation is to be restricted to two types of SPM:

i - 4-input, 2-output SPM - i.e. of 32 bits storage; and,

ii - 6-input, 2-output SPM - i.e. of 128 bits storage.

The original undecomposed machine requires a storage capacity of:

\[
M_0 = (Z + S \cdot 2^{S+X}) = 12 \cdot 2^{10} = 12,288 \text{ Bits;}
\]

whereas, the loop-free parallel structure, of Fig. 6.8.a, requires a storage capacity of:

\[
M_D = 2 \cdot 2^{S+4} + 4 \cdot 2^{4+4} + 6 \cdot 2^{4+6} = 7296 \text{ Bits.}
\]

Clearly then, the sizes of the three components, including the output matrix, are vastly different in size and structure and hence, cannot be realised using any one-size SPM except of a very large capacity which implies, the inclusion of a large degree of redundant storage.

In the following, the above machine is further decomposed so as to be realisable using either of the two SPMs mentioned:

i - Assuming that only 4 x 2 SPMs are available:

Starting from the loop-free structure given in Fig. 6.8.a, the state technique can be used to further decompose \( M_1 \), \( M_2 \) and \( M_3 \) as follows:

Component \( M_1 \) can be decomposed into 4 components, each with at most 4 inputs (i.e. \( X \)) and, 2 outputs \( (y_1 \) and \( y_2 \) - as shown in Fig. 6.8.b. Similarly, \( M_2 \) can be further decomposed into components that can be implemented using a 4 x 2 SPM; in this case, however, since \( M_2 \) has 4 state variables \( (y_3, y_4, y_5 \) and \( y_6) \), each output of the decoder \( D_2 \), is fed into the M/E of two SPMs in parallel and hence, each SPM realises only two next-state variables (at any instant of time, two SPMs in \( M_2 \) are activated so as to give rise to all 4 state-variables). Thus, \( M_2 \) has been decomposed into 32 components, each of 32 bits of storage.

* This assumes a Mealy-type machine, where the outputs are dependent on \( I \) and \( y \).

** Alternatively, the input technique can be used.
Fig. 6.8.c Structure in Fig. 6.8.a is realised using only 6x2 SPHs.
Finally, the output matrix $M_3$ in Fig. 6.8.a can be decomposed into components realisable through the use of $4 \times 2$ SPMs also shown in Fig. 6.8.b. As in the case of $M_2$, 3 SPMs are activated simultaneously since, each SPM in each row of 3 SPMs in this implementation leads to only 2 output functions when activated. Thus, $M_3$ has been decomposed into a total of 192 SPMs, each of 32 bits of storage.

Interconnecting the three components thus leads to a realisation of the original machine, requiring a total of 228 SPMs, each having 4 inputs, and 2 outputs; i.e. the total storage requirement is $228 \times 32 = 7296$ bits = $M_D$.

However, as was shown in Section 6.4, the elimination of redundant input variables from the various components, as would be the case in an actual design example, could lead to a substantial reduction in the storage requirement.

---

Assuming that only $6 \times 2$ SPMs are available:

$M_1$ of the loop-free structure of Fig. 6.8.a can be realised using only one such SPM, as shown in Fig. 6.8.c.

$M_2$ can be realised using input decomposition simply by decoding 2 external inputs $I_1$ and $I_2$ and feeding the 6 remaining variables $I_3$, $I_4$, $y_3$, $y_4$, $y_5$ and $y_6$ into 4 sets of 2 SPMs, each with 6 inputs and 2 outputs. Hence, $M_2$ is realised using 8 SPMs of 128 bits of storage each.

Finally, $M_3$ is realised by decoding all 4 external inputs and feeding its outputs into 16 sets of 3 SPMs; each SPM having 6 inputs and 2 outputs. Thus, $M_3$ is realised using 48 SPMs.

Fig. 6.8.c shows the complete realisation of all three components, the interconnection of which realises the original machine.

A total of 57 SPMs are used, each having 128 bits of storage. Thus, the total storage requirement of such a structure is given by:
\[ 57 \times 128 = 7296 \text{ Bits} = M_D. \]

Again, as in an actual design example, the elimination of redundant variables would lead to a substantial reduction in the storage requirement of the machine.

6.6 Realisations with Single-Output SPMs

The number of outputs from a module have a profound influence in determining the exact number of input variables that a particular component must have. The interdependence of variables, whether output functions or next-state variables, in the majority of sequential systems compels the inclusion of a certain amount of redundant storage in any realisation. As an example, if the following two logical functions:

\[ Y_1 = f \left( X_1, X_2, X_3, X_5, X_7 \right), \]
\[ Y_2 = g \left( X_2, X_4, X_5, X_6, X_7, X_8 \right), \]

are to be realised using a single SPM, then clearly it must have all variables, \( X_1, X_2, \ldots, X_8 \), as inputs to that module. In other words, its storage capacity must be:

\[ M_0 = 2 \cdot 2^8 = 512 \text{ Bits}. \]

However, if \( Y_1 \) and \( Y_2 \) are implemented separately using 2 SPMs, each having a single output, then the total storage is:

\[ M_D = 2^5 + 2^6 = 96 \text{ Bits}. \]

The main reason for the large reduction achieved is the fact that the size of an SPM is halved by reducing its number of inputs by one.

Since a particular function in a sequential system may be dependent on all of the external inputs as well as the present state variables, then in an implementation using a single SPM, all \( S + X \) variables must be fed as inputs to that SPM (if all outputs and next-state functions
Fig. 6.9 Machine of Fig. 6.8 a realised using 4x4 & 4x1 SPM structures.
are to be realised; i.e. in general, if $F_1$ is any of the $S+X$ variables, then:

$$F_1 = \hat{i}(I_1, I_2, \ldots, I_x; Y_1, Y_2, \ldots, Y_s).$$

Thus, in a sequential machine, all next-state variables and all output functions can be realised using individual SPMs, the inputs to which are only those variables on which a particular variable depends. Hence, using $S + 2$ SPMs, each having only one output and $S + X$ address inputs, any sequential system can be realised.

However, some of the next-state variables and some of the output functions of a sequential system may be independent of some of the external inputs and/or the state variables (such redundancies are almost always present in the internal structure of sequential systems). Single-output SPM realisations can eliminate all such redundancies and, in the majority of systems, would lead to considerable savings in the storage requirements.

As an example of the usefulness of such realisations, consider the example given in the previous section; and, assuming that only $4 \times 4$ or $4 \times 1$ SPMs are available, then the use of only $4 \times 4$ modules would lead to the incorporation of a large amount of redundant storage in the realisation of $M_1$ and $M_3$. However, through the use of the $4 \times 1$ module, no redundant storage is necessarily included. Fig. 6.9 shows the same machine realised using both $4 \times 4$ and $4 \times 1$ SPMs. The total number of modules required for this realisation is: 80 SPMs of $4 \times 4$ and, 136 SPMs of $4 \times 1$ structures; i.e. the total storage is $80 \times 64 + 136 \times 16 = 7296$ Bits = $M_D$; (where $M_D$ is the storage requirement of the assumed loop-free structure shown in Fig. 6.8.a).

The use of single output SPMs, in an actual design example, however, can result in substantial reduction in the storage requirement. Thus, single output SPMs can be used effectively, in conjunction with other realisation and decomposition techniques.
6.7 Comments and Conclusions

In this chapter, mainly two realisation techniques, that can be used in conjunction with any sequential system, have been outlined. They are general in the sense that they can be used to realise synchronous, as well as asynchronous, machines (in the latter's case a valid state assignment is assumed).

Both the state and input techniques result in realisations requiring a predetermined number of components, each having a predetermined size and structure. The pattern of connection of all the component machines is regular. Such properties are extremely suitable when designing at the sub-system level.

These techniques enable very large sequential systems to be realised from available modules. They can also be used in conjunction with the loop-free technique to further reduce the storage requirements of the system and to result in more uniform component machines.

In the following chapter, the uses of these techniques in the realisation of asynchronous sequential systems is outlined. It will be shown that modest variations on these basic techniques can result in asynchronous systems being implemented from similar hardware requirements to those of synchronous systems equivalent in size.
CHAPTER SEVEN

A STUDY OF THE STATE ASSIGNMENT OF SEQUENTIAL SYSTEMS

7.1 Introduction

The most critical factor which has to be taken into consideration when designing sequential systems is that of their internal state assignment. On its outcome, depends the resulting complexity of the system, its maximum speed of operation and its correct operation; the latter two points apply, in particular, to the asynchronous case.

The structure theory of Hartmanis achieves efficient state-assignments for a subset of synchronous sequential systems (the various types of decomposition structures that result from applying this theory, represent different state-assignments for the synchronous systems). However, when the system under consideration is of the asynchronous type, this theory offers little or no help in this respect. Other criteria are thus needed.

Numerous state-assignments have been developed that deal with this problem for a given set of constraints, such as the required maximum speed of operation, the amount of hardware required (whether combinational discrete logic or LSI modules), etc..

In this chapter, the state assignment problem of both types of machines is discussed. The usefulness of the realisation techniques, discussed in the previous chapter, in this context is examined. In particular, it is shown that the state and input realisation techniques lend themselves extremely well to the design of asynchronous systems requiring no external delays in their feedback loops.
It is shown that, using inertial delays of duration comparable to the expected variation in the response time of the feedback loops, in conjunction with the state and/or input realisation techniques, results in an asynchronous realisation requiring no special state assignment; in fact a purely arbitrary state assignment can be used without any risk of malfunctioning (due to hazards and races). Such realisations lead to asynchronous implementations requiring no more hardware than an equivalent, in size, synchronous system.

Practical implementations of the state and input techniques, for both the synchronous and asynchronous systems were carried out; their practical speed limitations were then determined. An asynchronous sequential system using inertial delays in its feedback loops and which had a completely arbitrary state assignment was also realised and was tested for correct operation.

7.2 State-Assignment of Synchronous Systems

Correct operation of synchronous sequential systems is not dependent on their internal state assignments, since it is the external timing signal which governs the transitions between the various internal states of the system. Thus, provided sufficient time interval is allowed for the system to arrive at its correct next-state, then correct operation is assured irrespective of what state assignment is used. Indeed, the state assignment of synchronous systems is relevant only in relation to

i - the reduction of the amount of hardware required, and

ii - the size and structure of the SPMs required (when using ISI modules only).

Thus, the internal states of a synchronous system can be assigned in any arbitrary manner without affecting the correct operation of the
system, provided care is taken with regard to the timing and frequency of the external signal.

The Structure Theory of Hartmanis offers an effective procedure for the state-assignment of such systems. As was discussed in Chapter 3, state assignments based on this theory result in reduced variable dependencies and thus lead to reduced hardware requirements. Appendix 1 illustrates the actual state assignment procedures for the various possible resulting structures.

However, not all synchronous systems possess the structural properties necessary for such reduced-dependency-type assignments. Augmentation of those undecomposable systems resulted mainly in undesirable component sizes and structures, which, when implemented using LSI modules, often implied the need for a larger store than that of the undecomposed system.

The realisation techniques outlined in the previous chapter can be used to implement synchronous systems using a predetermined number of components whose size and structure are also predetermined. All these techniques are very similar in terms of structure and speed of operation to the parallel loop-free structure.

7.3 General Outline of Asynchronous State Assignment

The absence of an external timing signal from asynchronous systems lead to the very complex problem of ensuring correct state transitions, irrespective of the order in which the state-variables may change. The presence of races and hazards\textsuperscript{12,47,48} in asynchronous systems leads, in the majority of cases, to the use of a large number of state-variables (as compared with the number of internal states) in order to ensure the correct operation of the system.

Another common restriction is in the changes in the external inputs
to asynchronous systems. Thus assuming the system to operate in the fundamental mode\textsuperscript{12}, normally only one external input is allowed to change at any one time. Multi-input changes are very similar in their effects to multi-state-variable changes, since it cannot be assumed that these changes happen simultaneously.

Thus, a satisfactory state assignment for asynchronous systems must take all these factors into consideration and must result in a practical realisation which is either independent of all these factors or that it specifies the constraints under which the circuit functions correctly.

Unger\textsuperscript{12} deals with the various state assignments in depth; upper bounds on the number of state-variables that would be required for a particular type of assignment; its corresponding maximum speed of operation, etc., are very clearly outlined. A resume of the main asynchronous state-assignments is given in Appendix 2.

7.4 Effects of Various Assignments on Speed and Economy of Realisation

The speed of operation of a particular realisation depends to a large extent on the type of assignment used; e.g. with an STT assignment\textsuperscript{12} (see Appendix 2), the speed of operation is the maximum possible for the type of hardware used in the implementation. When multi-transition assignments are used, the maximum speed of operation is greatly affected since the circuit then takes more than one cycle-time in order to facilitate some of the transitions, and thus reducing the speed of operation accordingly. Some assignments utilize non-critical races to facilitate some state transitions; in such cases the speed of operation is only marginally affected.

The hardware requirements of the various state assignments depends
largely on the number of variables required to carry out a satisfactory assignment for the operational constraints imposed by the design requirements. As an example, an STT assignment requires a relatively large number of state-variables (at least \( N-1 \) state variables for an \( N \)-state machine), whereas a multi-transition assignment requires a relatively small number of variables.

When using ISI modules for such asynchronous realisations, it is therefore of vital importance to utilize the absolutely minimum possible number of state variables without unduly affecting the maximum speed of operation possible with the type of hardware used.

7.5 Effects of the State/Input Techniques on the State-assignment Problem

The most serious shortcoming when realising an asynchronous machine is that of multi-input and multi-state variable changes, since such changes lead directly to races and hazards. Thus most, if not all, asynchronous machine realisations place restrictions upon the numbers of these functions changing simultaneously; indeed, generally only one state-variable and only one external input are allowed to change at any one time.

When realising an asynchronous sequential system using a single SPM, it is imperative, for correct operation, that these constraints are adhered to. However, due to the varying response times of the output functions of the SPMs, the circuit must be allowed the maximum response time before any further external input changes are allowed, i.e. the circuit must be allowed a sufficient time interval to reach a stable-state before any further external inputs are allowed to change. Thus in the vast majority of cases, no multi-variable changes can be guaranteed.

* This is the fundamental mode of operation.
to lead to correct operation due to the variation in the response times of the different functions and due to the variation of these parameters from one module to another.

This section outlines the positive aspects of asynchronous system implementations using the state and input realisation techniques. The two cases are discussed separately below:

I.  **The State Technique**

In this technique, the address to the active SPM changes only with a change in the external inputs, and provided all these changes are completed within a time interval considerably smaller than the response time of the SPMs used in the realisation, then the active SPM would still issue the correct next-state as its output irrespective of whether the changes in the external inputs implied a single or a multiple variable change.

Since, the active SPM changes with every change in the internal state of the system, and since this comes as a result of changes in the external inputs, it therefore follows that the changes in the state-variables are completely separated from changes in the external-inputs to the system.

Thus, the decoder in the state realisation technique - see Section 6.3 - acts to separate the changes in the external inputs from changes in the state-variables in the time domain.

II. **The Input Technique**

In this case, a change in the external inputs causes a change in the outputs of the decoder which then activates the appropriate SPM. With multiple-input changes, provided the period between any two changes does not approach the response time of the SPMs used, then the correct
SPM (i.e. the one intended after all changes in the external inputs) will be activated. After a short time interval, this newly activated SPM issues the appropriate next-state which, after a further short interval, becomes the new present-state of the system.

It is thus clear that provided all changes in the external inputs do not last for a period longer than the cycle time of the system,* and provided that any two consecutive changes are not separated by a time interval greater than the minimum response time of the SPMs, then multiple external input changes can take place without undue risk of malfunctioning.

However, in both these techniques, the SPMs used cannot be guaranteed to have a minimum response time of all its output functions, it therefore follows that what is needed is an implementation which is independent of the possible variations in the parameters of the modules used; and indeed independent of the type of SPM used - i.e. its technology.

In the following sections, two such techniques are illustrated; they both result in realisations of asynchronous sequential systems whose correct operations are technology-independent, and that they lead to asynchronous realisations having arbitrary state-assignments and facilitate multiple external input changes.

* since after disabling the active SPM by the very first change in the external inputs, the present-state of the system will have changed after a time interval decided by the turn-off period of the SPM plus the delays in the feedback loops.
Asynchronous Realisations with Arbitrary Assignments

The need for complicated state assignments in the case of asynchronous sequential systems stems from the imperfections of the physical devices used in the actual implementation of these systems. In fact, if all devices used in an asynchronous realisation can be assumed to be exactly predictable in terms of delays and response times, then the state-assignment problem would be made a very simple and straightforward one, since once these parameters are exactly known, then their variation can be compensated for.

However, with commercially mass-produced devices such testing and classification is practically impossible and thus, when implementing an asynchronous sequential system, exact compensation for parameter variation is impossible, except for individually tested circuits. The need thus arises for complex state-assignments that lead to correct system operation irrespective of the relative variation in the parameters of the various devices.

The state-realisation technique can be adapted so as to lead to an asynchronous sequential system's implementation with an arbitrary state-assignment which, provided appropriate care is taken in designing the system, leads to no malfunctioning.

Referring to the general outline of the state technique - see Fig. 6.3 - a change in the external inputs causes a change in the next-state variables, and if more than one state variable is allowed to change state, then these changes propagate through the active module, the external delays and the decoder at different speeds and the final variation may, in some cases, be large enough to cause a malfunctioning in the operation of the whole system.

The outputs of the decoder are all high (i.e. at logical level "1")
Fig. 7.1 State realisation technique with arbitrary state-assignment
except one (the one corresponding to the input at any particular instant); thus, with multi-state-variable changes, transient short-duration pulses are issued from the outputs of the decoder. It is these transient pulses that cause an erroneous SPM to be momentarily activated and thus lead to erroneous operation. The elimination of these bursts of transitory pulses from the outputs of the decoder, in the state technique, can thus lead to correct operation, irrespective of the number of state-variable changes. Clearly only the very short-duration pulses should be eliminated; whereas, pulses of duration exceeding the possible variation in the response time of the circuit should be allowed through.

One method of filtering out pulses of duration less than a predetermined magnitude is to pass that particular output line through an inertial delay\(^1\) of the appropriate magnitude. This ensures that only pulses of sufficient duration are passed; whereas, transitory pulses are filtered out and can thus cause no malfunctioning of the system.

Thus, in a state-realisation technique, provided all the outputs of the decoder are passed through inertial delays of the appropriate magnitude (which should be sufficient to compensate for the variation in the response time of the SPMs used), then the state-assignment problem can be made redundant in the design of asynchronous systems.

The need for the external delays, in the feedback loops of the system, for equalisation purposes (as with the conventional realisation of asynchronous systems) is eliminated since the inertial delays in this realisation act to equalise the propagation delays in the various paths of the circuit. Fig. 7.1 shows the final state realisation, of an asynchronous system having \(N\) internal states, incorporating \(N\) inertial delays.

---

* This implies an arbitrary state-assignment since any number of state-variable changes directly facilitate any possible transition in any state table.
delays at the outputs of the decoder.

The duration of the inertial delays, $t_i$, should cover the maximum possible variation in the response time of the SPMs as well as the variation in the response time of the decoder.

Thus, referring to Fig. 7.1 and assuming a minimum steady-state period of $t_s$, then the total cycle-time of this system would be:

$$t_c = t_e + t_d + t_i + t_s,$$

where $t_e$ and $t_d$ are the access time of the SPM and the propagation delay of the decoder respectively.

Thus, with this realisation, the external delays used in the design of asynchronous machines have been replaced with $N$ inertial delays of duration directly dependent on the characteristics of the hardware used. Hence, when very fast hardware is used, the duration of the inertial delays are reduced accordingly.

This method of realisation leads to a slight increase in the hardware requirement of the system, in that instead of having $S$ external delays, this realisation requires $N$ inertial delays - where $S \geq \lfloor \log_2 N \rfloor$. However, for most asynchronous implementations, the number of state variables $S$ approaches that of the number of internal states of the system if it does not actually exceed it $12$.

The advantages of such realisations are:

1 - Since the state assignment is done on a completely arbitrary basis, it therefore follows that the number of state-variables required would be the absolute minimum - i.e.,

$$S = \lfloor \log_2 N \rfloor$$

(which is the same as for a synchronous sequential system having the same number of internal states). This leads to a marked reduction in
the storage requirement of asynchronous sequential systems implemented using SPMs; in fact, the reduction is by a logarithmic factor which increases with the size of the system. For example, an 8-state asynchronous machine with \( X = 3 \), and \( Z = 5 \) when implemented using an STT-assignment requires at least \( S = 7 \) state-variables, and thus its storage requirement would be:

\[
M_1 = (7 + 5) \cdot 2^{7+3} = 12 \text{ K Bits ;}
\]

whereas, using the inertial delay method of implementation, the number of state-variables is \( S' = \lceil \log_2 8 \rceil = 3 \). Thus, the storage requirement is then given by:

\[
M_2 = (3 + 5) \cdot 2^{3+3} = 512 \text{ Bits.}
\]

2 - The speed of operation of such implementations is the maximum, or very near the maximum, for the type of hardware used. In other words, it is equivalent to an STT-assignment, where the circuit makes only one transition in arriving at any particular internal state. The only difference between the two is that the external delays in the STT assignments are replaced by inertial delays of approximately the same magnitude.

Thus, in terms of hardware requirements, such realisations enable an asynchronous system to be implemented using very nearly the same amount of hardware as an equivalent synchronous system - which is the minimum possible for a given size machine; and, in terms of speed of operation, the maximum possible for the type of hardware used in the realisation.
Fig. 7.2 Input realisation technique allowing any number of simultaneous external input changes.
7.7 Asynchronous Realisations with Multiple-Input Changes

Another restriction normally imposed on the operation of asynchronous sequential systems is the number of external inputs that could change state simultaneously; again, this restriction is a direct result of the variation in the response times of the physical circuits involved in the design.

Using the idea of inertial delays in conjunction with the input realisation technique any number of external inputs can be allowed to change with no risk of malfunctioning, provided, of course, the changes in the external inputs all take place within a predetermined interval.

Fig. 7.2 shows the modified input technique which achieves this; it is the exact counterpart of the system discussed in the previous section. The outputs of the decoder are fed through inertial delays, of magnitude equivalent to the maximum expected variation in the response time of the external-input source. Again, the function of the inertial delays is to filter out pulses below a predetermined duration; these pulses are the transitory pulses which if they reach the M/E pin of an SPM could cause erroneous operation.

In this case, however, a valid state assignment should be carried out on the internal states of the asynchronous system.

7.8 Multiple-Input-Change, Arbitrary-Assignments

This realisation technique combines both those discussed in the previous two sections so as to result in an asynchronous realisation technique requiring no special state-assignment and, at the same time, allowing any number of external inputs to change simultaneously. The only penalty being paid is a further slight increase in the hardware requirement of the system.
Fig. 7.3
State realisation with an arbitrary state assignment and multiple input changes.

Fig. 7.4
Input realisation resulting in arbitrary state assignment and multiple input changes.

Fig. 7.5
Single-SPN realisation of asynchronous systems with an arbitrary assignment and multiple-input changes.
Such realisations can be achieved in two ways:

I - **Using the State Technique**

Starting with the system developed in Section 7.6, and shown in Fig. 7.1, the external inputs are fed through a second decoder $D_2$ - see Fig. 7.3 - whose outputs are then fed through a set of inertial delays of appropriate durations and, the outputs of these delays are fed into an encoder which reverses the action of the decoder and restores the external inputs to their original number and texture. However, this "decoder-inertial delay-encoder arrangement" achieves the elimination of the transitory changes in the external inputs to the system, through the elimination of short-duration pulses from the outputs of $D_2$.

Thus, a multiple, but not necessarily simultaneous, change in the external inputs of the system in Fig. 7.3 propagates through the outputs of $D_2$, but is filtered out by the inertial delays, and the asynchronous system is thus made accessible only by the final state of the external inputs.

This system thus enables any asynchronous sequential system to be implemented using the least possible number of state-variables for a given $N$ and at the same time having no restrictions on the changes in its external inputs.

II - **Using the Input Technique**

Starting from the system developed in Section 7.7, and shown in Fig. 7.2, the state variables are in this case fed through the "decoder-inertial delays-encoder arrangement" of the appropriate dimensions and characteristics - as shown in Fig. 7.4. The outputs of the encoder are then regarded as the present-state variables of the system.

In this system, the "decoder-inertial delays-encoder arrangement"
eliminates the need for a special state-assignment.

Both systems in I and II are equivalent in as much as that they both achieve an asynchronous machine implementation requiring no special state assignment and at the same time allowing any number of simultaneous changes in the external inputs of the system. However, either of these two systems might be preferred on other grounds, such as the number of SPMs available, their size and structure, etc.

The maximum speed of operation of these systems is marginally affected due to the presence of these extra components in the circuit.

As a final development of the systems outlined previously, the "decoder-inertial delay-encoder arrangement" can be used in single-module asynchronous machine realisations - as shown in Fig. 7.5.

7.9 Practical Implementation and Results

Eight RAMs (Random-Access-Memories), type SN 7489, were used to, practically, implement both the state and input techniques. Each RAM has 4 address, and 4 output, lines; thus, giving rise to 16 words of 4 bits each.

Thus, using the state technique, a machine with up to 4 external inputs and 3 state-variables (since there are only 8 RAMs, where \( \log_2 8 = 3 \)) can be realised; whereas, with the input technique, a machine with 4 state-variables and 3 external inputs can be realised using 8 RAMs.

The system realised simulated both the synchronous and asynchronous-type sequential machines. In the asynchronous case, a provision was incorporated for varying the amount of delays in the feedback loops (the case of zero external delay was also considered).
The 4 external inputs to the systems were obtained from a 7-bit, maximal length, sequence generator.

The programming of the 8 SPMs was done manually. The state of the external inputs, the present-state variables, the next-state variables, and the logical state of the memory, and write, enables, were all monitored by LEDs.

The method used for testing the correct operation of the two cases is as follows:

1 - The synchronous case:

The proper method of testing synchronous sequential systems is to feed special testing sequences which would, when the system is operating correctly, result in known output sequences. Such tests would have had to be exhaustive, and very lengthy. However, even after such elaborate tests, the results cannot be treated as irrefutable and conclusive evidence that the sequential system would operate correctly under all conditions.

The state and input techniques were thus tested as follows:

a - The State Technique

One particular location in each of the 8 SPMs (i.e. corresponding to a certain input combination) were programmed so that the machine when operating cycled from one internal state to another. The pattern of occurrence of a particular state was then monitored and was compared with that expected theoretically.

The speed of operation was increased gradually until the circuit malfunctioned; the maximum speed was found to be about 7.5 MHz which, compared favourably with the maximum possible for the type of hardware used.*

*The total cycle time of the circuit is about 110 n Sec.
Table 7.1 Asynchronous state-table with double transitions; only one state variable changes its value in any transition.

<table>
<thead>
<tr>
<th>Present State</th>
<th>( I_1 ), ( I_2 ), ( I_3 ), ( I_4 )</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0000 0011 0110 0101 1100 1111 1110 1010 1001 1000</td>
<td></td>
</tr>
<tr>
<td>A 000</td>
<td>h A d b d A A d b h A A b b A h</td>
<td></td>
</tr>
<tr>
<td>B 000</td>
<td>g B c B C B b c a g B c a g B B a c</td>
<td></td>
</tr>
<tr>
<td>C 011</td>
<td>d b C C C b f f a C C d a D c e D e d D</td>
<td></td>
</tr>
<tr>
<td>D 010</td>
<td>D c D c D C a D c e D e d D</td>
<td></td>
</tr>
<tr>
<td>E 110</td>
<td>h f d E d E f f d f h E E d E d h</td>
<td></td>
</tr>
<tr>
<td>F 111</td>
<td>F F c e g F F c e e F e e F</td>
<td></td>
</tr>
<tr>
<td>G 101</td>
<td>G b h b G G f G b h f G G h f G</td>
<td></td>
</tr>
<tr>
<td>H 100</td>
<td>H a H e e e e e a H H e e g g H a H</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 7.6 a) Theoretical pattern of occurrence of state "F" for the machine in Table 7.1; the numbers indicating the "Mark-Space ratio.

b) Observed pattern of state "F", at 2.00 MHz.
b - The Input Technique

This was tested in a similar manner to the synchronous state technique in (a); the external inputs were fixed at a certain combination and the corresponding component machine was programmed so that it cycled from one state to another. The pattern of occurrence, of a particular state, was observed as the speed of operation was increased. The maximum speed in this case was similar to that of the state technique; it also varied slightly from one SPM to another, due to the variation in the access time of the different modules.

Since, in practice, the changes in the external inputs are restricted so as not to occur during the clock pulse, this simple test procedure was considered an effective method of testing the system.

II - The asynchronous case:

Both the state and input techniques were tested by programming the complete state tables of asynchronous sequential systems, with valid state assignments (STT, single-variable changes, and multi-variable changes were tested). In all cases, no restrictions were placed on the changes in the external inputs.

The magnitudes of the external delays in the feedback loops were gradually reduced until the maximum possible speed of operation was reached. This, for the type of hardware used in the implementation (i.e. TTL), was found to be 8.3 MHz when STT assignments were used, which corresponded very closely to that expected theoretically.

The pattern of occurrence, of a particular state, was observed and compared with that expected theoretically. As an example of these realisations, the system in Table 7.1 was realised using the state technique; multi-transitions are incorporated for the state assignment.
Table 7.2 An asynchronous machine with an arbitrary state assignment realised using the State technique in conjunction with inertial delays.
of this machine. Figs. 7.6 (a) and (b) show the expected and observed
patterns of occurrence of state F (the frequency of external input
changes being 2.00 MHz). The circuit operated correctly for frequencies
up to 4.00 MHz which is nearly half that of the STT type assignments
(clearly the two patterns correspond exactly to one another).

III - Arbitrary-Assignment, Asynchronous Realisations:

Asynchronous realisations with arbitrary state assignments were
realised using the state technique by short-circuiting the external
delays in the previous implementation and feeding the eight outputs of
the decoder, in the state technique through individual inertial delays
of 60 nS duration. This figure was arrived at as follows:

i - The possible variation in the access time of the SPMS used was about 35 nS.

ii - The possible variation in the response time of the decoder was allowed 15 nS.

Thus, total possible variation in the response time of the next-state
variables resulting from the different SPMS would be approximately 50 nS.
Thus, designing inertial delays with 60 nS was thought sufficient for the
purpose of eliminating transitory outputs from the decoder - see Fig. 7.1.
However, when using faster types of logic for such implementations, such
as Emitter-Coupled Logic, much shorter inertial delays would be required.

In the implementation carried out here, no compensation for multi-
external input changes was carried out; thus, only single input changes
were allowed. For this purpose, a synchronous 4-bit binary counter (type
SN 74191) was used and, its outputs converted to a Grey Code counter
(which changes only one output function with every clock pulse).

The asynchronous machine, given in Table 7.2, was programmed and
(a) OBSERVATION
Pattern of occurrence of state "F".

Pattern of state variable $y_1$.

Pattern of $y_2$.

Pattern of $y_3$.

(b) THEORETICAL
Pattern of occurrence of state F.

Pattern of state variable $y_1$.

Pattern of $y_2$.

Pattern of $y_3$.

Fig. 7.7 Observed and expected patterns of state "F" and its 3 state variables $y_1 y_2 y_3$. 
several tests were carried out by feeding single input changes and observing the internal state of the machine before and after these changes; all such tests proved that the system was functioning as expected. The main test, however, was carried out by feeding the complete Grey Code sequence (from 0000 to 1000) to the system, when it was started in a particular steady state "E" under input "0000", and observing the pattern of occurrence of state "F" as the system cycled from one column to the next. The state table was designed in such a way that it takes three full Grey Code sequences (i.e. 48 external input changes) to complete a full cycle of state changes.

Both the expected and observed sequence of occurrence of state "F" are given in Figs. 7.7 (b) and (a). It is clear that the mark-space ratios of both correspond exactly. Also shown, in Fig. 7.7.b is the full cycle of the states of all three state-variables \( y_1, y_2 \) and \( y_3 \) which constitute the coding of the eight internal states \( (A, \ldots, H) \); from these three graphs, the exact internal state transitions can be obtained, and this is given in the letter markings given underneath the three graphs. It can be easily verified, from the state-table in conjunction with the external input changes, that the transitions in Fig. 7.7.b correspond exactly to those expected from the state table.

The maximum theoretical speed of operation of this implementation was given by:

\[
\frac{1}{(t_i + t_e + t_d + t_s)}
\]

where, \( t_i \) = amount of inertial delay = 60 nS.  
\( t_e \) = access time of SPMs = 30 nS.  
\( t_d \) = propagation delay of decoder = 20 nS.  
and, \( t_s \) = minimum steady state period, before another change in the external inputs is assumed 20 nS.
Thus, maximum theoretical frequency of operation is \( \frac{1}{130} \approx 7.7 \) MHz.

The maximum frequency of operation obtained experimentally was about 5.00 MHz; however, this was not the maximum obtainable from the circuit, but the maximum obtainable from the pulse generator used at the time of the experiment. However, judging from the waveforms obtained at 5.00 MHz, it was felt that the theoretical and practical limits on the frequency of operation were closely related.

Similarly multi-input changes could be compensated for.

Thus, an asynchronous system with a completely arbitrary state assignment and, multi-input changes is implemented using the least amount of hardware possible.

7.10 Comments and Conclusions

In this chapter, a novel realisation of asynchronous sequential systems, requiring no special state assignment, and one which could allow for multi-input changes, is illustrated. The main feature of these realisations is the incorporation of inertial delays in the feedback loops of the system, which eliminate all transitory pulses in the feedback loops (which are a result of the variation in the characteristics of physical components used in their realisations).

As a consequence of this: i) the amount of hardware required to realise an asynchronous sequential system is appreciably reduced, since only the minimum number of feedback loops is needed (i.e. \( \lfloor \log_2 N \rfloor \) ) with such realisations; whereas, a minimum of \( N - 1 \) is needed for any comparable (in speed) conventional assignment. ii) The complexity of making a valid state assignment for asynchronous systems (which increases with the size of the machine) is reduced to a minimum; hence, for large systems
only a predetermined and predesigned inertial delay is required. Thus, the presence of inertial delays is the only feature that distinguishes these asynchronous realisations from their synchronous counterparts.
The advent of MSI/LSI technology has rendered the design and implementation methods of both the sequential and combinational switching systems almost totally redundant. These were developed mainly for designing such systems using discrete logic components (such as the Quine-McCluskey minimization method). This also implied a significant modification in the realisation and implementation methods of these systems. One such example is the Structure Theory of Hartmanis.

The criteria, on which the design of sequential switching systems using MSI/LSI modules are judged, are completely different from those using discrete logic components: in the latter case, the total number of discrete gates used in the realisation, or the total number of gate inputs are the two criteria; whereas, in the former case the only meaningful criterion that could be used in comparing the size and structure of different realisations is the total number of storage locations required by each — see Chapter 1. The two main fields in this thesis are discussed separately below:

A - Hartmanis's Structure Theory

When using discrete gates for the realisation of sequential systems, Hartmanis's Structure Theory resulted in appreciable reduction in the hardware requirements of a large proportion of systems. However, when using LSI memory modules, and due to the unavailability of some of the module sizes that could be required to realise some, or all, of the component machines that result from the application of this theory, the total storage requirements of some decomposed structures exceed those of the
undecomposed original machines. Thus the main object for which a decomposition is carried out is sometimes defeated by the need to utilize larger modules than those required theoretically.

Further to this, some decomposition structures are more suited for ISI realisations than others; as an example, parallel structures, generally, result in least storage as well as in most uniform component machines. Serial and Composite structures, however, result in a large degree of variation between the sizes of the predecessor and successor components, and this renders them less suited for ISI realisations than parallel structures.

When redundancy is added in order to make a machine decompose, the amount of added redundancy is critical if economical realisations are to be obtained; the main conditions are summarized as follows:

I - Parallel Structures

Absolutely minimum storage is obtained when each component machine in the decomposed structure has only one state-variable; as the size of the different component machines is increased, in order to reduce their number, the storage requirement of the structure is increased (Sec. 4.3).

The amount of redundancy which could be introduced into a sequential system and still results in an economical parallel realisation depends on:

i - The uniformity of the resulting components: the more uniform the different components are, the smaller is the storage requirement.

ii - The size and structure of the resulting components: the smaller the component machines are, the larger is the amount of redundancy that could be introduced into the system and still leads to an economical realisation.

Provided these two conditions are satisfied (or nearly so), the amount of redundancy which could be introduced into a system resulting
in a parallel structure can be quite considerable (Sections 4.3 and 4.6). Also, with such structures, sequential systems with almost any size can be augmented, in terms of the number of state-variables, and still result in economical realisations.

The closed partitions most suited for such structures should be as uniform as possible, and have the least degree of interdependence (Section 5.3). Thus, if there is a choice between two or more sets of closed partitions, all of which result in parallel structures, then the one that satisfies the conditions stated above is the one likely to lead to least storage. If one of these sets is more uniform than the others, then provided:

\[ S_{Du} \leq S_{Di} + \alpha_D - 2 \]

(where \( S_{Du} \), \( S_{Di} \) and \( \alpha_D \) are as defined in Section 5.3.2), the uniform one leads to least storage, in spite of having the larger number of state-variables - where \( \alpha_D \geq 2 \).

II - Serial Structures

As in the parallel case, minimum storage is obtained when each component machine has only one state variable (i.e., based on a 2-block partition). However, unlike the parallel case, the sizes and structures of the different components vary greatly (from the first predecessor component to the final successor one). These properties make such structures very unsuited for LSI realisations since, in order to implement such widely differing components, large amounts of redundant storage may have to be incorporated due to the unavailability of every SPM size and structure.

The formula, obtained in Section 4.4, and governing the best relative sizes (in terms of the storage capacity) of any two successive components
in a serial structure, namely:

\[ 1 + S_r \ln 2 = 2^{r+1} \]  

... (8.1)

can lead to component machines of uniform or near uniform sizes (at the cost of an increase in the total amount of storage required to realise the whole structure); thus, leading to structures more suited for LSI implementations than single-state-variable components.

This formula illustrates how vitally important it is to increase the sizes of the first predecessor components in a serial structure and to reduce the sizes of the final successor ones in order to obtain the maximum uniformity in such structures (Section 4.6). It also enables a sequential system to be decomposed into a serial structure of relatively small number of uniform, or near uniform, sizes.

The structure of the different components varies from the first predecessor to the final successor for the obvious reason that the numbers of inputs to the final successor components are larger than those to the first predecessor ones. This again places serial structures at a severe disadvantage when dealing with LSI modules (as compared with the parallel case).

When redundancy is introduced into a system in order to make it decompose into a serial structure, the amount of added redundancy (whether stemming from state-splitting or from the use of redundant and non-uniform closed partitions) has a profound effect on the storage requirement of the resulting structure. This could be classified as:

1. If the component machines are to have only one state variable each, then if \( S_D = S + 1 \) (Section 5.4), then \( S \geq 4 \); and if \( S_D = S + 2 \), then \( S \geq 8 \) etc. Clearly then, for the majority of sequential systems, a reasonable degree of redundancy can be tolerated provided single-state variable components result. The closed partitions that give rise to
such structures should be related as follows:

\[ D_i = \left\lceil \log_2 \left( \frac{\#(\pi_i)}{\#(\pi_{i-1})} \right) \right\rceil \], and that, for \( i = 1, \ldots, n \), \( D_i = D_{i-1} + 1 \);

where \( \pi_1 > \pi_2 > \cdots > \pi_{i-1} > \pi_i > \cdots > \pi_n = \pi(0) \).

b - Non-Uniform closed partitions

When there is a choice between two or more sets of closed partitions then, with 2-component structures, if:

\[ D_i + d_i \geq D_j + d_j + 2, \]

then the closed partition \( \pi_j \) results in least storage (for all practical realisations). If the machine decomposes into more than 2 serial components, then the important factor, for least number of state variables and, in the majority of cases, for least storage requirement, is the inter-partition dispersion ratio \( \gamma \) (Chapter 5) which dictates the number of state-variables of the intermediate component machines, and thus, of the final successor components (which mainly decide the storage requirements of serial structures).

ii - If the component machines are to have their state-variables distributed in accordance with equation (8.1), then the amount of redundancy that could be incorporated into a sequential system, such that \( M_D \leq M_o \), is appreciably reduced. As an example, only with sequential machines having \( S \geq 7 \) can \( S_D \) be \( \geq S + 1 \); and only with extremely large systems (i.e. \( S \geq 21 \)) can \( S_D \) be \( \geq S + 2 \). However, the advantage of such structures is the uniformity of the resulting component sizes.

As a general rule with structures conforming reasonably well to equation (8.1); if a machine decomposes into 3 or more components, the final successor ones that follow the first 2 should not, where possible (for least storage requirement and most uniformity in component sizes), have more than one state-variable each. This applies to all practically
realisable systems (only when $S \geq 18$ may this not be the case).

Thus, unlike the parallel case, only a limited amount of redundancy can be tolerated if the storage requirement of the resulting structures are not to exceed that of the original undecomposed machine (the exact amount being critically dependent on the resulting component sizes and the relative distribution of the state-variables of the augmented machine).

III - Composite Structures

The analyses of composite structures lie somewhere between the serial and parallel cases. The main desired feature for reduced storage requirement is that the parallel components should be as nearly as possible equal in size, and that the serial successor components should be as small as possible. Only when one of the parallel component machines is much larger than all the other parallel components may that particular parallel component be made smaller at the expense of increasing the size of the serial component (i.e. making the serial successor component have more than one state variable may result in smaller storage requirement - see Sub-Section 4.4.3).

Chapter 5 also analyses the effects of input and output consistent partitions on the choice of closed partitions. The effects of output consistencies is similar with all types of structures since, in all cases the output functions can be realised through a separate output matrix. The effects of input-consistencies can be summarised as follows:

i - With parallel structures comprising a large number of components, the closed partitions would be relatively large and thus have a high probability of being input-consistent. However, the effect on the storage requirement would then be very small.

When the structure comprises a small number of components, the closed
partitions on which the individual components are based would be small and hence have a small probability of being input-consistent. However, the effect of an input consistency on the overall storage would be large.

ii - With serial structures the effect of an input consistency in one of the final successor components, on the storage requirement is large, whereas that of an input consistency in one of the predecessor components is small except when the sizes of all components are uniform or near uniform. Also, unlike the parallel case, the smaller the number of components in a serial structure is, the higher the probability that some would be input-consistent.

The procedure developed in Chapter 5 can be used to simplify the choice of a particular loop-free structure by examining all possible structures in a logical manner, classifying all relevant structures and then eliminating all redundant, or costly, closed partitions according to the type of structure that these partitions form. The final choice is made on the bases of component uniformity and overall storage requirement.

B - Input and State Realisation Techniques

These two realisation techniques offer an alternative method of implementing such sequential switching systems which are either undecomposable economically,* or that they decompose into components of sizes and structures, if implemented using LSI modules, a large degree of redundant storage must be incorporated.

The input and state techniques offer a method of realising sequential machines which does not depend on the internal structure of the particular machine under consideration. They lead to components of predictable, and

* Using the loop-free structure theory.
predetermined, sizes and structures, and which can be varied so as to suit the ISI modules currently available; they depend on no complicated theory but are general (i.e. applicable to both synchronous and asynchronous types) and can be applied in a straightforward manner leading to component machines with a uniform pattern of interconnections.

The total storage requirement of such structures is less than, or at most equal to that of the original undecomposed machine. The elimination of redundancies present in the state table leads to substantial reductions in the storage requirement.

The input and state techniques can also be applied to any non-uniform component machine resulting from the application of the loop-free structure theory (discussed earlier in this thesis) thus, leading to components of more uniform structures.

However, the most important application of either of these techniques is in the realisation of asynchronous sequential systems. The method of designing asynchronous machines, outlined in Chapter 7, enables any such system to be realised using the minimum of hardware. The use of inertial delays in the feedback loops eliminates all transitory pulses from the outputs of the decoder. Again, this enables any asynchronous sequential machine to be implemented using the minimum of hardware and from SPMs of equal size and structure. This realisation technique also uses the minimally reduced state table as the machine to be decomposed and that no augmentation is necessary in order to carry out this decomposition.

The speed of operation of such realisations as was calculated theoretically, and demonstrated practically, was approximately equivalent to an STT assignment; the maximum speed of operation in such realisations was entirely dependent on the type of hardware used.
The savings in terms of storage that such asynchronous realisations achieve are substantial since, in order to realise a machine with $N$ internal states using an STT assignment, by conventional asynchronous methods, at least $(N - 1)$ state-variables are required; whereas, using this method (inertial delays in conjunction with state and/or input realisation techniques) only the minimum number of state-variables (i.e. $\lceil \log_2 N \rceil$) will be required.

The possible modifications that were outlined in Chapter 7 all lead to further substantial reductions in terms of storage and/or to a substantial relaxation in the conditions under which asynchronous systems may operate (such as the number of external inputs that may vary at any one time).

As a final summary, this thesis discusses the structure theory due to Hartmanis in detail pointing out the shortcomings that this theory may lead to when implementing sequential machines using currently available SPMs. The interpartition relationships that best lead to the various structures are also dealt with in depth.

However, for the reasons that: i - not all sequential systems are decomposable in their reduced form and, ii - the resulting structures may not be suitable for LSI implementations, alternative realisation techniques, which could be applied to both the synchronous and the asynchronous types of machines, were developed. They generally result in equi-sized components with a uniform structure and, which could be easily adapted to realise, economically, any asynchronous sequential system.

On the whole, this thesis presents a complete study in the design and implementation of sequential switching systems.
Table A1 State-assignment based on $\pi_1$, $\pi_2$ and $\pi_3/\tau$.

<table>
<thead>
<tr>
<th>State Code</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1(\pi_1)$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$y_2(\pi_2)$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$y_3(\pi_3)$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table A2 The complete state assignment based on $\pi_1$ and $\pi_2$.

<table>
<thead>
<tr>
<th>State Code</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1(\pi_2)$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$y_2(\pi_2)$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$y_3(\pi_2)$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$y_4(\pi_3)$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table A3 A sequential machine having the $M_m$ pairs given on P 144.

<table>
<thead>
<tr>
<th>Present State</th>
<th>$I_1$</th>
<th>$I_2$</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>A</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>01</td>
<td>C</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>11</td>
<td>D</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>10</td>
<td>B</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Present State</th>
<th>$I_1$</th>
<th>$I_2$</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>A</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>01</td>
<td>C</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>11</td>
<td>D</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>10</td>
<td>B</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>

Table A3 A sequential machine having the $M_m$ pairs given on P 144.
SYNCHRONOUS STATE ASSIGNMENT

A Using Closed Partitions:

The Structure Theory of Sequential Systems, due to Hartmanis, can be used to assign the internal states of synchronous systems so as to result in reduced variable dependency. The different cases are illustrated below:

I - Parallel

When a machine possesses two, or more, closed partitions such that:

\[ \pi_1 \cdot \pi_2 \cdot \ldots \cdot \pi_n = \pi(0) \]

then this machine can be decomposed into n components connected in parallel. As an example, assume that a particular machine possesses 3 closed partitions which satisfy the above condition. Let this machine have 8 states, 1 - 8, and:

\[ \pi_1 = \{1, 2, 3, 4\}; \quad \{5, 6, 7, 8\} \]

\[ \pi_2 = \{1, 2, 5, 6\}; \quad \{3, 4, 7, 8\} \]

\[ \pi_3 = \{1, 3, 5, 7\}; \quad \{2, 4, 6, 8\} \]

It therefore follows that this machine can be decomposed into 3 components, each with a single variable; the internal state-assignment for such a realisation is carried out in accordance with the closed partitions. One such assignment is given in Table A1.

II - Serial

When a machine possesses one, or more, closed partitions such that:

\[ \pi_1 \supseteq \pi_2 \supseteq \ldots \supseteq \pi_n \], where \( \pi_n = \pi(0) \),

then this machine can be decomposed into \( n + 1 \) components; the first predecessor machine is then based on \( \pi_1 \); the second on a non-closed
partition \( T_2 \) such that: \( \pi_1 \cdot T_2 = T_2 \); etc.

As an example consider a machine having 2 closed partitions:

\[
\pi_1 = 1,4; 2,5; 3,13; 6,15; 7,12; 8,11; 9,10; 14,16,
\]

\[
\pi_2 = 1,4,7,9,10,12,14,16; 2,3,5,6,8,11,13,15.
\]

Clearly \( \pi_2 > \pi_1 \). Hence, three possible serial structures can be envisaged:

i - Based on \( \pi_1 \) only; ii - based on \( \pi_2 \) only; and, iii - based on \( \pi_1 \) and \( \pi_2 \). However, as was demonstrated in Chapter 4, the third case leads to the most economical realisation since it facilitates a structure comprising 3 components: the first predecessor is then based on \( \pi_2 \) and has only one variable \( y_1 \); the intermediate component is based on \( T_2 \), where:

\[
\pi_2 \cdot T_2 = \pi_1,
\]

and has thus: \( \log_2 \left( \frac{\#(\pi_1)}{\#(\pi_2)} \right) = 2 \) state-variables \( y_2 \) and \( y_3 \). The final successor component is then based on \( T_3 \), where:

\[
\pi_1 \cdot T_3 = \pi(0) \quad \text{and has thus only one state variable} \quad y_4.
\]

The complete state assignment for such a structure is given in Table A 2 - where \( T_2 \) and \( T_3 \) are taken as:

\[
T_2 = 1,2,4,5; 3,7,12,13; 6,9,10,15; 8,11,14,16;
\]

and, \( T_3 = 1,2,3,6,7,8,9,14; 4,5,10,11,12,13,16 \).

III - Composite

When a machine possesses two, or more, closed partitions such that:

\[
\pi_1 \cdot \pi_2 \cdots \cdot \pi_n = \pi_p, \quad \text{where} \quad \pi_p \triangleright \pi(0),
\]

then this machine can be decomposed into \( n \) parallel components whose effective closed partition is \( \pi_p \). These are then succeeded by a serial component based on a non-closed partition \( \tau \) such that:

\[
\pi_p \cdot \tau = \pi(0).
\]
As an example consider a machine possessing two closed partition:

\[ \pi_1 = 1,2,3,4; 5,6,7,8 \]

and,

\[ \pi_2 = 1,2,5,6; 3,4,7,8 \]

Hence, \[ \pi_1 \cdot \pi_2 = 1,2; 3,4; 5,6; 7,8 = \pi_p \neq \pi(0) \]

One possible \( \tau \) is: \( \tau = 1,3,5,7; 2,4,6,8 \).

This machine is decomposed into 2 parallel components based on \( \pi_1 \) and \( \pi_2 \) respectively, and each having one variable (\( y_1 \) and \( y_2 \) respectively). The successor is based on \( \tau \) and also has one variable \( y_3 \). One possible state assignment can be taken as Table A1.

The case where a common component is "factored" out and then succeeded by two or more parallel components can be considered on similar lines.

**B Using Mm and p.p. pairs**

The use of Mm and p.p. pairs in reducing variable dependency in synchronous systems is well known \(^7,8\). However, the exact method of applying these concepts is not very well illustrated. The example below gives an exact and simple illustration on their use in state assignments.

The machine given in Table A3 has the following p.p.s:

\[
\begin{align*}
(\pi_1, \pi'_1) &= (A,B; C; B; E) \quad (A,C; B; D; E), \\
(\pi_2, \pi'_2) &= (A,C; B; D; E) \quad (A,B; B; C; E), \\
(\pi_3, \pi'_3) &= (A,B; B; C; E) \quad (A,B; C; B; E), \\
(\pi_4, \pi'_4) &= (A,B,C; B; E) \quad (A,C,B; B; E), \\
(\pi_5, \pi'_5) &= (A,B,D; C; E) \quad (A,B,C; D; E), \\
(\pi_6, \pi'_6) &= (A,B,D; B; E) \quad (A,B,D; C; E), \\
(\pi_7, \pi'_7) &= (A,C,B; B; E) \quad (A,B,D; C; E), \\
(\pi_8, \pi'_8) &= (A,B,D; C; E) \quad (A,B,C; D; E), \\
(\pi_9, \pi'_9) &= (A,B,C; B; E) \quad (A,C,B; B; E), \\
(\pi_{10}, \pi'_{10}) &= (A,B,C; B; E) \quad (A,B,C; B; E).
\end{align*}
\]
From this list, it is clear that \( \Pi_{10} \) is the only closed partition the machine possesses.

Studying the p.p. table reveals that since \( \Pi_8 \) and \( \Pi_8' \) have only two blocks each, and because \( \Pi_8 = M(\Pi_8') \) is \( \geq \Pi_6' \); and, moreover \( M(\Pi_6) \) has only 2 blocks, it therefore follows that one possible starting point in making a state assignment according to the p.p.s is to make \( \tau_1 = \Pi_8' \) and \( \tau_2 = \Pi_8 \).

Thus, \( \tau_1 \cdot \tau_2 = \overline{A,B}; \overline{C}; D; E \)

\[
(M(\tau_1), \tau_1) = (A,B,D; C,E) \quad (A,B,C; D,E),
\]

and, \( (M(\tau_2), \tau_2) = (A,C,D; B,E) \quad (A,B,D; C,E).\)

One possible \( \tau_3 \) is: \( \tau_3 = (A,C,B; D,E) \) and is \( \geq \Pi_9' \).

Thus, \( (M(\tau_3), \tau_3) = (A,B,C; D,E) \quad (A,C,B; D,E)\)

This choice for \( \tau_3 \) gives: \( \tau_1 \cdot \tau_3 = \overline{A,C}; \overline{B}; D; E; \)

and, \( \tau_1 \cdot \tau_2 = \overline{A,B}; \overline{C}; D; E \)

Hence:

\[
M(\tau_1) \geq \tau_2 ;
\]

\[
M(\tau_2) \geq \tau_3 ;
\]

and, \( M(\tau_3) \geq \tau_1 \cdot \tau_2 \) or \( \tau_1 \cdot \tau_3 \).

An assignment based on \( \tau_1, \tau_2 \) and \( \tau_3 \) (where \( \tau_1 \cdot \tau_2 \cdot \tau_3 = \Pi(0) \)) results in the following functional dependencies:

\[
Y_1 = f_1(y_2, x_1, x_2)
\]
\[
Y_2 = f_2(y_3, x_1, x_2)
\]

and \( Y_3 = f_3(y_1, y_2, x_1, x_2) \) or, \( f'_3(y_1, y_3, x_1, x_2) \);

where, \( Y \) and \( y \) represent the next and present state variables respectively.
APPENDIX - 2

ASYNCHRONOUS STATE ASSIGNMENTS

There are several methods for encoding the internal states of asynchronous machines, and several techniques employed in reducing the number of state-variables required; however, these techniques are generally employed at the expense of one or more performance parameters, such as the speed of operation. Unger \(^{12}\) deals with this problem extensively; however, a brief resume* of the main state assignment techniques used in conjunction with asynchronous systems is given below:

1 - **Single Transition-Time Assignments (STT)**

With such assignments, the machine reaches a stable state after a single transition time; thus attaining the maximum speed of operation for the type of hardware used. Two types of STT assignments exist:

i) - **The one-shot assignment**

Only one state-variable changes during any one transition. Thus, a large number of variables are required in order to facilitate all possible transitions. A machine with \(N\) internal states requires at least \(N-1\) state variables.

ii) - **Unicode STT (USTT)**

Each row represents a distinct state. Transitions between the internal states of the machine may occur by means of non-critical races among all variables distinguishing the initial and final states. Such assignments require a considerably smaller number of variables than the STT ones. However, such assignments affect the maximum speed of operation since some, if not all, transitions take a longer interval to be completed.
2 - Multi Transition Assignments

With such assignments, the sequential machine goes through one, or more, "transitory states" in arriving at some of its destined stable states. This reduces greatly the number of variables required but seriously reduces the maximum possible speed of operation since the machine in this case takes up to $k$ times the period it would take with STT assignments (where $k$ is the maximum allowable number of transitory states in any change of state).

3 - Connected Row Sets

A row set $R_i$ is defined as a set of $y$-states assigned to row $i$ of the flow table. No $y$-state occurs in more than one row set. When the system is in a $y$-state of $R_i$ and the input is $I_j$, the output is then $\lambda(i, I_j)$, and if $\delta(i, I_j) = k$, then the $y$-excitations will be such as to lead to a state of $R_k$ either directly or via a series of transitions through $y$-states in $R_i$. The number of state-variables necessary for a given table cannot easily be determined. However, universal row assignments that are valid for any $N$-row table can be constructed\textsuperscript{12}.

The basic idea is to construct connected row sets that are intermeshed in the sense that, given any two states $i$ and $j$, $R_i$ is adjacent to $R_j$. It then follows that regardless of what transitions are called for by the flow table, they can always be carried out by a sequence of single-step (i.e. non-critical race) transitions within the row set of the initial state, followed by a final single-step transition across the "border" into the row set of the destination state.

4 - Shared Row Assignments

A single $y$-state is assigned to each row, and other $y$-states are used as necessary to bridge transitions between rows whose $y$-states are
not adjacent. A "bridging" state which appears as a supplementary row of the flow matrix (i.e. the enlarged flow table), may be used in different columns to bridge transitions between different pairs of rows; hence, the term shared rows.
REFERENCES


155


