This is to certify that the

dissertation entitled

THE MODELING AND SYTHESIS
OF ASYNCHRONOUS SEQUENTIAL
LOGIC CIRCUITS

presented by

Jun-Woo Kang

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Electrical Engineering

David Fisher
Major professor

Date 2/26/93

THE MODELING AND SYTHESIS OF

ASYNCHRONOUS SEQUENTIAL LOGIC CIRCUITS

By

Jun-Woo Kang

A DISSERTATION

submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Electrical Engineering

1993

ABSTRACT

THE MODELING AND SYNTHESIS OF ASYNCHRONOUS SEQUENTIAL LOGIC CIRCUITS

By

Jun-Woo Kang

The modeling and synthesis procedure described efficiently generates asynchronous sequential logic circuits (ASLCs). From the high-level design specification, the functional behavior of inputs is analyzed to simplify the overall synthesis process. The analytical model developed delineates the inputs into three classes: **mode inputs, level inputs,** and **edge inputs.** By introducing transition variables for edge inputs, a set of equations referred to as **dynamic output equations** (DOEs) is generated, which describes the functional behavior of ASLCs in more compact form than the traditional one. The **state grouping** process based on the functional behavior of level inputs and edge inputs generates a state table without any difficulties from topological complexities inherent in the traditional procedure. The functionality of mode inputs facilitates the process of decomposing complex logic functions into smaller ones which can be more easily synthesized. Based on the characteristics of the state table, a race-free state assignment problem is formulated as a mapping of a **bipartite graph** into an n-cube. The race-free state assignment algorithm

developed features a **pattern matching** technique which predicts races and eliminates enumerative searches to get the near minimum number of state variables. The described procedure is well suited for the synthesis of large-scale ASLCs that have many data inputs but only a small number of control inputs. Moreover, it provides an efficient implementation with ASLCs for the sequential logic function which has been used with clocked sequential logic circuits. Therefore, circuit designers will have more choices to obtain better circuits using ASLCs than they could achieve using their clocked counterparts in certain sequential logic applications.

Dedicated to my parents, wife, and daughter.

ACKNOWLEDGEMENTS

I am gratefully indebted to Dr. P. David Fisher, my major advisor, for his financial support and encouragements for this research. I thank Dr. Chin-Long Wey for his editorial and technical guidance to improve this dissertation more than I care to admit. I also thank the guidance committee members, Dr. Michael A. Shanblatt and Dr. Jacob P. Plotkin, for their advice and sincere concern about this research. I am also indebted to Dr. Sheng-Fu Wu who has developed *MSUASLC* for his Ph.D. dissertation. Without this automated tool, the success of this research might be impossible.

To my parents who supported my study for years financially and spiritually, my wife, Tae-Won, and my daughter, Kil-Hyo, I dedicate this dissertation in token of gratitude for their patience and encouragements.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Sequential logic functions may be implemented in either clocked sequential logic circuits (CSLCs) or asynchronous sequential logic circuits (ASLCs). The former uses system-level clocks, and the latter solely uses sequences of input changes to initiate internal state transitions. Each of these classes of sequential circuits has advantages over its counterpart. ASLCs may be faster for certain applications since they do not have to wait for the arrival of a clock pulse before effecting a state transition. Moreover, they may require fewer logic gates since they do not use memory elements to store state information. However, CSLCs have been preferred by the designers because they have been simpler to design since there is no need to consider critical-races in the state assignments and hazards in the logic implementations, which are inherent in their ASLC counterparts [1-6].

On the other hand, as the integration scales and circuit complexity increase in high-speed digital system, the global clock signal in the CSLC may have a clock skew problem [7], which is a phase difference of the clock signal at different locations due to the capacitive load in the interconnection line. Multi-phase clocks can be used to absorb the

clock skew, but a dead time between clock phases degrades the performance by reducing the time available for computation. The proper clock distribution in the circuit layout is an alternative to reduce the clock skew problem. However, in the design automation tools available nowadays, the routing of clock wires as well as the load on the clock signal are globally considered, and it is difficult to extract from local connectivity [8, 9].

Asynchronous circuit design is a realistic approach to circumvent the clock skew problem. At the chip level, layout and simulation effort is greatly reduced since there is no global timing, and systems can be easily extended without problems in global synchronization by using pipeline architecture, where computation can be extended and sped up without any global constraint on the overall system throughput [10, 11].

1.1 Motivation

The functional behavior of an ASLC is traditionally described with a flow table [1-6, 12]. So, the synthesis procedure begins by generating a primitive flow table (PFT) which describes the functional design specification, and then states are merged and encoded to avoid critical races. Finally, a static hazard-free circuit is realized from a set of state equations and output equations. The PFT reveals difficulties to design large-scale asynchronous networks because the size of the PFT increases drastically. In general, the number of internal states increases with the number of inputs and outputs [13]. Since there are many ways to merge the rows of a PFT, it is hard to predict and obtain an optimum merged result without exhaustively searching the combinations. Once the state merging process is completed, the states are encoded in accordance with the merged flow table (MFT). An adjacency diagram which describes the relationship between any two nodes is generally employed for state encoding [2]. The complexity of the encoding process is

determined by the complexity of the adjacency diagram. Thus, the complexity of both state merging and encoding processes increases with the circuit complexity.

Merging rules are generally employed to develop a state merging algorithm. Those rules are generated in accordance with a merger diagram [2] but not with the functional behaviors of the input variables. In other words, the algorithms were developed to solve the graphical problems without taking the functional behavior into account. This is one reason why both state merging and encoding processes become so complicated.

The fact is that the input variables in the conventional PFT approach are treated equally. In practice, however, the input variables can be classified as either control inputs or data inputs from a design specification. The control inputs can be used as transition variables to simplify the synthesis procedure. This has motivated to the development of an analytical model for an alternative synthesis procedure in this study.

1.2 Problem Statement and Research Tasks

With *a priori* information given in a design specification, the input variables can be classified as either control inputs or data inputs. In this thesis, an analytical synthesis model is developed in which some control inputs are used as the transition variables. The transition variables are used to generate a set of dynamic output equations (DOEs). Based on the DOEs, an efficient state merging method is developed. With the merged state table, an efficient and effective state assignment method is also presented.

State merging methods have been studied significantly in the last few decades. Basically, states are merged if they are compatible, i.e., they have the same next state with the same input. Since the primitive flow table inherently has many "don't care" entries, a state can be compatible to any other states in many different ways. As a result, there are

many ways to merge the states. In this thesis, based on the analytical model, the transition variables are used to guide the derivation of the merged flow table. In other words, the merge strategy is based on the functional behavior of the circuit, but not only on the graphical relationship of the states in the merger diagram.

States are conventionally encoded based on an adjacency diagram which is derived from the merged flow table. Since the merged flow table were only based on the graphical relationship of the states in the merger diagram, the states in the adjacency diagram may not represent the functional behavior of the circuit very well. As a result, the state encoding problem becomes a very complicated graph problem as the number of states increases. However, based on the merged flow table presented in this thesis, the states in the adjacency diagram represent the functional behavior. The adjacency diagram drawn from the merged flow table can be represented as a bipartite graph. A state encoding problem can then be formatted as the embedding of a bipartite graph in an n-cube. In general, an n-cube can be also represented as a bipartite graph. Thus, the problem becomes to embed a bipartite graph from the adjacency diagram to another bipartite graph derived from the n-cube. This thesis presents a rule-based graph matching algorithm to encode states.

The tasks of this research are: (1) develop an analytical model for ASLC synthesis procedure; and (2) develop an efficient algorithm for state merging and encoding. The developed analytical synthesis model, state merging method, and state assignment algorithm can then be adopted to the ASLC synthesis system, *MSUASLC* [14], for the synthesis of larger scale circuits.

This research leads to the development of an efficient synthesis procedure which will be utilized to synthesize large-scale ASLCs or sequential logic functions which have been implemented with CSLCs.

1.3 Organization

This thesis is organized as follows: Chapter 2 briefly reviews the development of *MSUASLC* design automation system and discusses the major algorithms developed in [14]. Chapter 3 presents an analytical model for ASLC synthesis. A set of equations, referred to as dynamic output equations (DOEs), is developed first from a generalized model by introducing transition variables. Rules and examples for state grouping from the DOEs are described. Chapter 4 describes an efficient state assignment algorithm based on the bipartite characteristics of the adjacency diagram and an n-cube. Several rules and procedures are developed to map a bipartite adjacency table (BAT) to a bipartite representation table (BRT) of an n-cube. In order to demonstrate the effectiveness of the synthesis procedure developed in this study, several examples are presented in Chapter 5. Finally, a summary of this research work and future research are given in Chapter 6.

Chapter 2

MSUASLC Design Automation System

After Huffman [15] devised a general model for ASLCs and introduced the primitive flow table (PFT) as a design tool, many researchers [16-20] contributed to the development of a well-established ASLC design procedure which is described in many advanced logic design text books [1-6, 12]. The ASLC synthesis procedure generally consists of the following five steps: (1) generate the PFT from a design specification; (2) generate merged flow table (MFT) by merging the compatible states in the PFT; (3) encode the internal states for avoiding critical races; (4) generate the state excitation table and output table; and, (5) eliminate static hazards and implement the circuit using two-level logic or PLA-based architectures. Numerous synthesis systems have been developed to reduce the human effort dealing with the complexities of the procedure by automating parts of this procedure. An ASLC design automation system, *MSUASLC*, has been developed to produce design equations from a design specification. In this chapter, the features of *MSUASLC* are briefly described, and the state merging and encoding strategies developed in [14] discussed.

2.1 Features of MSUASLC

Based on Huffman's design procedure, an ASLC design system was first developed by Smith, et al. [21]. It receives input in the form of reduced primitive flow table, executes state assignment using Tracey's algorithms [22], and generates next-state equations and output equations which are realized by two-level logic.

The construction of a primitive flow table (PFT) requires the designer's intuitions and design experiences. Apparently, it becomes more difficult to handle as inputs and outputs increase. Since there exists no analytic approach to obtain the PFT directly from the verbal design specification, timing diagrams or state transition graphs are commonly used by designers.

Recently, Wu and Fisher [14, 23] developed a fully automated ASLC design system, *MSUASLC*, on SUN workstations in the C programming language. A block diagram of the *MSUASLC* Design System, as shown in Fig. 2-1, consists of the following five modules:

(1) **Behavioral Descriptor** ***(BD)***: It receives inputs with "IF... THEN..." format of design specifications, and generates a primitive state table and primitive output table which describe the circuit's functional behavior.

(2) **Merger**: It receives a primitive state table and primitive output table, and generates the merged state table and merged output table which contains a minimum number of states.

(3) **Connector**: It receives a merged state table and a merged output table, and converts it into an adjacency table which provides the relation between each state. It generates modified state table and modified output table which has race-free states relationships by adding states and generating cycles as needed.

(4) **Assigner**: It generates the excitation table and output table from the race-free state

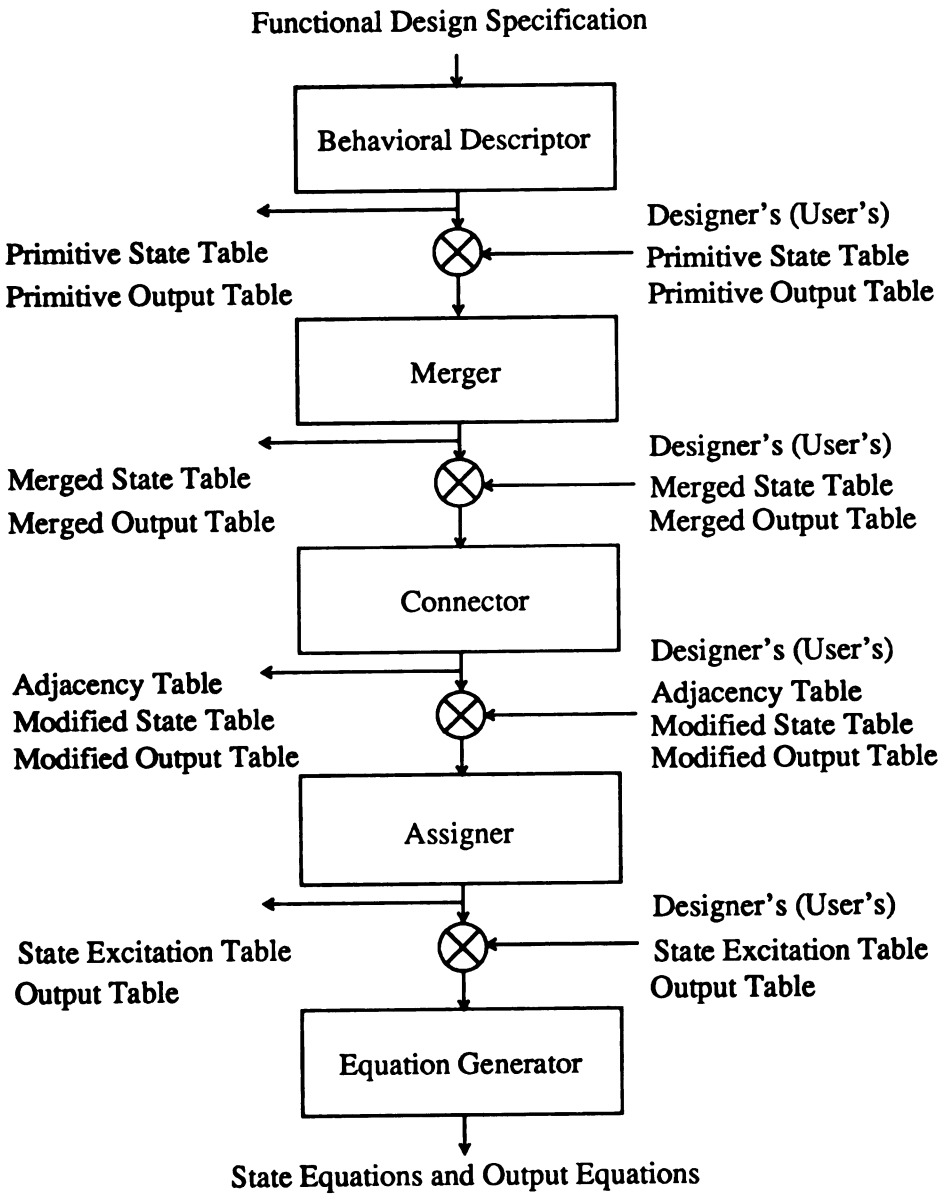


Figure 2-1. Block diagram of the *MSUASLC* design automation system. [14]

table and the output table.

(5) **Equation Generator** ***(EG)***: It generates state equations and output equations in sum-of-products forms from the excitation table and output table. They are static hazard-free.

Each output of the previous module can be directly sent to the next module or can be modified by the designer and then sent to the next module. This modularity of *MSUASLC* provides a convenient way to investigate alternative implementations of an ASLC. In the following sections, the modules, *Merger* and *Assigner*, are discussed in detail, which provide methods for alternative implementations of an ASLC.

2.2 State Merging Methods

The *Merger* in the *MSUASLC* design system generates first a merger diagram (MD) which is an undirected connected graph with nodes and edges. The nodes represent the primitive states in the PFT and the edges represent the compatibilities between the states in the rows. According to the compatibilities with other rows, each node has a degree of links, which is utilized to decide the priorities of merging sequence. The *MSUASLC* provides four different merging methods to merge rows. Method I starts from the first row to the last row for those have the same output. Method II starts from the rows with the minimum link degree and the least strongly connected subset is selected first. Method III is similar to Method II except the same outputs are not considered. Finally, Method IV starts from the rows with the same output, then extends to those with different outputs, if possible. These four different merge methods are used to find the minimum number of merged rows.

The classical method to find the minimum number of merged row is to find the largest strongly connected subsets of the rows which can be merged into a single row [2].

However, finding the largest strongly connected subsets of rows may require the comparison between states in each row. Such an exhaustive search may take an exponential time regarding to the number of rows. Since the *Merger* gets output by choosing the minimum from the outputs from several different methods, which require a polynomial computing time.

To illustrate these merging methods, the following design example, a positive edge-triggered J-K bistable element with postponed output (Petjk-FF) is considered. Fig. 2-2(a) shows the graphic symbol of the circuit. Its design specification is given as follows:

(1) There are three inputs, which are labeled J, K, and C.

(2) There are two outputs, which are labeled Q and Q_p.

(3) When C changes value from 0 to 1, the next states of the external output Q is $Q_n = J\,Q_c' + K'\,Q_c$, where Q_c is the current value of Q.

(4) The output Q_p is a postponed output of Q, and the transition of Q_p is postponed until C changes value from 1 to 0.

Conventionally, based on the timing diagram of Fig. 2-2(b), the PFT can be generated with tedious works. However, the PFT is easily generated by converting this design specification into "IF... THEN..." format [24]. For example,

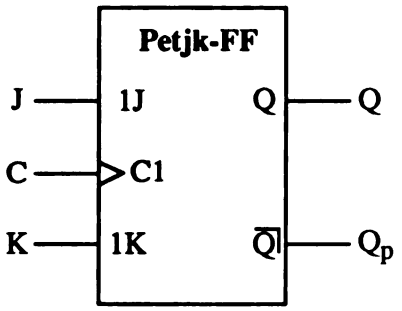
IF C = 2

THEN Q = J Q′ + K′ Q, P = P

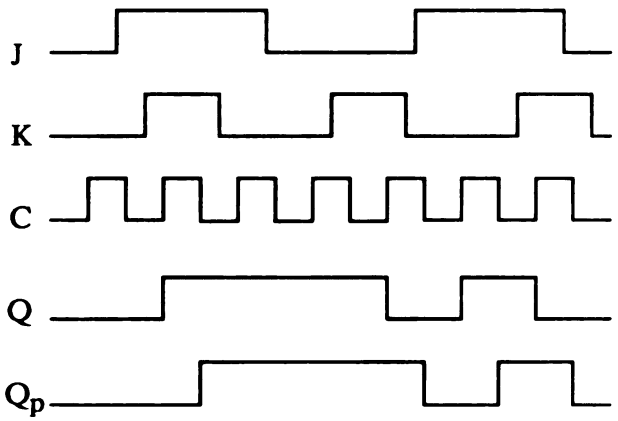
IF C = 3

THEN Q = Q, P = Q

Note that the P represents the output Q_p. Based on the *Behavioral Descriptor (BD)* of *MSUASLC*, the generated PFT is shown in Table 2-1. With four different merge methods, Tables 2-2, 2-3, 2-4, and 2-5 illustrate various Merged State Tables (MSTs). The relabeled MSTs and their adjacency diagrams are respectively shown in Figs. 2-3, 2-4, 2-5, and 2-6



(a)



(b)

Present Input (JKC)

Present Primitive State	000	001	011	010	110	111	101	100	Present Output (Q Qp)
0	(0)	4	-	8	-	-	-	16	0 0
3	(3)	7	-	11	-	-	-	19	1 1
4	0	(4)	12	-	-	-	20	-	0 0
5	0	(5)	13	-	-	-	21	-	0 1
6	3	(6)	14	-	-	-	22	-	1 0
7	3	(7)	15	-	-	-	23	-	1 1
8	0	-	12	(8)	24	-	-	-	0 0
11	3	-	13	(11)	27	-	-	-	1 1
12	-	4	(12)	8	-	28	-	-	0 0
13	-	5	(13)	8	-	29	-	-	0 1
14	-	6	(14)	11	-	30	-	-	1 0
15	-	7	(15)	11	-	31	-	-	1 1
16	0	-	-	-	24	-	22	(16)	0 0
19	3	-	-	-	27	-	23	(19)	1 1
20	-	4	-	-	-	28	(20)	16	0 0
21	-	5	-	-	-	29	(21)	16	0 1
22	-	6	-	-	-	30	(22)	19	1 0
23	-	7	-	-	-	31	(23)	19	1 1
24	-	-	-	8	(24)	30	-	16	0 0
27	-	-	-	11	(27)	29	-	19	1 1
28	-	-	12	-	24	(28)	20	-	0 0
29	-	-	13	-	24	(29)	21	-	0 1
30	-	-	14	-	27	(30)	22	-	1 0
31	-	-	15	-	27	(31)	23	-	1 1

(c)

Figure 2-2. Edge-triggered J-K bistable element with postponed output: (a) graphic symbol, (b) timing diagram, and (c) primitive flow table (PFT).

Table 2-1. Primitive State Table of the Petjk-FF sequential logic function.

*** MSU (WF) -- ASLC DESIGN AUTOMATION SYSTEM (BD) ***
*** Primitive State Table: "petjkwpo_PFT_PRN_1" ***

jkc (INPUT)	0	1	3	2	6	7	5	4	qp (OUTPUT)
	0 Y	4 N	-	8 N	-	-	-	16 N	0
	0 N	4 Y	12 N	-	-	-	20 N	-	0
	0 N	-	12 N	8 Y	24 N	-	-	-	0
	0 N	-	-	-	24 N	-	22 N	16 Y	0
	-	4 N	12 Y	8 N	-	28 N	-	-	0
	-	4 N	-	-	-	28 N	20 Y	16 N	0
	-	-	-	8 N	24 Y	30 N	-	16 N	0
	-	-	12 N	-	24 N	28 Y	20 N	-	0
	-	-	13 N	-	24 N	29 Y	21 N	-	1
	-	5 N	-	-	-	29 N	21 Y	16 N	1
	-	5 N	13 Y	8 N	-	29 N	-	-	1
	0 N	5 Y	13 N	-	-	-	21 N	-	1
	3 N	-	-	-	27 N	-	23 N	19 Y	3
	-	7 N	-	-	-	31 N	23 Y	19 N	3
	-	-	-	11 N	27 Y	29 N	-	19 N	3
	3 Y	7 N	-	11 N	-	-	-	19 N	3
	-	-	15 N	-	27 N	31 Y	23 N	-	3
	3 N	7 Y	15 N	-	-	-	23 N	-	3
	3 N	-	13 N	11 Y	27 N	-	-	-	3
	-	7 N	15 Y	11 N	-	31 N	-	-	3
	-	6 N	-	-	-	30 N	22 Y	19 N	2
	-	-	14 N	-	27 N	30 Y	22 N	-	2
	3 N	6 Y	14 N	-	-	-	22 N	-	2
	-	6 N	14 Y	11 N	-	30 N	-	-	2

Table 2-2. MST of the Petjk-FF with Method I.

*** MSU (WF) -- ASLC DESIGN AUTOMATION SYSTEM (MERGER) ***
*** Merged State Table: "petjkwpo_ALT0_MFT_PRN_1" ***

*** Merged same output (row by row) ***

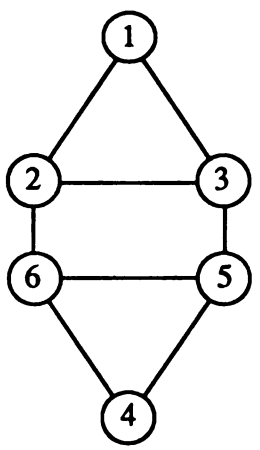
jkc (INPUT)	0	1	3	2	6	7	5	4	qp (OUTPUT)
	0 Y	4 Y	12 Y	8 Y	24 N	28 Y	20 Y	16 N	
	0 N	-	-	8 N	24 Y	30 N	22 N	16 Y	
	0 N	5 Y	13 Y	8 N	24 N	29 Y	21 Y	16 N	
	3 Y	7 Y	15 Y	11 N	27 N	31 Y	23 Y	19 Y	
	3 N	-	13 N	11 Y	27 Y	29 N	-	19 N	
	3 N	6 Y	14 Y	11 N	27 N	30 Y	22 Y	19 N	

Input (JKC)

	000	001	011	010	110	111	101	100
1	(1)	(1)	(1)	(1)	2	(1)	(1)	2
2	1	-	-	1	(2)	6	6	(2)
3	1	(3)	(3)	1	2	(3)	(3)	2
4	(4)	(4)	(4)	5	5	(4)	(4)	(4)
5	4	-	3	(5)	(5)	3	-	4
6	4	(6)	(6)	5	5	(6)	(6)	4

1 : (0, 4, 12, 8, 28, 20)
2 : (24, 16)
3 : (5, 13, 29, 21)
4 : (3, 7, 15, 31, 23, 19)
5 : (11, 27)
6 : (6, 14, 30, 22)

(a)



(b)

Figure 2-3. Method I: (a) relabeled MST and (b) adjacency diagram.

Table 2-3. MST of the Petjk-FF with Method II.

*** MSU (WF) -- ASLC DESIGN AUTOMATION SYSTEM (MERGER) ***
*** Merged State Table: "petjkwpo_ALT1_MFT_PRN_1" ***

** Merged same output (min. link degree first) **

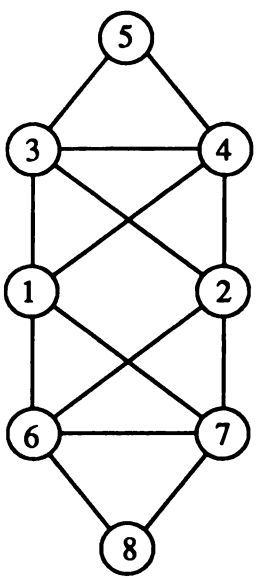
jkc (INPUT)	0	1	3	2	6	7	5	4	qp (OUTPUT)
	0 N	5 Y	13 Y	8 N	24 N	29 Y	21 Y	16 N	
	3 N	6 Y	14 Y	11 N	27 N	30 Y	22 Y	19 N	
	0 Y	4 N	12 Y	8 Y	24 N	28 N	22 N	16 Y	
	0 N	4 Y	12 N	8 N	24 Y	30 N	20 N	16 N	
	-	4 N	12 N	-	24 N	28 Y	20 Y	16 N	
	3 Y	7 Y	15 N	11 N	27 Y	29 N	23 N	19 Y	
	3 N	7 N	13 N	11 Y	27 N	31 N	23 Y	19 N	
	-	7 N	15 Y	11 N	27 N	31 Y	23 N	-	

Input (JKC)

	000	001	011	010	110	111	101	100
1	3	(1)	(1)	3	4	(1)	(1)	3
2	6	(2)	(2)	7	6	(2)	(2)	6
3	(3)	4	(3)	(3)	4	5	2	(3)
4	3	(4)	3	3	(4)	2	5	3
5	-	4	3	-	4	(5)	(5)	3
6	(6)	(6)	8	7	(6)	1	7	(6)
7	6	6	1	(7)	6	8	(7)	6
8	-	6	(8)	7	6	(8)	7	-

1 : (5, 13, 29, 21)
2 : (6, 14, 30, 22)
3 : (0, 12, 8, 16)
4 : (4, 24)
5 : (28, 20)
6 : (3, 7, 27, 19)
7 : (11, 23)
8 : (15, 31)

(a)



(b)

Figure 2-4. Method II: (a) relabeled MST and (b) adjacency diagram.

Table 2-4. MST of the Petjk-FF with Method III.

*** MSU (WF) -- ASLC DESIGN AUTOMATION SYSTEM (MERGER) ***
*** Merged State Table: "petjkwpo_ALT2_MFT_PRN_1" ***

** Min. link degree first (no care if output same or not) **

jkc (INPUT)	0	1	3	2	6	7	5	4	qp (OUTPUT)
	0 Y	4 N	13 N	8 N	24 N	29 Y	21 N	16 N	
	0 N	5 N	12 N	8 Y	24 N	29 N	21 Y	16 N	
	0 N	5 N	13 Y	8 N	24 N	29 N	22 N	16 Y	
	0 N	5 Y	13 N	8 N	24 Y	30 N	21 N	16 N	
	0 N	4 Y	12 Y	8 N	24 N	28 Y	20 Y	16 N	
	3 N	6 N	13 N	11 Y	27 N	30 N	22 Y	19 N	
	3 Y	7 N	14 N	11 N	27 N	30 Y	22 N	19 N	
	3 N	6 Y	14 N	11 N	27 Y	29 N	22 N	19 N	
	3 N	6 N	14 Y	11 N	27 N	30 N	23 N	19 Y	
	3 N	7 Y	15 Y	11 N	27 N	31 Y	23 Y	19 N	

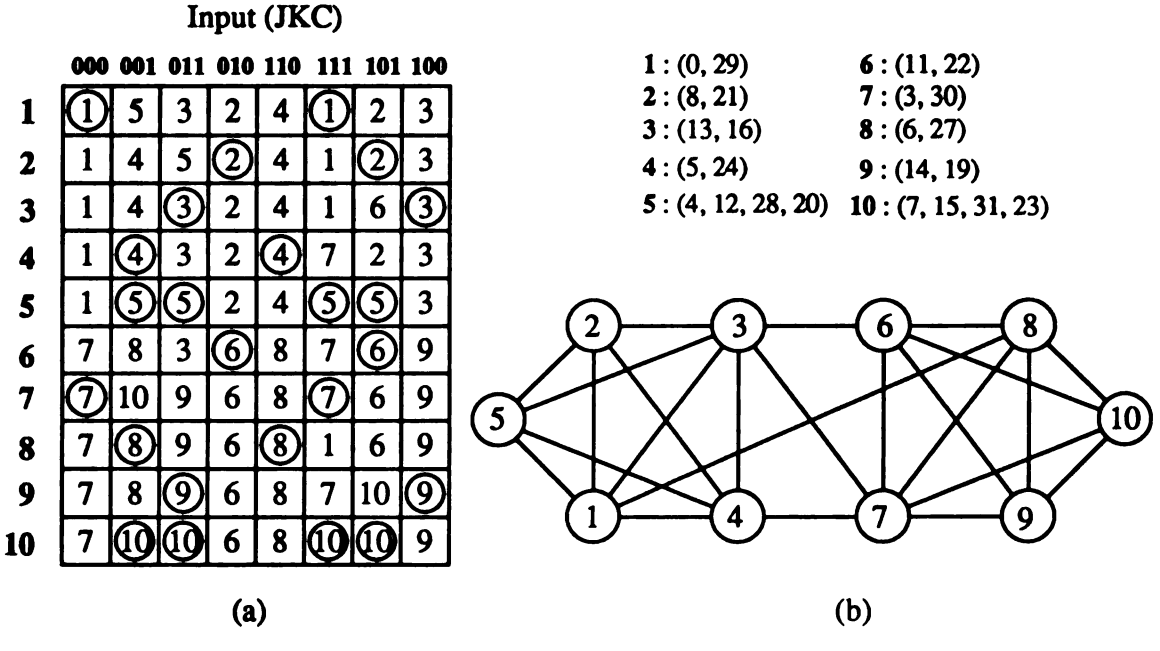


Figure 2-5. Method III: (a) relabeled MST and (b) adjacency diagram.

Table 2-5. MST of the Petjk-FF with Method IV

*** MSU (WF) -- ASLC DESIGN AUTOMATION SYSTEM (MERGER) ***
*** Merged State Table: "petjkwpo_ALT3_MFT_PRN_1" ***

** Merged same output (extend to different output) **

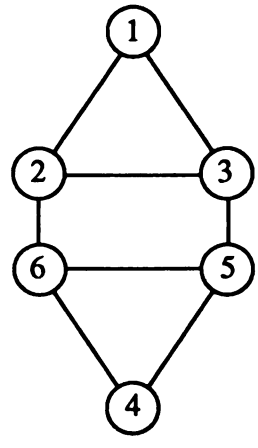
jkc (INPUT)	0	1	3	2	6	7	5	4	qp (OUTPUT)
	0 Y	4 Y	12 Y	8 Y	24 N	28 Y	20 Y	16 N	
	0 N	-	-	8 N	24 Y	30 N	22 N	16 Y	
	0 N	5 Y	13 Y	8 N	24 N	29 Y	21 Y	16 N	
	3 Y	7 Y	15 Y	11 N	27 N	31 Y	23 Y	19 Y	
	3 N	-	13 N	11 Y	27 Y	29 N	-	19 N	
	3 N	6 Y	14 Y	11 N	27 N	30 Y	22 Y	19 N	

Input (JKC)

	000	001	011	010	110	111	101	100
1	(1)	(1)	(1)	(1)	2	(1)	(1)	2
2	1	-	-	1	(2)	6	6	(2)
3	1	(3)	(3)	1	2	(3)	(3)	2
4	(4)	(4)	(4)	5	5	(4)	(4)	(4)
5	4	-	3	(5)	(5)	3	-	4
6	4	(6)	(6)	5	5	(6)	(6)	4

1 : (0, 4, 12, 8, 28, 20) **4** : (3, 7, 15, 31, 23, 19)
2 : (24, 16) **5** : (11, 27)
3 : (5, 13, 29, 21) **6** : (6, 14, 30, 22)

(a)



(b)

Figure 2-6. Method IV: (a) relabeled MST and (b) adjacency diagram.

for the comparison purpose.

Method I and Method IV have the same result with 6 rows, which is the minimum number of rows. They also have the least complex adjacency diagram. However, the adjacency diagrams of Method I and Method IV show that there are hidden intrinsic races (HIR)[14] in this MST because it has loops with 3 nodes. This implies that these six internal states may not be the minimum number of states because some more states may be added later for race-free state assignments.

It is clear that the results of merging depend not only on the merging method but also on the merging sequence [14]. Also, additional states may be required to eliminate the races. Therefore, obtaining a minimum number of states in the merging step does not guarantee the optimal circuit realization. Chapter 3 will describe an efficient merging method which utilizes the information from the design specification and leads to an efficient circuit realization.

2.3 State Assignment Methods

Before the *Assigner* in *MSUASLC* assigns a binary vector to each state, the *Connector* generates a race-free state table by utilizing a Node Weight Diagram (NWD) [25]. The NWD is a binary n-cube connection diagram which provides a geometric representation of binary numbers for race-free state assignments. Fig. 2-7(b) shows a 4-NWD, where the circled numbers represent the states to be assigned to an n-cube, while the boldfaced numbers represent the decimal number of the binary vectors assigned for those nodes. The *Assigner* provides alternative state assignments by allowing the designer to assign a specific binary vector to a specific state.

Table 2-6 illustrates the modified state table generated by the *Connector* for the Petjk-FF example, where there are seven internal states and eight modified unstable states.

Table 2-6. Modified state table of Petjk-FF sequential logic function

*** MSU (WF) -- ASLC DESIGN AUTOMATION SYSTEM (CONNECTOR) ***
*** Modified (ADJ) State Table: "petjkwpo_ADJ_MFT_PRN_1" ***

jkc (INPUT)	0	1	3	2	6	7	5	4	qp (OUTPUT)
1	1 Y	1 Y	1 Y	1 Y	2 N	1 Y	1 Y	2 N	
2	1 N	-	-	1 N	2 Y	6 N	6 N	2 Y	
3	1 N	3 Y	3 Y	1 N	1 N	3 Y	3 Y	1 N	
4	4 Y	4 Y	4 Y	6 N	6 N	4 Y	4 Y	4 Y	
5	6 N	-	7 N	5 Y	5 Y	7 N	-	6 N	
6	4 N	6 Y	6 Y	5 N	5 N	6 Y	6 Y	4 N	
7	-	-	3 N	-	-	3 N	-	-	

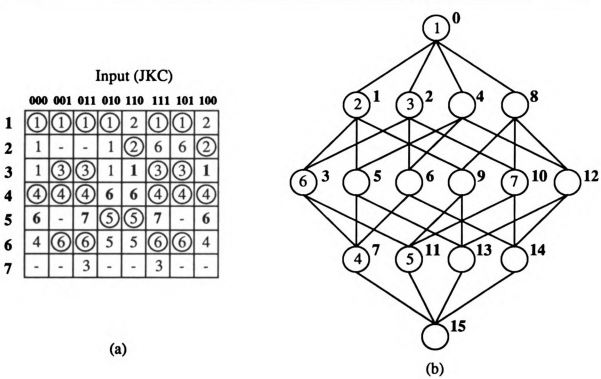


Figure 2-7. State encoding of the Petjk-FF sequential logic function: (a) modified state table and (b) state assignments with 4-NWD.

The table is retabulated in Fig. 2-7(a), where the boldfaced numbers are the modified unstable states. Fig. 2-7(b) shows the state assignment with 4-NWD for this example.

Table 2-7 shows the unique state assignments generated from the *Assigner* for this modified state table. For seven internal states and eight possible encoded binary vectors, it is possible to generate 56 distinctive combinations which exclude the permutation of these assignments. However, the *Assigner* provides 16 distinct state assignments as shown in Table 2-7. The circuit designer must choose the "best" among these assignments for an

Table 2-7. Unique state assignments in the *Assigner* of *MSUASLC* for the Petjk-FF sequential logic function design.

State Number	1	2	3	4	5	6	7
Assign # 1	0	1	2	7	11	3	10
Assign # 2	1	0	3	6	10	2	11
Assign # 3	2	3	0	5	9	1	8
Assign # 4	7	6	5	0	12	4	13
Assign # 5	11	10	9	12	0	8	1
Assign # 6	3	2	1	4	8	0	9
Assign # 7	10	11	8	13	1	9	0
Assign # 8	9	8	11	14	2	10	3
Assign # 9	8	9	10	15	3	11	2
Assign # 10	4	5	6	3	15	7	14
Assign # 11	15	14	13	8	4	12	5
Assign # 12	14	15	12	9	5	13	4
Assign # 13	5	4	7	2	14	6	15
Assign # 14	6	7	4	1	13	5	12
Assign # 15	13	12	15	10	6	14	7
Assign # 16	12	13	14	11	7	15	6

optimum realization. However, the optimum one may not be included in these assignments. In addition, those state assignments require 4 state variables provided by the *Connector*.

It should be mentioned that, with an alternative modification of the state table, the same example can be realized using only three state variables, as shown in Figs 2-8 and 2-9. Instead of using 8 modified unstable states in Fig. 2-7(a), Figs. 2-8 and 2-9 employ 10 and 4 modified unstable states, respectively. In other words, the state table in Fig. 2-9 is better than that in Fig. 2-7 generated by *MSUASLC* in the numbers of state variables and modified unstable states.

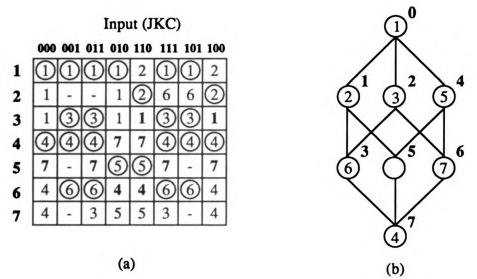


Figure 2-8. Alternative state assignment 1 of the Petjk-FF sequential logic function: (a) modified state table and (b) state assignments with 3-NWD.

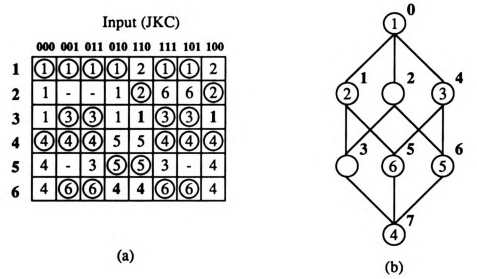


Figure 2-9. Alternative state assignment 2 of the Petjk-FF sequential logic function: (a) modified state table and (b) state assignments with 3-NWD.

Chapter 3

Synthesis Model

The primitive flow table (PFT) has traditionally been used in the synthesis procedure to capture the functional behavior of ASLCs. However, its size increases exponentially according to the number of input and output. This chapter presents an analytical model which efficiently captures the functional behavior of an ASLC. The model is represented with a set of equations named Dynamic Output Equations (DOEs). These equations are functionally equivalent to PFT, but they don't have exponential increase of the entries. In addition, the DOEs provides an efficient way to identify the compatible states so that the number of states can be reduced considerably with a reasonably low computation time.

3.1 Analytical Model for an ASLC

The input variables in the conventional PFT approach are treated equally. In practice, however, the input variables can be classified as either control inputs or data inputs from the design specification.

A generalized ASLC model is introduced in which three distinct classes of inputs are delineated, i.e., **mode inputs**, **level inputs**, and **edge inputs**. The transition variables are defined to model the dynamic behavior of the edge inputs. Using transition variables, a set of equations, referred to as **dynamic output equations** (DOEs), is developed to characterize the functional behavior of an ASLC. As the transition variables are defined from the edge inputs which may cause state transitions to occur, the states can be grouped according to the values of the transition variables. This state grouping leads to simpler states encoding procedure.

3.1.1 Generalized Model

A generalized ASLC model is illustrated in Fig. 3-1. The external input data word is decomposed into three parts; namely, M, L, and E. M is a p-bit input and is referred to as the mode input. L is a q-bit input and is referred to as the level input. And E is an r-bit input and is referred to as the edge input. The ASLC has one external output, which is labeled Z and is an n-bit data word. Internally, the ASLC contains two combinational logic elements, NSG and POG. NSG and POG are the next-state generator and present-output generator, respectively. The present state and next state of the ASLC are represented by the m-bit data words y and Y, respectively.

The M, L, and E inputs are distinguished as follows: Mode inputs (M) do not cause internal state transitions to occur nor do they effect the internal state of the ASLC. Hence, the next state Y is not a function of M. Level inputs (L) do not cause state transitions to occur but may effect the state of the ASLC; so, the next state Y is a function of L. Edge inputs (E) may cause state transitions to occur, and, moreover, the next state Y may be a function of E. The present output Z will depend upon M, provided $p \geq 1$ (see Fig. 3-1). In this generalized model, the output Z may also be dependent upon the present values of the L and E inputs. With reference to Fig. 3-1, q_0 and r_0 represent the number of bits in the current values of level input and edge input data words, respectively, that effect the present value of the output Z, where $0 \leq q_0 \leq q$, and $0 \leq r_0 \leq r$. And, finally, a subset of the bits in the present state y may effect the present output, where $0 \leq m_0 \leq m$ (see Fig. 3-1).

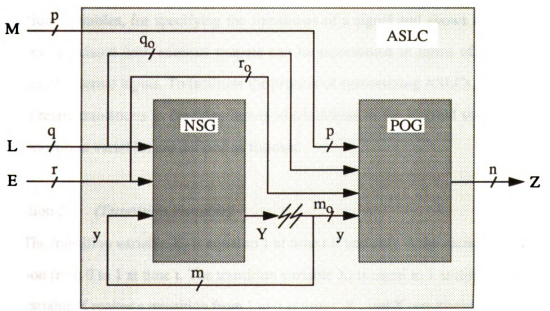


Figure 3-1. A generalized model for an asynchronous sequential logic circuit (ASLC).

The special case where $m_o = 0$ implies that the external outputs are not a function of the internal state of the ASLC, which means that the logic element is a combinational logic element and not a sequential logic element. Thus, combinational logic elements represent a limiting case for the ASLC modeling for synthesis methodology presented here.

The mode inputs (M) may be viewed as a subset of the level inputs (L), which are in turn a subset of the edge inputs (E). If all inputs for a particular ASLC are viewed as edge inputs, then the modeling approach presented here would reduce to the traditional approach for describing the functional behavior of an ASLC for purposes of synthesis. This is so because each input would be treated equally. Hence, this limiting case could lead to expressing the functional behavior of the ASLC in the form of the traditional primitive flow table.

In this ASLC model, only transitions in edge inputs may cause internal state transitions to occur. McCluskey [4] describes a convenient notation, which he terms **transition variables**, for specifying the transitions of a signal and shows how transition variables associated with external outputs can be represented in terms of the transition variables of external inputs. To facilitate the process of synthesizing ASLCs, the concept is used to relate transitions in the edge inputs to transitions in the internal state of ASLCs. These transition variables are defined as follows.

Definition 3.1 *(Transition Variable)*

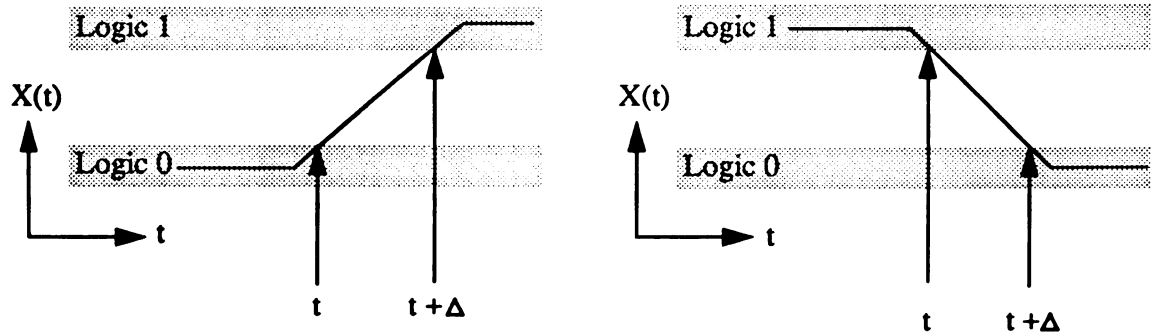
The transition variable X_r is equal to 1 at time t if and only if the variable X makes a transition from 0 to 1 at time t. The transition variable X_f is equal to 1 at time t if and only if the variable X makes a transition from 1 to 0 at time t. X_r and X_f are known as the rising-edge and falling-edge transition variables, respectively.

They possess both a physical and logical significance. From a physical perspective, X_r and X_f equate to the time rate of change of the electrical signal observed for X(t); the width of the X_r and X_f pulses relate directly to the rise time and fall time of X(t), respectively. From a logical perspective, X_r and X_f indicate the occurrence of a transition of the signal X. And, from a mixed-mode (i.e., physical and logical) perspective these transition variables can be defined as follows (see Fig. 3-2):

$$X_r = X'(t) \cdot X(t+\Delta) \tag{3.1}$$

$$X_f = X(t) \cdot X'(t+\Delta) \tag{3.2}$$

The transition variables X_r and X_f describe the dynamic behavior of the external inputs that lead to changes in the ASLC's internal state. The functional behavior of the ASLC is captured with a set of sequential logic functions, or DOEs. For the generalized



$X_r = X'(t) \bullet X(t+\Delta)$ (a)

$X_f = X(t) \bullet X'(t+\Delta)$ (b)

Figure 3-2. Mixed-mode representation of the signal X(t)'s (a) rising edge transition variable X_r and (b) falling edge transition variable X_f.

ASLC model, the set of DOEs express the functional dependence of the ASLC's next output Z_n on the current output Z_c; the current values of the input variables D and E; and the current values of the transition variables E_r and E_f. This functional relationship can be expressed as follows:

$$Z_n = f(Z_c, M, L, E, E_r, E_f) \tag{3.3}$$

This generalized model can be used to implement Finite State Machines (FSMs) as either a Mealy type or a Moore type machines. When the output Z is a function of current states, mode inputs, level inputs, and edge inputs, or Z = f(y, M, L, E), and the next state Y is a function of current states, level inputs, and edge inputs, or Y = h(y, L, E), then the ASLC can be viewed as a Mealy machine. When there is no mode control input, the output is a function of current states only or Z = g(y), and the next state Y is a function of current states, level inputs, and edge inputs, or Y = h(y, L, E), then the ASLC can be viewed as a Moore machine.

Through the generalized ASLC model, a relationship between inputs and outputs are derived as $Z_n = f(Z_c, M, L, E, E_r, E_f)$, where Z_n is the next output and Z_c is the present output. This relationship, or DOE, represents the circuit's functional behavior, and which can be obtained by considering the relationship between each type of input and present output independently.

3.1.2 Rules for Generating DOEs

Consider an ASLC with a single edge input E_i, referred to as the present value of input E_i. Let E_{ir} and E_{if} denote rising-edge and falling-edge transition variables, respectively. From the definition of these transition variables, it follows that

$$E_i \cdot E_{ir} = 0 \qquad E_i' \cdot E_{if} = 0 \qquad E_{ir} \cdot E_{if} = 0 \tag{3.4}$$

Using these relationships, it can be shown that following relationships among E_i, E_{ir}, and E_{if} are true:

$$\begin{aligned} &E_i' \cdot E_{ir} = E_{ir} \qquad E_i \cdot E_{if} = E_{if} \qquad E_{ir} \cdot E_{if}' = E_{ir} \\ &E_i \cdot E_{ir}' = E_i \qquad E_i' \cdot E_{if}' = E_i' \qquad E_{ir}' \cdot E_{if} = E_{if} \end{aligned} \tag{3.5}$$

Equations (3.4) and (3.5) lead to the following rules for developing the set of DOEs:

Rule 3.1.1 A sum-of-products (SOP) term in a DOE never contains more than one uncomplemented transition variable.

Rule 3.1.2 If the transition variable for an edge input is present in a SOP term, then that term is vacuous in the edge-input switching variable(s) associated with that transition variable.

These two rules deal with SOP terms which contain a literal, a transition variable E_i or its complement E_i'. Consider a SOP term that does not contain literals E_i, E_i', E_{ir}, or E_{if} but does contain the literals E_{ir}' and E_{if}'. From the definitions for these transition variables, it can be shown that

$$E_{ir}' \cdot E_{if}' = E_i'(t) \cdot E_i'(t+\Delta) + E_i(t) \cdot E_i(t+\Delta) \tag{3.6}$$

If $E_{ir}' \cdot E_{if}' = 1$, the ASLC is in a steady state condition since no transitions in the edge inputs occur at that instant of time. Hence the next ASLC output is not due to an internal state change caused by transitions in the edge input E_i.

This property of the ASLC can then be stated in terms of the following rule for constructing DOEs:

Rule 3.1.3 If an SOP term only contains complements of the transition variables, then the ASLC is in steady state with respect to internal state transitions when $E_{ir}' \cdot E_{if}' = 1$.

The above rules were developed by only considering the existence of a single edge input E_i. When more than one edge input is present, the fundamental-mode constraint can be applied to produce additional rules to guide the development of the DOEs. The fundamental-mode constraint only allows one transition variable to be asserted at any instant in time. For the case of two edge variables, E_i and E_j, the fundamental-mode constraint implies that

$$E_{ir} \cdot E_{jr} = 0 \qquad E_{ir} \cdot E_{jf} = 0 \qquad E_{if} \cdot E_{jr} = 0 \qquad E_{if} \cdot E_{jf} = 0 \tag{3.7}$$

Using these relationships, it can be shown that following relationships among the transition variables of E_i and E_j and the complements of the transition variables are valid:

$$E_{ir}' \cdot E_{jr} = E_{jr} \qquad E_{ir}' \cdot E_{jf} = E_{jf} \qquad E_{if}' \cdot E_{jr} = E_{jr} \qquad E_{if}' \cdot E_{jf} = E_{jf} \tag{3.8}$$

Equations (3.7) and (3.8) lead to the following rule for developing the set of DOEs:

Rule 3.1.4 If transition variable E_t is present in an SOP term of the DOE, then the term is vacuous in all other transition variables, as well as their complements.

This is a generalization of Rules 3.1.1 and 3.1.2 when the number of edge inputs E is equal to p; $p \geq 1$. Rule 3.1.3 can also be extended to the general case when one or more than one edge inputs are present. Let p denote the number of edge inputs; let p_r and p_f represent the number of rising edges and falling edges of the edge inputs that cause internal state transitions, respectively.

Rule 3.1.5 If an SOP term contains the complements of the p_r rising-edge transition variables and p_f falling-edge transition variables that cause internal state transitions to occur, then the ASLC is in steady state if

$$\prod_{i=1}^{p_r} \prod_{j=1}^{p_f} E_{ir}' \cdot E_{jf}' = 1 \tag{3.9}$$

By steady state we mean that the next external output Z_n does not result from a change in the ASLC's internal state even though one of the external inputs may have changed and thereby will cause a change in the external output.

Rules 3.1.4 and 3.1.5 lead directly to a more refined model for the set of DOEs than that presented in Equation (3.3). The DOEs are a set of switching functions which describe the next ASLC output state Z_n in terms of the current values of the mode (M) inputs, the level (L) inputs, and the edge (E) inputs; and the rising-edge (E_{ir}) and falling-edge (E_{jf}) transition variables. Each DOE can be expressed in a standard sum-of-products form as follows:

$$Z_n = \sum_{i=1}^{p_r} F_{ir} \cdot E_{ir} + \sum_{j=1}^{p_f} F_{jf} \cdot E_{jf} + F_s \cdot \prod_{i=1}^{p_r} \prod_{j=1}^{p_f} E_{ir}' \cdot E_{jf}' \tag{3.10}$$

Because of Rules 3.1.4 and 3.1.5, the sum-of-products terms are linearly independent in the transition variables. The coefficients F_{ir}, F_{jf}, and F_s have the following interpretation in the context of the functional behavior of the ASLC: F_{ir} and F_{jf} are the next external outputs when $E_{ir} = 1$ and $E_{jf} = 1$, respectively. F_{ir} and F_{jf} may be a function of Z_c, M, and L, as well as any of the edge input switching variables except E_i or E_j, respectively. F_s is the next output when no internal state transition occurs. F_s represents the steady-state output and may be a function of Z_c, M, L, and E.

Equation (3.10) provides the framework for developing an analytical model to describe the functional behavior of ASLC for purposes of implementing this sequential function in an ASLC. The utility of this modeling methodology is demonstrated in Chapter 5 with synthesis examples.

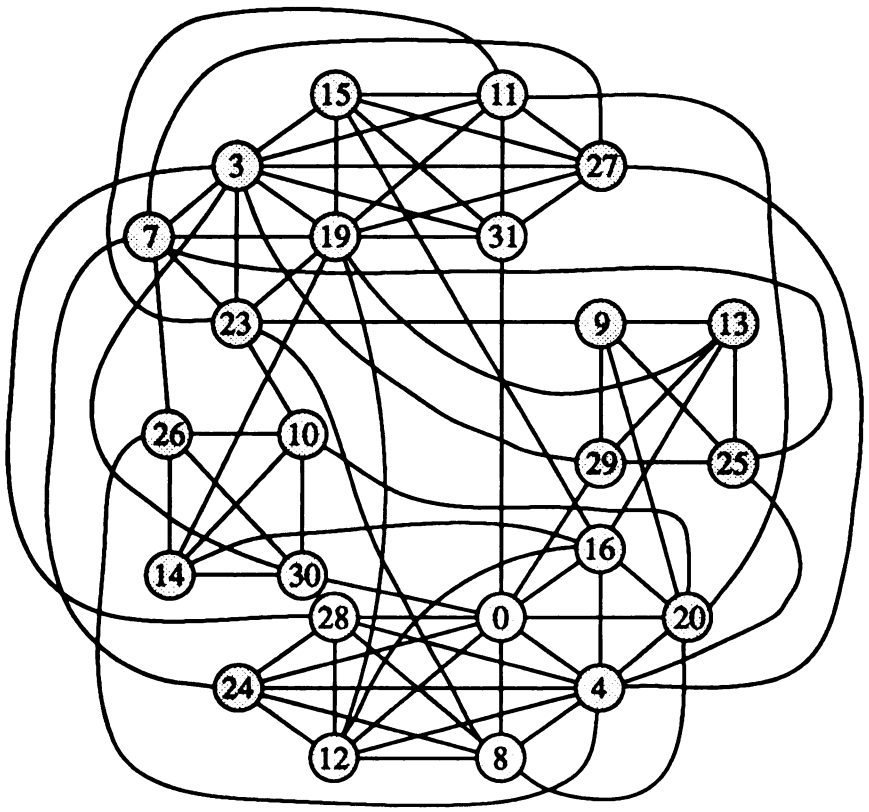
3.2 Traditional State Merging Methods

The traditional synthesis procedure of ASLCs begins with the development of the primitive flow table (PFT), and follows with the state reduction of the PFT to generate the merged flow table (MFT). The major purpose of the state reduction is to get a MFT with the fewest number of internal states. Many researchers contributed to the development of an efficient method to obtain a minimum-state flow table [26-34]. Since the PFT is constructed with a fundamental-mode assumption, it always has many entries unspecified, and there is no unique minimum-state flow table for an incompletely specified flow table. Accordingly, there are many possible ways to merge the primitive states which may lead to different circuits. However, no general rule which allows the designer to choose a merger leading to an efficient circuit has existed.

To illustrate above procedures, the design specification of a sequential logic function for Petjk-FF, in Fig. 2-2(a) is considered.

Table 2.1 shows the developed PFT. When the PFT is generated, the state reduction is possible by finding compatible sets of rows and the largest strongly connected sets in the PFT [4]. To find compatible sets of rows, the comparisons between rows are needed, which requires considerable amounts of computation time, and the identification of the largest strongly connected set also requires considerable amounts of computation time when the number of internal states is large. A merger diagram is commonly used in the traditional procedure to identify the largest strongly connected sets. Fig. 3-3(a) shows the merger diagram of this example.

The resulting merged flow table (MFT) is shown in Fig. 3-3(b), and its adjacency diagram for state assignments is shown in Fig. 3-3(c). One of the most important parts in the ASLC synthesis procedure is to get race-free state assignments for encoding the internal states. The race-free state assignment procedures are generally divided into two parts: first one is to identify and eliminate an intrinsic races, and second one is to assign binary codes to the states without generating races [14]. The flow table may have races if its m states cannot be encoded by n state variables without adding states or cycles, where $n = \lceil \log_2 m \rceil$. This case is easily identified when the number of states in each loop of the adjacency diagram is odd. So the adjacency diagram in Fig. 3-3(c) has races. When there exist races, generating cycles with or without additional states is required to remove them. If the state table has no intrinsic races, a race-free state assignment can be easily obtained in the synthesis procedure.



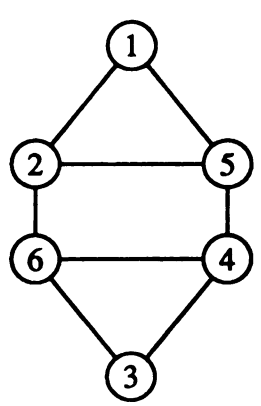
(a)

Present Input (JCK)

	000	001	011	010	110	111	101	100
1	(1)	(1)	(1)	(1)	(1)	(1)	2	2
2	1	1	-	-	6	6	(2)	(2)
3	(3)	4	(3)	(3)	(3)	(3)	4	(3)
4	3	(4)	5	-	-	5	(4)	3
5	1	1	(5)	(5)	(5)	(5)	2	2
6	3	4	(6)	(6)	(6)	(6)	4	3

1 : (0, 4, 12, 8, 24, 28) **4** : (7, 23)
2 : (20, 16) **5** : (13, 9, 25, 29)
3 : (3, 15, 11, 27, 31, 19) **6** : (14, 10, 26, 30)

(b)



(c)

Figure 3-3. Edge-triggered J-K bistable element with postponed output: (a) merger diagram, (b) merged flow table, and (c) adjacency diagram for state assignments.

3.3 State Grouping

Two states are said to be compatible in the traditional synthesis procedure if and only if whose outputs and next states agree whenever either or both are specified [4]. But, the compatibility is defined slightly differently in this state grouping process. A set of DOEs can be used to calculate the allowed output values. Since the concatenation of input variables and output variables produces the primitive states, the primitive states which are compatible can be easily recognized and grouped together by applying the following two rules:

Rule 3.3.1 Identify and make groups with compatible primitive states; i.e., they are compatible if they have the same outputs and transition variables.

Rule 3.3.2 Generate a state table with stable state groups and unstable states in each value of transition variable.

The unstable states in a state table can be calculated from the DOEs with the stable states and input values of transition variables. To illustrate the application of above rules, the Petjk-FF example is again considered. From the design specification in Section 2.2, two transition variables may be defined, namely, C_r and C_f. Moreover, the following additional information can be extracted and applied to Equation (3.10) for each output Q and Q_p: For Q, $p_r = 1$; $p_f = 1$; $E_{1r} = C_r$; $E_{1f} = C_f$; $F_{1r} = J\,Q_c' + K'\,Q_c$; $F_{1f} = 0$; and $F_s = Q_c$. For Q_p, $p_r = 1$; $p_f = 1$; $E_{1r} = C_r$; $E_{1f} = C_f$; $F_{1r} = Q_c$;; $F_{1f} = Q_c$; and $F_s = Q_{pc}$.

Hence, the DOEs for the Petjk-FF are

$$Q_n = (JQ_c' + K'Q_c)C_r + Q_cC_r' \tag{3.11}$$

$$Q_{pn} = Q_cC_r + Q_cC_f + Q_{pc}C_r'C_f' \tag{3.12}$$

By Equations (3.7) and (3.8), the allowed outputs are 00 and 11, when C = 0 ($C_f = 1$ $C_f' = 0$, $C_r' = 0$, and $C_r' = 1$), and 00, 01, 10, and 11, when C = 1 ($C_r = 1$, $C_r' = 0$, $C_f = 0$, and $C_f' = 1$). So the states are grouped into 2 groups when C = 0 such as ($CJKQQ_p$) = (0--00) and (0--11), and 4 groups when C = 1 such as ($CJKQQ_p$) = (1--00), (1--01), (1--10), and (1--11) as shown in Fig. 3-4(c).

The number in each group are primitive states which are decimal representations of the input-output combinations. The state table is derived as Fig. 3-4(d) by applying above rules. In this table, state transitions only occur on the rising edge ($0 \rightarrow 1$) of the edge input C, and the adjacency diagram of this state table is a bipartite graph as shown in Fig. 3-4(e).

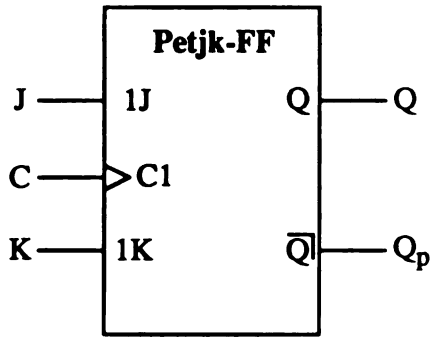
This state table has the following characteristics [35]:

(1) The stable states can be grouped into sets according to the values of the edge input variables.

(2) There are no internal state transitions among the states in the same set. In other words, there is no adjacencies between states which belong to the same group.

(3) Cycles may be generated by adding unstable states instead of modifying the existing unstable states.

These special characteristics are the basis of developing an efficient state assignment algorithm which will be discussed in the next chapter.



(a)

$$Q_n = (J\,Q_c' + K'\,Q_c)\,C_r + Q_c C_r'$$

$$Q_{pn} = Q_c C_r + Q_c C_f + Q_{pc} C_r' C_f'$$

(b)

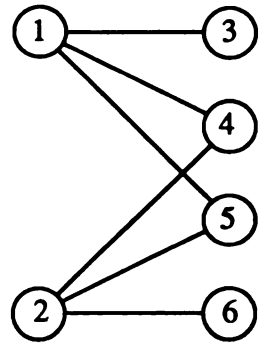
1 : (0, 4, 8, 12) **3** : (16, 20, 24, 28)
2 : (3, 7, 11, 15) **4** : (17, 21, 25, 29)
5 : (18, 22, 26, 30)
6 : (19, 23, 27, 31)

(c)

Present Input (CJK)

	000	001	011	010	110	111	101	100	Present Output (Q Qp)
1	(1)	(1)	(1)	(1)	5	5	3	3	0 0
2	(2)	(2)	(2)	(2)	6	4	4	6	1 1
3	1	1	1	1	(3)	(3)	(3)	(3)	0 0
4	1	1	1	1	(4)	(4)	(4)	(4)	0 1
5	2	2	2	2	(5)	(5)	(5)	(5)	1 0
6	2	2	2	2	(6)	(6)	(6)	(6)	1 1

(d)



(e)

Figure 3-4. Edge-triggered J-K bistable element with postponed output: (a) graphic symbol, (b) dynamic output equations, (c) state groups, (d) state table, and (e) adjacency diagram.

3.4 Circuit Decomposition

As mentioned in the previous sections, the edge inputs are used as transition variables to generate a set of DOEs to simplify the state merging process. Synthesis process can be further simplified if a circuit can be decomposed into many smaller sub-circuits. In CSLCs, a circuit is decomposed based on the topological connection of the graph representing the circuit, but not on the functional behavior. In this section, the mode inputs are used to achieve the circuit decomposition.

The mode inputs (M), as shown in Fig.3-1, do not cause internal state transitions to occur nor do they effect the internal state of the ASLC. Hence, the next state Y is not a function of M. If the output Z is a function of the mode inputs (M), the generalized ASLC model in Fig. 3-1 can be refined as shown in Fig. 3-5, referred to as a refined model, where the output Z is also a function of the present values of the L and E inputs. q_{1o} and q_{no} represent the number of bits in the current values of level input data words of NSG#1 and NSG#n, respectively, and r_{1o} and r_{no} represent the number of bits in the current values of edge input data words of NSG#1 and NSG#n, respectively. The p-bit mode inputs may be regarded as control signals for multiplexing the outputs of the NSGs. The selected next states from many NSGs may become the present output Z.

Note that the complexity of the synthesis process increases nearly exponentially with the circuit size. Thus, the complexity of synthesizing many smaller NSG circuits in total should be much lower than that of synthesizing a large NSG. It is clear that the use of mode inputs for circuit decomposition will make the ASLC synthesis process much simpler.

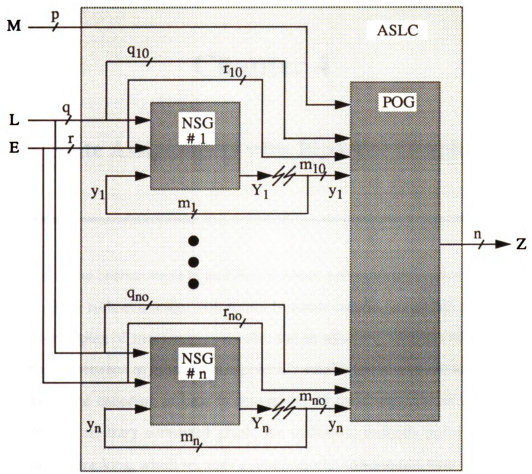


Figure 3-5. A refined model of an ASLC for mode inputs.

Chapter 4

State Assignments with Bipartite Graphs

Encoding the internal states of an ASLC has been a research topic since Huffman first introduced a flow table as a design tool. A race-free state encoding of an ASLC is a problem of finding a mapping of states onto an n-cube, and an adjacency diagram which describes the relationship between any two states is generally employed for state encoding. Since the complexity of the encoding process is determined by the complexity of the adjacency diagram, for an arbitrary connected graph, the traditional methods utilize enumerative searches or backtracking, which requires a combinatorial computation time.

Based on the salient features of bipartite graphical representations of both adjacency diagram and n-cube, this chapter presents an efficient state assignment algorithm using pattern matching techniques. First, the previous works on state assignments are briefly discussed. The bipartite representations of graphs are described in the following section, and the developed algorithm is presented in Section 4.3 with examples. Finally, comparisons with other algorithms are discussed in Section 4.4.

4.1 Background

Many techniques are developed to assign the states without races, but there always exist trade-offs between the goal of minimizing the number of state variables, and reducing the enumerative efforts. The classical state encoding methods are based on the enumerative way to assign the states to an n-cube which has the smallest dimension as possible, and to generate cycles to make them a race-free state encoding. Since the complexity of the circuit is dependent on the binary codes chosen in the state assignment step, this method may generate the most compact circuit realization.

More recent developments are based on the partition theory [36] associated with inputs. The Single Transition Time (STT) state assignment is the result of this approach. The column partition method is utilized to generate a Boolean matrix in the STT state assignment [22, 37]. It contributes to the reduction of the enumerative efforts to assign the states to an n-cube, but it requires additional efforts to find the minimum cover for the matrix to obtain the minimum number of state variables. Moreover, it requires more state variables than the classical ones. The multicode Single Transition Time (MSTT) state assignment [38, 39] is developed to reduce the dimension of an n-cube by allowing the doubly assigned codes for the different column partitions, but it still generally requires more state variables than the classical ones. Universal state assignments [39-44] which consider only the dimension of an n-cube according to the number of states to be assigned help reduce the enumeration efforts, but it is far from the compact circuit implementations. The one-hot code state assignment [45] requires the least efforts with the largest number of state variables to implement an ASLC. For compact circuit realization, the classical techniques may generate the best results, but they are difficult to apply to the large number of states because of their enumerative search requirements.

While recent techniques adopt the partition theory associated with inputs, the classical ones utilize an adjacency diagram [2], which visualizes the adjacency relationships between the states to be assigned. This adjacency diagram approach was regarded as impractical because it becomes more complex as the number of states increases, and enumerative efforts are required to find a race-free state encoding. However, with the development of graph theory it becomes more feasible to analyze the adjacency diagram using CAD tools.

Saucier [46] employed a partition method to get a maximal spanning tree, and devised an embedding algorithm to fit it onto an n-cube by employing backtracking techniques to find one possible embedding. The results are better than the covering method in terms of the dimension of the internal variable. Recently, Wu [14] utilized an adjacency diagram, and developed rules to predict races from a Node Weight Diagram (NWD) which is a binary n-cube connection diagram, but it requires an enumerative search to assign the states and to find the shortest path for generating cycles in an n-cube.

4.2 Bipartite Representation of Graphs

An adjacency diagram can be represented by a graph $G_a = \langle S_a, L_a \rangle$, where S_a is a set of nodes or states and a link $(u, v) \in L_a$ denotes that the states u and $v(\in S_a)$ have an adjacency relationship. The adjacency graph G_a may be considered as a set of states S_a and a function G_a: $S_a \times S_a \rightarrow \{0, 1\}$, such that $G_a(u, v) = G_a(v, u)$ and $G_a(u, u) = 0$ for all u, $v \in S_a$. $G_a(u, v) = 1$ means that there is an adjacency relationship between u and v, i.e., the pair $(u, v) \in L_a$.

The graph of an n-cube can be denoted by $G_n = \langle V_n, E_n \rangle$, where V_n is the set of vertices and E_n, the edges, represent the interconnection pattern of the nodes. The graph of

an n-cube G_n may be considered as a set of vertices V_n and a function G_n: $V_n \times V_n \rightarrow \{0, 1\}$. Without loss of generality, we may assume that $|S_a| \leq |V_n|$.

A bipartite graph is a graph whose vertex set V can be partitioned into two subsets V_r and V_c such that no edge in the graph joins two vertices in the same subset. Thus, each edge in the bipartite graph has one end in V_c and the other end in V_r. Every cycle in a bipartite graph contains an even number of edges. This follows since each edge joins vertices in different subsets and the cycle must return to its subset in which it originated [47].

Two graphs G_1 and G_2 are *isomorphic* to each other if there is a one-to-one correspondence between their vertices and between their edges such that the number of edges joining any two vertices in G_1 is equal to the number of edges joining the corresponding two vertices in G_2. This may be stated more formally: Two graphs G_1: $V_1 \times V_1 \rightarrow \{0, 1\}$ and G_2: $V_2 \times V_2 \rightarrow \{0, 1\}$ with $|V_1| = |V_2|$ are isomorphic if there exists a function e: $V_1 \rightarrow V_2$ such that $G_1(u, v) = G_2(e(u), e(v))$ for all $u, v \in V_1$.

The problem of determining whether two graphs are isomorphic is one of the classical unsolved combinatorial problems [48]. The state assignment problem, finding a mapping function f_m: $S_a \rightarrow V_n$, is computationally equivalent to the graph isomorphic problem. No exact polynomial time algorithm exists for the problem and they are solved approximately using heuristic algorithms.

4.2.1 Bipartite Adjacency Table (BAT)

A bipartite adjacency diagram is a connected graph whose states are partitioned into two subsets X_c and X_r such that no link in the graph joins two states in the same subset. For simplicity, the states in X_c and X_r are denoted as S_i's and s_j's, respectively, i.e., $X_c=\{S_1, S_2,..., S_l\}$ and $X_r=\{s_1, s_2,..., s_m\}$, where l and m are the numbers of states in X_c and X_r,

respectively. Without loss of generality, we assume that $l \leq m$. Let L be the set of all links between X_c and X_r, i.e., $L=\{(S_i, s_j) | S_i \in X_c \text{ and } s_j \in X_r\}$.

Based on the characteristics of the state table, the links in a bipartite adjacency table (BAT) can be classified as **adjacency links** and **bipartite links** as follows:

(1) *Adjacency link:* It represents state transitions in the state table. A pair of states related to each link respectively belong to the distinct state groups.

(2) *Bipartite link*: It represents the state transition between a pair of states according to the values of edge inputs regardless of other input changes.

Note that a bipartite link may also be considered as an adjacency link, but the bipartite link cannot be a break to generate cycles. Let's consider the edge-triggered J-K bistable element with postponed output discussed in Section 3.3. Since the adjacency diagram of the state table is a bipartite graph, it can be represented with the table as shown in Fig. 4-1(c). In this table, (1, 3), (1, 5), (2, 4), and (2, 6) are the adjacency links, and (1, 3), (1, 4), (2, 5), and (2, 6) are the bipartite links. The links (1, 3) and (2, 6) are bipartite links as well as adjacency links.

4.2.2 Bipartite Representation Table (BRT)

An n-cube is a subset of a hypercube and its interconnection structure is difficult to visualize when $n \geq 4$. Since an n-cube is always bipartite [49], the interconnection structure of an n-cube can be represented by a table as described below. For simplicity, we first consider when n = 3. Fig. 4-2(a) illustrates a 3-cube structure. A 3-cube is comprised of two 2-cubes and the links between them. There exist three sets of separated 2-cubes: (a, c | b, d) and (f, h | e, g); (a, f | e, b) and (h, c | d, g); and (a, h | d, e) and (c, f | b, g). Consider the first representation. In addition to the two cubes, there exists four links between these two 2-cubes. They are (a, e), (c, g), (f, b), and (h, d). The 3-cube structure can be represented by

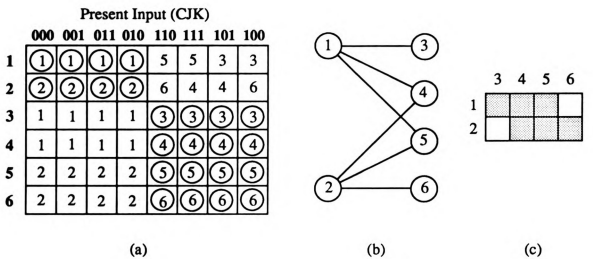


Figure 4-1. Edge-triggered J-K bistable element with postponed output: (a) state table, (b) adjacency diagram, and (c) bipartite adjacency table (BAT).

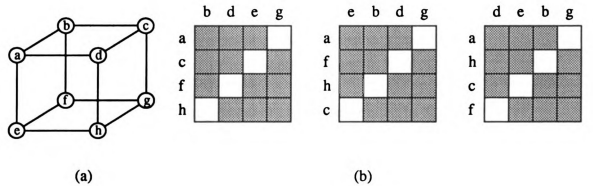


Figure 4-2. Illustration of bipartite representations of 3-cube: (a) geometric symbol, and (b) three different representations of 3-BRT.

the leftmost bipartite table in Fig. 4-2(b), referred to as a bipartite representation table (BRT), where the shaded block represents the existence of a link between two nodes and the blank block indicates no link. Similarly, the BRTs for the 3-cube with the remaining two 2-cube representations are also shown in Fig. 4-2(b). In general, an n-cube is comprised of two (n-1)-cubes and the links between them. Fig. 4-3 illustrates various BRTs. The bold typed numbers are decimal numbers of binary vectors assigned to the states in the row and column of the BRT. The BRT for an n-cube, referred to as n-BRT, is constructed by the following recursive algorithm.

Procedure 4.2.2 (n-BRT Generation)

Input: Dimension of n-cube. (n)
Output: Ordered lists of column nodes (X_c) and row nodes (X_r).

```
Gen_BRT(n) {
    If n > 1
        Gen_BRT(n-1)
        If n is odd
            Add complement of each element in Xc to Xr;
            Add complement of each element in Xr to Xc;
        Else
            Add complement of each element in Xc to Xc;
            Add complement of each element in Xr to Xr;
        Ascending ordering of Xc and Xr according to the decimal value.
    Else
        0 → Xc ; 1 → Xr
    Return (Xc, Xr)
        }
```

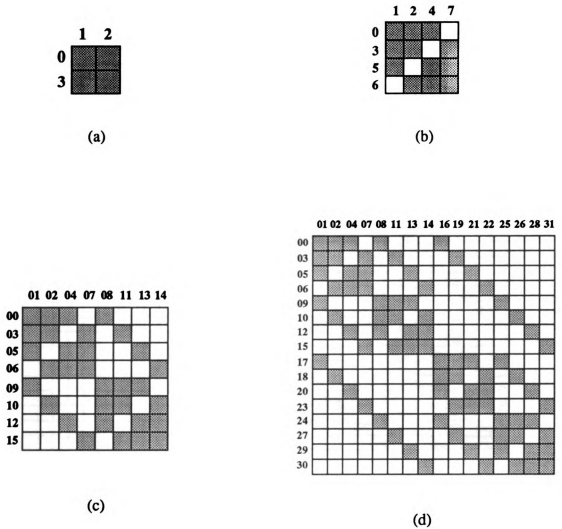
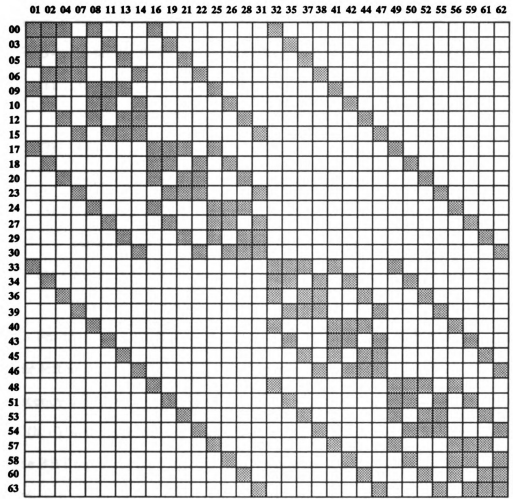


Figure 4-3. Bipartite representation table (BRT) for (a) 2-cube, (b) 3-cube, (c) 4-cube, (d) 5-cube, and (e) 6-cube.



(e)

Figure 4-3. (Continued)

Note that each node in X_c (or X_r) of (n-1)-BRT is represented in (n-1) bits, and the expanded bit for the n-BRT is considered as "0". The memory size for such a representation requires 2^n for an n-BRT. For simplicity of discussion, the links in an n-BRT are classified as follows: **bipartite links, adjacency links, boundary links,** and **forbidden links.**

(1) *Bipartite link:* It represents an isolated edge in an n-cube, which does not share nodes between them. It appears in the diagonal of an n-BRT.

(2) *Adjacency link*: It represents an edge which connects the nodes which are included in the two separated edges.

(3) *Boundary link*: It is an edge in an n-cube which connects the nodes of two separated (n-1)-cubes included in the n-cube.

(4) *Forbidden link*: It is an imaginary link between the nodes which have no edges. This link is generally preserved for any bipartite representations of n-cubes.

For simplicity, the first three links are referred to as *non-forbidden links*. For example, in the leftmost BRT in Fig. 4-2(b), the bipartite links are (a, b), (c, d), (f, e), and (h, g), the adjacency links are (a, d), (c, b), (f, g), and (h, e), the boundary links are (a, e), (c, g), (f, b), and (h, d), and the forbidden links are (a, g), (c, e), (f, d), and (h, b). The unshaded blocks in the BRT represent the forbidden links, while the shaded blocks are for the non-forbidden links.

4.3 State Assignments Using Bipartite Graph Techniques

Based on the construction of an n-BRT, where $n \geq 2$, each node in X_c (or X_r) has n non-forbidden links, i.e., one bipartite link, (n-2) adjacency links, one boundary link, and $(2^{(n-1)}-n)$ forbidden links. Thus, an n-BRT contains $2^{(n-1)}$ bipartite links. An n-BRT can be partitioned into four quadrants: Quadrant I (Upper-left), Quadrant II (Upper-right),

Quadrant III (Lower-right), and Quadrant IV (Lower-left). Quadrant I (III) contains an (n-1)-BRT, while Quadrant II (IV) consists of boundary links.

Note that an n-BRT contains two (n-1)-BRTs. If we recursively bisect the BRT, the final component will be a 2-BRT. Thus, the basic component of an n-BRT is a 2-BRT for the 2-cube. It should be mentioned that the states in X_c (or X_r) can be partitioned into two state groups and that the number of states in one group which share links with one state in the other group is exactly (n-1) and they share exactly two non-forbidden links. In the following three properties of an n-BRT, we consider a state pair (S_1, S_2) in X_c and a state pair (s_1, s_2) in X_r.

Property 4.1 Suppose that both state pairs form a 2-cube. Then,

(1) S_1 and S_2 are in the same state group in X_c if and only if s_1 and s_2 are in the same state group in X_r; or

(2) S_1 and S_2 are in different state groups in X_c if and only if s_1 and s_2 are in different state groups in X_r.

Consider the 4-BRT in Fig. 4-3(c), there are two 3-BRTs, and the states are grouped as (0, 3, 5, 6) and (9, 10, 12, 15) for X_c, and (1, 2, 4, 7) and (8, 11, 13, 14) for X_r. The links (3, 1), (3, 11), (9, 1), and (9, 11) form a 2-cube with (3, 9 | 1, 11). The states 3 and 9 belong to the different state groups, so do the states 1 and 11.

Property 4.2 If (S_1, s_2) and (S_2, s_1) are two boundary links located in Quadrants II and IV, respectively, then (S_1, s_1) and (S_2, s_2) are the same type of links located in Quadrants I and III, respectively.

For example, consider the 4-BRT in Fig. 4-3(c). Since the links (5, 13) and (10, 2) are two boundary links located in Quadrants II and IV, respectively, both (5, 2) and (10, 13) are forbidden links. Similarly, the boundary links (10, 2) and (6, 14) result that both (10, 14) and (6, 2) are adjacency links. Finally, both (0, 1) and (9, 8) are bipartite links because both (0, 8) and (9, 1) are boundary links.

Property 4.3 If both (S_1, s_1) and (S_2, s_2) are bipartite links, then both (S_1, s_2) and (S_2, s_1) have the same type of links.

In the 4-BRT in Fig. 4-3(c), since (0, 1) and (6, 7) are bipartite links, both (6, 1) and (0, 7) are forbidden links. On the other hand, since (0, 1) and (5, 4) are bipartite links, both (0, 4) and (5, 1) are adjacency links.

4.3.1 Problem Statement

The race-free state assignment problem can be formulated as the problem of embedding a bipartite connected graph in an n-cube. In this implementation, the problem can be formulated as a mapping of the adjacency diagram, represented with a bipartite graph, to an n-cube also represented with a bipartite graph, i.e., embedding of a BAT in a BRT. A BAT is generally comprised of l rows and m columns, where $l \leq m$. Since we always can expand the BAT by adding some dummy states to increase the number of rows, without loss of generality, we consider the case that $m = l$. Thus, we consider the BAT having 2m states, referred to as a m-BAT, for simplicity.

Definition 4.1 A m-BAT is ***mappable*** if it can be embedded in an n-BRT, where $n = \lceil log_2 m \rceil + 1$. Otherwise, it is ***unmappable***.

Definition 4.2 The *m*-BAT is ***m*-bisectable** if

(1) The states in X_c and X_r can be partitioned into two pairs of state groups, (X_{c1}, X_{c2}) and (X_{r1}, X_{r2}), respectively;

(2) For every state in X_{c1}, there exists at most one link to a state in X_{r2}, where no states in X_{r2} have linked the same state in X_{c1}; and

(3) For every state in X_{c2}, there exists at most one link to a state in X_{r1}, where no states in X_{r1} have linked the same state in X_{c2}.

Note that the m-BAT is partitioned into four blocks. For simplicity, both (X_{c1}, X_{r1}) and (X_{c2}, X_{r2}) are referred to as diagonal blocks, while both (X_{c2}, X_{r1}) and (X_{c1}, X_{r2}) are called as off-diagonal blocks. By Definition 4.2, the states in each diagonal block form a *l*-BAT, where $l = \lceil log_2 m \rceil$. Again, if the *l*-BAT satisfies the conditions in Definition 4.2, then it is *l*-bisectable. Therefore, if the states in both X_c and X_r are properly partitioned so that the conditions in Definition 4.2 can be satisfied by a newly generated *k*-BAT, $k \leq m$, then the *k*-BAT is *k*-bisectable.

Definition 4.3 A m-BAT is **bisectable** if the states in X_r and X_c can be partitioned in such a way that all the corresponding k-BATs, $k = l,..., 2$, are *k*-bisectable.

Theorem 4.1 A bisectable m-BAT is mappable.

Proof: Since the m-BAT is bisectable, it is also m-bisectable. By Definition 2, both X_c and X_r can be partitioned into two pairs of state groups, (X_{c1}, X_{c2}) and (X_{r1}, X_{r2}), respectively, where the links in each off-diagonal block can be treated as the boundary links of an n-BRT, where $n = \lceil \log_2 m \rceil + 1$. Since the diagonal blocks are *l*-BAT, they are *l*-bisectable, where $l = \lceil \log_2 m \rceil$. Similarly, the links in each corresponding off-diagonal block are the boundary links of a (n-1)-BRT. The BAT formed by the states in each diagonal block can

be further partitioned until all the diagonal blocks are 2-BATs each of which can be mapped to a 2-BRT. This concludes that a bisectable m-BAT is mappable to an n-BRT.

Q.E.D.

Corollary 4.1 A m-BAT is mappable if it consists of bipartite links only.

Proof: A m-BAT with bipartite links only shows that, for every state in X_c, there exists exactly one link to a state in X_r, and vice versa. Thus, the states in X_c and X_r can be permuted so that the links are all on the diagonal entries of the BAT. This implies that the m-BAT is bisectable. By Theorem 4.1, the m-BAT is mappable. *Q.E.D.*

Consider the 7-BAT in Fig. 4-4(a), for example, where X_c and X_r are partitioned into two ordered pairs of state groups, {(1, 7, 5, 6), (3, 4, 2)} and {(10, 11, 13, 9), (8, 14, 12)}, respectively. Based on Definitions 4.2 and 4.3, the 7-BAT is bisectable and thus is

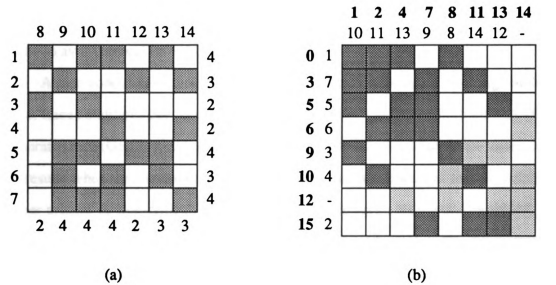


Figure 4-4. An example illustrating Theorem 4.1: (a) mappable 7-BAT and (b) depiction of a mapping to 4-BRT. (The shaded blocks in (b) represent links in 4-BRT, and the darker blocks indicate the mapped links of 7-BAT)

mappable. Fig. 4-4(b) depicts the mapping results and the corresponding state encoding.

Theorem 4.2 A m-BAT is unmappable if either

(1) Each row or column of the m-BAT contains more than n links, or

(2) The link set of the m-BAT includes $\{(S_i, s_j) \mid i=1, 2, 3 \text{ and } j=1, 2\}$ or $\{(S_i, s_j) \mid i=1, 2 \text{ and } j=1, 2, 3\}$.

Proof: Since each node, or state, in X_c and X_r of n-BRT has only n links, it follows that the m-BAT is unmappable. On the other hand, since any two rows or columns in a n-BRT share at most two links, the m-BAT containing more than three common links is thus unmappable. *Q.E.D.*

An ummapable m-BAT cannot be mapped to an n-BRT, where $n = \lceil log_2 m \rceil + 1$, i.e., the 2m states in the m-BAT cannot be encoded with n state variables. In other words, encoding the 2m states with n state variables will definitely create critical races when the BAT is unmappable. Therefore, it is necessary to add cycles by adding some intermediate states to avoid critical races [1-6].

A race condition occurs when two or more state variables are to change at the same time. Race conditions can naturally be classified as being either an intrinsic race (IR), or a generated race (GR) [14, 25]. A GR is caused by a careless encoding of the states, while an IR results when the minimum possible Hamming distance is greater than 1. A GR can be either critical or non-critical. Two types of intrinsic races have been identified: visible intrinsic race (VIR) and hidden intrinsic race (HIR). A VIR occurs when the maximum link degree exceeds the number of state variables or the corresponding adjacency diagram has at least one loop of odd number of states. Otherwise, it is a HIR. In [14], race conditions are avoided by first identifying and eliminating IRs and then making race-free state assignments to guarantee that no GRs are produced. This can be achieved by encoding the

adjacent states to have a Hamming distance equal to 1. This state assignment is referred to as unit-distance code (UDC) state assignments. To facilitate the UDC state assignments, the NWD is implemented. Note that some intermediate states may be added or some unstable states may be modified to generate cycles for avoiding races. Results in [14] have demonstrated that the algorithm provides better results than any others. However, the algorithm uses an enumerative search approach to assign the states and to generate cycles by exhaustively trying all possible encodings.

In this study, based on the special bipartite characteristics of an adjacency diagram, the conditions in Theorems 4.1 and 4.2 identify the mappability of a BAT under consideration. If a BAT is identified as mappable, the resultant mapping is the assignment of the states. On the other hand, if it is not mappable, some cycles are generated. The following theorem provides an alternative rule to identify the mappability of the modified BAT.

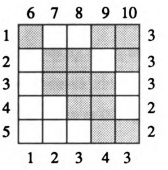
Definition 4.4 For the link (S_i, s_t) in a BAT, we define

(1) $\text{adj}(S_i) = \{(S_i, s_k) \mid (S_i, s_k) \in L, s_k \in X_r, \text{ and } s_k \neq s_t\}$,

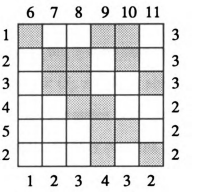
(2) $\text{adj}(s_t) = \{(S_k, s_t) \mid (S_k, s_t) \in L, S_k \in X_c, \text{ and } S_k \neq S_i\}$, and

(3) $\langle \text{adj}(S_i), \text{adj}(s_t) \rangle = \{(S_k, s_l) \mid (S_k, s_l) \in L, S_k \in X_c, s_l \in X_r, \text{ and } \text{adj}(\text{adj}(S_i)) = \text{adj}(\text{adj}(s_t))\}$.

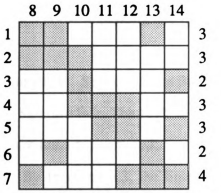
In the BAT shown in Fig. 4-5(a), L = {(1, 6), (1, 9), (1, 10), (2, 7), (2, 8), (2, 10), (3, 7), (3, 8), (3, 9), (4, 8), (4, 9), (5, 9), (5, 10)}. For the link (2, 7), adj(2) = {(2, 8), (2, 10)}, and adj(7) = {(3, 7)}, and ⟨adj(2), adj(7)⟩ = {(3, 8)}.



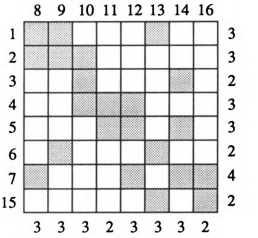
(a)



(b)



(c)



(d)

Figure 4-5. An example illustrating Theorem 4.3 and Corollary 4.3: (a) 5-BAT which has a pattern defined in Theorem 4.3, (b) expanded BAT of (a), (c) 7-BAT which has a pattern defined in Corollary 4.3, and (d) expanded BAT of (c).

Theorem 4.3 Suppose that a BAT contains the following adjacency links: (S_i, s_r), (S_i, s_t), (S_j, s_r), (S_j, s_t), (S_j, s_u), (S_k, s_t), and (S_k, s_u). The BAT is unmappable if there exists more than one link such as $\langle adj(S_i), adj(s_u)\rangle$ or $\langle adj(S_k), adj(s_r)\rangle$.

Proof: The states S_i, S_j, S_k and s_r, s_t, s_u can be mapped to the 3-cube domain, and they generate two 2-cubes which are interconnected with the link (S_j, s_t). They also have two forbidden links such as (S_i, s_u) and (S_k, s_r). By Property 4.2, the states belonging to the 3-cube have only one adjacency link and one boundary link, so only one link is allowed, which is a boundary link such as either $\langle adj(S_i), adj(s_u)\rangle$ or $\langle adj(S_k), adj(s_r)\rangle$, but not both. When considering a mapping of the BAT to an n-BRT, where n > 3, by Property 4.2, the links in $\langle adj(S_i), adj(s_u)\rangle$ should be a forbidden link when $adj(S_i)$ exists outside of the 3-cube. The same condition can be applied to $\langle adj(S_k), adj(s_r)\rangle$. Therefore, other links except one which is either $\langle adj(S_i), adj(s_u)\rangle$ or $\langle adj(S_k), adj(s_r)\rangle$ should be forbidden links.

Q.E.D.

Corollary 4.3 Suppose that a BAT does not satisfy Theorem 4.1 and 4.2, and the link (S_i, s_j) is a bipartite link. The BAT is unmappable if there exist links such as $\langle adj(S_i), adj(s_j)\rangle$, which are not a bipartite link.

Proof: By Property 4.3, the links in $\langle adj(S_i), adj(s_j)\rangle$ should be a bipartite link, if they exist. *Q.E.D.*

Consider a BAT in Fig. 4-5(a) which satisfies the condition in Theorem 4.3. There is a pattern between the column state group (2, 3, 4) and row state group (7, 8, 9). The links (1, 9) and (5, 9) are adj(9), and the link (2, 10) is adj(2). Since there exist two links such as (1, 10) and (5, 10), which are $\langle adj(2), adj(9)\rangle$, this BAT is unmappable. The expanded BAT is shown in Fig. 4-5(b) by breaking the link (3, 9) and eliminating the pattern defined in Theorem 4.3.

The BAT in Fig. 4-5(c) shows the pattern defined in Corollary 4.3. The link (7, 8) is adj(8), and the link (1, 13) is adj(1). The link (7, 13) should be a bipartite link for a mappable BAT because the link (1, 8) is a bipartite link and the link (7, 13) is the link ⟨adj(1), adj(8)⟩. The expanded BAT in Fig. 4-5(d) is generated by eliminating this link, and it becomes a mappable BAT.

4.3.2 State Assignment Algorithm

The developed race-free state assignment algorithm is divided into three major procedures: The first procedure examines if the given BAT is mappable. If it is not mappable to an n-cube, the second procedure eliminates the races by expanding the table with additional states and generating cycles. The third procedure maps a mappable BAT to an n-cube. The first procedure applies a sequence of rules to predict the races which make the BAT to be unmappable. When the BAT is identified as unmappable, rules are applied to determine a break which should be eliminated to generate a mappable BAT. An unmappable BAT is expanded according to the characteristics of the BAT by two different procedures: symmetric expansion and non-symmetric expansion. Since the n-BRT is symmetric on the basis of bipartite links in nature, a symmetric unmappable BAT needs a special expansion, which makes the expanded BAT to be symmetric again. The third procedure applies constraints to the states in the BAT to find a mapping of an n-cube. To find a mapping is to find two independent sets of states recursively. Detailed description of each procedure is as follows.

4.3.2.1. Procedure to identify a mappable BAT

The procedure for identifying a mappable BAT is divided into two parts. The first part is to identifying an unmappable BAT. It begins by counting the link degrees in the column or row of a given BAT and all 2-cubes in the BAT are searched to generate a list. The unmappable condition (1) and (2) in Theorem 4.2 and Theorem 4.3 are examined next. All required breaks between the separated and intersected 2-cubes are examined in the second part to determine breaks for expanding the unmappable BAT. All links are weighted when they should be breaks, and two of the most heaviest links are selected as breaks which should be eliminated by generating cycles. The links and required breaks between the separated and intersected 2-cubes are shown in Fig. 4-6. The procedures to identify an unmappable BAT and to generate a mappable BAT by determining breaks are described as follows.

Procedure 4.3.2.1.1 (Unmappable BAT Identification)

Input: BAT matrix. (m_0 by m_0)

Outputs: Ordered list of link degrees of states in X_c and X_r.
2-cubes list

Identify_Unmappable_BAT (BAT) {

1. Count the link degree of states in X_c and X_r.
2. Search all 2-cubes to generate 2-cubes list.
3. IF the maximum link degree is greater than n, THEN go to Step 8, where $n = \lceil log_2 m_0 \rceil + 1$.
4. Search the pattern defined in Theorem 4.2(2) from the 2-cubes list.
   4.1 IF it is found, THEN go to Step 8.
5. Search the pattern defined in Theorem 4.3 from the 2-cubes list.
   5.1 Count the number of links in ⟨adj(i), adj(j)⟩ for each forbidden link (i, j) in the defined blocks.
   5.2 IF it is more than 1, THEN go to Step 8.

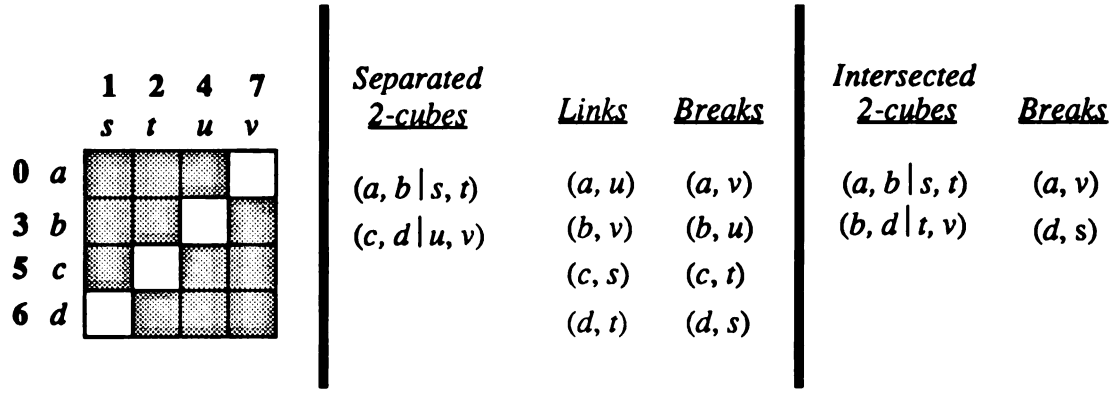


Figure 4-6. Links and breaks in two separated 2-cubes and two interconnected 2-cubes.

6. Find the ⟨adj(i), adj(j)⟩ for each bipartite link (i, j).
   6.1 IF they are not bipartite links, THEN go to Step 8.
7. End. Next is to map the BAT to an n-BRT (See Procedure 4.3.2.3).
8. Generate a mappable BAT (See Procedure 4.3.2.1.2).

}

Procedure 4.3.2.1.2 (Break Determination)

Input: BAT matrix. (m_0 by m_0)
2-cubes List

Outputs: 2-cubes List (modified)
Breaks List

Break_Determination (BAT) {

1. Find the breaks in the separated 2-cubes:
   1.1 Find the maximum number of separated 2-cubes in the 2-cubes list.
   1.2 IF there are no separated 2-cubes, THEN go to Step 2.
   1.3 Mark all required breaks for the separated 2-cubes.
   1.4 Eliminate all 2-cubes in the 2-cubes list which have required breaks.
2. Find the breaks in the intersected 2-cubes:
   2.1 Find all interconnected 2-cubes in the list.
   2.2 IF there are no intersected 2-cubes, THEN go to Step 3.

2.3 Mark all required breaks for the interconnected 2-cubes.
2.4 Eliminate all 2-cubes in the 2-cubes list which have required breaks.
3. IF the break is a bipartite link in the BAT, THEN remove it from the break list.
4. Arrange the breaks in descending order according to their weights.
5. Select first two breaks from the list. Next is to expand the BAT by adding additional states to the BAT. (See Section 4.3.2.2)

}

4.3.2.2. Procedure for BAT expansion

The cycles can be easily generated by adding states and by adding links between the states which have breaks. However, the expansion of the BAT should be symmetric if the original BAT is symmetric. The symmetric BAT is recognized from the order of the link degrees in the X_c and X_r. If the order of link degrees of X_c is the same as the order of link degrees of X_r after arranging the link degrees, the BAT is symmetric. The symmetric expansion is by counting the number of forbidden links in the BAT. If non-symmetric expansion is applied to the symmetric BAT, the BAT cannot be mapped onto an n-cube or results in having a larger dimension of an n-cube.

For example, consider BATs in Fig. 4-7(b) and (c). Both BATs are symmetric on the basis of bipartite links for $m \leq 3$. Both BATs are expanded from the BAT in Fig. 4-7(a), which is symmetric on the basis of bipartite links for $m \leq 4$. Since the BAT in Fig. 4-7(c) has a symmetric expansion, it is also symmetric on the basis of bipartite links. However, the BAT in Fig. 4-7(b) is not symmetric because it has an non-symmetric expansion. The BAT in Fig. 4-7(c) is mappable to 4-BRT as depicted in Fig. 4-7(d), but the BAT in Fig. 4-7(b) is unmappable because it has a pattern defined in Theorem 4.3 with states groups (3, 4, 9) and (7, 8, 10), and the link (1, 5) is ⟨adj(9), adj(7)⟩.

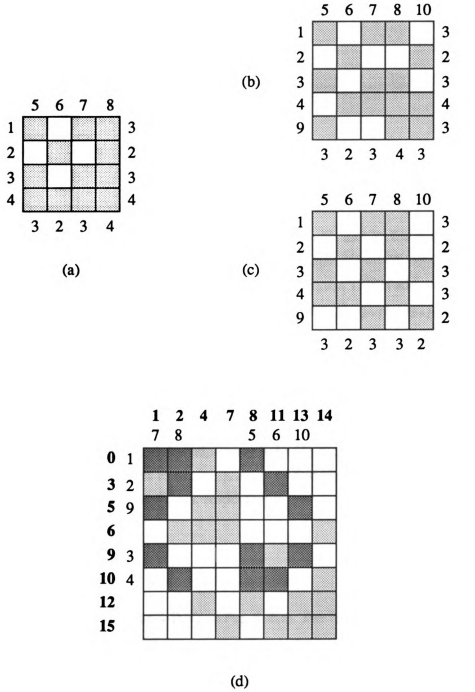


Figure 4-7. An example of BAT expansions: (a) 4-BAT, (b) non-symmetric expansion of (a), (c) symmetric expansion of (a), and (d) depiction of a mapping of (c) to 4-BRT.

The non-symmetric and symmetric BAT expansion procedures are as follows:

Procedure 4.3.2.2.1 (Non-symmetric Expansion)

Inputs: BAT matrix. (k by k)
Breaks List.

Outputs: BAT matrix. (k+1 by k+1)
Cycles List.

Non-symmetric_Expansion(BAT){

1. Generate a cycles by adding links in X_c and X_r for two breaks whose one state in each pair is the same:
   1.1 IF two breaks are (S_i, s_j) and (S_i, s_l), THEN add one bipartite link (S_{k+1}, s_{k+1}) and adjacency links (S_i, s_{k+1}), (S_{k+1}, s_j), and (S_{k+1}, s_l).
   1.2 IF two breaks are (S_a, s_j) and (S_b, s_j), THEN add one bipartite link (S_{k+1}, s_{k+1}), and adjacency links (S_a, s_{k+1}), (S_b, s_{k+1}), and (S_{k+1}, s_j).
2. IF there is only one break such as (S_i, s_j), THEN add one bipartite link (S_{k+1}, s_{k+1}) and adjacency links (S_i, s_{k+1}), and (S_{k+1}, s_j).
3. List cycles as $(S_i, s_{k+1}, S_{k+1}, s_j)$ and $(S_i, s_{k+1}, S_{k+1}, s_l)$ for the breaks (S_i, s_j) and (S_i, s_l), respectively.

Return (BAT)

}

For example, consider the 4-BAT in Fig. 4-8(a). The BAT is not symmetric. Since it has a pattern defined in Theorem 4.2, it has races. In this example, the break (2, 8) is selected as a break for BAT expansion. The expanded BAT with a cycle (2, 10, 9, 8) is shown in Fig. 4-8(b), and the resulting mapping to 4-BRT is depicted in Fig. 4-8(c).

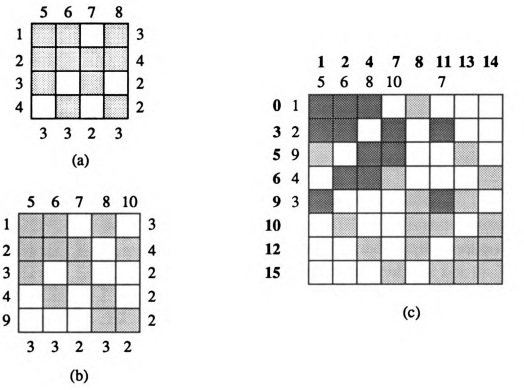


Figure 4-8. An example of non-symmetric BAT expansion: (a) non-symmetric 4-BAT, (b) 5-BAT with non-symmetric expansion, and (c) depiction of a mapping to 4-BRT.

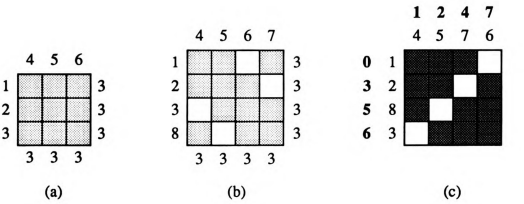


Figure 4-9. An example of symmetric BAT expansion: (a) symmetric 3-BAT, (b) 4-BAT with symmetric expansion, and (c) depiction of a mapping to 3-BRT.

Procedure 4.3.2.2.2 (Symmetric Expansion)

Inputs: BAT matrix. (k by k, where $m \geq k \geq m_0$)
Breaks List.

Outputs: BAT matrix. (k+1 by k+1, where $m \geq k \geq m_0$)
Cycles List.

Symmetric_Expansion(BAT){

1. Find two breaks which are symmetric to each other:
   1.1 IF there is one break (S_a, s_j), THEN the symmetric break (S_b, s_l) is such that the order of state S_a in the X_c is the same as the order of the state s_l in the X_r, and the order of state s_j in the X_c is the same as the order of the state S_b in X_r.
   1.2 IF there is no symmetric break for (S_a, s_j) in the breaks list, THEN add the symmetric break (S_b, s_l) to the breaks list.
2. Generate cycles for two symmetric breaks:
   2.1 IF two symmetric breaks are (S_a, s_j) and (S_b, s_l), THEN add one bipartite link (S_{k+1}, s_{k+1}), and four adjacency links such as (S_a, s_{k+1}), (S_{k+1}, s_j), (S_b, s_{k+1}), and (S_{k+1}, s_l).
   2.2 List cycles as $(S_a, s_{k+1}, S_{k+1}, s_j)$ and $(S_b, s_{k+1}, S_{k+1}, s_l)$.

Return (BAT)
}

For example, consider a BAT in Fig. 4-9(a). This BAT is symmetric, but it is unmappable because it has the pattern defined in the Theorem 4.2. It requires two breaks such as (1, 6) and (3, 4), which are symmetric to each other. The expanded BAT is shown in Fig. 4-9(b) with cycles (1, 7, 3, 6) and (3, 7, 8, 4). The resultant mapping to 3-BRT is depicted in Fig. 4-9(c).

4.3.2.3. Procedure to map a mappable BAT to an n-BRT

Once a BAT is identified as mappable with or without BAT expansions, there is no race in the BAT, and it is bisectable from Theorem 1. Since the bisecting procedure requires not only to partition the states into two pairs of state groups recursively, but also to satisfy the condition that each state in one group of X_c (or X_r) has at most one boundary link to the state in the other group of X_r (or X_c), the mapping procedure begins first by partitioning the states in X_c (or X_r) into 2 subsets recursively until it becomes a 2-cube with applying the following two constraints. They are derived from the properties of an n-BRT.

Constraint 4.3.1 Suppose that there is a 2-cube such as $(S_1, S_2 \mid s_1, s_2)$ in a mappable BAT and that states in X_c is partitioned into X_{c1} and X_{ck}, and states in X_r is partitioned into X_{r1} and X_{rk}. Then,

(1) IF $S_1, S_2 \in X_{c1}$, THEN $s_1, s_2 \in X_{r1}$.
(2) IF $S_1, S_2 \in X_{ck}$, THEN $s_1, s_2 \in X_{rk}$.
(3) IF $S_1 \in X_{c1}$, $S_2 \in X_{ck}$, and $s_1 \in X_{r1}$, THEN $s_2 \in X_{rk}$.
(4) IF $S_1 \in X_{c1}$, $S_2 \in X_{ck}$, and $s_1 \in X_{rk}$, THEN $s_2 \in X_{r1}$.

Constraint 4.3.2 Suppose that states in X_c is partitioned into X_{c1} and X_{ck}, and states in X_r is partitioned into X_{r1} and X_{rk}. The state S_k (or s_k) in X_{ck}(or X_{rk}) should have at most one link with at most one state in X_{r1} (or X_{c1}).

Whenever there exist multiple choices, the selection refers to above constraints. The second part of the procedure arranges the order of states in each subset according the constraints of the boundary links, and merges the ordered subsets to expand the state space to an n-cube.

The partition process can be executed by moving some states in the original state set to the new subset by applying some selection constraints. The link degree of each state is the selection criterion in this procedure. Let one state S_k is first selected from X_c, which has the maximum link degree with the states in X_r, then the state s_k separated from the states in X_r should be a state which has a link with S_k, but it has the minimum link degree with the states in X_c. When the state s_k is moved to the new subset of X_r, then the states in X_c, except S_k, which have links with s_k should be moved to the new subset of X_r because of Constraint 4.3.2. By repeating this process, the states which satisfy the bisectable condition is moved from the original set to the new subset, and it results in partitioning the original states into 2 subsets. This partition process continues until the number of states in the original set becomes 2, in which boundary links cannot be applied. Once the states in X_c and X_r form a 2-cube, the state space is now expanded from a 2-cube to an n-cube by arranging the order of states in each subset and merging those ordered subsets. The ordering of the states in each subset is executed by considering the boundary links between (n-1)-BRTs. The detailed mapping procedure is described as follows:

Procedure 4.3.2.3 (Mapping a mappable BAT to an n-BRT)

Inputs: BAT matrix. (m by m)
2-cubes List.

Outputs: List of assigned numbers for column states
List of assigned numbers for row states

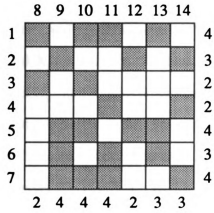
BAT_to_ BRT_mapping (BAT) {

1. Assign the column (or row) states set to X_{c1}, which has the maximum link degree, and assign the other states set to X_{r1}. Set i = 1 and k = 1.
2. Select S_{pk} among the unmarked states in X_{c1} such that $|adj(S_{pk})|$ is the maximum, where $p = 2^{(n-2-i)} + 1$.
3. Select s_{pk} among the unmarked states in X_{r1} such that $adj(s_{pk}) \in \{adj(S_{pk})\}$ and $|adj(s_{pk})|$ is the minimum.

4. $X_{rp} = X_{rp} + s_{p1}$, $X_{r1} = X_{r1} - s_{p1}$, $X_{cp} = X_{cp} + \{S \mid S \in X_{c1}, adj(S) = adj(s_{p1}), S \neq S_{p1}\}$, and $X_{c1} = X_{c1} - \{S \mid S \in X_{c1}, adj(S) = adj(s_{p1}), S \neq S_{p1}\}$.
5. IF the states in X_{rp} and X_{r1} (X_{cp} and X_{c1}) cannot satisfy Constraint 4.3.1, THEN
   5.1 Mark the state S_k.
   5.2 Move all states in X_{rp} and X_{cp} to X_{r1} and X_{c1}, respectively.
   5.3 Go to Step 2.
6. Repeat Step 2 to 5 with increasing k until $| X_{c1}|$ and $| X_{r1}| \leq 2^{(n-1-i)}$.
7. IF the states in X_{rp} and X_{r1} (or X_{cp} and X_{c1}) cannot satisfy Constraint 4.3.2, THEN
   7.1 Mark the states in X_{cp} and X_{rp} which violate Constraint 4.3.2.
   7.2 Move all states in X_{rp} and X_{cp} to X_{r1} and X_{c1}, respectively.
   7.3 Go to Step 2.
8. Repeat Step 2 to 7 with increasing i by 1 until it becomes (n-2).
9. Set i = 1.
10. Arrange the order of states in the subsets $\{X_{c(j+1)};...; X_{ck}\}$ (or $\{X_{r(j+1)};....; X_{rk}\}$) by examining links with the states of $\{X_{r1};...; X_{rj}\}$(or $\{X_{c1};...; X_{cj}\}$), where $j = 2^{(i-1)}$ and $k = 2^i$.
11. Repeat Step 10 with increasing i by 1 until it becomes (n-2).
12. Assign the binary vectors of n-BRT to the states of $\{X_{c1};...; X_{cl}\}$and $\{X_{r1};...; X_{rl}\}$ in the order as they appear, where $l = 2^{(n-2)}$.

}

For example, consider a mappable 7-BAT in Fig. 10-(a). It is identified as mappable to 4-BRT. The states in X_{c1} = (1, 2, 3, 4, 5, 6, 7) and X_{r1} = (8, 9, 10, 11, 12, 13, 14). From the states in X_{c1}, the state 1 is selected as S_{31} in Step 2, and the state 8 is selected as s_{31} in Step 3. So X_{r3} = (8) and X_{r1} = (9, 10, 11, 12, 13, 14), X_{c3} =(3), and X_{c1} = (1, 2, 4, 5, 6, 7). This state partition satisfy the Constraint 4.3.1 because there exist a 2-cube (1, 3 | 8, 10) and the state 1 and state 3 is separated as the state 8 and state 10 is separated. Next, S_{32} is the state 5 and s_{32} is the state 12. So X_{r3} = (8, 12) and X_{r1} = (9, 10, 11, 13, 14), X_{c3} = (3, 2), and X_{c1} = (1, 4, 5, 6, 7). This partition does not violate the Constraint 4.3.1. Next, S_{33}



2-cubes

(1, 3 | 8, 10) (2, 5 | 9, 12)
(1, 5 | 10, 13) (2, 7 | 9, 14)
(1, 6 | 11, 13) (4, 7 | 11, 14)
(1, 7 | 10, 11)

X_{c1} : (1, 2, 3, 4, 5, 6, 7)
X_{r1} : (8, 9, 10, 11,12, 13, 14)

(a)

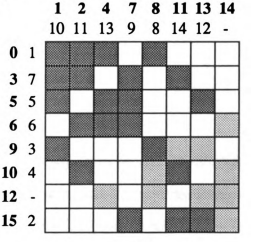
Step 2 and 3:	X_{c3} : (3)	X_{c1} : (1, 2, 4, 5, 6, 7)
	X_{r3} : (8)	X_{r1} : (9, 10, 11,12, 13, 14)
Step 4 to 7:	X_{c3} : (3, 2, 4)	X_{c1} : (1, 5, 6, 7)
	X_{r3} : (8, 12, 14)	X_{r1} : (9, 10, 11, 13)
Step 8:	X_{c2} : (5, 6)	X_{c1} : (1, 7)
	X_{r2} : (10, 11)	X_{r1} : (13, 9)
Step 10:	X_{c3} : (3, 4)	X_{c4} : (-, 2)
	X_{r3} : (8, 14)	X_{r4} : (12, -)

(b)

(0, 3, 5, 6, 9, 10, 12, 15)
(1, 7, 5, 6, 3, 4, -, 2)

(1, 2, 4, 7, 8, 11, 13, 14)
(10, 11, 13, 9, 8, 14, 12, -)

(c)



(d)

Figure 4-10. An example of the Procedure 4.3.2.3: (a) mappable BAT with a 2-cubes list, (b) steps to find a mapping, (c) results from the procedure, and (d) depiction of a mapping to 4-BRT.

is the state 7 and s_{33} is the state 14. So X_{r3} = (8, 12, 14) and X_{r1} = (9, 10, 11, 13), X_{c3} = (3, 2, 4), and X_{c1} = (1, 5, 6, 7). This partition does not violate Constraint 4.3.1. Since the number of X_{c1} and X_{r1} becomes 4, apply Constraint 4.3.2. Since there is no state which has more than one link with the corresponding state group, go to Step 8. From the states in X_{c1}, the state 1 is selected as S_{21} in Step 2, and state 13 is selected as s_{21} in Step 3. So X_{r2} = (13) and X_{r1} = (9, 10, 11), X_{c2} = (5, 6), and X_{c1} = (1, 7). Next, S_{22} is state 9. So X_{r2} = (13, 9) and X_{r1} = (10, 11), X_{c2} = (5, 6), and X_{c1} = (1, 7). Since the number of X_{c1} and X_{r1} becomes 2, apply Constraint 4.3.2. Since there is no violation, go to Step 8. Now the states in each subset are as follows: X_{c1} = (1, 7), X_{c2} = (5, 6), X_{c3} = (3, 2, 4), X_{r1} = (10, 11), X_{r2} = (13, 9), and X_{r3} = (8, 12, 14). Each state in these subsets does not violate the Constraint 4.3.1 and 3.2. In Step 10, by searching links of the states in X_{c1}, the order of states in X_{r2} becomes (13, 9), and by searching links with the states in X_{r1}, the order of states in X_{c2} becomes (5, 6). By searching links of the states in $\{X_{c1}; X_{c2}\}$, $\{X_{r3}; X_{r4}\}$becomes (8, 14, 12, -), and by searching links with the states in $\{X_{r1}; X_{r2}\}$, $\{X_{c3}; X_{c4}\}$ becomes (3, 4, -, 2). The final results from this procedure is $\{X_{c1}; X_{c2}; X_{c3}; X_{c4}\}$ = (1, 7, 5, 6, 3, 4, -, 2), and $\{X_{r1}; X_{r2}; X_{r3}; X_{r4}\}$ = (10, 11, 13, 9, 8, 14, 12, -). Each assigned state number with 4-BRT is shown in Fig. 10-(c), and it is depicted in Fig. 10-(d).

4.3.3 State Assignment Examples

The developed algorithm accepts inputs with adjacency matrix of state table and generates outputs as an assigned binary number for each state. Two examples are selected to illustrate the procedure to identify a mappable BAT, to expand the BAT to generate cycles, and to map the mappable BAT to an n-BRT.

Example 4.1: The example illustrated in Fig. 4-11 has a hidden intrinsic race (HIR) because it has a pattern defined in Theorem 4.2(b). From the given BAT, all 2-cubes are searched and listed as shown in Fig. 4-11(c) by Procedure 4.3.2.1.1. By Procedure 4.3.2.1.2, all interconnected 2-cubes are searched to find a break. However, all links have the same weights. So the link (1, 6) is arbitrary selected as a break. Since the BAT is symmetric, the link (1, 6) and its symmetric link (3, 4) are removed and by Procedure 4.3.2.2.2. New links such as (1, 7), (3, 7), (8, 4), (8, 6), and (8, 7) are added as depicted in Fig. 4-11(d). All 2-cubes are searched in the new BAT by Procedure 4.3.2.1.1 again.

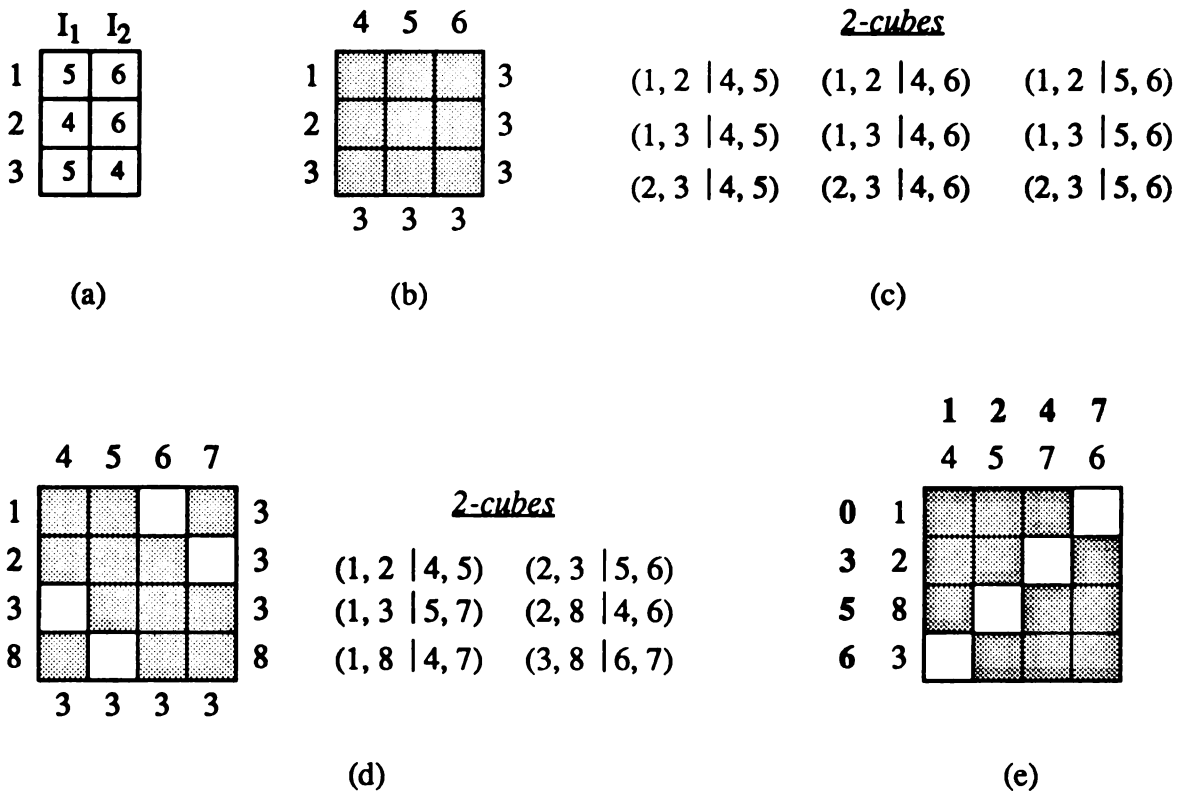


Figure 4-11. Example 4.1: (a) reduced state table, (b) 3-BAT, (c) 2-cubes list, (d) expanded BAT with a 2-cubes list, and (e) depiction of a mapping to 3-BRT.

Separated 2-cubes such as (1, 3 | 5, 7) and (2, 8 | 4, 6) are found by Procedure 4.3.2.1.2. From the interconnection of these separated 2-cubes, there is no required breaks. Hence, this BAT is mappable. This mappable 4-BAT is mapped to 3-BRT by Procedure 4.3.2.3, which is depicted in Fig. 4-11(e).

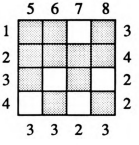
Example 4.2: The next example has intrinsic races including a visible intrinsic race (VIR) because the maximum number of links in the BAT exceeds the number of state variable. From Procedure 4.3.2.1.1, a 2-cubes list of the BAT is generated as shown in Fig. 4-12(c). Two interconnected 2-cubes are selected, and required breaks are marked accordingly in Procedure 4.3.2.1.2. Two links such as (2, 6) and (2, 8) have equal weights to be a break. Since the link (2, 6) is a bipartite link, the link (2, 8) is selected as a break and eliminated. The BAT is expanded with Procedure 4.3.2.2.1. The new BAT is examined by Procedure 4.3.2.1.1 again, and it is identified as mappable. The mapping of this 5-BAT to 4-BRT is performed by Procedure 4.3.2.3, and the outputs are shown as Fig. 4-12(e). This result can be depicted as Fig. 4-12(f). From this assignment we can get the modified state table with cycles as shown in Fig. 4-12(g).

4.4 Discussion

The algorithm is tested with several examples. Some examples generate less number of state variables than the results from Wu's and Saucier's algorithm. The computation time is not easy to compare because the computing device is much different from the time when those results were obtained. However, the elimination of an exhaustive search apparently gives benefits to the performance of our algorithm.

	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
1	5	5	6	6	6	6	8	8
2	8	7	8	7	8	5	8	5
3	5	5	5	5	7	7	7	7
4	6	8	6	8	6	8	6	8

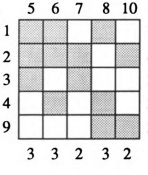
(a)



(b)

2-cubes	*Breaks*
(1, 2 | 5, 6)	(2, 6)
(1, 2 | 5, 8)	(2, 8)
(1, 4 | 6, 8)	(1, 6)
(2, 3 | 5, 7)	(1, 8)

(c)



2-cubes
(1, 2 | 5, 6)
(1, 4 | 6, 8)
(2, 3 | 5, 7)

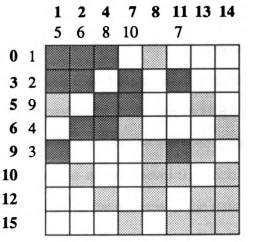
(d)

Figure 4-12. Example 4.2: (a) reduced state table, (b) 4-BAT, (c) 2-cubes list and breaks list, (d) expanded BAT with a 2-cubes list, (e) results of the Procedure 4.3.2.3, (f) depiction of a mapping to 4-BRT, and (g) modified state table.

(0, 3, 5, 6, 9, 10, 12, 15)
(1, 2, 9, 4, 3, -, -, -)

(1, 2, 4, 7, 8, 11, 13, 14)
(5, 6, 8, 10, -, 7, -, -)

(e)



(e)

	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
1	5	5	6	6	6	6	8	8
2	10	7	10	7	10	5	10	5
3	5	5	5	5	7	7	7	7
4	6	8	6	8	6	8	6	8
9	8	-	8	-	8	-	8	-
10	9	-	9	-	9	-	9	-

(f)

Figure 4-12. (Continued)

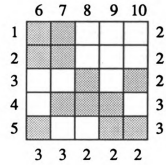
The example which has better results than the algorithm by Wu is shown in Fig. 4-13. The reduced state table and BAT are shown in Fig. 4-13(a) and (b), respectively. According to our algorithm, the BAT has one 2-cube such as (1, 2 | 6, 7), and it is identified as mappable. The BAT mapping to 4-BRT is depicted in Fig. 4-14(c). However, Wu's algorithm requires two additional states to generate cycles and 5 state variables to represent the states by generating a cycle for the link (1, 7), though there is no VIR or HIR.

This is because of a bad selection process for the first state to be assigned to the Node Weight Diagram (NWD). It assigns the first state to the node in level 0, which has the maximum number of links, and increases the level by filling the nodes with the states which are adjacent to the nodes which already have been assigned. If no vacant node is found for the state to be assigned, the shortest path is searched, and it generates a cycle by adding states. Since there is no *a priori* information about the structure of an n-cube, this algorithm adds the levels of the NWD as much as it satisfies the adjacencies for specific states.

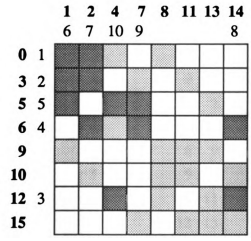
Consider the BAT in Fig. 4-14(b). According to our algorithm, the BAT is unmappable and symmetric. So a symmetric expansion is needed. The expanded BAT is shown in Fig. 4-14(c). A mapping of this 7-BAT to 4-BRT is depicted in Fig. 4-14(d). The Saucier's algorithm is mainly divided into two parts. The first part extracts a maximal spanning tree from the partition information of the state table as shown in Fig. 4-14(e). Next, an embedding is obtained by completing the spanning tree. This is achieved by adding edges and supplementary vertices and expanding the dimension of the cube, as necessary. Here difficulties occur in trying to find the maximal spanning tree because all weights of the edges are the same. Suppose that the edges for the spanning tree are chosen arbitrarily as indicated in Fig. 4-14(g). This algorithm searches with this spanning tree for finding an embedding for an 3-cube. Since there is no way to map the adjacency diagram to a 3-cube, it should be expanded to 4-cube after an exhaustive search. Therefore, the computation time of Saucier's algorithm is proportional to the number of states and the

	I_1	I_2	I_3	I_4
1	6	6	7	7
2	7	6	7	6
3	8	10	8	10
4	9	7	8	8
5	10	10	6	9

(a)



(b)

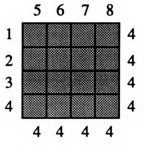


(c)

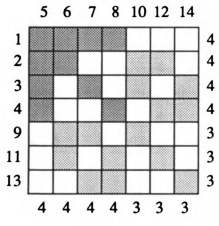
Figure 4-13. An example for the comparison with Wu's algorithm: (a) reduced state table, (b) 5-BAT, and (c) depiction of a mapping to 4-BRT.

	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
1	5	6	7	8	8	7	6	5
2	6	7	8	5	5	8	7	6
3	7	8	5	6	6	5	8	7
4	8	5	6	7	7	6	5	8

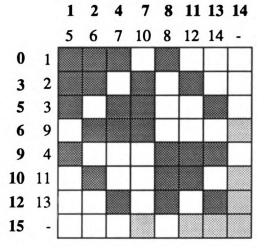
(a)



(b)



(c)



(d)

Figure 4-14. An example for the comparison with Saucier's algorithm: (a) reduced state table, (b) 4-BAT, (c) expanded BAT, (d) depiction of a mapping to 4-BRT, and Saucier's (e) partitions, (f) weighted directed transition graph, and (g) spanning tree which requires an exhaustive search to find an embedding before expanding the dimension of the cube.

$\pi_1 = (1\ \mathbf{5}, 2\ \mathbf{6}, 3\ \mathbf{7}, 4\ \mathbf{8})$

$\pi_2 = (1\ \mathbf{6}, 2\ \mathbf{7}, 3\ \mathbf{8}, 4\ \mathbf{5})$

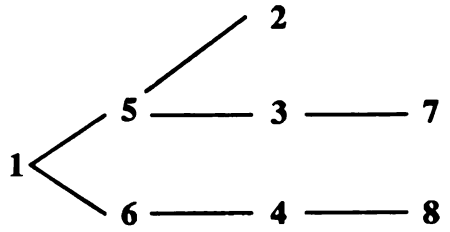
$\pi_3 = (1\ \mathbf{7}, 2\ \mathbf{8}, 3\ \mathbf{5}, 4\ \mathbf{6})$

$\pi_4 = (1\ \mathbf{8}, 2\ \mathbf{5}, 3\ \mathbf{6}, 4\ \mathbf{7})$

(e)

Edges	Weight	Edges	Weight
1 5	3	3 5	3
1 6	3	3 6	3
1 7	3	3 7	3
1 8	3	3 8	3
2 5	3	4 5	3
2 6	3	4 6	3
2 7	3	4 7	3
2 8	3	4 8	3

(f)



$n_0 = 3$

(g)

Figure 4-14. (Continued)

edges. Furthermore, it will be very complex when the adjacency diagram is a complete graph, in which every state in S_c has edges to every state in S_r. Our algorithm searches the pattern of the BAT, modifies the pattern to be a mappable BAT before finding an embedding of n-cube from the properties of an n-BRT. Hence, the computation time will be much less than Saucier's algorithm.

Chapter 5

Synthesis Examples

In order to demonstrate the synthesis procedure presented in the previous chapters, several synthesis examples for sequential logic elements and finite state machines (FSMs) are given, where the FSMs are from the MCNC benchmark examples.

The synthesis procedure can be summarized as follows:

(1) Generate DOEs using transition variables.

(2) Derive state groups.

(3) Encode state variables.

(4) Generate hazard-free state equations and output equations.

5.1 Sequential Logic Elements

Synthesis procedures of three sequential logic elements such as FKW-1, FKW-2, and FKW-3 are discussed in this section.

5.1.1 *FKW-1* Sequential Logic Function

Consider the following design specification of a sequential logic function named as FKW-1.

(1) There are two inputs, which are labeled A and B.

(2) There are two outputs, which are labeled D and E.

(3) The outputs D and E may change values on the falling edge of B. At this time $D_n = A\ E_c$ and $E_n = A + D_c$, where E_c and D_c are the current outputs, and D_n and E_n are the next outputs.

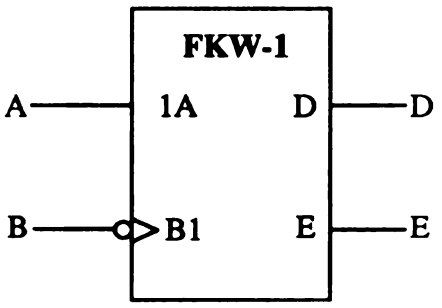
Based on the design specification, B_f can be defined as the transition variable. According to Equation (3.10), we obtain $p_r = 0$; $p_f = 1$; $E_{1f} = B_f$; $F_{1f} = AE_c$; and $F_s = D_c$ for output D, and $p_r = 0$; $p_f = 1$; $E_{1f} = B_f$; $F_{1f} = (A + D_c)$; and $F_s = E_c$ for output E.

Therefore, by applying above information to Equation (3.10), we can obtain the following set of DOEs:

$$D_n = AE_cB_f + D_cB_f' \tag{5.1}$$

$$E_n = (A + D_c)B_f + E_cB_f' \tag{5.2}$$

From the above DOEs, the states groups can be made by determining the allowed outputs and values of B. The allowed outputs calculated from above DOEs are 00, 01, and 10 when B=0 or B=1, so there are three state groups when B = 0, which are (BADE)



(a)

$D_n = A\,E_c\,B_f + D_c\,B_f'$

$E_n = (A + D_c)\,B_f + E_c\,B_f'$

(b)

1 : (0, 4)
2 : (1, 5)
3 : (3, 7)
4 : (8, 12)
5 : (9, 13)
6 : (11, 15)

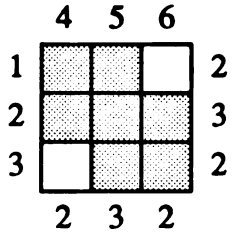
(c)

BA

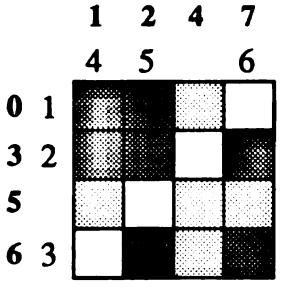
	00	01	11	10	Present Output (DE)
1	(1)	(1)	4	4	**0 0**
2	(2)	(2)	5	5	**0 1**
3	(3)	(3)	6	6	**1 1**
4	1	2	(4)	(4)	**0 0**
5	1	3	(5)	(5)	**0 1**
6	2	3	(6)	(6)	**1 1**

(d)

Figure 5-1. FKW-1 sequential logic function: (a) graphic symbol, (b) DOEs, (c) state groups, (d) state table, (e) 3-BAT, (f) depiction of a mapping to 4-BRT, (g) state excitation table, (h) present output table for D, and (i) present output table for E.



(e)



(f)

	00	01	11	10
0	0	0	1	1
3	3	3	2	2
6	6	6	7	7
1	0	3	1	1
2	0	6	2	2
7	3	6	7	7

(g)

	00	01	11	10
0	0	0	-	-
3	0	0	-	-
6	1	1	-	-
1	-	-	0	0
2	-	-	0	0
7	-	-	1	1

(h)

	00	01	11	10
0	0	0	-	-
3	1	1	-	-
6	1	1	-	-
1	-	-	0	0
2	-	-	1	1
7	-	-	1	1

(i)

Figure 5-1. (Continued)

=(0-00), (0-01), and (0-10), and three state groups when B = 1, which are (BADE) = (1-00), (1-01), and (1-10). The state table is generated from these state grouping as shown in Fig. 5-1(d).

A 3-BAT, as shown in Fig. 5-1(e), is generated based on the adjacency diagram of the state table. The race-free state assignment algorithm is applied and the 3-BAT is mappable to a 4-BRT as depicted in Fig. 5-1(f), which shows that state 1 is encoded as 0, or a 3-bit binary value 000, state 2 as 3 (or 011), state 3 as 6 (or 110), state 4 as 1 (or 001), state 5 as 2 (or 010), and state 6 as 7 (or 111). Finally, the hazard-free next state equations and present output equations are generated as follows from the state excitation table and present output tables in Fig. 5-1(g), (h), and (i):

$$Y_0 = \bar{y}_1 B + y_2 B + y_2 y_1 y_0 \bar{A} + y_1 y_0 \bar{B} \bar{A} + \bar{y}_2 \bar{y}_1 y_0 A + \bar{y}_2 y_0 \bar{B} A + \bar{y}_2 y_1 y_0 \bar{B}$$

$$Y_1 = y_2 + y_1 A + y_1 B + y_1 y_0 + y_0 \bar{B} A$$

$$Y_2 = y_2 A + y_2 B + y_2 \bar{y}_0 + y_1 \bar{y}_0 \bar{B} A$$

$$D = y_2$$

$$E = y_1$$

5.1.2 *FKW-2* Sequential Logic Function

Consider the following I/O functional design specification of a sequential logic element. (See Fig. 5-2)

(1) There are three external inputs, which are labeled A, B and M, where A is a level input, B is an edge input, and M is a mode input.

(2) There are two external outputs. They are labeled D and E_p, where E_p is the postponed output of E.

(3) The outputs D and E may change values on the rising edge of B. At this time $D_n = A\,E_c$ and $E_n = A + D_c$, if M = 0, and $D_n = A\,E_c'$ and $E_n = A' + D_c$, if M = 1, where E_c and D_c are the current outputs, and D_n and E_n are the next outputs.

(4) The output E_p is a postponed output of E, and the transition of E_p is postponed until B changes value from 1 to 0 (falling edge).

To obtain a set of DOEs for this design specification, a pseudo output E must be introduced. This pseudo output represents a temporary variable of the ASLC needed to generate the required state equations and output equations. By applying Equation (3.10), one can obtain the following set of DOEs:

$$D_n = (M'AE_c + MAE_c')B_r + D_cB_r' \tag{5.3}$$

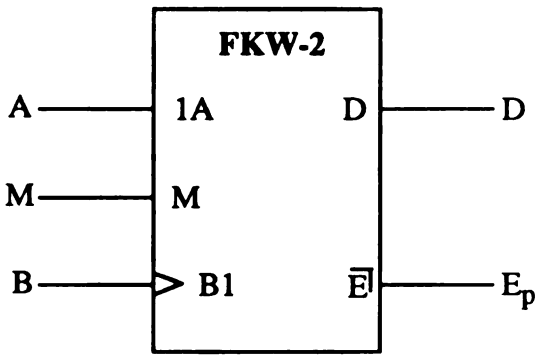
$$E_n = (M'A + MA' + D_c)B_r + E_cB_r' \tag{5.4}$$

$$E_{pn} = E_cB_r + E_cB_f + E_{pc}B_r'B_f' \tag{5.5}$$

These DOEs can be used to generate other support documentation, such as the timing diagram illustrated in Fig. 5-2(d).

The state groupings can be made by determining the output states and considering the values of B. This example has 4 allowed outputs (000, 011, 100, and 111) when B = 0, and 7 allowed outputs (000, 001, 010, 011, 100, 110, and 111) when B = 1. So the state table has 11 state groups, and we can come up with a state table shown in Fig. 5-2(f).

This example needs dummy states to generate a BAT because two states have bipartite relationships with one state such as (1, 5) and (1, 6), (2, 7) and (2, 8), and (4, 10) and (4, 11). The dummy states are used only to avoid the breaks in the bipartite links when generating cycles. The BAT with dummy states 1′, 2′, and 4′ are shown in Fig. 5-



(a)

When M = 0,

$D_n = A\ E_c$, $E_n = A + D_c$

When M =1,

$D_n = A\ E_c'$, $E_n = A' + D_c$

(b)

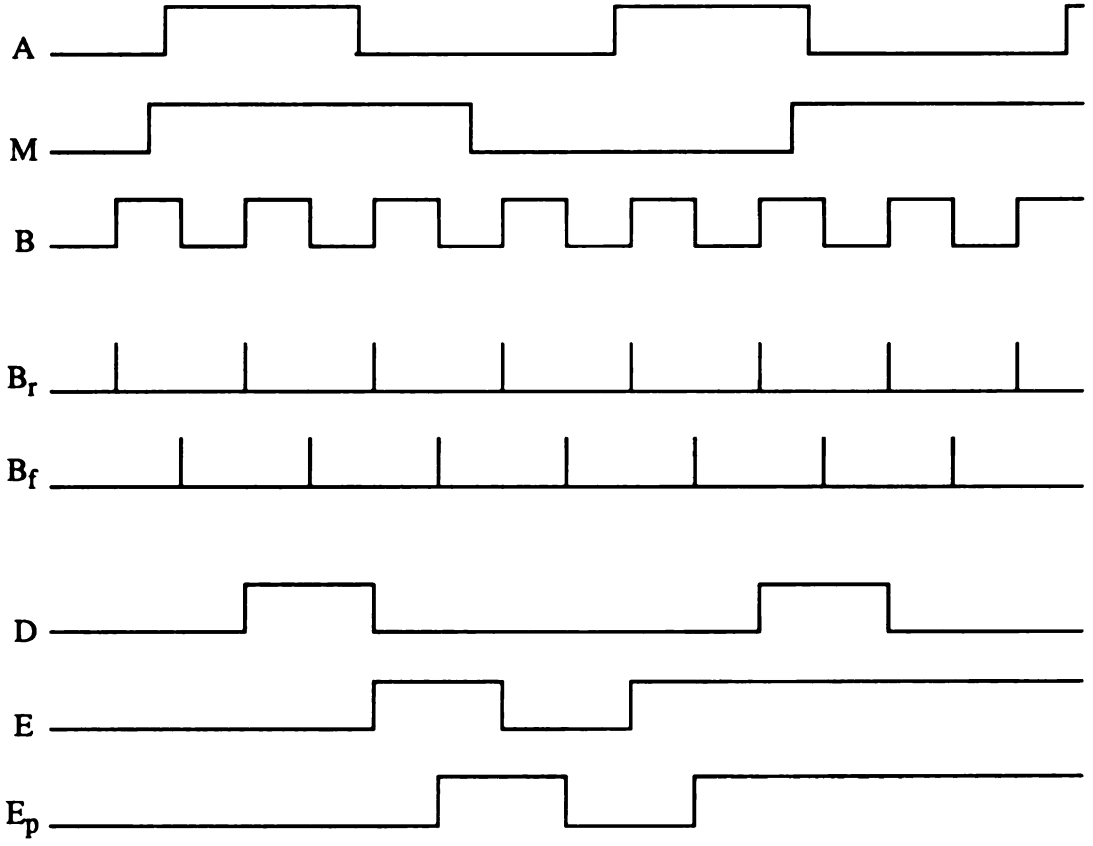
$$D_n = (M'\ A\ E_c + M\ A\ E_c')\ B_r + D_c\ B_r'$$

$$E_n = (M'\ A + M\ A' + D_c)\ B_r + E_c\ B_r'$$

$$E_{pn} = E_c\ (B_r + B_f) + E_{pc}\ B_r'\ B_f'$$

(c)

Figure 5-2. FKW-2 sequential logic function; (a) graphic symbol, (b) output definitions, (c) DOEs, (d) timing diagram, (e) state groups, (f) state table, (g) 7-BAT, (h) depiction of a mapping to 4-BRT, (i) state excitation table, (j) present output table for (DE_p), (k) circuit decomposition, (l) SUB-1 state table, (m) SUB-2 state table, and (n) state table generated from state tables of (l) and (m).



(d)

Figure 5-2. (Continued)

1 : (0, 8, 16, 24)
2 : (3, 11, 19, 27)
3 : (4, 12, 20, 28)
4 : (7, 15, 23, 31)
5 : (32, 40, 48, 56)
6 : (33, 41, 49, 57)
7 : (34, 42, 50, 58)
8 : (35, 43, 51, 59)
9 : (36, 44, 52, 60)
10 : (38, 46, 54, 62)
11 : (39, 47, 55, 63)

(e)

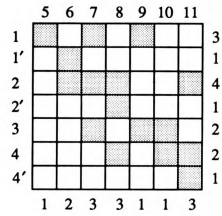
BAM

	000	001	011	010	110	111	101	100	
1	①	①	①	①	7	9	7	5	**000**
2	②	②	②	②	11	6	8	6	**011**
3	③	③	③	③	7	10	7	7	**100**
4	④	④	④	④	11	8	8	8	**111**
5	1	1	1	1	⑤	⑤	⑤	⑤	**000**
6	1	1	1	1	⑥	⑥	⑥	⑥	**001**
7	2	2	2	2	⑦	⑦	⑦	⑦	**010**
8	2	2	2	2	⑧	⑧	⑧	⑧	**011**
9	3	3	3	3	⑨	⑨	⑨	⑨	**100**
10	4	4	4	4	⑩	⑩	⑩	⑩	**110**
11	4	4	4	4	⑪	⑪	⑪	⑪	**111**

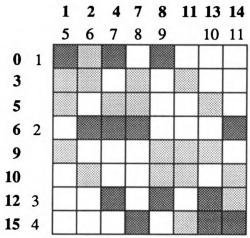
Present Output (DEE_p)

(f)

Figure 5-2. (Continued)



(g)



(h)

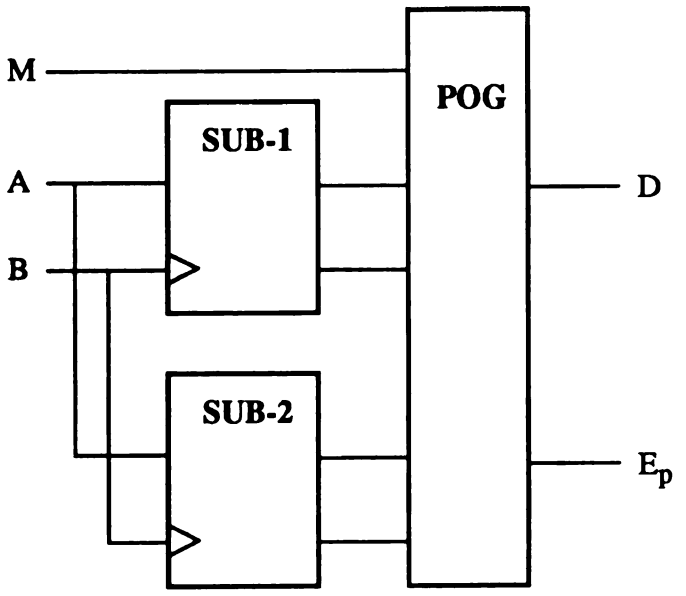
	000	001	011	010	110	111	101	100
0	0	0	0	0	4	8	4	1
6	6	6	6	6	14	2	7	2
12	12	12	12	12	4	13	4	4
15	15	15	15	15	14	7	7	7
1	0	0	0	0	1	1	1	1
2	0	0	0	0	2	2	2	2
4	6	6	6	6	4	4	4	4
7	6	6	6	6	7	7	7	7
8	12	12	12	12	8	8	8	8
13	15	15	15	15	13	13	13	13
14	15	15	15	15	14	14	14	14

(i)

	000	001	011	010	110	111	101	100
0	0	0	0	0	-	-	-	-
6	1	1	1	1	-	-	-	-
12	2	2	2	2	-	-	-	-
15	3	3	3	3	-	-	-	-
1	-	-	-	-	0	0	0	0
2	-	-	-	-	1	1	1	1
4	-	-	-	-	0	0	0	0
7	-	-	-	-	1	1	1	1
8	-	-	-	-	2	2	2	2
13	-	-	-	-	2	2	2	2
14	-	-	-	-	3	3	3	3

(j)

Figure 5-2. (Continued)



(k)

BA

	00	01	11	10	Present Output (DEE_p)
1	⓪	⑧	26	16	0 0 0
2	③	⑪	31	17	0 1 1
3	⑦	⑮	31	19	1 1 1
4	0	8	㉔	⑯	0 0 0
5	0	8	㉕	⑰	0 0 1
6	3	11	㉖	⑱	0 1 0
7	3	11	㉗	⑲	0 1 1
8	7	15	㉛	㉓	1 1 1

(l)

BA

	00	01	11	10	Present Output (DEE_p)
1	⓪	⑧	28	18	0 0 0
2	③	⑪	25	19	0 1 1
3	④	⑫	30	18	1 0 0
4	⑦	⑮	27	19	1 1 1
5	0	8	㉕	⑰	0 0 1
6	3	11	㉖	⑱	0 1 0
7	3	11	㉗	⑲	0 1 1
8	4	12	㉘	⑳	1 0 0
9	7	15	㉚	㉒	1 1 0

(m)

Figure 5-2. (Continued)

BAM

	000	001	011	010	110	111	101	100	Present Output (DEE_p)
1	(1)	(1)	(1)	(1)	7	9	7	5	000
2	(2)	(2)	(2)	(2)	11	6	8	6	011
3	-	(3)	(3)	-	-	10	7	-	100
4	(4)	(4)	(4)	(4)	11	8	8	8	111
5	1	-	-	1	(5)	-	-	(5)	000
6	1	1	1	1	(6)	(6)	(6)	(6)	001
7	2	2	2	2	(7)	(7)	(7)	(7)	010
8	2	2	2	2	(8)	(8)	(8)	(8)	011
9	-	3	3	-	-	(9)	(9)	-	100
10	-	4	4	-	-	(10)	(10)	-	110
11	4	-	-	4	(11)	-	-	(11)	111

(n)

Figure 5-2. (Continued)

2(g). A mapping of the 7-BAT to 4-BRT is depicted in Fig. 5-2(h).The state excitation table and present output table for outputs are derived from the encoded states as shown in Fig. 5-2(i) and (j), respectively. Note that the present output table for E is not necessary because it is a pseudo output which will not be realized.

The circuit decomposition according to the model in Section 3.4 is shown in Fig. 5-2(k). The NSG is divided into SUB-1 and SUB-2. And, the primitive states of each sub-circuit are grouped separately to generate state tables. The number of states groups of SUB-1 is 3 when B = 0, and 5 when B=1. The number of states group of SUB-2 is 4 when B = 0, and 5 when B=1. Each sub-circuit can be synthesized easily from this simpler state table, and the POG can be easily constructed with few gates. The combined state table is shown in Fig. 5-2(n), in which more "don't cares" exist than the state table in Fig. 5-2(f). These don't care states are unallowed states in the operation of the circuits which may be utilized to minimize the logic when realizing the circuits.

5.1.3 *FKW-3* Sequential Logic Function

Consider another sequential logic element that would traditionally be implemented with a clocked sequential logic circuits but whose speed could be increased and chip area reduced by implementing it using an asynchronous circuits. The circuit divides the frequency of the input signal by three and has the following I/O functional specification: (see Fig. 5-3)

(1) There is one external input C.

(2) There is one external output Q.

(3) Q changes state every third transition of C. The timing diagram of this element is illustrated in Fig. 5-3(b).

To generate the DOE for this design specification, pseudo outputs are needed to store the first and the second transitions of the external input C. These pseudo outputs represent temporary variables and are used to store the transition count.

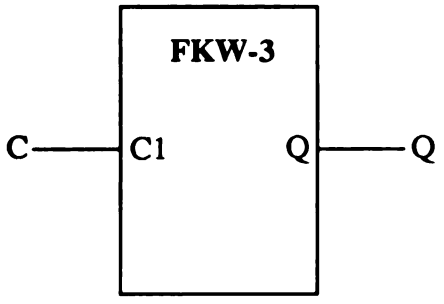
Let G be a pseudo output which makes a transition according to the first transition of C, and H as a pseudo output which makes a transition according to the second transition of C. Then $p_r = 1$ and $p_f = 1$ for each output, and the output Q can be represented by transition variables and pseudo outputs according to Equation (3.10) as follows:

$$G_n = Q_c'C_r + Q_c'C_f + G_cC_r'C_f' \tag{5.6}$$

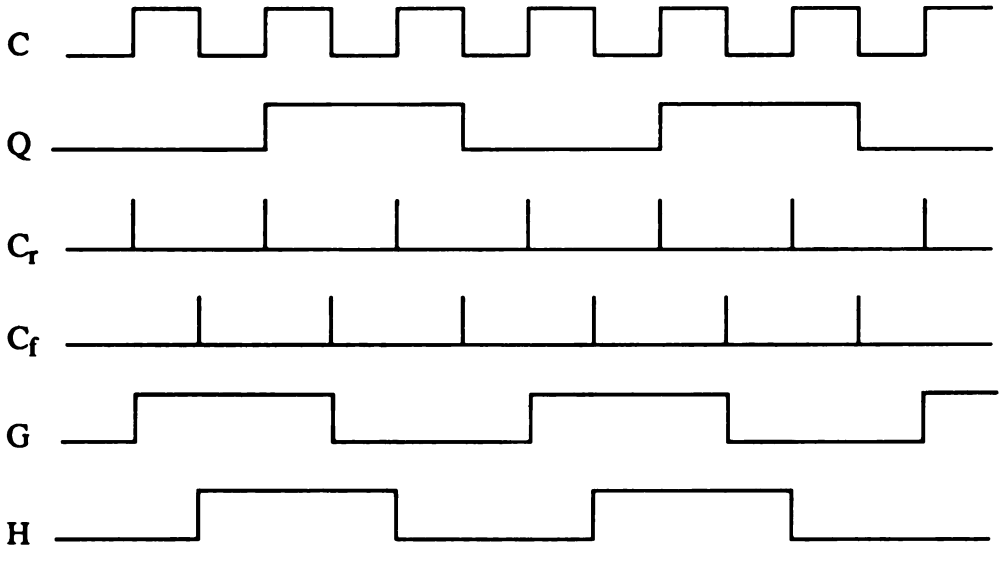
$$H_n = G_cC_r + G_cC_f + H_cC_r'C_f' \tag{5.7}$$

$$Q_n = H_cC_r + H_cC_f + Q_cC_r'C_f' \tag{5.8}$$

The state groups are generated by considering the output states and the values of C. This example has 3 allowed outputs when C=0, and accordingly it has 3 state groups, which are (CGHQ) = (0000), (0011), and (0110), and 3 allowed outputs and state groups when C = 1, which are (CGHQ) = (1001), (1111), and (1100). So the state table has 6 rows, which is shown in Fig. 5-3(e). From the state table, a 3-BAT is derived as shown in Fig. 5-3(f). The encoded state number for each state is depicted in Fig. 5-3(g). The state excitation table is generated from the encoded states as shown in Fig. 5-3(h). The output tables for pseudo outputs are not needed because they will not be implemented in the circuit. The output table for Q is derived as shown in Fig. 5-3(i).



(a)



(b)

Figure 5-3. FKW-3 sequential logic function: (a) graphic symbol, (b) timing diagram, (c) DOEs, (d) state groups, (e) state table, (f) 3-BAT, (g) depiction of a mapping to 3-BRT, (h) state excitation table, and (i) present output table for Q.

$$G_n = Q_c' (C_r + C_f) + G_c C_r' C_f'$$

$$H_n = G_c (C_r + C_f) + H_c C_r' C_f'$$

$$Q_n = H_c (C_r + C_f) + Q_c C_r' C_f'$$

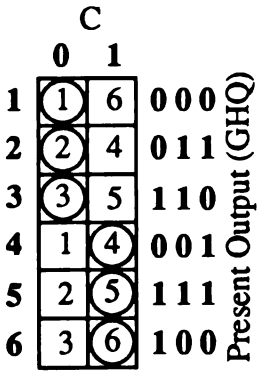
(c)

1 : (0000) **4** : (1001)

2 : (0011) **5** : (1111)

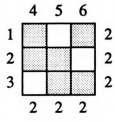
3 : (0110) **6** : (1100)

(d)

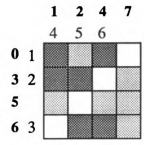


(e)

Figure 5-3. (Continued)



(f)



(g)

	0	1
0	0	4
3	3	1
6	6	2
1	0	1
2	3	2
4	6	4

(h)

	0	1
0	0	-
3	1	-
6	0	-
1	-	1
2	-	1
4	-	0

(i)

Figure 5-3. (Continued)

5.2 Finite State Machines

Due to high complexity of synthesizing finite state machines (FSMs) in asynchronous way, the FSMs have been implemented with CSLCs. With an external clock that synchronizes the circuit operation, both hazards and races can be avoided. In addition, the circuit is generally simpler to design than the ASLC counterpart. As mentioned, the ASLC implementation may offer the advantage of speed performance over the CSLC implementation. This subsection describes the implementation of the developed synthesis procedure to asynchronous FSMs.

Since the clock signal C can be employed to simplify the FSM design in the CSLC implementation, the signal can also be used as a transition variable [4]. Consider the state transition diagram of a FSM, as shown in Fig. 5-4(a). The corresponding state transition table is shown in Fig. 5-4(b). For asynchronous implementation, the state table is expanded based on the transition variable C, as shown in Fig. 5-4(c). States are changed only when the signal C is changed. More specifically, suppose the circuit is stabilized at the state 1, the change of C from 0 to 1 causes a state transition to either state 1′ or 2′ depending upon the values of inputs, and the circuit will stabilize at that state, say, state 2′. Once the signal C is changed from 1 to 0, the other transition occurs again to change the state from 2′ to 2.

If the state table in Fig. 5-4(c) is partitioned into four sub-tables in terms of edge input C and the present states, the sub-tables have the following special characteristics:

(1) The states of the sub-tables in the upper left hand and in the lower right hand are all stable, when C = 0 and C = 1, respectively, regardless of the changes of the inputs.

(2) The sub-table in the upper right hand is exactly the same as the truth table in Fig. 5-4(b), except that state "a" is renamed as state "a′".

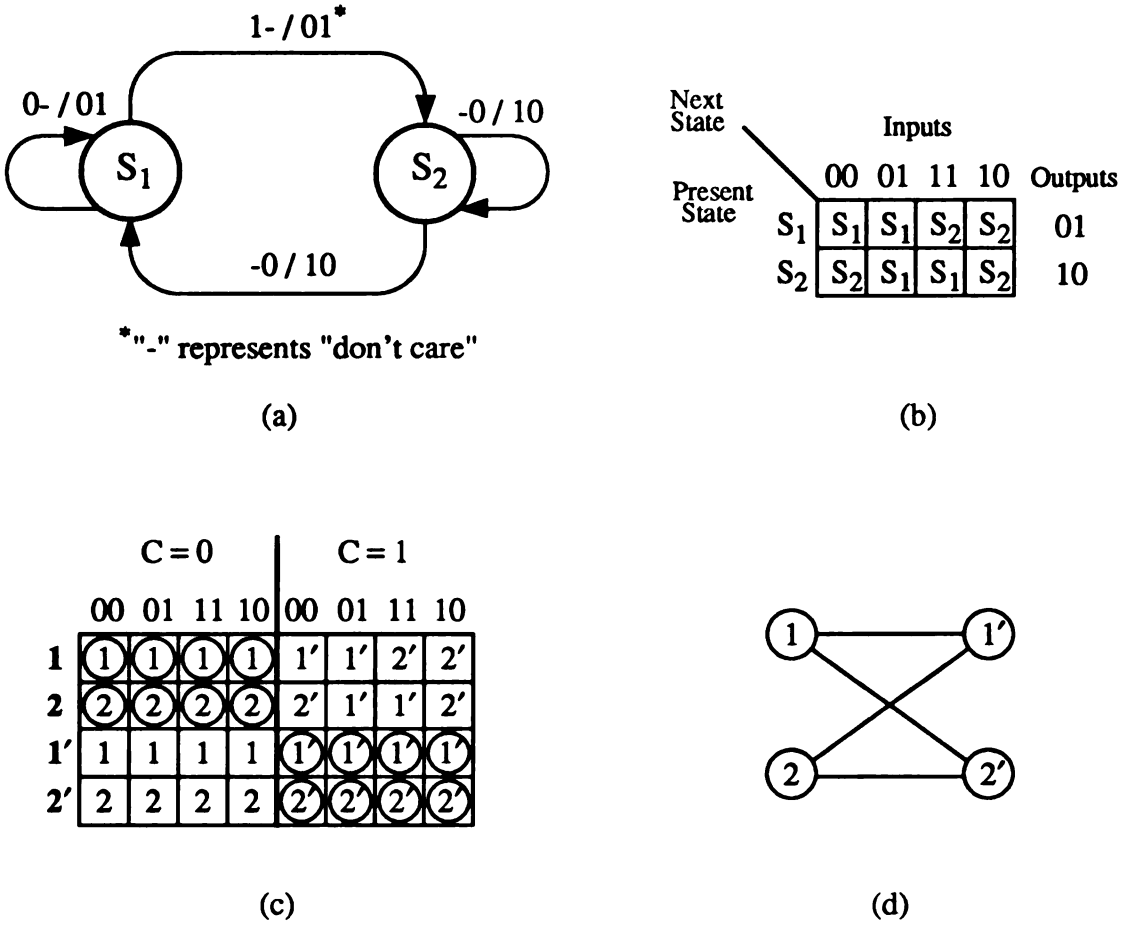


Figure 5-4. A FSM description: (a) state transition diagram, (b) state transition table, (c) expanded state table, and (d) bipartite adjacency diagram.

(3) The sub-table in lower left hand is to map a state "a′" back to state "a" regardless of the changes of the inputs.

Moreover, the adjacencies among the states can be represented with a bipartite graph as shown Fig. 5-4(d). This ASLC implementation needs neither the generation of the primitive flow table nor the merging process of the primitive states. In addition, according to the adjacency diagram of the state table, the race-free state assignment problem is a mapping a bipartite graph into an n-cube which can be solved by the race-free state assignment algorithm developed in Chapter 4.

In order to demonstrate the synthesis procedure for asynchronous FSMs, some FSMs in the MCNC benchmarks have been tested. Table 5-1 listed the simulation results.

Consider the s8 FSM, which has 4 inputs, 5 states, and one output. The state transition table is listed in Fig. 5-5(a), while the expanded state table is shown in Fig. 5-5(c). Based on the adjacency diagram generated from the table in Fig. 5-5(c), a 5-

Table 5-2. Synthesis results of the FSM in the MCNC benchmark

Name	Inputs	States	Outputs	BAT*	Links**	Intrinsic Races	Additional States	n-BRT
bbara	4	10	2	(10, 10)	(9, 4)	Yes	12	6
dk14	3	7	5	(7, 7)	(7, 6)	Yes	10	5
dk15	3	4	5	(4, 4)	(4, 4)	Yes	4	4
dk17	2	8	3	(8, 8)	(6, 5)	Yes	10	5
dk27	1	7	2	(7, 7)	(4, 3)	Yes	2	4
dk512	1	15	3	(15, 15)	(7, 3)	Yes	8	6
ex3	2	10	2	(10, 10)	(9, 5)	Yes	12	6
ex4	6	14	9	(14, 14)	(4, 3)	No	0	5
ex6	5	8	8	(8, 8)	(8, 5)	Yes	14	5
lion	2	4	1	(4, 4)	(3, 3)	No	0	3
lion9	2	9	1	(9, 9)	(3, 3)	No	0	5
mark1	5	15	16	(15, 15)	(7, 8)	Yes	12	7
mc	3	4	5	(4, 4)	(2, 2)	No	0	3
modulo12	1	12	1	(12, 12)	(2, 2)	No	0	5
s8	4	5	1	(5, 5)	(3, 3)	No	0	4

* BAT size (Column, Row)
** Maximum link degree in the BAT (Column, Row)

	I_1	I_2	I_3	I_4
S_1	S_1	S_1	S_2	S_2
S_2	S_2	S_3	S_2	S_1
S_3	S_3	S_5	S_3	S_5
S_4	S_4	S_2	S_3	S_3
S_5	S_5	S_5	S_1	S_4

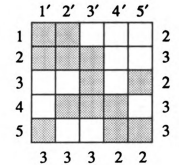
(a)

	C=0				C=1			
	I_1	I_2	I_3	I_4	I_1	I_2	I_3	I_4
1	1	1	1	1	1′	1′	2′	2′
2	2	2	2	2	2′	3′	2′	1′
3	3	3	3	3	3′	5′	3′	5′
4	4	4	4	4	4′	2′	3′	3′
5	5	5	5	5	5′	5′	1′	4′
1′	1	1	1	1	1′	1′	1′	1′
2′	2	2	2	2	2′	2′	2′	2′
3′	3	3	3	3	3′	3′	3′	3′
4′	4	4	4	4	4′	4′	4′	4′
5′	5	5	5	5	5′	5′	5′	5′

(b)

	I_1	I_2	I_3	I_4
1	1′	1′	2′	2′
2	2′	3′	2′	1′
3	3′	5′	3′	5′
4	4′	2′	3′	3′
5	5′	5′	1′	4′

(c)



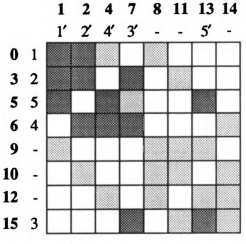
(d)

Figure 5-5. S8 FSM example: (a) state transition table, (b) expanded state table, (c) reduced state table, (d) 5-BAT, (e) state assignments to 4-BRT, (f) depiction of a mapping to 4-BRT, and (g) resultant encoded state table.

0	**3**	**5**	**6**	**9**	**10**	**12**	**15**
1	2	5	4	-	-	-	3

1	**2**	**4**	**7**	**8**	**11**	**13**	**14**
1′	2′	4′	3′	-	-	5′	-

(e)



(f)

	C=0				C=1			
	I_1	I_2	I_3	I_4	I_1	I_2	I_3	I_4
0	0	0	0	0	1	1	2	2
3	3	3	3	3	2	7	2	1
15	15	15	15	15	7	13	7	13
6	5	5	5	5	4	2	7	7
5	6	6	6	6	13	13	1	4
1	0	0	0	0	1	1	1	1
2	3	3	3	3	2	2	2	2
7	15	15	15	15	7	7	7	7
4	5	5	5	5	4	4	4	4
13	6	6	6	6	13	13	13	13

(g)

Figure 5-5. (Continued)

BAT is derived as shown in Fig. 5-5(d). The race-free state assignment algorithm maps the 5-BAT to a 4-BRT, as shown in Figure 5-5(e). Finally, the resultant state encoding is given in Fig. 5-5(g).

In this FSM, the BAT consists of 5 columns and 5 rows. Both rows and columns have the maximum link degree of 3. It has been checked that the BAT contains no intrinsic races and no additional states are added.

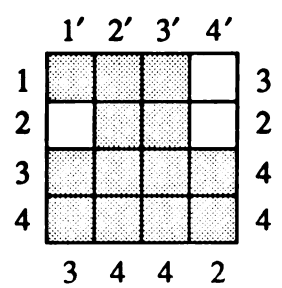
Similarly, consider the dk15 FSM, as shown in Fig. 5-6, which has 3 inputs, 4 states, and 5 outputs. The maximum link degrees in both columns and rows are 4. Based on the 4-BAT in Fig. 5-6(c), it has been found that there exist some intrinsic races in the table. The race-free state assignment algorithm maps the 4-BAT to a 4-BRT with four additional sates, as shown in Fig. 5-6(f).

	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
S_1	S_1	S_2	S_3	S_2	S_3	S_1	S_2	S_3
S_2	S_2	S_2	S_3	S_2	S_3	S_3	S_2	S_3
S_3	S_1	S_2	S_3	S_1	S_1	S_1	S_2	S_4
S_4	S_2	S_2	S_3	S_1	S_1	S_1	S_2	S_3

(a)

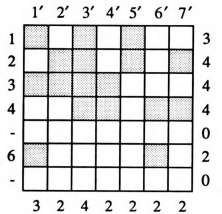
	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
1	1′	2′	3′	2′	3′	1′	2′	3′
2	2′	2′	3′	2′	3′	3′	2′	3′
3	1′	2′	3′	1′	1′	1′	2′	4′
4	2′	2′	3′	1′	1′	1′	2′	3′

(b)

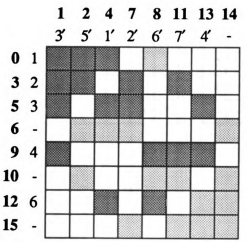


(c)

Figure 5-6. Dk15 FSM example: (a) state transition table, (b) reduced state table, (c) 4-BAT, (d) expanded BAT, (e) depiction of a mapping to 4-BRT, and (f) resultant encoded state table.



(d)



(e)

Figure 5-6. (Continued)

	C=0								C=1							
	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
0	0	0	0	0	0	0	0	0	4	2	1	2	1	4	2	1
3	3	3	3	3	3	3	3	3	7	7	1	7	1	1	7	1
5	5	5	5	5	5	5	5	5	4	7	1	4	4	4	7	13
9	9	9	9	9	9	9	9	9	11	11	1	8	8	8	11	1
4	0	0	0	0	0	0	0	0	4	4	4	4	4	4	4	4
7	3	3	3	3	3	3	3	3	7	7	7	7	7	7	7	7
1	5	5	5	5	5	5	5	5	1	1	1	1	1	1	1	1
13	9	9	9	9	9	9	9	9	13	13	13	13	13	13	13	13
2	-	-	-	-	-	-	-	-	-	3	-	3	-	-	3	-
8	-	-	-	-	-	-	-	-	-	-	-	12	12	12	-	-
12	-	-	-	-	-	-	-	-	-	-	-	4	4	4	-	-
11	-	-	-	-	-	-	-	-	3	3	-	-	-	-	3	-

(f)

Figure 5-6. (Continued)

Chapter 6

Summary and Conclusions

6.1 Summary

The traditional ASLC synthesis procedure with the Huffman model begins with the generation of a primitive flow table (PFT). Though this table generally contains sparse entries, the size of the PFT is of $O(2^{2I+N})$, where I and N are the total number of inputs and outputs, respectively. Note that the table size grows nearly exponentially with the number of inputs and outputs. For any but the most simple cases, this makes generating the PFT a very complex task. Once the PFT is generated, states are merged and encoded. State merging is generally achieved by examining the corresponding merger diagram. Due to the high complexity of a merger diagram, the state merging problem was generally formulated and resolved by a graph theory. This is followed by state encoding process. It is clear that both state merging and encoding processes were developed based on the topological structures, but not on the functional behavior.

In this thesis, an analytical model for synthesizing ASLCs is presented. While the inputs are treated equally in the flow table with the traditional synthesis procedure, *a priori* information about the behavior of a specific sequential logic circuit is utilized to delineate its inputs into three distinct classes: **mode inputs, level inputs,** and **edge inputs.** The transition variables are defined to model the dynamic behavior of the edge inputs. Using transition variables, a set of equations to characterize the functional behavior of an ASLC has developed, which is known as the **dynamic output equations** (DOEs).

The DOEs are more compact than the traditional primitive flow table (PFT) in terms of representing the state transitions because they are only a set of equations according to the number of outputs. Moreover, the functional behavior of mode inputs facilitates to decompose a large circuit into smaller ones which may be easily synthesized.

As the transition variables are defined from the edge inputs which may cause state transitions to occur, the states can be grouped according to the values of the transition variables. This **state grouping** does not need to generate a complex merger diagram or merger table which was conventionally utilized to reduce the number of internal states. Moreover, this state grouping leads to simpler states encoding procedures.

A race-free state assignment of an ASLC is formulated to find an embedding of an n-cube. For an arbitrary connected adjacency diagram, to map the states to an n-cube which has the minimum dimension requires a combinatorial computation time with enumerative efforts. The race-free state assignment algorithm presented here features two improvements over existing methods. First, a pattern matching technique to map the states onto an n-cube results in better performance compared with an enumerative search approach. Secondly, the bipartite representation of an n-cube simplifies the procedure for the race-free state assignment for large number of states.

The **bipartite representation table** (BRT) developed provides a geometric visualization of an n-cube. From the n-BRT, we can easily draw out the characteristics of the adjacency diagram which can be mapped onto an n-cube. The characteristics are further utilized to develope rules for a race-free state assignment algorithm. Moreover, the bipartite characteristics of an n-BRT efficiently reduce the computation time with bisecting the total state space recursively. Therefore, this algorithm is useful for the synthesis of complex ASLCs which requires encoding of large number of states.

The synthesis procedure developed is summarized as follows:

(1) Generate DOEs for behavioral modeling using transition variables.

(2) Derive state groups according to the values of allowed outputs and values of transition variables to get a state table.

(3) Assign the states in the bipartite adjacency table (BAT) to an n-BRT.

(4) Generate hazard-free state equations and output equations.

The analytical modeling plays an important role in simplifying the process of step (1) to (3) described above. In step (1), the generation of a large primitive flow table which is an essential step to begin the traditional synthesis procedure is eliminated by the generation of simple equations. In step (2), a simple state grouping process reduces the computation time compared with the conventional way to find the strongly connected states sets from a complex merger diagram. In step (3), the bipartite characteristics of the adjacency diagram of the state table eliminates enumerative efforts to map the adjacency diagram to an n-cube, or to find the shortest path to the nearest states for generating cycles to avoid races.

This synthesis procedure is well suited for the integration into CAD tools. Moreover, this simplified synthesis procedure can be applied to the synthesis of large asynchronous digital networks which contain many ASLCs. It works best when it is used to synthesize an asynchronous sequential logic element (ASLE) which has many data inputs but only a small number of control inputs.

6.2 Contributions

This research presents an analytical model for efficiently synthesizing ASLCs. With *a priori* information about the input signals, three different types of input signals, mode inputs, level inputs, and edge inputs, are identified. The edge inputs are used as the transition variables for generating a set of DOEs which simplifies the state merging and encoding processes significantly. In addition, the mode inputs can be used to decompose a large ASLC to many smaller ones. As mentioned, the complexity of synthesizing an ASLC grows nearly exponentially with the size of the circuit. Thus, the identification of the mode input signals facilitates to synthesize large-scale ASLCs efficiently in practical applications.

With the identification of edge inputs, or transition variables, primitive states in the conventional primitive flow table approach can be merged to a compact and regular structure as discussed in Chapter 3. In addition, the structure reveals the special characteristics of bipartite relationship among the states. As a result, the characteristics are implemented to develop an efficient state assignment algorithm presented in Chapter 4. As the experimental results illustrated in Chapter 5, the developed state assignment algorithm can handle those "very large" finite state machines which cannot be handled by *MSUASLC*.

In the practical applications of developed synthesis procedure, the finite state machines as well as sequential logic elements which have large number of inputs and outputs may be implemented with ASLCs with much less design complexities, which previously have been mainly implemented with clocked sequential logic circuits.

6.3 Future Research

A research issue related to the developed synthesis model is an investigation of alternative architectures of ASLCs. In the synthesis model developed, three different classes of inputs are delineated. However, each input defined in the design specification may not be unique. In other words, the input labelled A in the design specification can be classified as an edge input for one architecture, but it may be classified as a data input, which results in another architecture. These two different architectures apparently satisfy different design constraints and one will be more efficient than the other in some design criterions such as to minimize the chip area, to maximize the speed, or to have special functions. One possible approach to investigate alternative architectures is to start its synthesis by assuming that each input in the given design specification be the same class of input. Then enumerate possible classifications of each input to implement the circuits. The developed synthesis procedure in this research and our synthesis system, *MSUASLC*, will be powerful tools for this research work because the simplified synthesis procedure reduces the researcher's design efforts, and the synthesis system allows researcher's intervention between each design module to generate logic implementations.

This thesis has presented an efficient state assignment algorithm which maps a bipartite adjacency table (BAT) to a bipartite representation table (BRT) of an n-cube. The resultant mapping is the state assignment. This study has generated several rules to determine the mappability. Both Theorems 4.1 and 4.2 provide the sufficient, but not necessary, conditions for determining if a given BAT is mappable and unmappable, respectively. In other words, if a BAT satisfies the conditions in Theorem 4.1, it is definitely mappable. However, Theorem 4.1 does not imply that the BAT is not

mappable if the conditions are not satisfied. Similarly, Theorem 4.2 only determines if the BAT is definitely not mappable. The question is how to determine the mappability and unmappability if a given BAT does not satisfy the conditions in Theorem 4.1, nor those in Theorem 4.2. Theorem 4.3 provides some rules to determine the unmappability in this case. In addition, it also provides the information to determine how to generate cycles in an unmappable BAT so that the modified BAT becomes mappable. As shown in Chapter 4, the developed algorithm can efficiently determine the mapping results if the BAT is mappable. On the other hand, if the BAT is definitely unmappable, the developed algorithm can efficiently generate the necessary cycles. However, if the BAT does not satisfy Theorem 4.1, it is assumed to be unmappable and required to modify the BAT by generating cycles. Apparently, the efficiency of mapping is highly determined by identifying the mappbility of a given BAT. When an efficient method exists to identify all mappable BATs, there requires no additional processing to generate a mappble BAT from an unmappable BAT by generating cycles. This leads to developing a more efficient race-free state assignment algorithm for future study.

In this study, the hazard-free state equations and output equations are realized by two-level sum-of-product (SOP) logic implementations. Multi-level logic implementation has been commonly used in CSLCs to reduce chip area and delay. Technology mapping algorithms have developed for CSLCs to map the resultant optimal multi-level logic network to the available standard cells or gate arrays in cell library. It is desirable to develop a logic synthesis procedure for decomposing the two-level logic to multi-level logic and to map the network to standard cells or gate arrays. In ASLCs, the hazard-free state equations and output equations are all realized by two-level logic. No multi-level logic implementation has been attempted. It seems very trivial to decompose a two-level logic to a multi-level logic. For example, when the OR gates available in the cell library are either 2-input or 3-input, a 5-input OR function can

be realized by a 3-input OR gate and a 2-input OR gate, where both OR gates outputs are fed to a 2-input OR gate at the second stage. For ASLC implementation, gate delay acts a very important role in the logic synthesis. Due to the different gate delays, the inputs to the 2-input OR gate at the second stage may result in a delay hazard at the output of that OR gate. Thus, how to resolve the delay hazard so that the synthesized ASLCs can be implemented with multi-level logic is a very interesting and practical problem for future study.

The synthesis system, *MSUASLC*, provides an automated tool for effectively synthesizing ASLCs. With the successful development of the analytical model and synthesis procedure, it is readily to adopt the developed algorithm and to upgrade the system to handle the practically large ASLCs. This is also a desirable development for future work.

LIST OF REFERENCES

LIST OF REFERENCES

[1] J. F. Wakerly, *Digital Design Principles and Practices*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[2] K. J. Breeding, *Digital Design Fundamentals*, Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[3] A. E. A. Almaini, *Electronic Logic Systems*, 2nd ed., Prentice Hall International (UK) Ltd., Cambridge, 1989.

[4] E. J. McCluskey, *Logic Design Principles*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[5] A. D. Friedman, *Fundamentals of Logic Design and Switching Theory*, Computer Science Press Inc., Rockville, MD, 1986.

[6] W. I. Fletcher, *An engineering Approach to Digital Design*, Prentice Hall, Englewood Cliffs, New Jersey, 1980.

[7] C. Mead and L. Conway, *Chapter 7, Introduction to VLSI Systems*, Addison-Wesley, Reading, Massachusetts, 1980.

[8] D. F. Wann and M. A. Franklin, "Asynchronous and Clocked Control Structures for VLSI Based Interconnection Network," *IEEE Transactions on Computers*, vol. C-32, no. 3, pp. 284-293, Mar., 1983.

[9] H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*, Addison-Wesley Publishing Co., Inc., Reading, Massachusetts, 1990.

[10] D. G. Messerschmitt, "Synchronization in Digital Systems Design," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 10, pp. 1404-1419, Oct., 1990.

[11] T. H. Meng, *Synchronization Design for Digital Systems*, Kluwer Academic Publishers, Norwell, Massachusetts, 1991.

[12] S. H. Unger, *Asynchronous Sequential Circuits*, Wiley Interscience, New York, 1969.

[13] T-A. Chu, "On the Models for Designing VLSI Asynchronous Digital Systems," *Integration, the VLSI Journal*, vol. 4, pp. 99 - 113, Apr., 1986.

[14] S-F. Wu, *Automating the Design of Large-Scale Asynchronous Sequential Logic Circuits*, Ph.D. Dissertation, Department of Electrical Engineering, Michigan State University, May 1991.

[15] D. A. Huffman, "The Synthesis of Sequential Switching Circuits," *J. Franklin Inst.*, vol. 257, pp. 161 -190, March 1954 and pp. 275 -203, Apr. 1954

[16] S. H. Caldwell, *Switching Circuits and Logical Design*, Wiley, New York, 1958.

[17] M. Krieger, *Basic Switching Circuit Theory*, The Mcmillan company Ltd., New York, 1967.

[18] D. D. Givone, *Introduction to Switching Circuit Theory*, McGraw-Hill, New York, 1970.

[19] C. L. Sheng, *Introduction to Switching Logic*, Haddon Craftsman Inc., Scranton, 1972.

[20] D. L. Dietmeyer, *Logic Design of Digital Systems*, 2nd ed., Allyn and Bacon Inc., Boston, 1978.

[21] R. J. Smith, II, J. H. Tracey, W. L. Schoeffel and G. K. Maki, "Automation in the Design of Asynchronous Sequential Circuits," *AFIPS Proc.*, vol. 32, pp. 53-60, 1968.

[22] J. H. Tracey, "Internal State Assignments for Asynchronous Sequential Machines," *IEEE Transactions on Electronic Computers*, vol. EC-15, pp. 551-560, Aug.1966.

[23] S-F. Wu and P. David Fisher, "Automating the Design of Asynchronous Sequential Logic Circuits," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 3, pp. 364-370, March, 1991

[24] S-F. Wu, and P. D. Fisher, "An Artificial Intelligence Approach to the Behavioral Modeling of Asynchronous Sequential Logic Circuits," *Proc. 33rd Midwest Symposium on Circuits and Systems*, pp. 1115-1118, Aug. 1990.

[25] P. D. Fisher and S-F. Wu, "Race-Free State Assignments for Synthesizing Large-Scale Asynchronous Sequential Logic Circuits," Accepted to appear in *IEEE Transactions on Computers.*

[26] M. C. Paull and S. H. Unger, "Minimizing the Number of States in Incompletely Specified Sequential Switching Functions," *IRE Transactions on Electronic Computers*, vol. EC-8, pp. 356-367, Aug. 1959.

[27] A. Grasselli, and F. Luccio, "A Method for Minimizing the Number of Internal States in Incompletely Specified Sequential Networks," *IEEE Transactions on Electronic Computers*, vol. EC-14, pp. 350-359, June 1965.

[28] W. S. Meisel, "A Note on Internal State Minimization in Incompletely Specified Sequential Networks," *IEEE Transactions on Electronic Computers*, vol. EC-16, pp. 508-509, Aug. 1967.

[29] S. C. de Sarkar, A. K. Basu, and A. K. Choudhury, "Simplification of Incompletely Specified Flow Tables with the Help of Prime Closed Sets," *IEEE Transactions on Computers*, vol. C-18, pp. 953-956, Oct. 1969.

[30] R. W. House and D. W. Stevens, "A New Rule for Reducing CC Tables," *IEEE Transactions on Computers*, vol. C-19, pp. 1108-1111, Nov. 1970.

[31] C. C. Yang, "Closure Partition Method for Minimizing Incompletely Sequential Machines," *IEEE Transactions on Computers*, vol. C-22, pp. 1109-1122, Dec. 1973.

[32] N. N. Biswas, "State Minimization of Incompletely Specified Sequential Machines," *IEEE Transactions on Computers*, vol. C-23, pp. 80-84, Jan. 1974.

[33] C. V. S. Rao. and N. N. Biswas, "Minimization of Incompletely Specified Sequential Machines," *IEEE Transactions on Computers*, vol. C-24, pp. 1089-1100, Nov. 1975.

[34] M. Yamamoto, "A Method for Minimizing Incompletely Specified Sequential Machines," *IEEE Transactions on Computers*, vol. C-29, pp. 732-736, Aug. 1980.

[35] J-W. Kang, C-L. Wey, and P. David Fisher, "An efficient modeling and Synthesis Procedure of Asynchronous Sequential Logic Circuits," *Proc. 35th Midwest Symposium on Circuits and Systems*, Washington, D.C., Aug. 1992. (In press)

[36] J. Hartmanis and R. E. Stearns, *Algebraic Structure, Theory of Sequential Machines*, Prentice Hall, Englewood Cliffs, New Jersey, 1966.

[37] C. N. Liu, "A State Variable Assignment Method for Asynchronous Sequential Switching Circuits," *J. ACM*, vol. 10, pp. 209-216, Apr. 1963.

[38] J. G. Kuhl and S. M. Reddy, "Multicode Single Transition Time State Assignment for Asynchronous Sequential Circuits," *IEEE Transactions on Computers*, vol. C-27, pp. 927-934, Oct. 1978.

[39] T. Nanya and Y. Tohma, "Universal Multicode STT State Assignments for Asynchronous Sequential Machines," *IEEE Transactions on Computers,* vol. C-28, pp. 811-818, Nov. 1979.

[40] A. D. Friedman, R. L. Graham, and J. D. Ullman, "Universal Single Transition Time Asynchronous State Assignments," *IEEE Transactions on Computers,* vol. C-18, pp. 541-547, June 1969.

[41] C. J. Tan, "State Assignments for Asynchronous Sequential Machines," *IEEE Transactions on Computers,* C-20, pp. 382-391, Apr. 1971.

[42] G. K. Maki and J. H. Tracy, "A State Assignment Procedure for Asynchronous Sequential Circuits," *IEEE Transactions on Computers,* vol. C-20, pp. 666-668, June 1971.

[43] R. J. Smith, II, "Generation of Internal State Assignments for Large Asynchronous Sequential Machines," *IEEE Transactions on Computers,* vol. C-23, pp. 924-932, Sept. 1974.

[44] T. Nanya and Y. Tohma, "On Universal Single Transition Time Asynchronous State Assignments," *IEEE Transactions on Computers,* vol. C-27, pp. 781-782, Aug. 1978.

[45] L. A. Hollaar, "Direct Implementation of Asynchronous Control Circuits," *IEEE Transactions on Computers,* vol. C-31, no. 12, pp. 1133-1141, Dec. 1982.

[46] G. Saucier, "State Assignment of Asynchronous Sequential Machines Using Graph Techniques," *IEEE Transactions on Computers,* vol. C-21, pp. 282-288, Mar. 1972.

[47] F. Harary, *Graph Theory,* Addison-Wesley, New York,1969.

[48] A. Wagner and D. G. Corneil., "Embedding Trees in a Hypercube is NP-complete," *SIAM J. Comput.*, vol. 19, No. 4, pp. 570-590, June 1990.

[49] S. Foldes, "A Characterization of Hypercubes," *Discrete Mathematics 17,* pp. 155-159, 1977.