This is to certify that the

dissertation entitled

DESIGN AND SYNTHESIS OF
TESTABLE ASYNCHRONOUS SEQUENTIAL
LOGIC CIRCUITS

presented by

Ming-Der Shieh

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Electrical Engineering

Chi-Sang Cho
P. David Fisher
Major professor

Date March 25, 1993

DESIGN AND SYNTHESIS OF TESTABLE ASYNCHRONOUS SEQUENTIAL LOGIC CIRCUITS

By

Ming-Der Shieh

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

DESIGN AND SYNTHESIS OF TESTABLE ASYNCHRONOUS SEQUENTIAL LOGIC CIRCUITS

By

Ming-Der Shieh

Test generation is much more difficult for ASLCs than for CSLCs because of races, hazards, and state oscillations. Often a designer does not know whether or not all transitions in a circuit are free of races and hazards. The situation is even more uncertain if a fault is present. Even though fault-free circuits can be designed to be free of races, hazards, and state oscillations, they may occur as a result of faults. And even though the fault-free circuit can be designed to change one state variable at a time during a state transition to avoid races, the condition cannot be guaranteed in the presence of faults. Based on the single stuck-at fault model, fault effects such as redundant faults and state oscillation faults for both Huffman-modeled and STG-modeled ASLCs have been presented in this research. With the observed fault effects, numerous synthesis properties have also been described for both Huffman-modeled and STG-modeled ASLCs with or without SR-latches. Those properties can be implemented to reduce the complexity of the test generation process.

The similarities and differences between the Huffman-modeled ASLC and the STG-modeled ASLC have been presented in this research. A STG-modeled ASLC

without input/output concurrency can be viewed as a special Huffman-modeled ASLC. The input/output concurrency is a special feature in STG which resolves the fundamental mode problem in Huffman model. Due to such a salient feature, asynchronous control circuits have been successfully designed and implemented using STG. However, with the input/output concurrency, a test input that must be applied at the same time when the output changes is obvious not an easy task. This study shows that the faults due to the input/output concurrency cannot be tested without scan structure.

Due to the lack of controllability and observability of state variables, testing of sequential circuits has been recognized as a very difficult task. This research presents the development of the ASLCScan method, a scan design structure for ASLCs. The scan structure is free of races and hazards in both normal operation and test modes, and achieves full testability for all single stuck-at faults. With the ASLCScan design, the test generation problem of ASLCs is reduced to one of just testing the combinational logic and the introduced races in the combinational circuit of ASLC can be identified easily.

ACKNOWLEDGMENTS

The author wishes to express his sincere gratitude to his major advisors, Dr. Chin-Long Wey and Dr. P. David Fisher, for the guidance, support and encouragement given in the course of this study. Gratitude is also extended to his committee members, Dr. David Yen and Dr. Diane Rover. In particular, the author wishes to express his deep appreciation to his family for their support, understanding and exemplary patience.

TABLES OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Sequential logic circuits may be implemented either as clocked sequential logic circuits (CSLCs) or asynchronous sequential logic circuits (ASLCs). CSLCs require a global clock to synchronize the internal transition, while ASLCs do not have to wait for the arrival of a clock pulse before effecting a transition. Thus, ASLC implementation may provide faster operation than CSLC implementation. Due to the simplicity of clock synchronization, however, CSLC implementation has been extensively used in today's LSI/VLSI system designs. Moreover, as the speed of the computing system is increased, the distribution and synchronization of a global clock will soon become a bottleneck to system throughput [1] and, thus, ASLC implementation is an obvious solution. On the other hand, ASLCs also provide the capability of modular design for complex system which greatly facilitates system integration.

Two main models of operations for ASLCs are conventionally used: *Huffman model* and *Muller model.* The Huffman-modeled circuit [2,3] is decomposed into a combinational circuit and a set of feedback paths, where both wire and gate delay are bounded. In addition, the circuit assumes the fundamental mode of operation, i.e., input cannot change until the circuit stabilizes, with the constraint of single input change. The Muller-modeled circuit [4] is decomposed into gates with an arbitrary interconnection and assumes that the wire delays are zero and the gate delays are unbounded, but finite. In

general, the Muller-modeled circuits are referred to as *speed-independent* circuits. Recently, an alternative design approach using a graph model, *Signal Transition Graph* (STG) [1,5-7], has been presented to synthesize asynchronous controllers. Based on these models, many automated synthesis systems have been developed [1,26,39].

Test generation is much more difficult for ASLCs than for CSLCs because of races, hazards, and state oscillations. Often a designer does not know whether or not all transitions in a circuit are free of races and hazards. The situation is even more uncertain if a fault is present. Even though fault-free circuits can be designed to be free of races, hazards, and state oscillations, they may occur as a result of faults. And even though the fault-free circuit can be designed to change one state variable at a time during a state transition to avoid races, the condition cannot be guaranteed in the presence of faults. This has motivated the development of synthesizing testable ASLCs.

1.1 Problem Statement

When a fault is presented in an ASLC, one of two fault effects occurs [8]: (1) the circuit remains in a transition path and becomes stable, but not in the proper stable state; or (2) the circuit is out of the correct transition path. Theoretically speaking, when a circuit becomes stable, it can be checked whether or not the stable state is proper. If not, a fault has occurred. In practice, however, the state variables are unobservable during the off-line testing unless a scan structure is applied. Therefore, by observing the primary outputs of a non-scan structure the following effects may occur due to a fault: (1) the outputs are stable and proper; (2) the outputs are stable, but improper; and (3) outputs are unstable (the duration exceeds a predetermined time period for the worst case). The latter two cases imply that the fault effects are obviously observable at the primary outputs and thus the faults are detectable. On the other hand, the first case results in that the faulty circuit is equivalent to the fault-free circuit and thus detecting such fault effects becomes rather difficult. Such fault effects are generally caused by either *logic redundancy* or *state*

oscillation.

Redundancy is sometimes desirable for hazard protection in ASLCs. However, it may also be introduced unintentionally due in part to the implementation of an improper synthesis procedure, such as improperly encoded state variables or improper assignment of the don't care terms. Since redundant circuits cannot be tested, fault detection can then be a formidable problem. State oscillations do not occur in CSLCs because the clock prevents the state variable values from being used to generate excitations until they become stable, thus effectively ensuring that they appear to change simultaneously [9]. However, state oscillations may occur in ASLCs in the presence of hazards and/or races. They may be avoided by either adding extra delay elements, or eliminating hazards and races. But, they may occur as a result of faults. If the resultant outputs associated with the oscillating states are different in the presence of faults, then the unstable outputs indicate the existence of faults. However, if the resultant outputs are the same as desired values, the faulty circuit is interpreted to be fault-free. As a result, the indeterminate stable state and the unpredictable next state make the test generation process rather complicated. Therefore, in order to simplify the test generation process, while still increasing fault coverages, for ASLCs, it is necessary to identify and eliminate redundant faults and to avoid state oscillations in the presence of faults.

Due to the lack of controllability and observability, testing of sequential circuits has been recognized as a very difficult task. In order to reduce the complexity of the test generation problem, scan designs [20-24] have been proposed and successfully implemented for CSLCs. With scan structures, all state variables of a sequential circuit are completely controllable and observable from primary inputs and outputs. Thus, the test generation problem is reduced to one of just testing the combinational logic. In other words, scan structures can be used to further improve the controllability and observability of the synthesized ASLCs.

Recently, research efforts have successfully and extensively dealt with test

generation, design for testability, and synthesis for testability of CSLCs [10-14]. Little emphasis, however, has been devoted to their ASLC counterparts [15-19]. Since CSLCs may be considered to be a special case of ASLCs with certain constraints on input changes, the fault effects that occur in CSLCs will also occur in ASLCs. If an ASLC is synthesized such that each fault can be tested without generating critical races, then the circuit is testable. Otherwise, state oscillations may result during testing, and some faults may not be properly detected.

1.2 Research Tasks

This research achieves the following two tasks: (1) fault effects and synthesis properties for both Huffman-modeled and STG-modeled ASLCs; and (2) asynchronous scan design methodology.

Based on the single stuck-at fault model, this study investigates the fault effects such as redundant faults and state oscillations in ASLCs. Based on those fault effects, several synthesis properties are presented to synthesize testable ASLCs. Both Huffman-modeled and STG-modeled ASLCs are considered. In order to reduce the complexity of the testing process, while still increasing testability of a synthesized ASLC, an asynchronous scan design methodology, ASLCScan, is developed for both Huffman and STG modeled circuits, in which a race-free and hazard-free scan structure is implemented.

1.3 Organization

This thesis is organized as follows: Chapter 2 reviews the previous works that achieved these research tasks. Chapter 3 describes the fault effects and synthesis properties in both Huffman-modeled and STG-modeled ASLCs. The scan design methodology, ASLCScan, is presented with some design examples in Chapter 4. Finally, conclusions and future research topics are given in Chapter 5.

Chapter 2

Background

This chapter briefly reviews the existing models for designing and synthesizing ASLCs. Two models have been commonly implemented. In addition to the conventional Huffman model, a signal transition graph (STG) model has been recently developed for designing control circuits. Taking advantage of these two models, the development of a self-timed model for designing asynchronous finite state machines (AFSMs) is also described. The major problems in each synthesis model which are related to this thesis research are also discussed. In Section 2.2, the single stuck-at fault model commonly used in CSLCs is presented. Based on the fault model, fault effects and techniques for testability design and synthesis are discussed.

2.1 Synthesis Procedures for ASLCs

In this section, the synthesis procedure of Huffman model [2,3] is first presented. Those problems such as the state assignments and hazards are discussed. This is followed by the discussion of the STG model [1,5-6] and its synthesis procedure. The issues on how to generate a realizable circuit and how to avoid hazards are also addressed. Finally, the self-timed model for AFSMs developed in this study is presented.

2.1.1 Huffman-modeled ASLCs

A Huffman-modeled ASLC is generally described by a quintuple (S,I,O,δ,ω), where S is a finite set of internal states; I is a finite set of inputs; O is a finite set of outputs; the mapping δ is the *next-state function*, δ: I×S → S; and the mapping ω is the *output function*, ω: I×S → O (*Mealy machine*) or ω: S → O (*Moore machine*). Figure 2.1 shows the block diagram of a Huffman-modeled ASLC, where the input state $I=(x_1,x_2,..,x_r)$, the present state $y=(y_1,y_2,...,y_n)$, the next state $Y=(Y_1,Y_2,..,Y_n)$, and the output state $O=(Z_1,Z_2,..,Z_m)$. A Huffman-modeled ASLC is generally defined in terms of a *flow table*

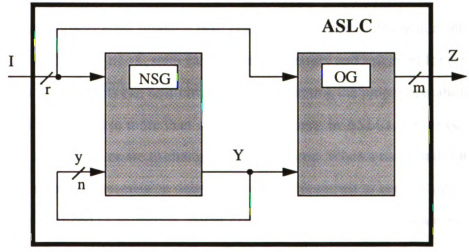


Figure 2.1: A Huffman-modeled ASLC with next-state generator (NSG) and output generator (OG).

in which each column of the flow table represents an input state and each row represents an internal state.The combination of the internal state y and the input state I is called the *total state*. The entries of the flow table represent the next internal states of the circuit. When the next internal state Y is the same as the present state y, the total state is said to be *stable*, otherwise it is an *unstable* state. In a *normal-mode* flow table, any state transition leads directly to a stable state and no output is required to change more than once during

the transition. The Huffman-modeled ASLC assumes that the line and gate delays are bounded with no restrictions imposed on their relative magnitude and it is operated in the fundamental mode, i.e., the input state cannot change until the circuit stabilizes, with only one input allowed to change at a time. Design procedure for ASLCs is generally comprised of five major steps [3,25]: (1) map a functional design specification into a primitive flow table; (2) derive the merged flow table; (3) encode the internal state variables to avoid critical races; (4) generate the state transition table and output table; and (5) eliminate static hazards and implement the circuit using two-level logic. Among these five steps, the state assignments and hazard problems are of major concern in this study.

State Assignments

The state assignment problem is to encode the internal states with a set of binary vectors. In CSLCs, $\lceil \log_2 n \rceil = s_0$ state variables are necessary and sufficient for representing n states. State assignments can be arbitrary without affecting the proper operations, as long as no coding is assigned to more than one state. However, in ASLCs, a *race* occurs when two or more state variables are to change at the same time. When a race condition exists in a circuit, the unequal transmission delays may cause the circuit to reach stable state other than the one intended. The race is called *critical*; otherwise the races are *non-critical.* Therefore, s_0 state variables may not be sufficient to encode the internal states in an ASLC for avoiding critical races. Races can naturally be classified as being either an *intrinsic race* (IR), or a *generated race* (GR) [26]. A GR is caused by encoding 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. The necessary, but not sufficient, condition for making race-free state assignments is that all the link degrees in a given flow table must be less than or equal to the number of state variables. Note that the reason why so many state assignment algorithms exist is that it is not always possible to derive an unicode state assignment in which each state transition only requires a single state variable to change. Among the existing algorithms, the following two commonly used algorithms are employed in this

study: the *totally sequential state assignment* [26-28], which is also referred to as an unit-distance code (UDC) state assignment, and the *single transition time* (STT) *state assignment* [29-33].

In the UDC state assignment, the internal state variables are assigned in such a way that each inter-row transition is represented by a set of adjacent states (two states differ in only one single variable) and only one state variable is excited at any time during a state transition. Races in ASLCs with the UDC state assignment can always be eliminated by creating cycles, i.e., directing the circuit through intermediate unstable states before it reaches its final desirable stable state with or without additional internal states. A *node-weight diagram* (NWD), a variation of a binary n-cube connection diagram, as shown in Figure 2.2, has been implemented to facilitate the UDC state assignment [26]. The level simply represents the weight of a code, i.e., the number of 1's in that code. It is obvious that the Hamming distance between two codes in the same level is always greater than or equal to 2.

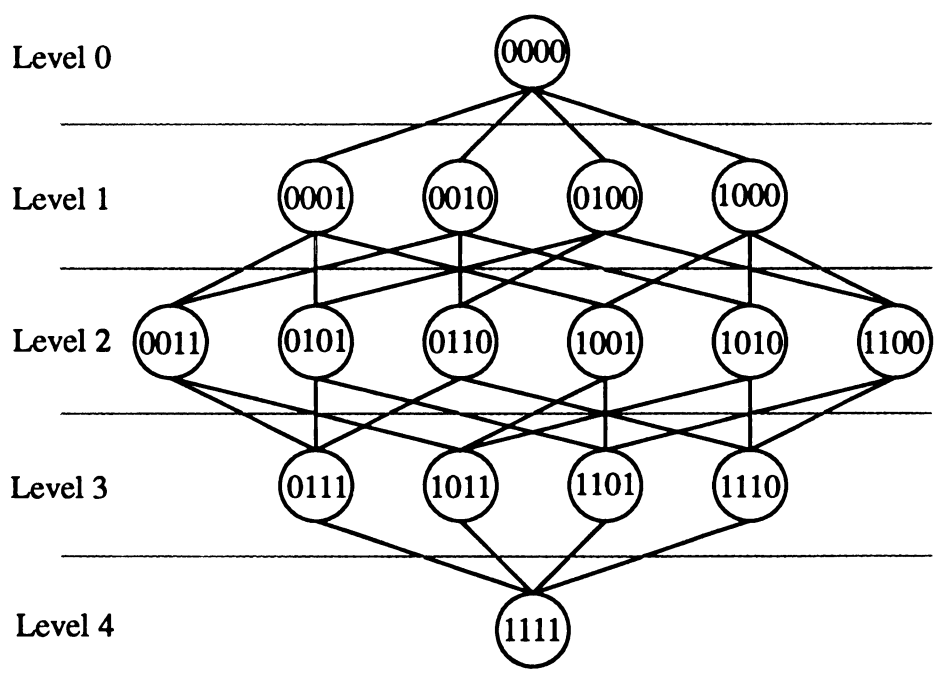


Figure 2.2: Geometric representation of a 4-NWD.

A *direct* transition from an internal state S_i to S_j, denoted as $[S_i,S_j]$, is a transition whereby all internal state variables that are to undergo a change of state are simultaneously excited. A *K-set* exists in a single column of a flow table and consists of all (k-1) unstable state entries leading to the same stable state, together with that stable state. A pair of states consisting of one unstable state and the stable state of a K-set is called a *transition pair.* The STT state assignment generally implements a partitioning scheme to distinguish the transition pairs [30,34]. Based on Tracey's Theorem, the state transitions with different stable states in the same column of the flow table can be distinguished by at least one state variable. As a result, all the state transitions are direct and all the races are non-critical. In other words, such a state assignment avoids all critical races and, as its name implies, each state transition is executed within a single transition time. Results in [26] have shown that the ASLC design implementation with the UDC state assignment requires fewer states than that with the STT state assignment.

Hazards

Any physical switching circuit has delays associated with gates and interconnecting lines which are usually called *stray delays*. Delays are not important in CSLCs due to the existence of clock synchronization. However, in ASLCs, delays may cause hazard problems and result in an incorrect operation. A hazard condition exists if there exist some possible combinations of values of stray delays which will produce a glitch for some input changes. The hazards can be further classified into two categories: combinational and sequential hazards [3,35,36]. The static and dynamic hazards are combinational hazards, while the essential hazard is a sequential hazard. The hazards are generally avoided by either adjusting the path delays or adding the delay elements in the feedback lines of the internal state variables. Techniques have been reported in [3,28,37] that identify the hazard conditions and design ASLCs which are free of delay elements.

2.1.2 STG-Modeled ASLCs

ASLCs are traditionally designed by using the FSM model. Two modifications to the Huffman model have been suggested [7]: (1) the use of total states instead of input, output, and state variables; (2) the use of a signaling protocol to alleviate the problems due to controlled changes of inputs. A total state $s \in S$ (a set of state variables) is a binary vector representing the state of all signals in a circuit, where the signals are the input and output terminals of the logic components. A set of the signal transitions $T = M \times \{+,-\}$, where M is all the I/O terminals of a circuit. Therefore, a machine behavior can be represented simply by (S,T,N), where N is the next-state function and $N: S \times T \rightarrow S$. Using the signal protocol, the signal changes can be controlled if the request/acknowledge protocol is used at the input and output ports of a signal.

The *signal transition graph* (STG) is an event-based specification for ASLCs [1,5,38]. It specifies the behavior of both the circuit and the environment where it operates. Further, the STGs can be viewed as *interpreted free-choice Petri-nets*, where the *transitions* in nets are interpreted as the value changes on input/output signals of a specified circuit, and the *places* in nets are specified by causal relations. A place can be *marked* with one or more *tokens*, meaning that the corresponding condition holds in the circuit. The causal relations joining pairs of transitions represent how the circuit and its environment can react to signal transitions. Note that the STG treats the set of input and non-input signal transitions in the same way. The fundamental difference between the transitions of input and non-input signals is that the former are caused by the external environment while the latter are caused by the system. The firing rules can be specified as follows [38,39]: (1) When all the pre-conditions of a transition are marked, the transition may fire, i.e. the transition is *enabled*; (2) When the transition is fired, tokens are removed from its pre-conditions and added to its post-conditions. The firing rules specify the desired operation sequences which are independent of the time metric, therefore the concurrency and ordering of signal changes can be easily specified.

For simplicity, places with exactly one predecessor and one successor can be omitted. However, if a place has more than one successor, then the free-choice place must be presented in the nets. Therefore, for the STG without choices [5], it can be represented by a directed graph with a triple (T, R, T_{EO}), where T = M ×{+,-} is a set of the signal transitions in which the rising transition, 0⟶1, of a signal x is denoted as x^+ and a falling transition, 1⟶0, as x^-, R ⊆ T×T is the causal relation which specifies the relation over the set of transitions, and T_{EO} is the set of transitions which are enabled in the initial state of the circuit. For example, Figure 2.3(a) shows a STG representation, where a signal transition is represented by a vertex, arcs between transitions represent the causal relation, the circle represents the token, and x^+ is enabled in this making. Based on the firing rules, after x^+ is fired, a token is removed from each of the incoming arcs, $y^-\rightarrow x^+$, and adds to each outgoing arc, $x^+\rightarrow z^+$ and $x^+\rightarrow y^+$, as shown in Figure 2.3(b). Further, since x^+

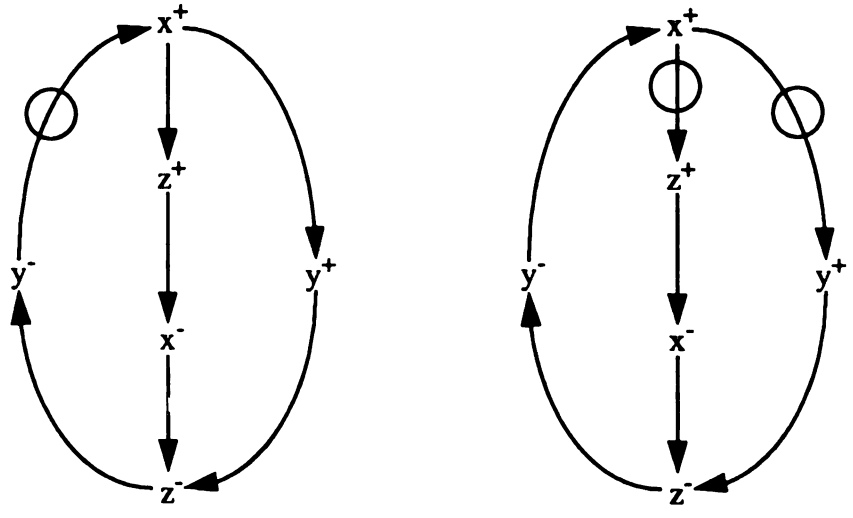


Figure 2.3: An STG representation (a) Initial marking with x^+ enabled; and (b) The marking after x^+ fired.

causes transitions z^+ and y^+ to be enabled, therefore x^+ and z^+ (y^+) are ordered. However, transitions z^+ and y^+ do not share any pre-condition, so they can fire in any order, i.e. z^+ and y^+ can be operated concurrently. This process can be continued by considering all possible token flows in the STG.

Two problems of concern in this study are how to achieve a realizable STG-modeled ASLC and how to avoid hazards.

Realizable STG-modeled ASLCs

A realizable STG [1,39] must first satisfy the conditions of *liveness* and *safeness*. A marking is *live* if every transition can be enabled through some sequence of firing transition. *Safeness* means that no more than one token can be present in a place in any reachable marking. Further, for a free-choice net, if a place has more than one successor, then it must be the only predecessor of those transition. In addition, since each marking in the STG implies a value of each signal, two more conditions must also be satisfied for being a realizable STG: *consistent labeling* and *unique state coding* (USC) [5,39,40]. More specifically, the *current state* in a STG is given by the concatenation of all current input and output values, and the value label attached to each marking must be consistent across different firing sequences. For example, the current states (xyz) are (000) and (100) in Figures 2.3(a) and 2.3(b), respectively. A signal t is said to be *consistent* for the current state, if the transition t^+ (t^-) is enabled in the current state, then the value of t in this state must be 0 (1). For consistent labeling, every signal in the current state must be consistent. The USC property implies that two markings must enable the same set of transitions, if they have the same label. Otherwise, the behavior of the circuit will be unpredictable. An immediate solution to satisfy the USC property is to add an extra internal signal to distinguish different behavior with the same label.

The synthesis procedure for STG-modeled ASLCs is comprised of (1) constructing a STG which satisfies the five conditions discussed above; and (2) deriving the next state values for the current state. The next state values can be derived by the

following rules: (1) For a signal t in the current state, if there is no signal transition labelled t*, where $t^* \in \{t^+, t^-\}$, from this current state, then the next value of t is unchanged; and (2) If there exists a signal transition labelled t* from this current state, then the next value of t is complemented [39]. Therefore, by an exhaustive simulation of all reachable markings, the logic implementation can be achieved.

Hazards

Although the implementations of STG-modeled ASLCs are straightforward, hazards may still occur due to the unbalanced path delays. Hazards in STG-modeled ASLCs can be simulated by applying all possible allowed input sequences. By checking all the 1-set and 0-set covers in the realized logic of the input sequences, the relative delays among different paths can be identified. If hazards occur, delay elements may be added to compensate the path delays [39,41]. The tool, *astg* in *sis*, developed by University of California at Berkeley can be used to produce hazard-free ASLCs. However, it is a time-consuming process and the resultant circuit always degrades in speed due to the added delay elements.

2.1.3 Self-Timed AFSMs

Since Huffman-modeled ASLCs assume the fundamental mode operation with single input change, the design of *asynchronous finite state machines* (AFSMs) has been limited because multiple-input changes are not allowed. *Self-timed logic* provides a method for designing ASLCs such that their correct behavior depends neither on the speed of their components nor on the delay along communication wires. Various self-timed ASLC designs have been reported [3,42-45] in which input spacer words or encoded input and output data are used to generate completion signal. The completion signal is generated whenever both outputs and internal states are stable. The completion-signal concept has been presented for the design of a parallel binary adder which is an iterative combinational circuit [46]. It was realized that, depending on the numbers being added, the time required

for the outputs to respond varies over a wide range. When a carry bit must "ripple" through the entire circuit, the delay is maximum. Cases in which there are no carry, the delay may be less than the maximum value by a factor of about n for an n-bit adder. Results have shown that, if the numbers are randomly chosen, the average time required varies approximately as $\log_2(5n/4)$. However, if the adder does not return some signals indicating when it has completed each computation, then it is necessary to allow the maximum time in every case.

Since conventional Huffman-modeled ASLCs cannot handle unrestricted input changes [7], it is necessary to impose certain restrictions on the input changes [47]. By nature, datum signals can assume a large number of states and one datum may follow another in a random order. In contrast, timing or control signals have a limited number of states and a limited number of sequences of states changed. Therefore, datum signals can make unrestricted changes while control signals can be specified such that they change in a predictable and orderly manner. Based on this concept, the model for self-timed AFSM can be generalized as shown in Figure 2.4 [48]. The environment supports the input data I and a control signal R_{eq} to trigger the operation of an AFSM. After the AFSM completes its task, it produces the output values O associated with a completion signal C_p to the environment such that the next operation can be processed. In addition, the controller is used to guarantee a correct operation sequence between the AFSM and the environment.

By taking advantage of the design procedures of both the Huffman-modeled and STG-modeled ASLCs, a self-timed AFSM is designed as follows [48]: The operating ASLC is a Huffman-modeled ASLC with a self-timed SR-latch as shown in Figure 2.5 [3,49], in which the signal C_{Zi} is used to indicate the status of the SR-latch, while the controller can be synthesized with a STG-modeled circuit.

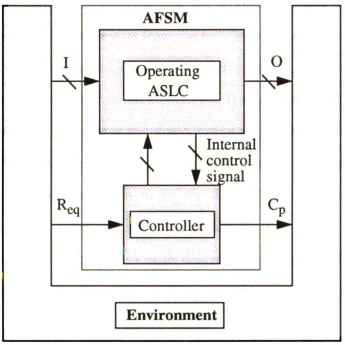


Figure 2.4: An AFSM with data and control signals.

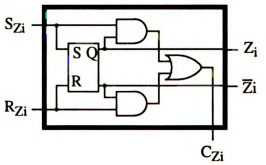


Figure 2.5: A self-timed SR-latch.

Hazards

When designing self-timed circuits, the problem of delay hazards [3,42] must be considered. The delay hazard results from the fact that several of the OR-gate inputs go on during the course of a transition. Once one of them goes on, it is impossible to determine from the output signal whether or not another one is on. If the circuit is designed to be that only one input to each OR gate is energized during a given transition, then we are able to tell from the output when the circuit has reached a stable state after an input change. Under the unbounded gate delay assumption and the use of SR-latches to implement the logic circuit as shown in [42], we only need to consider the problem caused by delay hazards and to eliminate them. The strategy is to design the circuit such that no more than one input to any OR gate is 1 at any time.

2.2 Testing Problems in CSLCs

The testability of a logic circuit has a great effect on the cost of generating and applying tests for the circuit. The greater the testability of a circuit is, the easier the testing will be. The problem of test generation is in general NP-complete and thus very intractable for very large circuits. This demonstrates the necessity of designing easily testable circuits. With the advent of VLSI circuits, the need for methods of design for testability becomes more and more urgent. Many methods for measuring testability in logic circuits have been proposed [35,50]. These methods in common introduce two measures, controllability and observability, to estimate testability. *Controllability* is defined as a measure of how easily the internal logic of the circuit can be controlled from its primary inputs. *Observability* is defined as a measure of how easily the internal logic of the circuit can be observed at its primary outputs. The process of test generation consists of the tasks of controlling and observing internal logic values. Representatives of controlling and observing tasks in test generation are the consistency and D-drive operations of the D-algorithm [50,51], respectively. Therefore, the main issue for the testability of CSLCs is

strongly related to the controllability and observability of internal state variables.

2.2.1 Fault Model

A *fault* in a circuit is a physical defect of one or more components. Faults can be classified as *logical* or *parametric*. A *logical fault* is one that causes the logic function of a circuit element or an input signal to be changed to some other logic function; a *parametric fault* alters the magnitude of a circuit parameter, causing a change in some factors such as circuit speed, currents, or voltages [50,51].

Fault models serve two purposes. First, they help generate tests, and second, they help evaluate test quality defined in terms of coverage of modeled faults. Stuck-at faults are only the simplest faults to analyze, but they also have been proven to be very effective in representing the faulty behavior of actual devices. In this study, single stuck-at faults are assumed, i.e., only one stuck-at-0(1) fault occurs at a gate input or output at a time.

2.2.2 Synthesis for Testability

The techniques of synthesis for testability in CSLCs are to improve the controllability and observability of internal state variables without extra input and output pins. Therefore, the strategy is to identify all the redundant faults in the circuit and then to eliminate them.

Redundant Faults in CSLCs

In a CSLC, a fault may be redundant [11-14], i.e., untestable. Redundant faults may be *combinationally redundant* (CRFs) or *sequentially redundant* (SRFs). The effect of a CRF cannot be propagated to the primary outputs or the next state lines, beginning from any state, with any input vector. Three SRFs are classified: (1) *Invalid-state faults*; (2) *Isomorphic faults*; and (3) *Equivalent-state faults.*

A CSLC can be generally represented by a *state transition graph* (C_STG to distinguish from the STG for signal transition graph in ASLCs), where an edge joins s_i

and s_j if there is any vector of primary input values that causes the circuit to evolve from state s_i to state s_j. A state which can be reached from the reset state via some input vector sequence is called a *valid state* in the C_STG; otherwise, it is an *invalid state*. An edge is said to be *corrupted* by a fault if either the fan-out state or an output of this edge is changed because of the existence of the fault. The *invalid-state fault* does not corrupt any fan-out edge of a valid state in the C_STG, but does corrupt the fan-out edge of an invalid state [11].

An *isomorphic fault* results in a faulty machine that is isomorphic (with different encoding) to the original machine. In [11], it was shown that stuck-at faults in a sequential machine implemented by a two-level combinational network could not cause isomorphism. A *differentiating sequence* for states s_1 and s_2 in a machine is a sequence of inputs $i_1, i_2, .., i_N$ such that if the machine begins in state s_1, the output associated with input i_N differs from that if the machine begins in state s_2. Two states in a C_STG are *equivalent* if they do not have a differentiating sequence. The *equivalent-state faults* cause the interchange/creating of equivalent states in a C_STG.

Example 1: (Equivalent-State Sequential Redundant Faults)

Consider a C_STG of a FSM, as shown in Figure 2.6(a), consisting of three states. Figure 2.6(b) shows its next-state and output equations where the invalid state (11) is used as don't care for logic minimization. Figure 2.6(c) illustrates the complete C_STG for this circuit. Assume that a stuck-at-0 fault occurs at the x-input of the AND gate, $x\bar{y}_1\bar{y}_0$, of Y_0, the corresponding faulty C_STG is shown in Figure 2.6(d). Starting from state (00) any test sequence will produce the same output sequence for both fault-free and faulty C_STGs, where states (00) and (01) are equivalent. Since the equivalent states are all valid states, the fault is referred to as a *valid/valid equivalent-state fault.* Similarly, assume that a stuck-at-1 fault occurs at the x-input of the AND gate, $x\bar{y}_1\bar{y}_0$, of Y_0, the faulty C_STG is illustrated in Figure 2.6(e), where states (00) and (11) are equivalent. Since state (11) is an invalid state, the fault is referred to as a *valid/invalid equivalent-state fault.*

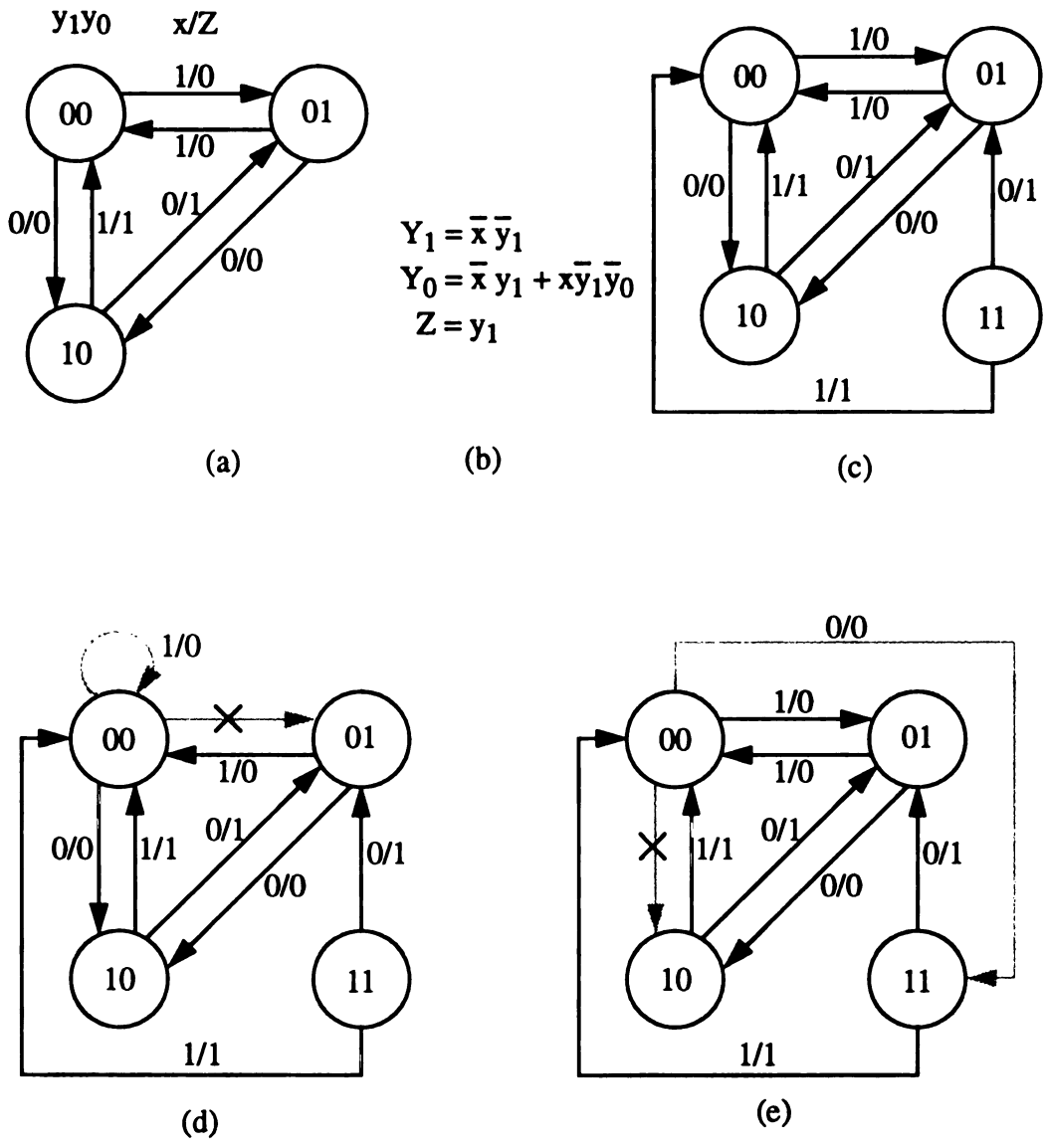


Figure 2.6: SRFs in CSLCs (a) State Transition Graph (C_STG); (b) Logic implementation of (a); (c) The complete C_STG for (b); (d) Valid/valid equivalent-SRF; and (e) Valid/invalid equivalent-SRF.

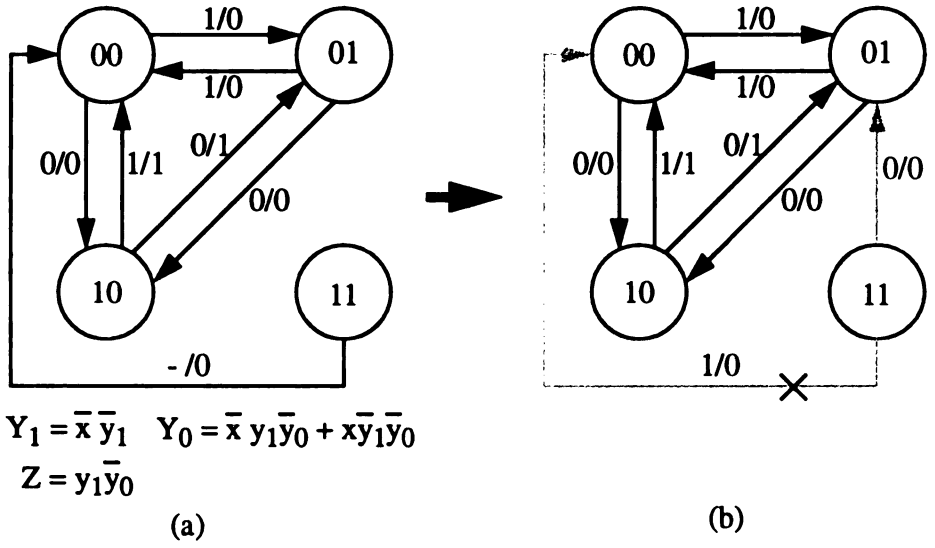
Example 2: (Invalid-State Sequential Redundant Faults)

Invalid-state SRFs may occur as a result of improperly assigning the don't care states. Consider the invalid state (11). Suppose that its next state, a don't care entry, is assigned to state (00) and output is 0. Figure 2.7(a) shows the logic implementation and the complete C_STG. Assume that a stuck-at-1 fault occurs at the $\bar{y}_0$-input of the AND gate, $\bar{x}y_1\bar{y}_0$, of Y_0, the corresponding faulty C_STG is illustrated in Figure 2.7(b). The fault can be excited by state (11), but the invalid state (11) cannot be reached from any valid state. Thus, the fault is an *invalid-state redundant fault.*

Example 3: (Isomorphic Sequential Redundant Faults)

The *isomorphic fault* results in a faulty machine that is isomorphic (with different encoding) to the original machine. Figure 2.8 illustrates such a fault effect, where states (01) and (10) are interchanged in the presence of a fault. It has been shown that stuck-at faults in a sequential machine implemented by a two-level combinational network could not cause states to be isomorphic [11].

CRFs can be eliminated via combinational logic optimization alone [10-14]. General methods for eliminating the above SRFs have been introduced in [14]. Isomorph-SRFs can be eliminated if the combinational part of a sequential circuit is irredundant and implements either a two-level logic or an inversion-parity invariant multi-level logic. One solution to eliminating equivalent-SRFs and invalid-state SRFs is to expand the original circuit so that 2^n reachable and distinguishable states are all assigned properly, where n is the number of flip-flops. An alternative solution is to implement a state minimization process eliminating valid/valid equivalent-SRFs and invalid-state SRFs, or to carefully assign the invalid states eliminating valid/invalid equivalent-SRFs. However, the latter approach is a time-consuming process.



$Y_1 = \bar{x}\,\bar{y}_1 \quad Y_0 = \bar{x}\,y_1\bar{y}_0 + x\bar{y}_1\bar{y}_0$

$Z = y_1\bar{y}_0$

(a) (b)

Figure 2.7: Invalid-SRF (a) Fault-free C_STG; and (b) Faulty C_STG.

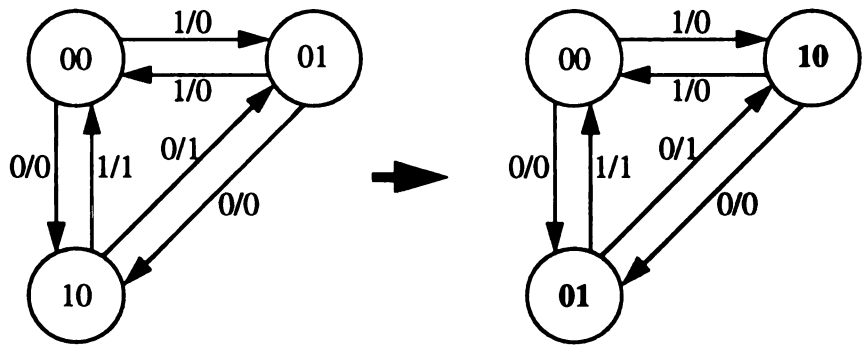


Figure 2.8: Isomorph-SRF.

Test Generation for CSLCs

Figure 2.9 shows the conventional iterative array model used for the test generation in CSLCs. Each identical cell in this array represents a different time frame from 1 to n. The symbols PI, PO, PS, and NS correspond to primary input, primary output, present state, and next state, respectively. Based on this model, the sequential test generation [10] is decomposed into two steps. First, an input sequence, T1 = (PI^i, PI^{i+1}, ..., PI^n), and an initial state S0, that excites and propagates the effect of the fault to the PO^n are found. This step is called *fault excitation and propagation* and T1 is called the *fault propagation sequence*. Next, *state justification* on S0 is performed; i.e. a path from reset state to S0 is found consisting of another sequence of input vectors, T0 = (PI^1, PI^2, ..., PI^{i-1}). T0 is called the *justification sequence*. Finally, the test sequence is the concatenation of T0 and T1.

Similar concepts have been implemented for test generation of ASLCs [15,16]. Since each state transition in the ASLCs may take more than one time step before stabilizing, each cell in Figure 2.9(b) is again represented by iterative arrays as shown in Figure 2.10. The number of repeated arrays *p* in each time frame *i* depends on the number of state transition steps.

2.2.3 Design for Testability

Design for testability techniques are divided into two categories [21,35]: *ad hoc techniques* and *structured approaches*. Ad hoc techniques consist of heuristic methods to solve testing problems for specific designs, but not directed at solving the general sequential problem. This is contrasted with the second category of structured approaches which are trying to solve the general problem with a design methodology. The structured designs are generally applicable and usually involve a set of design rules for which designs are implemented. Due to the lack of controllability and observability, testing of sequential circuits has been recognized as a very difficult task. In order to reduce the

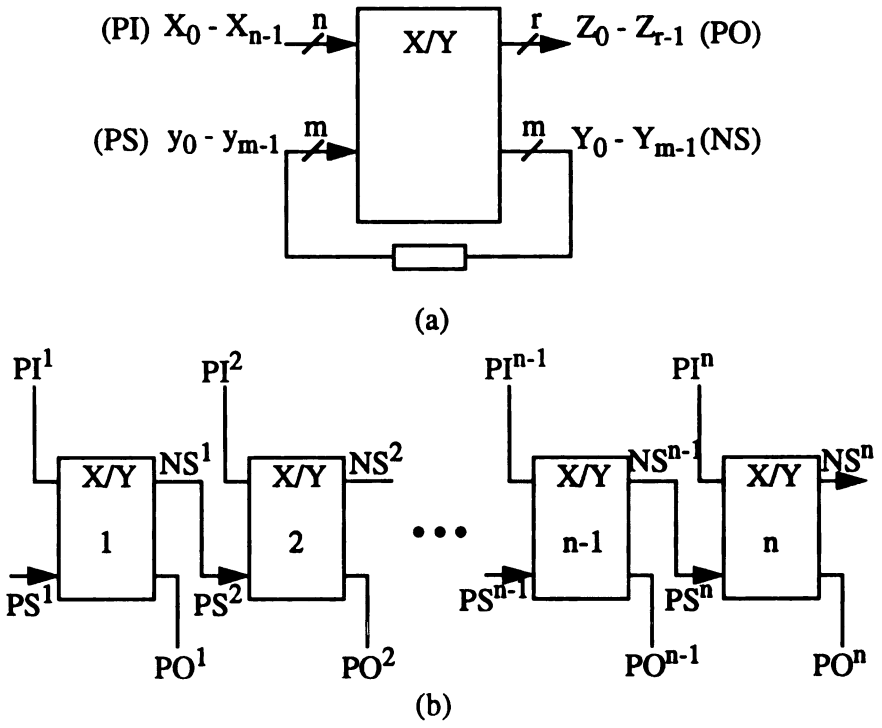


Figure 2.9: (a) A sequential circuit; and (b) Iterative array model.

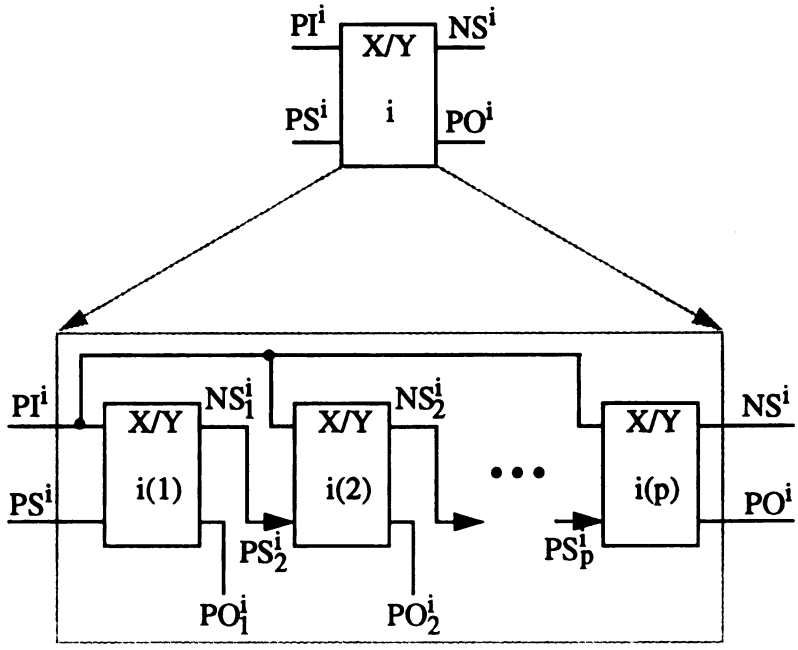


Figure 2.10: Test generation iterative array model for ASLCs.

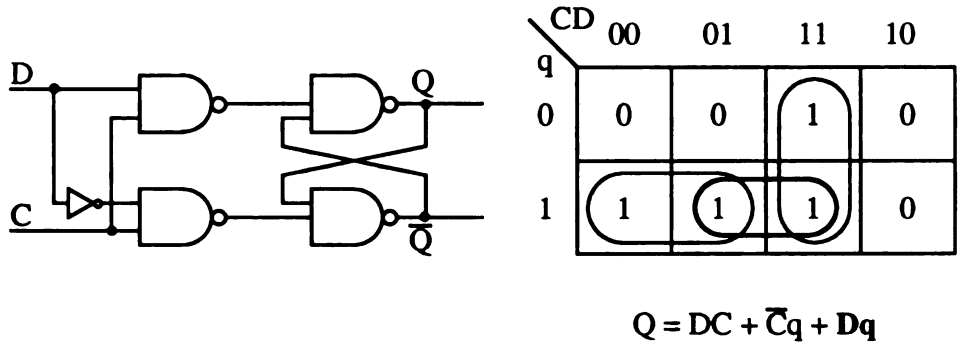
complexity of test generation problem, scan designs have been proposed and successfully implemented for CSLCs [20-24,55]. With scan structures, all state variables of a sequential circuit are completely controllable and observable from primary inputs and outputs. Thus, the test generation problem is reduced to one of just testing the combinational logic. In this subsection, a scan design, namely, *Level-Sensitive Scan Design* (LSSD) is discussed.

Level-Sensitive Scan Design

Level Sensitive Scan Design (LSSD) is a technique developed by IBM to deal with the testability issues [20]. The LSSD method, coupling a level-sensitive design concept with scan operation, ensures race-free system operation as well as race-free testing. The term *level sensitive* refers to the fact that the steady-state response of a circuit to any allowable input change is independent of delays within the circuit. Also, the steady-state response of the circuit must be independent of the order of input-value changes in the event of simultaneous multiple changes. The key elements in the LSSD methodology are the Polarity-Hold latch (PH) and polarity-hold Shift Register Latch (SRL) shown in Figures 2.11(a) and 2.11(b). The SRL shown in Figure 2.11(b) is simply a master-slave, D-type flip-flop that is provided with an extra input stage for signals I and A. During normal operation, the test input stage of the flip-flop is disabled and the circuit performs as a normal master-slave, D-type flip-flop. During testing, the normal input stage is disabled to allow the flip-flop to be loaded with the test inputs and scan out the values of the internal state variables by connecting all the SRLs to form one or more independent shift registers. Figure 2.12 shows the double-latch LSSD implementation [20]. Race conditions are avoided by using separate, non-overlapping clocks for the master-slave flip-flops, thereby providing the level-sensitive operation.

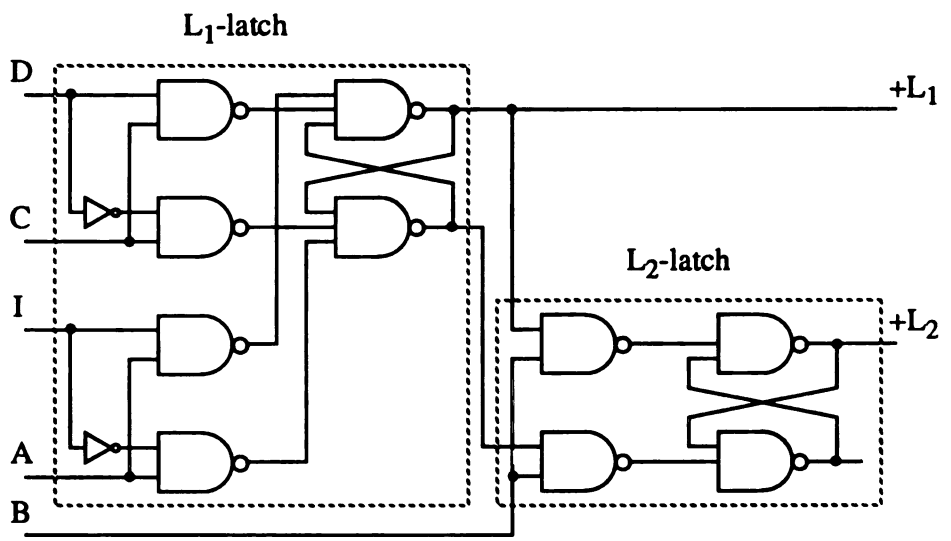
The LSSD methodology requires that the designer adheres to a set of basic design rules [20]. The purpose of the LSSD rules is to improve the testability of the resulting design. There are two fundamental constraints to the LSSD design [52]: (1) The value of

input datum should last for a long enough time when the pulse of storing signal returns to its inactive value; (2) The pulse width of the storing signal must be larger than the minimum time for regeneration in the feedback loop to take over control and maintain the new value. Without satisfying these two constraints, races and hazards are unavoidable. When these two constraints are satisfied, a key factor in the design of LSSD polarity-hold latches is that the hazard may occur during clock turns off when storing a "1". A variety of latch design techniques have been presented in [52,53] that solve the hazard problem. As shown in Figure 2.11(a), the problems of this hazard are eliminated by adding the hazard protection term, Dq, so that reliable operation can be achieved. Further, the correct shifting responses have been verified by a set of test patterns proposed by Bottorff et. al. [54], referred to as *BFGO test patterns*. Basically, the test, referred to as *BFGO test*, consists of two parts: a *flush test*, in which all shift clocks are turned on and a signal is flushed through the register from the scan input to the scan output; and a *shift test*, in which a 00110011 pattern is shifted through each register.



$$Q = DC + \overline{C}q + Dq$$

(a)



(b)

Figure 2.11: LSSD scan design (a) Polarity-hold latch and its corresponding K-map; and (b) Polarity-hold Shift Register Latch.

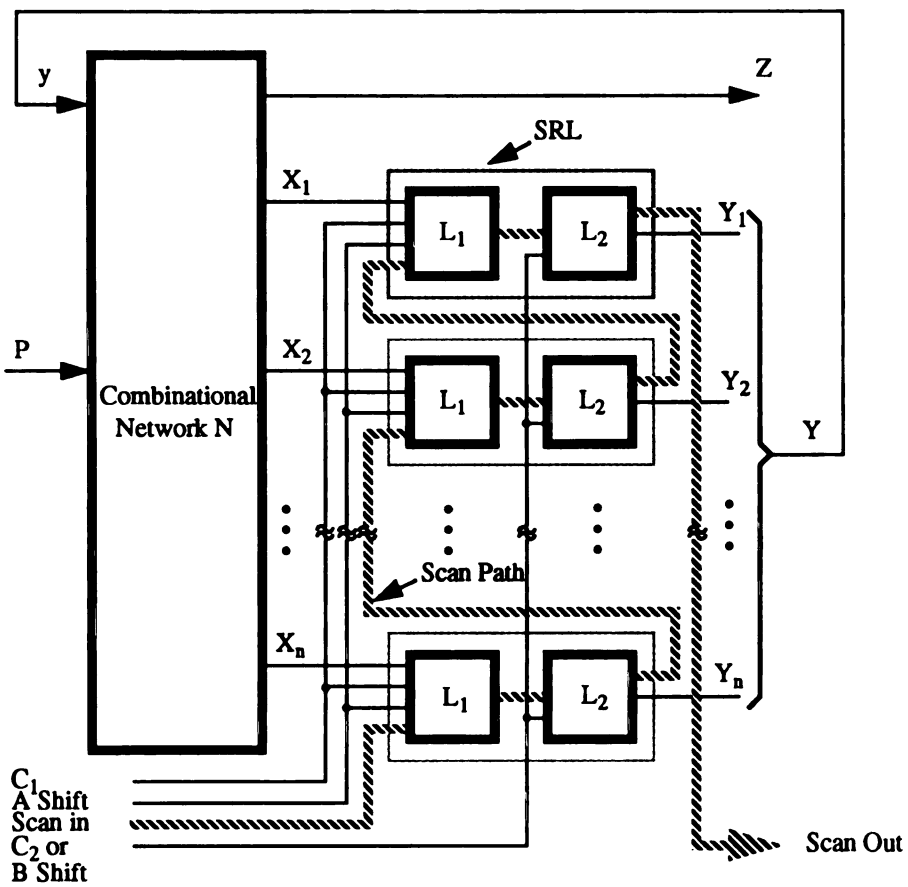


Figure 2.12: Double-latch LSSD implementation.

Chapter 3

Fault Effects and Testability Synthesis Properties in ASLCs

An ASLC can be implemented either with asynchronous SR-latches for its state variables, or without SR-latches [3,35]. The latter has also been referred to as with line feedback [35]. In Section 3.1, the fault effects and testability synthesis properties of the Huffman-modeled ASLCs implemented with line feedback is first discussed. Then, the Huffman-modeled ASLCs implemented with SR-latches are described in Section 3.2. In Section 3.3, the similarities and differences between Huffman-modeled and STG-modeled ASLCs are presented.

3.1 Huffman-Modeled ASLCs with Line Feedback

In this section, the Huffman-modeled ASLCs are synthesized with the following assumptions: (1) The circuits are operated and tested in the normal fundamental mode with single input change; (2) Each circuit has a reset state and all input test sequences begin from the reset state; (3) The circuits are implemented with two-level logic which consists of only prime and irredundant implicants except the redundant logic for static hazard protection, and there exist no shared logic for the next state equations and output equations; and (4) The single stuck-at fault model is employed. The assumptions (3) and (4) imply that a single stuck-at fault only affects either one next-state equation or one output equation at a time. This simplifies the testing problem significantly. For state assignments, this chapter

considers two commonly used algorithms: *critical-race-free Single-Transition-Time* (*STT*) state assignment [3,30], and *race-free Unit-Distance-Code* (*UDC*) state assignment, also referred to as *totally sequential state assignment* [26,27]. For simplicity, the ASLC encoded with the STT (UDC) state assignment is referred to as the *STT-generated (UDC-generated) ASLC.*

Based on assumption (3), Figure 3.1 illustrates the block diagram of an ASLC implemented with line feedback, where OL and NSL represent the output logic and the next state logic, respectively.

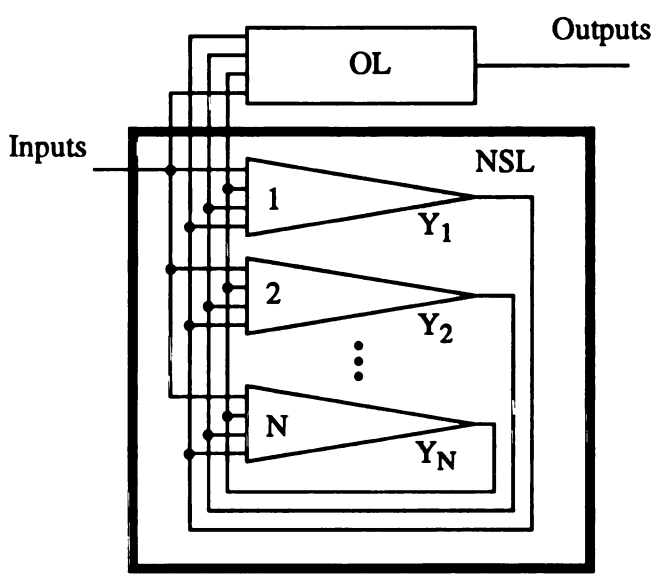


Figure 3.1: Partitioned ASLC without shared logic.

3.1.1 Fault Effects

CSLCs can be considered as a special case of ASLCs with certain constraints on input changes. Thus, the fault effects such as redundant faults that occur in CSLCs may also occur in ASLCs. In addition, the state oscillation is a special fault effect in ASLCs due to

critical races.

3.1.1.1 Redundant Faults

Similar to CSLCs, combinationally and sequentially redundant faults [10-14] may also occur in ASLCs. However, since stuck-at faults could not cause isomorphism in a circuit with two-level logic implementation, the sequentially redundant faults encountered here are *equivalent-state faults* and *invalid-state faults*.

Equivalent-State Redundant Faults

Two states are *inequivalent* if there exists at least a differentiating sequence for them [10]. Equivalent states are generally eliminated by state minimization process. However, equivalent states may occur as a result of faults. Equivalent-state redundant faults in ASLCs may be generated due to (1) *the constraint of single input change*, (2) *the non-critical races*, or (3) *delays*. In the CSLC implementations, the inputs can be any arbitrary combination. Thus, two states are *equivalent* if they are indistinguishable with all input combinations. Under the constraint of single input change in an ASLC, however, two states are equivalent if they are indistinguishable with the current input and its adjacent inputs. The fault that causes the equivalent states is redundant. On the other hand, a non-critical race generally causes more than one transition paths to stabilize at the same stable state. If the effect of a fault causes one of the transition paths to stabilize at a state that can be distinguished from the corresponding fault-free stable state, (of course, the current state must be reachable from the reset state), then the fault is detectable. However, if the fault effect forces the circuit to take one of the transition paths for the non-critical race, then the faulty circuit will end up with stabilizing at the same stable state as the fault-free circuit. Thus, it is also an equivalent-state redundant fault.

Equivalent-state redundant faults may also occur due to delays. More specifically, suppose that there exists a non-critical race between y_1 and y_2, and suppose also that the fault effect is detectable only through a transition path in which the change of y_2 is faster

than y_1. If the change of y_2 becomes slower than y_1 in the presence of fault, i.e. the fault propagation path is blocked, the fault is then an equivalent-state fault. In order to demonstrate the fault effects in ASLCs, Figure 3.2(a) shows a flow table of an ASLC. Figures 3.2(b) and 3.2(c) respectively illustrate the STT-generated ASLC and the UDC-generated ASLC, where the circled entries in the table represent the stable states, and the boldfaced terms in the final equations indicate the redundant terms for static hazard protection.

Example 1:

Consider the STT-generated ASLC shown in Figure 3.2(b). Assume that a stuck-at-1 fault occurs at the x_2-input of the AND gate, $\bar{x}_1 x_2 \bar{y}_1$, of Y_2. Figure 3.2(d) illustrates the transition table of the corresponding faulty circuit, where the entries with asterisks represent the contaminated entries caused by this fault and the boldfaced bits indicate the bit change due to the fault. The fault causes the entries $(x_1x_2\text{-}y_1y_2y_3)$=(00-000), (00-001), (00-010), and (00-011), to change from (000) to (010). Assume that the current state is (011) and the current input is (10). When the input is changed from (10) to (00), the next state will stabilize at (000) in the fault-free circuit and at (010) in the faulty circuit. With single input change constraint, the next applicable input is either (01) or (10). However, both faulty and fault-free circuits result in stabilizing at the same state (011), for the input (01), or (110), for the input (10). Thus, the fault is an equivalent-state redundant fault which is due to the single input change.

Consider a stuck-at-1 fault that occurs at x_2-input of the AND gate, $x_2y_1y_2$, of Y_1. Figure 3.2(e) shows the transition table of the corresponding faulty circuit. There are two essential prime entries $(x_1x_2\text{-}y_1y_2y_3)$=(00-110) and (10-111), as indicated by asterisks, that can be used to detect such a fault. The fault causes the entry $(x_1x_2\text{-}y_1y_2y_3)$=(00-110) to change from (000) to (100) and the entry $(x_1x_2\text{-}y_1y_2y_3)$=(10-111) to change from (011) to (111). We first consider the entry (00-110). Assume that the current state is (110) and the current input is (10). When the input is changed from (10) to (00), the state is changed from

	x_1x_2 00	01	11	10	x_1x_2 00	01	11	10
1	(1)	2	(1)	3	0	1	1	1
2	1	(2)	3	(2)	0	1	0	0
3	1	4	(3)	(3)	0	1	0	1
4	(4)	(4)	1	2	1	1	1	0

(a)

$Y_1Y_2Y_3$	Present Input (x_1x_2) 00	01	11	10	Z 00	01	11	10	
000	(000)	011	(000)	110	0	1	1	1	(1)
001	000	011	000	011	0	1	1	0	
011	000	(011)	110	(011)	0	1	0	0	(2)
010	000	011	110	110	0	1	0	1	
110	000	101	(110)	(110)	0	1	0	1	(3)
111	---	101	110	011	---	1	0	0	
101	(101)	(101)	000	011	1	1	1	0	(4)
100	000	101	000	110	0	1	1	1	

Present State($y_1y_2y_3$)

$$Y_1 = \bar{x}_1y_1y_3 + \bar{x}_1x_2y_1 + x_1x_2y_2 + x_1\bar{x}_2\bar{y}_3 + x_1y_2\bar{y}_3 + x_2y_1y_2$$
$$Y_2 = \bar{x}_1x_2\bar{y}_1 + x_1y_2 + x_1\bar{x}_2 + x_2\bar{y}_1y_2$$
$$Y_3 = \bar{x}_1y_1y_3 + \bar{x}_1x_2 + x_1\bar{x}_2y_3 + \bar{x}_2y_1y_3$$
$$Z = \bar{x}_1y_1y_3 + \bar{x}_1x_2 + x_2\bar{y}_2 + x_1\bar{x}_2\bar{y}_3 + x_1\bar{y}_2\bar{y}_3$$

(b)

Figure 3.2: An ASLC for Example 1 (a) Merged flow table and output table; (b) STT-generated ASLC; (c) UDC-generated ASLC; and (d)&(c) Transition tables of the corresponding faulty circuits.

$Y_1Y_2Y_3$ \ Present Input (x_1x_2)	00	01	11	10	
000	(000)	001	(000)	010	(1)
001	000	(001)	011	(001)	(2)
011	---	---	010	---	
010	000	110	(010)	(010)	(3)
110	---	100	---	---	
111	---	---	---	---	
101	---	---	---	001	
100	(100)	(100)	000	101	(4)

Present State ($y_1y_2y_3$)

$Y_1Y_2Y_3$ \ Present Input (x_1x_2)	00	01	11	10
000	(000)	001	(000)	010
001	000	(001)	011	(001)
011	000	110	010	000
010	000	110	(010)	(010)
110	100	100	000	101
111	100	100	010	001
101	100	100	011	001
100	(100)	(100)	000	101

$$Y_1 = \bar{x}_1y_1 + \bar{x}_2y_1\bar{y}_3 + \bar{x}_1x_2y_2$$
$$Y_2 = x_2\bar{y}_1y_2 + x_1x_2y_3 + x_1\bar{x}_2\bar{y}_1\bar{y}_3 + x_1\bar{y}_1y_2\bar{y}_3$$
$$Y_3 = x_1\bar{x}_2y_1 + \bar{x}_1x_2\bar{y}_1\bar{y}_2 + x_1\bar{y}_2y_3 + x_2\bar{y}_1\bar{y}_2y_3$$
$$Z = \bar{x}_1x_2 + x_2\bar{y}_2\bar{y}_3 + x_1\bar{x}_2\bar{y}_1\bar{y}_3 + \bar{x}_1y_1 + x_1\bar{y}_1\bar{y}_2\bar{y}_3$$

(c)

$Y_1Y_2Y_3$ \ Present Input (x_1x_2)	00	01	11	10	
000	010*	011	(000)	110	(1)
001	010*	011	000	011	
011	010*	(011)	110	(011)	(2)
010	010*	011	110	110	
110	000	101	(110)	(110)	(3)
111	101	101	110	011	
101	(101)	(101)	000	011	(4)
100	000	101	000	110	

Present State ($y_1y_2y_3$)

(d)

$Y_1Y_2Y_3$ \ Present Input (x_1x_2)	00	01	11	10	
000	(000)	011	(000)	110	(1)
001	000	011	000	011	
011	000	(011)	110	(011)	(2)
010	000	011	110	110	
110	100*	101	(110)	(110)	(3)
111	101	101	110	111*	
101	(101)	(101)	000	011	(4)
100	000	101	000	110	

Present State ($y_1y_2y_3$)

(e)

Figure 3.2: (Cont'd).

(110) to (000) in the fault-free circuit, as shown in Figure 3.2(b). Due to the non-critical race, the change may follow either one of these two paths: 110→010→000 or 110→100→000, and the circuit stabilizes at the stable state (000) in the fault-free circuit. On the other hand, in the presence of such a fault with input changed from (10) to (00), the state is changed from (110) to (100). This fault forces the faulty circuit to take the transition path 110→100→000 and to stabilize at the stable state (000). Thus, both faulty and fault-free circuits stabilize at the same stable state. This equivalent-state redundant fault is caused by the non-critical race.

Consider the entry (10-111) for the above stuck-at-1 fault. Assume that the current state is (101) and the input is (00). When the input is changed from (00) to (10), the state is changed from (101) to (011). The transition may take either 101→001→011, if the change of y_1 is faster than y_2, or 101→111→011, if the change of y_2 is faster than y_1, and the circuit is finally stabilized at the stable state (011). On the other hand, in the presence of such a fault, the transition path 101→111→011 is changed as 101→111→111. This implies that, if the change of y_2 is faster than y_1 and the stable states (011) and (111) are distinguishable, then this fault can be detected. However, the fault is an equivalent-state redundant fault due to the delay if the change of y_1 is faster than y_2. ♦

Equivalent states, under the single input change assumption, are inherently allowed in the fault-free STT-generated ASLCs. For example, states (001) and (011), in Figure 3.2(b), have the same state entries and output entries at the input columns (00), (01), and (10), i.e., both states are equivalent if the current input is (00). The existence of such equivalent states is because that non-critical race conditions are allowed in the STT state assignment. Thus, the race-free UDC-generated ASLCs generally have less equivalent-state redundant faults than the STT-generated ASLCs.

Invalid-State Redundant Faults

A state that cannot be reached from the reset state via some input sequences is

called an *invalid state*. The invalid-state redundant fault does not corrupt any fan-out edge of a valid state in the state transition diagram, but does corrupt the fan-out edge of an invalid state, i.e., the fault may only be excited by an invalid state. In CSLC implementations, the invalid-state redundant faults are caused by the existence of invalid state(s). The invalid-state redundant faults in ASLCs are caused not only by the existence of invalid states, but also by improperly assigning the don't care terms in the flow table.

Example 2:

Consider the transition table of a STT-generated ASLC shown in Figure 3.3(c). Since only one state variable is changed during every state transition, it is also an UDC-generated ASLC. Assume a stuck-at-0 fault that occurs at the output of the AND gate, $y_1\bar{y}_2$, of Y_1. Figure 3.3(d) illustrates the transition table of the corresponding faulty circuit. The essential prime entries indicated by asterisks can be used as test patterns for fault detection. This fault causes the entry $(x_1x_2\text{-}y_1y_2y_3)$=(10-101) to change from (101) to (001) and the entries (10-100) and (11-100) to change from (100) to (000). Suppose that the current state is (101) and the input is (00). When the input is changed from (00) to (10), the fault-free circuit stabilizes at the state (101) with an output (11), but the faulty circuit will stabilize at the state (001) with an output (00). Sine the faulty and fault-free states are distinguishable, the fault is detectable. However, the current state (101) is an invalid state, as shown in Figure 3.3(b), and is not reachable from the reset state. Thus, the fault is an invalid-state redundant fault.

On the other hand, assume a stuck-at-1 fault that occurs at the $\bar{x}_2$-input of the AND gate, $\bar{x}_1\bar{x}_2\bar{y}_1y_3$, of Y_2. Figure 3.3(e) shows the transition table of the corresponding faulty circuit. The fault causes the entry $(x_1x_2\text{-}y_1y_2y_3)$=(01-001) to change from (001) to (011). Since both stable states (001) and (011) produce the outputs (00) and (01), respectively, they are distinguishable. Unfortunately, the entry is a don't care entry, as shown in Figure 3.3(b), and is not reachable from the reset state. Thus, the fault is also an invalid-state redundant fault. ♦

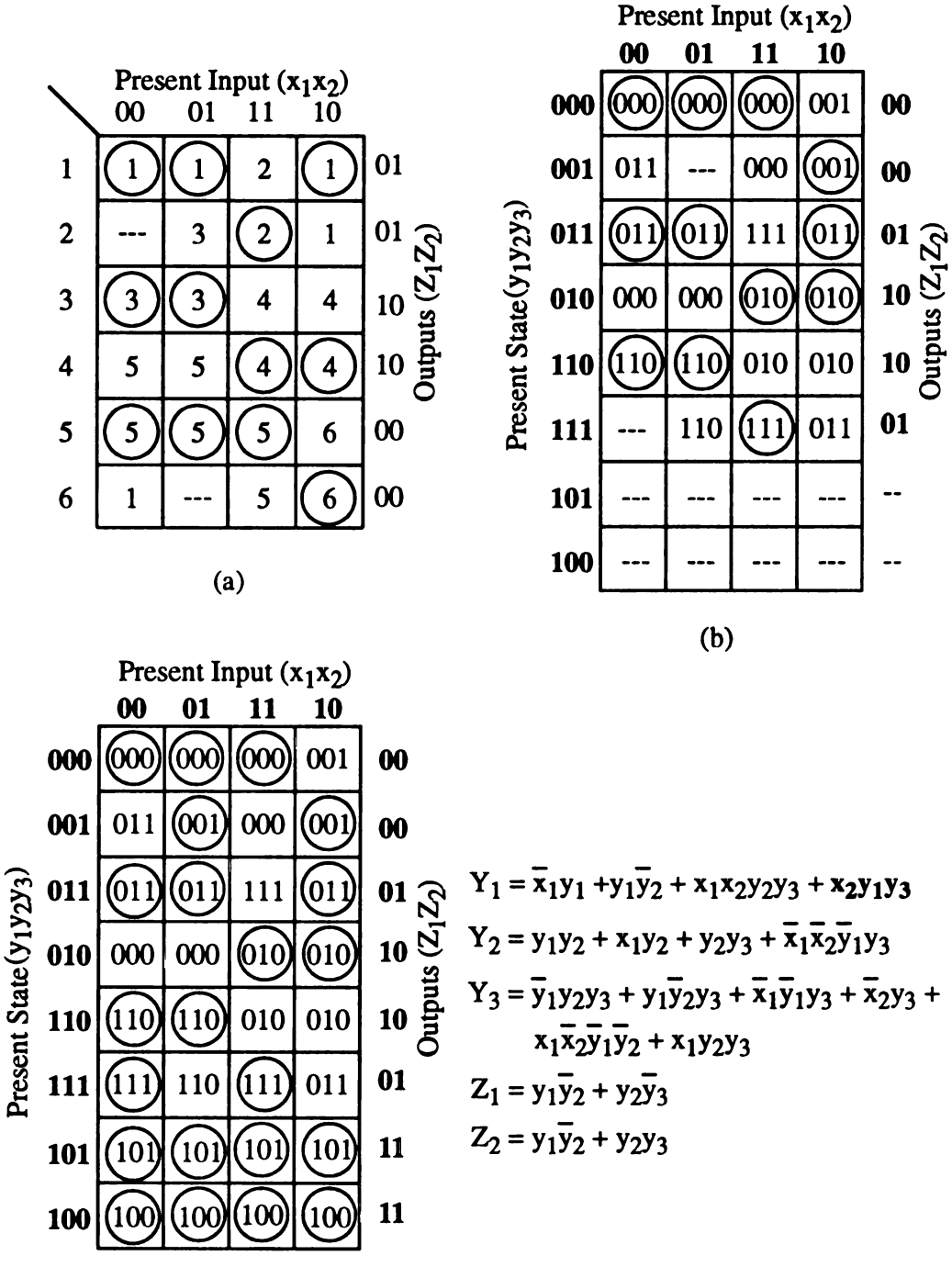
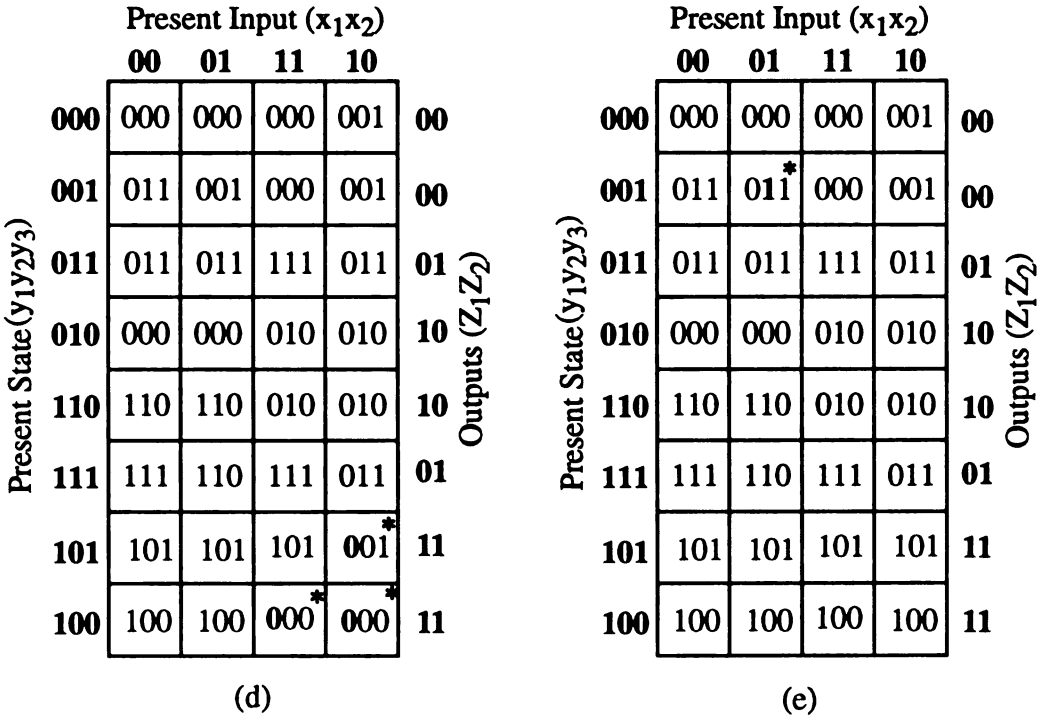


Figure 3.3: An ASLC for Example 2 (a) Flow table; (b) Transition table with don't cares; (c) Transition table without don't cares and its two-level logic realization; and (d)&(e) Transition tables of the corresponding faulty circuits.



Figuer 3.3: (Cont'd).

3.1.1.2 State Oscillations

Consider a flow table and its state assignment and realization shown in Figure 3.4 [9]. Let the circuit be stable in State 2 (represented by y_1=0, y_2=1) and input x=1. For the state transition when x changes value from 1 to 0, G_2 will begin to change to 1 and both Y_1 and Y_2 will begin to change their values. If y_2 changes to 0 before y_1 changes to 1, G_2 will change to 0 and the variable y_1 may oscillate, i.e., the circuit causes a state oscillation between State 1 and State 3, or even stabilizes at y_1=0 instead of y_1=1 if the delay associated with G_4 is sufficiently large. The resultant state oscillation or undesired stable state is caused by the presence of critical-race from State 2 (01) to State 3 (10). In practice, implementing a critical-race-free state assignment can avoid such a problem. However, the state oscillations may occur as a result of faults. The following example is to show that, in the presence of fault, both UDC- and STT-generated circuits suffer from state oscillations.

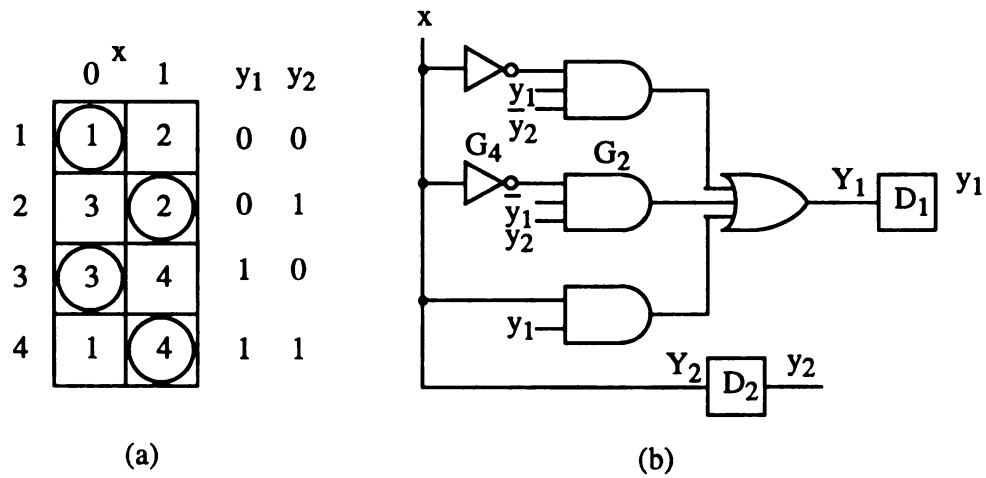


Figure 3.4: State oscillation due to critical races [9].

Example 3:

Consider the STT-generated ASLC shown in Figure 3.2(b). Assume that a stuck-at-0 fault occurs at the output of the AND gate, $\bar{x}_1x_2y_1$, of Y_1. Figure 3.5(a) shows the transition table of the corresponding faulty circuit. The fault causes the entry (x_1x_2-$y_1y_2y_3$)=(01-100) to change from (101) to (001) and a critical-race condition results. According to the Karnaugh-map (K-map) for Y_1-expressions shown in Figure 3.5(a), (x_1x_2-$y_1y_2y_3$)=(01-100) is the only essential prime entry which can be used to detect such a fault. Thus, a state oscillation may occur if the change of y_2 is faster than those of y_1 and y_3. Similarly, consider the UDC-generated ASLC shown in Figure 3.2(c). Suppose that a stuck-at-0 fault occurs at the AND gate, $x_2\bar{y}_1y_2$, of Y_2. Figure 3.5(b) shows the transition table of the corresponding faulty circuit. The fault causes the entries (x_1x_2-$y_1y_2y_3$)=(01-011) and (01-010) to change from (110) to (100) and critical-race conditions result. Similarly, state oscillations may occur if the change of y_1 is slower than that of y_2. ♦

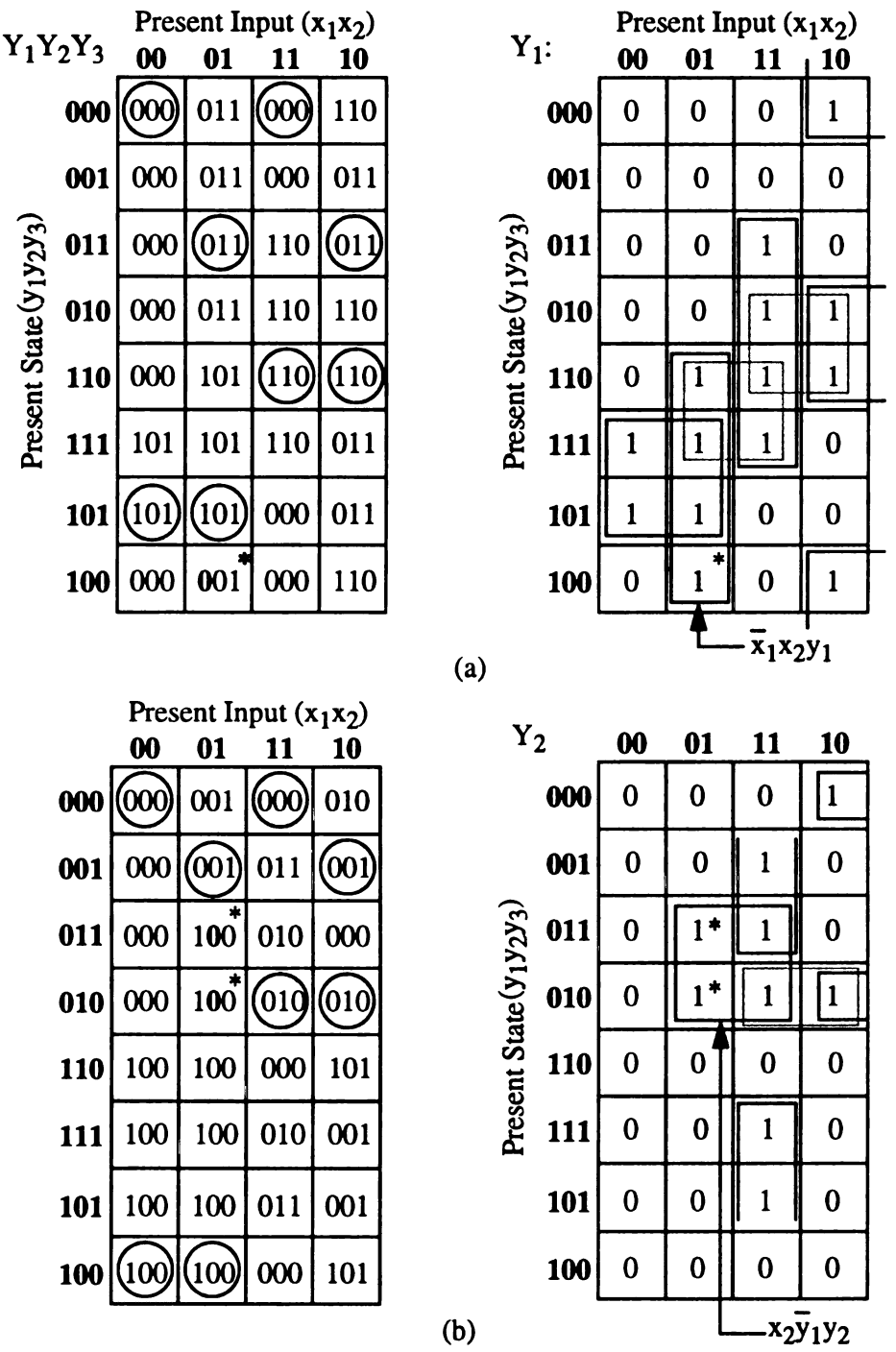


Figure 3.5: Example 3 (a) Transition table of the corresponding faulty circuit for STT-generated ASLC and K-map for Y_1; and (b) Transition table of the corresponding faulty circuit for UDC-generated ASLC and K-map for Y_2.

Example 3 has shown that both state assignment schemes suffer from the state oscillation due to the critical races that occur as a result of fault. An interesting problem that naturally arises is *whether or not the circuit is free of state oscillations if its faulty circuit is race-free.*

Let S_i be a state of a fault-free UDC-generated ASLC, and S_{ni} and S_{fi} be its next states with the application of an input to the fault-free and faulty circuits, respectively. In the fault-free circuit, $S_{ni}=S_i$ if S_i is a stable state, and the *Hamming distance* $d_H(S_{ni},S_i)=1$ if S_i is unstable. On the other hand, for the affected next-state entry in the presence of a stuck-at fault, $d_H(S_{fi},S_i)=1$ if S_i is stable, and $S_{fi}=S_i$ or $d_H(S_{fi},S_i)=2$ if S_i is unstable. Note that $d_H(S_{fi},S_i)=2$ implies the existence of a race condition in the faulty circuit which contradicts to the assumption of race-free faulty circuit. Also, $S_{fi}=S_i$ concludes that S_i is stable in the faulty circuit and no oscillation occurs. In other words, no state oscillations occur when a stuck-at fault affects the unstable state S_i. By a fault that affects the state S_i, we mean that the fault causes the next state to be $S_{fi}\neq S_{ni}$, i.e., both fault-free and faulty circuits have different next states. Otherwise, the fault does not affect that state. Therefore, *the state oscillation may occur only when the fault affects those stable states S_i's such that $d_H(S_{fi},S_i)=1$.* Note that, state oscillations may not occur in some cases even though a stuck-at fault affects a stable state. For example, consider a *transition pair* (S_1, S_2) [30], where S_1 is unstable, S_2 is stable, and $S_{n1}=S_{n2}$. In the logic implementation, both S_{n1} and S_{n2} are generally realized by the same gate. As a result, a fault that affects the stable state S_2 will definitely affect the unstable state S_1. Thus, state oscillations will never occur in such a case.

An *oscillation loop* is defined as a set of states S_i's, i=1,2,.., r, r≥2, where $S_{ni}=S_{i+1}$, i=1,2,.., r-1, and $S_{nr}=S_1$. If r=2, the oscillation loop is referred to as a *two-state oscillation.* Otherwise, it is a *multi-state oscillation.*

Based on the above conclusion, we first consider the two-state oscillations.

Property 1: Two-state oscillations will never occur in a race-free faulty circuit.

Proof: [By Contradiction]. Let S_1 and S_2 be any two states of a fault-free circuit. Assume that they form an oscillation loop in the presence of fault, i.e., $S_{f1}=S_2$, $S_{f2}=S_1$ and $d_H(S_1,S_2)=1$, referred to as *Condition A*. Without loss of generality, we let

$$S_1= (y_1,..,y_{k-1},y_k=z,y_{k+1},..,y_n), \text{ and}$$
$$S_2= (y_1,..,y_{k-1},y_k=\bar{z},y_{k+1},..,y_n),$$

where z is either 0 or 1, and $\bar{z}$ is the complement of z. Three cases can be identified for S_1 and S_2: (a) both are stable; (b) both are unstable; and (c) one is stable and the other is unstable.

For case (a), since S_1 and S_2 are stable, we have $S_{n1}=S_1$ and $S_{n2}=S_2$. If a stuck-at fault affects S_1 (or, S_2) at the state variable y_t, $t\neq k$, then $S_{f1}\neq S_2$ (or, $S_{f2}\neq S_1$) which contradicts to Condition A. On the other hand, A stuck-at-z (or, $\bar{z}$) fault at y_k causes $S_{f1}=S_{n1}=S_1\neq S_2$ (or, $S_{f2}=S_{n2}=S_2\neq S_1$) which also contradicts to Condition A. For case (b), a fault that affects the unstable states will never cause any state oscillation.

Finally, for case (c), without loss of generality, let S_1 be unstable and S_2 be stable. No oscillations occur if the fault affects the unstable state S_1, or if the fault affects the stable state S_2 and $S_{n1}=S_{n2}$ which makes (S_1,S_2) as a transition pair. Thus, we only consider the case that the fault affects S_2, but not S_1, and $S_{n1}\neq S_{n2}$. However, since $S_{n2}=S_2$, for stable state S_2, and $S_{f1}=S_{n1}$, for unstable state S_1, we conclude that $S_{f1}=S_{n1}\neq S_{n2}=S_2$ which contradicts to Condition A. *Q.E.D.*

Property 1 has shown that two-state oscillations will never occur in a race-free faulty circuit. The next question that arises is *whether or not multi-state oscillations will occur in a race-free faulty circuit.*

Races in UDC-generated ASLCs can always be eliminated by creating cycles, i.e., directing the circuit through intermediate unstable states before it reaches its final stable state. For a UDC-generated ASLC, each column of the flow table represents an input state

and each row represents an internal state with the corresponding output state. In each column, the states may be either stable or unstable, but each unstable state will transit to a stable state at the same input column. In other words, there always exists a *connected path*, denoted as P_{ab}, from the current state, S_a, to the next stable state, S_b, where $S_a=s_1\rightarrow s_2\rightarrow..\rightarrow s_k=S_b\rightarrow S_b$ and s_j's are the intermediate states with $d_H(s_j,s_{j+1})=1$, $j=1,2,..,k-1$, and $d_H(s_j,S_b)\geq 2$, $j=1,2,..,k-2$, for any integer k. The connected path with a length of k is referred to as *k-connected path* and $P_{ab}=(S_a,S_b)$ denotes the *connected pair.* It is obvious that the single stable state in an input column is a *1-connected path* and a connected path consisting of two states is a *2-connected path.* For example, consider the left flow table in Figure 3.2(c). In the input column 00, the state (100) is a stable state and forms a 1-connected path. The connected pair (000,001) is a 2-connected path in column 01. Similarly, the connected pair (001,010) in column 11 forms a 3-connected path.

Based on the above definitions, a sufficient, but not necessary, condition of synthesizing a multi-state oscillation-free circuit is given in the following property.

Property 2: If each input column contains either

(1) at most one k-connected path, $k > 3$, or

(2) k-connected paths, $k \leq 3$,

then no multi-state oscillation occurs in a race-free faulty circuit.

Prior to the proof of Property 2, we first consider the following lemmas.

Lemma 1: Any states in a k-connected path will never form an oscillation loop.

Proof: Consider a k-connected path $P_{ab}=(S_a,S_b)$. By definition, all states s_j's, except S_b, are unstable and $d_H(S_b,s_j)\geq 2$, $j=1,2,..,k-2$. Since no oscillations occur when a fault affects unstable states, we only consider the case that the fault affects the stable state S_b, i.e., the only possibility of forming an oscillation loop is that the fault causes S_b to loop with s_j's, $j=1,2,..,k-1$. However, since $d_H(S_b,s_j)\geq 2$, for $j=1,2,..,k-2$, implies that $S_{fb}\neq s_j$, hence, S_b will

never form an oscillation loop with these s_j's. On the other hand, by Property 1, two states, s_{k-1} and S_b, will never form an oscillation loop even though $d_H(S_b,s_{k-1})=1$. This concludes the lemma. *Q.E.D.*

Lemma 1 shows that any states in a k-connected path will never form an oscillation loop. The question is *whether or not the k-th connected path $P_{ab}=(S_a,S_b)$ forms an oscillation loop with any other states which are not included in that path.* Apparently, the necessary condition should be that there exist two states S_p, $S_q \notin P_{ab}$: $S_a=s_1\rightarrow s_2\rightarrow..\rightarrow s_k=S_b\rightarrow S_b$, such that $d_H(S_p,S_i)=1$, $S_i\in P_{ab}$, and $d_H(S_b,S_q)=1$, where S_p is stable, but S_q can be either stable or unstable. For forming an oscillation loop, it is necessary that a stuck-at fault affects all the stable states, but not the unstable states. Thus, in the presence of fault, the conditions for forming an oscillation loop are

$d_H(S_p,S_i)=1$, $S_i\in P_{ab}$, $d_H(s_j,s_{j+1})=1$, j=1,2,..,k-1, and $d_H(S_b,S_q)=1$;
$s_{fj}=s_{nj}=s_{j+1}$, for unstable states s_j's, j=1,2,..,k-1; and *Condition B*
$S_{fp}=S_i$ and $S_{fb}=S_q$.

Based on *Condition B*, the following lemma examines the possibility of forming oscillation loops.

Lemma 2: In each input column, any k-connected path, $k \leq 3$, will never form an oscillation loop with any other states in the same column.

Proof: [By Contradiction] For k=1, i.e., the connected path consists of only a stable state S_b. Assume that the connected path forms an oscillation loop with any other states. According to Condition B, we have the following conditions, referred to *Condition B1*: $d_H(S_p,S_b)=d_H(S_b,S_q)=1$, $S_p\neq S_b$, $S_q\neq S_b$, $S_{fp}=S_b$, and $S_{fb}=S_q$. Based on the first condition, without loss of generality, we assume that S_p and S_b differ at the state variable y_{k1}, while S_b and S_q differ at y_{k2}. So, their binary codes are

$S_p=(y_1,y_2,...,y_{k1-1},y_{k1}=z_1,y_{k1+1},...,y_{k2-1},y_{k2}=z_2,y_{k2+1},...,y_n)$,
$S_b=(y_1,y_2,...,y_{k1-1},y_{k1}=\bar{z}_1,y_{k1+1},...,y_{k2-1},y_{k2}=z_2,y_{k2+1},...,y_n)$, and
$S_q=(y_1,y_2,...,y_{k1-1},y_{k1}=\bar{z}_1,y_{k1+1},...,y_{k2-1},y_{k2}=\bar{z}_2,y_{k2+1},...,y_n)$,

where $z_1, z_2 \in \{0,1\}$. Note that $S_{np}=S_p$ and $S_{nb}=S_b$ for stable states S_p and P_b. Suppose that a stuck-at fault affects S_p and S_b at the state variable y_t, $t \neq k_1$. Since both S_{nb} and S_{np} differ only at y_{k1}, we have $S_{fp} \neq S_b$ which contradicts to Condition B1. Suppose that the fault occurs at y_{k1}, a stuck-at-z_1 (or, $\bar{z}_1$) fault causes $S_{fp}=S_p \neq S_b$ (or, $S_{fb}=S_b \neq S_q$) which also contradicts to Condition B1. This concludes that 1-connected path will never form an oscillation loop with any other states.

For k=2, the connected path $P_{ab}=(S_a,S_b)$. Suppose that the connected path forms an oscillation loop with any other states. Connecting two states S_p and S_q to the path, two possible oscillation loops may be formed: (1) S_p connects to S_a, i.e., $S_p \rightarrow S_a \rightarrow S_b \rightarrow S_q \rightarrow ...$; or (2) S_p connects to S_b, i.e., $S_p \rightarrow S_b \rightarrow S_q \rightarrow ...$ The latter loop is similar to the case of k=1 which will never form an oscillation. Thus, we only consider case (1). According to Condition B, we have the following conditions, referred to as *Condition B2*: $d_H(S_p,S_a)=d_H(S_a,S_b)=d_H(S_b,S_q)=1$; S_p, $S_q \notin \{S_a,S_b\}$; and $S_{fp}=S_a$, $S_{fa}=S_{na}=S_b$, and $S_{fb}=S_q$. Without loss of generality, we assume the following binary codes for these variables

$$S_p=(y_1,...,y_{k1}=z_1,...,y_{k2}=z_2,...,y_{k3}=z_3,...,y_n),$$
$$S_a=(y_1,...,y_{k1}=\bar{z}_1,...,y_{k2}=z_2,...,y_{k3}=z_3,...,y_n),$$
$$S_b=(y_1,...,y_{k1}=\bar{z}_1,...,y_{k2}=\bar{z}_2,...,y_{k3}=z_3,...,y_n), \text{ and}$$
$$S_q=(y_1,...,y_{k1}=\bar{z}_1,...,y_{k2}=\bar{z}_2,...,y_{k3}=\bar{z}_3,...,y_n), \text{ or} \quad (q1)$$
$$S_q=(y_1,...,y_{k1}=z_1,...,y_{k2}=\bar{z}_2,...,y_{k3}=z_3,...,y_n), \quad (q2)$$

where $z_1, z_2, z_3 \in \{0,1\}$. Similar to the proof for k=1, a stuck-at fault at y_t, $t \neq k_1$, or a stuck-at-z_1 at y_{k1}, results in $S_{fp}=S_p \neq Sa$, regardless of implementing with (q1) or (q2), which contradicts to Condition B2. On the other hand, the stuck-at-$\bar{z}_1$ at y_{k1} forces $S_{fb}=S_{nb}=S_b \neq S_q$, in (q1), which also contradicts Condition B2. This concludes the Lemma for k=2.

Similarly, for k=3, the connected path $P_{ab}=(S_a,S_b)$:$S_a \rightarrow S_c \rightarrow S_b$. Connecting the states S_p and S_q may result in three possible loops: (1) $S_p \rightarrow S_a \rightarrow S_c \rightarrow S_b \rightarrow S_q \rightarrow ...$; (2) $S_p \rightarrow S_c \rightarrow S_b \rightarrow S_q \rightarrow ...$; or (3) $S_p \rightarrow S_b \rightarrow S_q \rightarrow ....$ The latter two cases are similar to k=2 and

k=1, respectively. Thus, we only consider case (1). The same procedure in the proof for k=2 can be used to prove for k=3. *Q.E.D.*

Proof of Property 2. Based on Lemmas 1 and 2, no multi-state oscillations occur in a race-free faulty circuit. *Q.E.D.*

The following discussion will show that the conditions in Property 2 are sufficient, but not necessary. Consider a 4-connected path $P_{ab}=(S_a,S_b):S_a\rightarrow S_c\rightarrow S_d\rightarrow S_b$ and there exist two states S_p and S_q with the following binary codes

$$S_p=(y_1,\ldots,y_{k1}=z_1,\ldots,y_{k2}=z_2,\ldots,y_{k3}=z_3,\ldots,y_n),$$
$$S_a=(y_1,\ldots,y_{k1}=\bar{z}_1,\ldots,y_{k2}=z_2,\ldots,y_{k3}=z_3,\ldots,y_n),$$
$$S_c=(y_1,\ldots,y_{k1}=\bar{z}_1,\ldots,y_{k2}=\bar{z}_2,\ldots,y_{k3}=z_3,\ldots,y_n),$$
$$S_d=(y_1,\ldots,y_{k1}=z_1,\ldots,y_{k2}=\bar{z}_2,\ldots,y_{k3}=z_3,\ldots,y_n),$$
$$S_b=(y_1,\ldots,y_{k1}=z_1,\ldots,y_{k2}=\bar{z}_2,\ldots,y_{k3}=\bar{z}_3,\ldots,y_n), \text{ and}$$
$$S_q=(y_1,\ldots,y_{k1}=\bar{z}_1,\ldots,y_{k2}=\bar{z}_2,\ldots,y_{k3}=\bar{z}_3,\ldots,y_n),$$

where z_1, z_2, and z_3 are either 0 or 1. Apparently, $d_H(S_p,S_a)=d_H(S_a,S_c)=d_H(S_c,S_d)=d_H(S_d,S_b)=d_H(S_b,S_q)=1$, and S_p, $S_q \notin \{S_a,S_c,S_d,S_b\}$; On the other hand, in the presence of a stuck-at-$\bar{z}_1$ at y_{k1}, it is easy to verify that $S_{fa}=S_{na}=S_c$, $S_{fc}=S_{nc}=S_d$, $S_{fd}=S_{nd}=S_b$, $S_{fa}=S_a$, and $S_{fb}=S_q$. Thus, these binary codes satisfy Condition B and, thus, state oscillation may occur. The following example illustrates the detailed encoding and state assignment, where the circuit contains an input which consists of two 4-connected paths.

Example 4:

Consider the partial flow table of a given ASLC shown in Figure 3.6(a). The boldfaced states, as shown in Figure 3.6(b), indicate races. Since States 1, 3, and 5 have the same next state 4, the UDC state assignment creates cycles by modifying the entries of states 1 and 2 as intermediate states to form a connected path 3(011)→1(001)→5(101)→4(100). Similarly, the other connected path, 0(000)→2(010)→6(110)→7(111), is also generated. Both paths are illustrated by the boldfaced arrows in Figure 3.6(d). Suppose that a stuck-at-0 fault occurs at the output of the AND gate, y_1I_p, of Y_1, Figure 3.6(e) shows

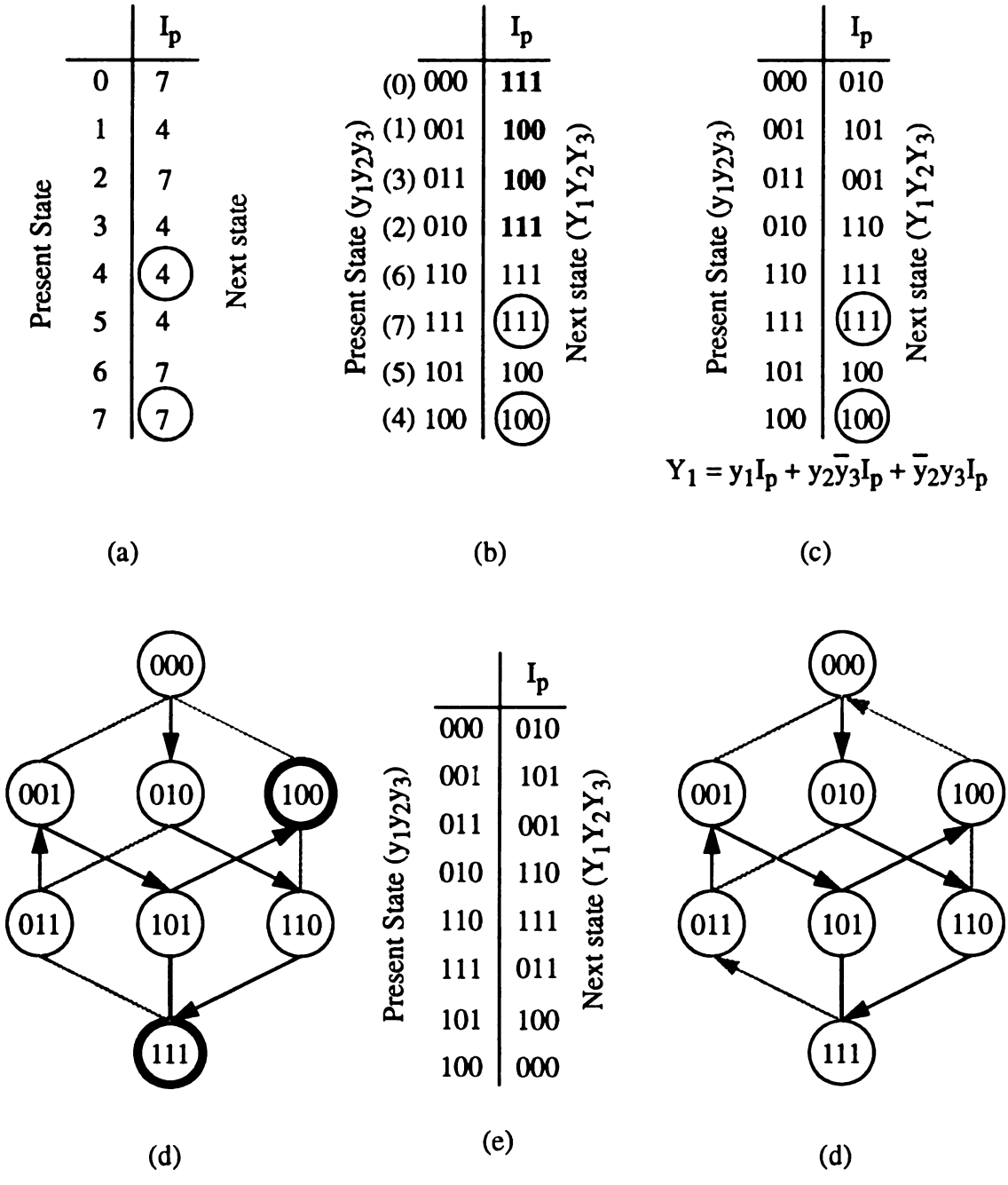


Figure 3.6: An ASLC example (a) Partial flow table; (b)&(c) Transition table; (d) Connected path; (e) Transition table of the corresponding faulty circuit; and (f) Connected paths form an oscillation loop.

the partial transition table of the corresponding faulty circuit which does not have races. However, in the presence of such a fault, both connected paths form an oscillation loop, 011→001→101→100→ 000→010→110→111→011, in Figure 3.6(f). ♦

An alternative sufficient, but not necessary, condition for synthesizing oscillation-free circuit is also given as follows. Two connected paths are *disjoint* if they have different final stable states.

Property 3: If there exist two disjoint k-connected paths, k>3, and $d_H(S_x,S_y)>1$ for all S_x and S_y in different connected paths, then no multi-state oscillations occur in the race-free faulty circuit.

Proof: By Lemma 2, the k-th connected paths, k≤3, will never form an oscillation with any other states. Thus, the only possible multi-state oscillation loop is formed by these two k-th connected paths, k>3. However, since $d_H(S_x,S_y)>1$ for all S_x and S_y in different connected paths, Condition B will not be satisfied. Thus, no oscillation will occur. *Q.E.D.*

Properties 2 and 3 provide simple rules for synthesizing oscillation-free UDC-generated ASLCs if the faulty circuit is race-free. although races in the UDC-generated ASLCs can always be eliminated by creating cycles, Properties 2 and 3 guarantee that no state oscillations occur in a race-free faulty circuit if the circuit is synthesized in such a way that each input column allows to have either only a long connected path, or two connected paths, with k>3, and $d_H(S_x,S_y)>1$ for all S_x and S_y in different connected paths.

For detecting a fault, it may need one or more test sequence. If a circuit is synthesized by the rules provided in Properties 2 and 3, and the faulty circuit is race-free with the application of such the test sequences, we guarantee that the faulty circuit is free of state oscillation. On the other hand, if the faulty circuit is not free of races with the application of all possible test sequences for detecting a fault, state oscillations may occur. In such a case, detecting such a fault is rather a difficult task due to the unpredictable

"stable" states.

3.1.2 Testability Synthesis Properties

This section discusses the testability properties for synthesizing both *STT-generated ASLCs* and *UDC-generated ASLCs*. Here, the combinational circuit of an ASLC is optimized with prime and irredundant implicants except the redundant logic for static hazard protection.

A fault is detectable if its fault effects can be excited and propagated to the primary outputs and the fault site can be reached from the initial state. Thus, a fault is detected by applying a test sequence which is comprised of a fault *propagation sequence* and a *justification sequence*. Let S_i be a state of a UDC- or STT-generated ASLCs. Let T^p_{i1} and T^p_{i0} denote the fault propagation sequence and justification sequence for the faulty next-state entry of S_i in the input column I_p, respectively. As defined in Section 3.1.1.2, S_{ni} and S_{fi} are the next states of S_i in fault-free and faulty circuits, respectively. S^p_{ni} and S^p_{fi} are denoted as the S_{ni} and S_{fi} in the input column I_p, respectively. A fault is detectable if there exists a state S_i such that (1) $S^p_{ni} \neq S^p_{fi}$; and (2) there exist the sequences T^p_{i1} and T^p_{i0}. For simplicity of discussion, the first condition is referred to as *Condition C.1*, while the second as *Condition C.2*. Note that no critical races are allowed when both sequences T^p_{i1} and T^p_{i0} are applied.

For simplicity, State S_i is represented by a n-tuple binary vector $(y_1,y_2,..,y_n)$, and S_{ij} denotes as the y_j bit of S_i. $D_H(S_1,S_2)=j$ means that $S_{1i}=S_{2i}$ for all i except j, i.e., S_1 and S_2 differ in one bit which is the y_j bit. When a state transition occurs, some bits may be changed. Let $X_u(S_i,S_j)=\{S_{ik} | S_{ik}=S_{jk},\ k=1,2,..,n\}$ and $X_c(S_i,S_j)=\{S_{ik} | S_{ik} \neq S_{jk},\ k=1,2,..,n\}$ denote the sets of unchanged and changed bits in a state transition from S_i to S_j.

Based on *Condition C*, the testability synthesis properties are derived. Moveover, a state S_1 is said to be *distinguishable* from another state S_2 in the input column I_p if there exist an input sequence such that the output sequence starting from S^p_{f1} is different from

that starting from S^p_{f2}.

STT-generated ASLCs without Latches

As defined in Section 2.1, a *K-set* in a single column of a flow table consists of all (k-1) unstable entries leading to the same stable state, together with the stable state [30,34]. A pair of states $[S_i,S_j]$ consisting of one unstable state S_i and the stable state S_j of a K-set is called a *transition pair.* In generally, a K-set may contain one or more transition pairs and each input column may contain one or more K-sets. The set of states which the circuit may assume in undergoing a transition between the states of a transition pair $[S_i,S_j]$ is called the *transition path* (TP) of $[S_i,S_j]$ and denoted as TP_{ij}. For example, if S_i=(1000) and S_j=(1110), then TP_{ij}=(1--0)={(1000), (1010), (1100), (1110)}, where $X_u(S_i,S_j)$={1,4} and $X_c(S_i,S_j)$={2,3}. Therefore, if y_k in $X_u(S_i,S_j)$, then y_k=0 or 1 in TP_{ij}; if y_k in $X_c(S_i,S_j)$, then y_k= "-" (don't care).

For a STT state assignment, every state transition between a transition pair is direct [30]. The STT state assignment was developed based on the theory of *partitions*. A partition π on a set S is a collection of the subsets of S such that their pairwise intersection is the null set and the disjoint subsets are called the blocks of π. Each state variable is determined by a partition by assigning 1 to a block and 0 to another block. An internal state variable which partitions one transition path from another is called a *partition variable*.

Tracey's Theorem [30]: A state assignment alloting one y-state per state can be used for the realization of STT flow tables without critical races if, and only if the following conditions hold for every transition pair $[S_i,S_j]$, S_j a stable state.

(1) If $[S_m,S_n]$ is another transition pair, $j\neq n$, in the same column, then at least one partition partitions the pair $[S_i,S_j]$ and the pair $[S_m,S_n]$ into separate blocks;

(2) If S_k is a stable state in the same column $j\neq k$, then at least one partition partitions the pair $[S_i,S_j]$ and the state S_k into separate blocks; and

(3) For $i\neq j$, S_i and S_j are in separate blocks of at least one partition.

During the state transition of a transition pair, some state variables are changed and some others are unchanged. Conditions (1) and (2) show that $X_u(S_i,S_j)\neq\phi$ (null set), while Condition (3) implies $X_c(S_i,S_j)\neq\phi$. That is, the conditions imply that bot X_u-set and X_c-set are non-empty sets.

It should be mentioned that, using the K-sets instead of transition pairs, Tracey's Theorem can be expressed as follows [30,34]: A direct transition in a K-set k_1 does not race critically with a direct transition in a K-set k_2 if an assignment has been made such that at least one y-variable partitions the elements of k_1 and the elements of k_2 into separate blocks. Apparently, this theorem keeps the first two conditions in Tracey's Theorem, but it might not generate minimum number of state variables.

Consider a transition pair $[S_i,S_k]$ in a K-set and in input column I_p. Faults may occur at the bits in $X_c(S_i,S_k)$ or in $X_u(S_i,S_k)$. We first consider the former case, where the bit is changed during the state transition.

Lemma 3: If S_j in TP_{ik} and a stuck-at fault occurs at y_r in $X_c(S_i,S_k)$, then S^p_{nj} in TP_{ik} and S^p_{fj} in TP_{ik}.

Proof: According to the STT state assignment, $S^p_{nj}=S_k$ in TP_{ik}. On the other hand, y_r in $X_c(S_i,S_k)$, hence y_r="-" in TP_{ik}. Thus, in the presence of a stuck-at fault at y_r bit, S^p_{fj} in TP_{ik} with either $S_{fj}=S_{nj}$ or $S_{fj}\neq S_{nj}$. *Q.E.D.*

Based on Lemma 3, the following theorem results.

Theorem 1: A stuck-at fault that occurs at y_r in $X_c(S_i,S_k)$ is detectable if (1) $S^p_{fk}\neq S^p_{nk}$, where S_k is the stable state in a K-set; (2) there exists a state S_j such that S_j in TP_{ik} and $D_H(S_j,S_k)=r$, S_j is distinguishable from S_k in the input column I_p; and (3) $[S_i,S_k]$ is reachable from the reset state.

Proof: Since S_j in TP_{ik}, it implies that $S^p_{nj}=S_k$. By condition (1) and $D_H(S_j,S_k)=r$, this fault will cause $S^p_{fk}=S_j$. Two cases can be distinguished: Case 1: If $S^p_{fj}=S^p_{nj}=S_k$, it will result

in state oscillation between S_k and S_j. However, from the result of Property 1, this case will not occur. Case 2: $S^p_{fj} \neq S^p_{nj}$, it implies that the S_j will become a stable state in the presence of fault and it satisfied *Condition C.1*. Further, for an unstable state S_u, where S_u in TP_{ik} and $S_u \neq S_j$, either $d_H(S_u,S_k) \geq 2$ or $D_H(S_u,S_k)=m$ with $m \neq r$. A stuck-at fault at y_r bit will not result in S_u to become a stable state. As a result, S_j is the only stable state in TP_{ik} in the presence of faults. By Lemma 3, the state transition will also be in TP_{ik}. Conditions (2) and (3) imply T^p_{j1} and T^p_{j0} exists respectively. Therefore, *Condition C.2* is also satisfied and this fault is detectable. *Q.E.D.*

The condition $S^p_{fk} \neq S^p_{nk}$ in Theorem 1 implies that the redundant faults as shown in Example 1 will not occur in the presence of faults. Now, consider the stuck-at fault that occurs at y_r in $X_u(S_i,S_k)$. If such a fault occurs and there exist no other transition pairs which satisfy the conditions in Theorem 1, then critical races may occur as a result of faults. The following lemma is used to guarantee that the introduced races are non-critical such that the stuck-at fault can be detected.

Lemma 4: Let $[S_i,S_{k1}]$ be a transition pair in a K_1-set, in an input column I_p. A stuck-at fault that occurs at y_r in $X_u(S_i,S_{k1})$ is detectable, if (1) $S^p_{fk1} \neq S^p_{nk1}$and C is the minimal cube that contains all the states S_t's in TP_{ik1} with $S^p_{nt} \neq S^p_{ft}$; (2) A cube D in TP_{jk2}, the TP of a K_2-set, where $K_2 \neq K_1$, and $D_H(D,C)=r$; (3) S_{k1} is distinguishable from S_{k2} in the input column I_p; and (4) $[S_i,S_{k1}]$ is reachable from the reset state.

Proof: By $S^p_{fk1} \neq S^p_{nk1}$, *Condition C.1* is satisfied. Further, when a stuck-at fault occurs y_r in $X_u(S_i,S_{k1})$, a critical-race condition results in the cube C. Let $E=C \cup D$. The introduced races in C will become non-critical in E. Since $S^p_{fk1} \neq S^p_{nk1}$, the faulty circuit will finally stabilize at the state S_{k2}. By conditions (3) and (4), both sequences T^p_{k11} and T^p_{k10} exist and satisfy *Condition C.2*. Thus, the fault is detectable. *Q.E.D.*

Similar to the results of Theorem 1, if the stable state of the transition pair is not affected, then critical races may result and the behavior of a faulty circuit is unpredictable. Note that a K-set may contain one or more transition pairs and each input column may contain one or more K-sets. Similar to Lemma 4, the following theorem results.

Theorem 2: Let y_r be the bit to distinguish K_1-set from other K-sets. A stuck-at fault that occurs at y_r is detectable if (1) $S^p_{fk1} \neq S^p_{nk1}$ and C is the minimal cube contains that all the states S_t's in TP_{ik1} with $S^p_{nt} \neq S^p_{ft}$; (2) A cube D in TP_{jk2}, the TP of a K_2-set, where $K_2 \neq K_1$, and $D_H(D,C)=r$; (3) S_{k1} is distinguishable from S_{k2} in the input column I_p; and (4) $[S_i,S_{k1}]$ is reachable from the reset state.

Proof: A fault is detectable if there exists a test sequence which can distinguish the behavior of fault-free and faulty circuits. Therefore, by Lemma 4, this fault is detectable.

Q.E.D.

For an ASLC implementation, Theorem 1 can be first used to check whether or not a stuck-at fault is detectable. If not, then Theorem 2 is applied.

UDC-generated ASLCs without Latches

Those synthesis properties for STT-generated ASLCs can be further modified for UDC-generated ASLCs. Races in a UDC-generated ASLC can always be eliminated by creating cycles. Thus, for a connected pair (S_a,S_b), where S_a is the current state and S_b is the next stable state with the application of the input I_p, there always exists a connected path $S_a=s_0 \rightarrow s_1 \rightarrow .. \rightarrow s_k=S_b \rightarrow S_b$, where s_j's are the intermediate states with $d_H(s_j,s_{j+1})=1$ for each adjacent pair (s_j,s_{j+1}), j=0,1,..,k-1, and k is an arbitrary integer. Further, two connected paths with different stable states must be disjoint. Let $X_u(S_i,S_{ni})=\{S_{ik} | S_{ik}=S_{nik}, k=1,2,..,n\}$ and $X_c(S_i,S_{ni})=\{S_{ik} | S_{ik} \neq S_{nik}, k=1,2,..,n\}$ denote the sets of unchanged and changed bits in a state transition from S_i to its next state S_{ni}. In UDC state assignments, the cardinality $|X_c(S_i,S_{ni})| \leq 1$. If $|X_c(S_i,S_{ni})|=0$, S_i is a stable state. If $|X_c(S_i,S_{ni})|=1$, S_i is unstable

Theorem 3: A stuck-at fault that occurs at y_r in $X_c(S_i,S_{ni})$ is detectable if (1) S_i in P_{ab}, where S_i is an unstable state in the input column I_p, and $S^p_{fi} \neq S^p_{ni}$; (2) S_i is distinguishable from the state S_b in the input column I_p; and (3) T^p_{i0} exists.

Proof: For a UDC-generated ASLC with $D_H(S_i,S_{ni})=r$, the first condition satisfies *Condition C.1* and results in $S^p_{fi}=S_i$ in the presence of fault at y_r-bit. The second and third conditions satisfy *Condition C.2*. Thus, this fault is detectable. *Q.E.D.*

Consider a connected path P_{ab} in the input column I_p, if a stuck-at fault occurs at y_r in $X_u(S_b,S_{nb})$ with $S^p_{fb} \neq S^p_{nb}$, S_b will become an unstable state and no critical races occur. On the other hand, if a stuck-at fault occurs at y_r in $X_u(S_i,S_{ni})$ with $S^p_{fi} \neq S^p_{ni}$, where S_i is an unstable state in P_{ab}, critical races may occur. The following theorem considers the testability synthesis property when a stuck-at fault occurs at y_r in $X_u(S_i,S_{ni})$ in the UDC-generated ASLCs.

Theorem 4: A stuck-at fault that occurs at y_r is detectable if (1) $S^p_{fb} \neq S^p_{nb}$ and $P_{tb} \in P_{ab}$, where S_t is either the stable state or the first intermediate state with $S^p_{ft} \neq S^p_{nt}$ and y_r in $X_u(S_t,S_{nt})$; (2) a path P, which is part of another disjoint connected path P_{cd}, differs from P_{tb} in y_r-bit; (3) S_d is distinguishable from S_b; and (4) P_{ab} is reachable from the reset state.

Proof: Similar to the proof of Theorem 2. With $S^p_{fb} \neq S^p_{nb}$, whenever races occur in P_{tb} in the presence of fault, condition (2) ensures that the races will force the state transition to enter one of the states in P_{cd} and finally stabilize at S_d. Together with conditions (3) and (4), this fault is detectable. *Q.E.D.*

In general, a fault may or may not result in race conditions in the faulty circuit. Without race conditions, either the affected unstable state become stable or the affected stable state become unstable. Theorem 3 deals with the affected unstable state without races and Theorem 4, with $S_t=S_b$, is concerned with the affected stable state without races. On the other hand, when S_t is the intermediate state, Theorem 4 consists of testable conditions

for the faulty circuit with races. Condition (2) in Theorem 4 is needed for avoiding state oscillations and eliminating equivalent-state redundant faults. However, to find a disjoint connected path that includes the path P may not be practical, particularly for a long connected path P_{tb}. As discussed previously, multi-state oscillations can be avoided if the length of the connected path is limited (Property 3). Similarly, if the length of the connected path is limited during the state assignment phase, then synthesizing testable circuit can be easier. To have a better understanding of previous Lemmas and Theorems, the following example is used to demonstrate those results for STT-generated ASLCs. Similar concepts can also be applied to UDC-generated ASLCs.

Example 5:

Consider the ASLC used in Example 1. There exist two K-sets in input (11) column, where K_1-set consists of only one transition pair [101,000] with transition path $TP_{101,000}$ ={(101),(100),(001),(000)} and K_2-set consists of one transition pair [011,110] with $TP_{011,110}$={(011),(010),(111),(110)}. Both transition pairs (or sets) are partitioned and distinguished by y_2-bit, where y_2=0 for [101,000]and y_2=1 for [011,110] with S_{k1}=(000) and S_{k2}=(110). Assume that a stuck-at-1 fault occurs at y_1-input of the AND gate, $x_1x_2y_2$, of Y_1. Figure 3.7(a) shows the transition table of the corresponding faulty circuit, where the y_1-bit of the states in $TP_{101,000}$ are all changed from 0 to 1 as indicated by asterisks. It is obvious that the affected states are still in the same transition path. Consider the state (100) in $TP_{101,000}$, which is adjacent to the stable state (000). If states (100) is distinguishable from (000) and the state (101) can be reached from the reset state, by Theorem 1, such a fault is detectable. Assume that a stuck-at-1 fault occurs at the $\bar{x}_2$-input of the AND gate, $x_1\bar{x}_2\bar{y}_3$, of Y_1. Figure 3.7(b) shows the transition table of the corresponding faulty circuit. By Theorem 1, since the stable state is also affected, the fault is detectable if states (100) is distinguishable from (000) and state (101) is reachable from the reset state. Consider the input (01) column which consists of two K-sets, where K_1-set consists of only one transition pair [000,011] with $TP_{000,011}$={(000),(010),(001),(011)} and K_2-set consists of

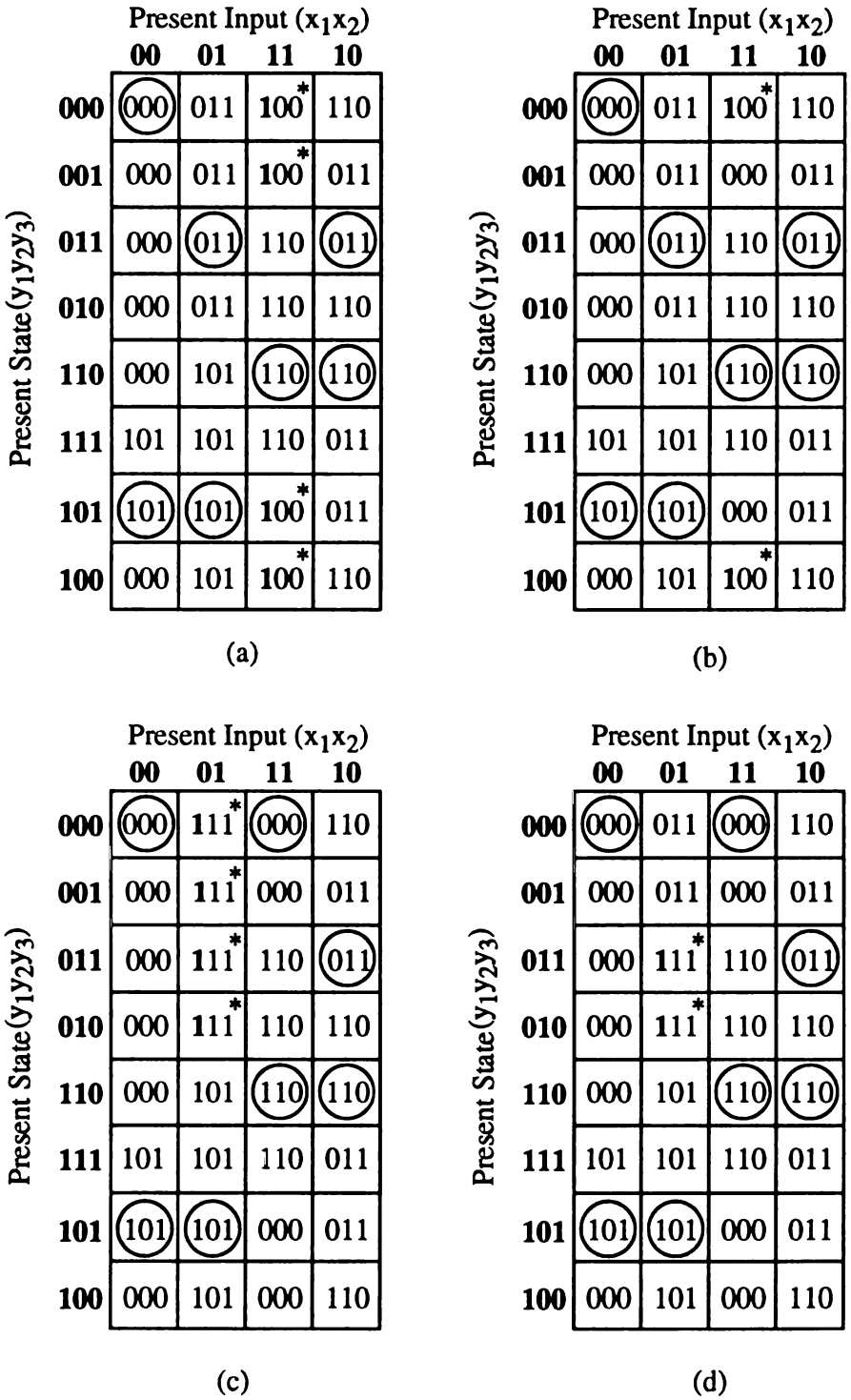


Figure 3.7: Transition tables for Example 5.

one transition pair [110,101] with $TP_{110,101}$={(110),(111),(100),(101)}. Both transition pairs are partitioned and distinguished by y_1-bit, where y_1=0 for [000,011] and y_1=1 for [110,101]. Figure 3.7(c) shows the transition table for a stuck-at-1 fault that occurs at the y_1-input of the AND gate, $\bar{x}_1x_2y_1$, of Y_1. By Theorem 2, such a fault is detectable if the state (000) can be reached from the reset state. Similarly, Figure 3.7(d) shows the transition table for a stuck-at-1 fault that occurs at the y_1-input of the AND gate $\bar{x}_1x_2y_1$, of Y_1. The states (011) and (010) in $TP_{000,011}$ are affected, where this subset includes the stable state (011). Moreover, both states, (111) and (110), are included in the TP of the transition pair [110,101] in another K-set. Thus, the fault is detectable if the affected state (000) can be reached from the reset state. ♦

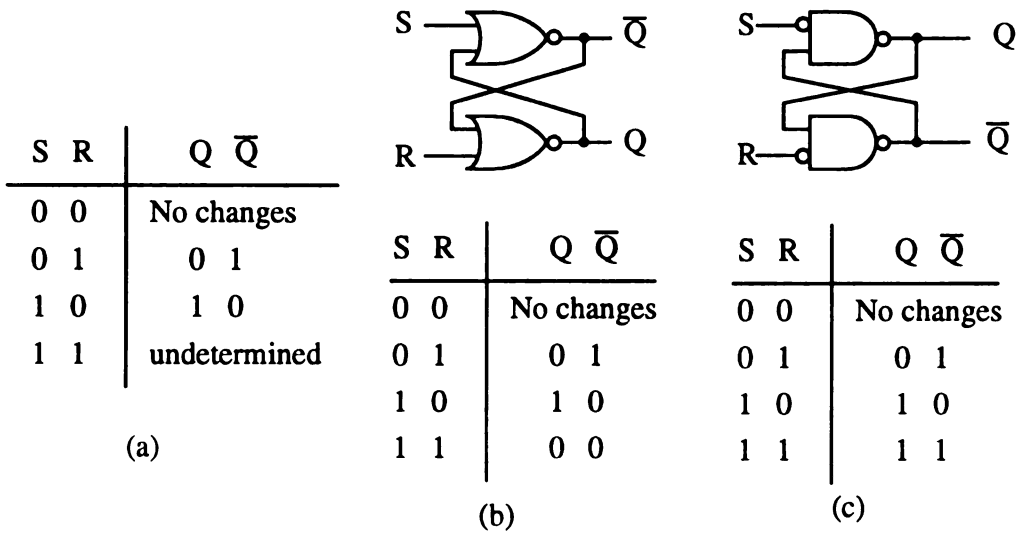
3.2 Huffman-Modeled ASLCs with SR-Latches

Figure 3.8(a) illustrates the truth table of a SR-latch, where $(Q,\bar{Q})$ are undetermined when (S,R)=(1,1). The values will be assigned when the SR-latch is implemented either with NOR or NAND gates, as shown in Figures 3.8(b) and 3.8(c), respectively. Note that when the inputs (S,R) are changed from (1,1) to (0,0), the next state, which is either (0,1) or (1,0), is determined by the race between the signals S and R. For a reliable design, (S,R)=(1,1) has been avoided. However, it may occur as a result of faults.

This section describes the fault effects for Huffman-modeled ASLCs implemented with SR-latches where the excitation table for the latch is shown in Table 3.1.

3.2.1 Fault Effects

Let S_{Yi} and R_{Yi} respectively denote as the set (S) and reset (R) inputs of a SR-latch for the state variable Y_i. Suppose that the ASLCs under consideration, as shown in Figure 3.9, are synthesized under the assumptions described in Section 3.1, where each CLB (combinational logic block) does not share any logic with any other CLBs. Because the output values of the SR-latch remain unchanged when S=R=0, no static logic 1-hazard



S R	Q $\overline{Q}$
0 0	No changes
0 1	0 1
1 0	1 0
1 1	undetermined

(a)

S R	Q $\overline{Q}$
0 0	No changes
0 1	0 1
1 0	1 0
1 1	0 0

(b)

S R	Q $\overline{Q}$
0 0	No changes
0 1	0 1
1 0	1 0
1 1	1 1

(c)

Figure 3.8: (a) Operation of SR-latch; (b) NOR-gate implementation; and (c) NAND-gate implementation.

protection terms are needed [37]. This section describes the fault effects such as redundant faults and state oscillations.

3.2.1.1 Redundant Faults

Based on the assumptions described in Section 3.1 with no hazard protection terms, the logical functions S_{Yi} and R_{Yi} are implemented with two-level logics which are prime and irredundant. Thus, there exist no combinationally redundant faults in that circuit. Two types of sequential redundant faults are discussed.

Equivalent-State Redundant Faults

The following example demonstrates the equivalent-state redundant faults that may occur due to (1) *the constraint of single input change*, (2) *the non-critical races*, or (3) *delays*.

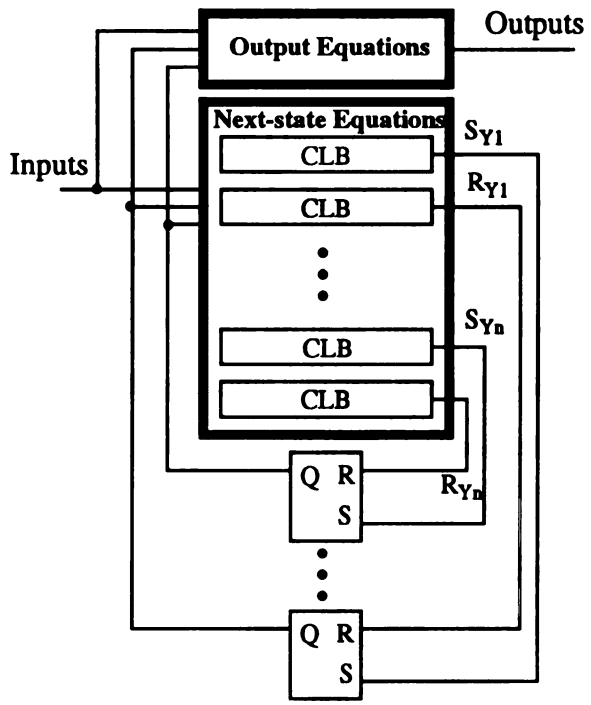


Figure 3.9: SR-latches implementation without shared logics.

Table 3.1: SR-Latch Excitation Table

Q(t)	Q(t+1)	S	R
0	0	0	-
0	1	1	0
1	0	0	1
1	1	-	0

Example 6:

Consider the STT-generated ASLC shown in Figure 3.10. The circuit is realized with the SR-latches and its logical equations for S_{Yi} and R_{Yi} are shown in Figure 3.10(c). Assume that a stuck-at-0 fault occurs at the output of the AND gate, $x_1x_2y_2$, of S_{Y1}. The fault causes the entry (x_1x_2-$y_1y_2y_3$) =(11-011) to change from (110) to (010) and the entry (x_1x_2-$y_1y_2y_3$)=(11-010) to change from (110) to (010). Assume also that the current state is (011) and the current input is (10). When the input is changed from (10) to (11), the next state will stabilize at (110) in the fault-free circuit and at (010) in the presence of this fault. With the single input change constraint, the next applicable input is either (01) or (10). As a result, both faulty and fault-free circuits will stabilize at the same state (000) for the input (01), or at (110) for the input (10). This is an equivalent-state redundant fault due to the constraint of single input change.

Consider a stuck-at-0 fault that occurs at output of the AND gate, $x_1\bar{x}_2$, of S_{Y2}. The fault causes the entry (x_1x_2-$y_1y_2y_3$)=(10-101) to change from (011) to (001) and the entry (x_1x_2-$y_1y_2y_3$)= (10-100) to change from (110) to (100). We first consider the entry (10-101) and assume that the current state is (101) and the current input is (00). When the input is changed from (00) to (10), the state is changed from (101) to (011) in the fault-free circuit as shown in Figure 3.10(b). Due to the non-critical race, the change may follow either one of these two paths: 101→001→011 or 101→111→011, and the circuit stabilizes at the stable state (011) in the fault-free circuit. On the other hand, in the presence of such a fault with input changed from (00) to (10), the state is changed from (101) to (001) and the fault forces the faulty circuit to take the transition path 101→001→011 and to stabilize at the stable state (011). Thus, both faulty and fault-free circuits stabilize at the same stable state. This is an equivalent-state redundant fault due to the non-critical race.

Consider the entry (10-100) for the above stuck-at-0 fault. Assume that the current state is (000) and the current input is (11). When the input is changed from (11) to (10), the state is changed from (000) to (110). The transition may take either 000→010→110, if the

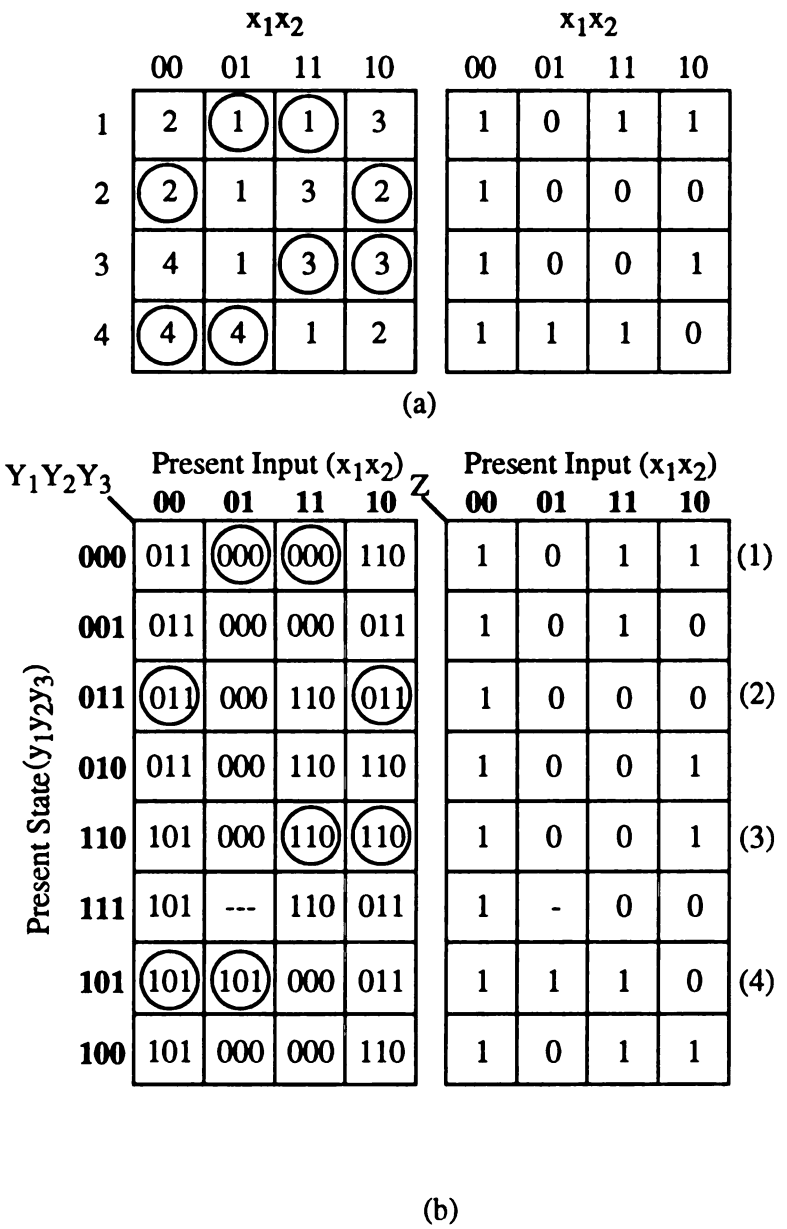


Figure 3.10: An ASLC example (a) Merged flow table and output table; (b) STT-generated ASLC; and (c) Logic implementation for (b).

$S_{Y1}R_{Y1}$ / Present State $(y_1y_2y_3)$	Present Input (x_1x_2) 00	01	11	10
000	0 -	0 -	0 -	1 0
001	0 -	0 -	0 -	0 -
011	0 -	0 -	1 0	0 -
010	0 -	0 -	1 0	1 0
110	- 0	0 1	- 0	- 0
111	- 0	- -	- 0	0 1
101	- 0	- 0	0 1	0 1
100	- 0	0 1	0 1	- 0

$S_{Y2}R_{Y2}$ / Present State $(y_1y_2y_3)$	Present Input (x_1x_2) 00	01	11	10
000	1 0	0 -	0 -	1 0
001	1 0	0 -	0 -	1 0
011	- 0	0 1	- 0	- 0
010	- 0	0 1	- 0	- 0
110	0 1	0 1	- 0	- 0
111	0 1	- -	- 0	- 0
101	0 -	0 -	0 -	1 0
100	0 -	0 -	0 -	1 0

$S_{Y3}R_{Y3}$ / Present State $(y_1y_2y_3)$	Present Input (x_1x_2) 00	01	11	10
000	1 0	0 -	0 -	0 -
001	- 0	0 1	0 1	- 0
011	- 0	0 1	0 1	- 0
010	1 0	0 -	0 -	0 -
110	1 0	0 -	0 -	0 -
111	- 0	- -	0 1	- 0
101	- 0	- 0	0 1	- 0
100	1 0	0 -	0 -	0 -

$$S_{Y1} = x_1x_2y_2 + x_1\bar{x}_2\bar{y}_3$$
$$R_{Y1} = \bar{x}_1x_2\bar{y}_3 + x_1x_2\bar{y}_2 + x_1\bar{x}_2y_3$$
$$S_{Y2} = \bar{x}_2\bar{y}_1 + x_1\bar{x}_2$$
$$R_{Y2} = \bar{x}_1x_2 + \bar{x}_1y_1$$
$$S_{Y3} = \bar{x}_1\bar{x}_2$$
$$R_{Y3} = x_2\bar{y}_1 + x_1x_2$$

(c)

change of y_1 is faster than y_2, or 000→100→110, if the change of y_2 is faster than y_1, and the circuit finally stabilizes at the stable state (110). On the other hand, in the presence of such a fault, the transition path 000→100→110 is changed as 000→100→100. This implies that if the change of y_2 is faster than y_1 and the states (110) and (100) are distinguishable, then the fault is detectable. However, if the change of y_1 is faster than y_2, then the fault is an equivalent-state redundant fault due to the delay. ♦

Examples 1 and 6 show that the equivalent-state redundant fault occur in the STT-generated ASLCs with and without SR-latch implementations.

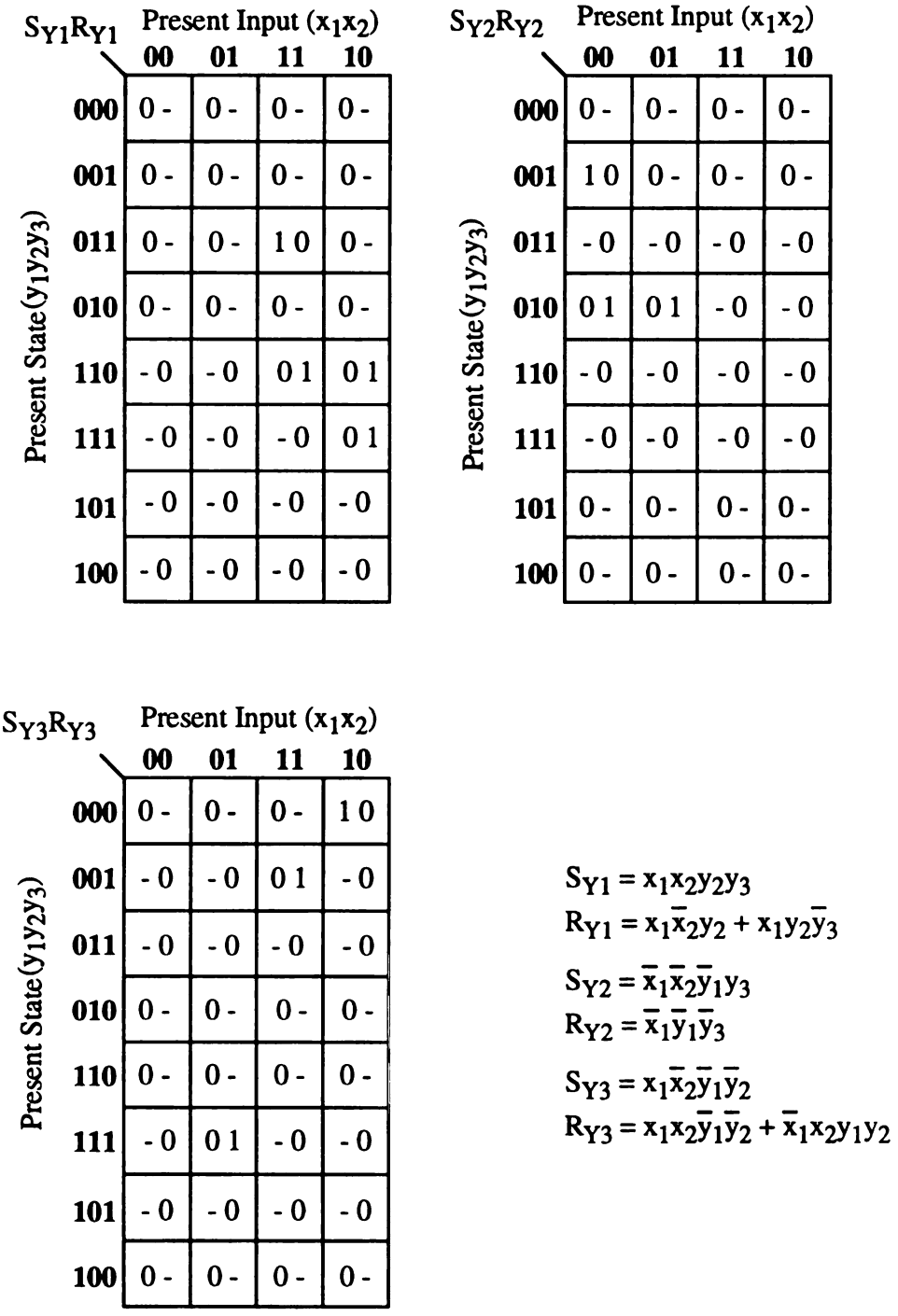
Invalid-State Redundant Faults

The following example demonstrates the invalid-state redundant faults that may occur due to either the existence of invalid states or improperly assign the don't care terms in the flow table.

Example 7:

Consider the UDC-generated ASLC in Figure 3.3(c) for Example 2. The transition table in Figure 3.3(c) is realized with SR-latches and its logical equations for the S_{Yi}'s and R_{Yi}'s are shown in Figures 3.11. Assume that a stuck-at-1 fault occurs at the y_2-input of the AND gate, $x_1\bar{x}_2y_2$ of R_{Y1}. This fault causes the entry $(x_1x_2\text{-}y_1y_2y_3)$=(10-101) to change from (101) to (001) and the entry (10-100) to change from (100) to (000). Suppose that the current state is (101) and the current input is (00). When the input is changed from (00) to (10), the fault-free circuit will stabilize at the state (101) with an output (11), but the faulty circuit will stabilize at the state (001) with an output (00). Since the faulty and fault-free states are distinguishable, the fault is detectable. However, as shown in Figure 3.3(b), the current state (101) is an invalid state which is not reachable from the reset state. Thus, the fault is an invalid-state redundant fault.

On the other hand, assume that a stuck-at-1 fault occurs at the $\bar{x}_2$-input of the AND gate, $\bar{x}_1\bar{x}_2\bar{y}_1y_3$, of S_{Y2}. The fault causes the entry $(x_1x_2\text{-}y_1y_2y_3)$=(01-001) to change from



$S_{Y1}R_{Y1}$ | Present Input (x_1x_2)

Present State ($y_1y_2y_3$)	00	01	11	10
000	0 -	0 -	0 -	0 -
001	0 -	0 -	0 -	0 -
011	0 -	0 -	1 0	0 -
010	0 -	0 -	0 -	0 -
110	- 0	- 0	0 1	0 1
111	- 0	- 0	- 0	0 1
101	- 0	- 0	- 0	- 0
100	- 0	- 0	- 0	- 0

$S_{Y2}R_{Y2}$ | Present Input (x_1x_2)

Present State ($y_1y_2y_3$)	00	01	11	10
000	0 -	0 -	0 -	0 -
001	1 0	0 -	0 -	0 -
011	- 0	- 0	- 0	- 0
010	0 1	0 1	- 0	- 0
110	- 0	- 0	- 0	- 0
111	- 0	- 0	- 0	- 0
101	0 -	0 -	0 -	0 -
100	0 -	0 -	0 -	0 -

$S_{Y3}R_{Y3}$ | Present Input (x_1x_2)

Present State ($y_1y_2y_3$)	00	01	11	10
000	0 -	0 -	0 -	1 0
001	- 0	- 0	0 1	- 0
011	- 0	- 0	- 0	- 0
010	0 -	0 -	0 -	0 -
110	0 -	0 -	0 -	0 -
111	- 0	0 1	- 0	- 0
101	- 0	- 0	- 0	- 0
100	0 -	0 -	0 -	0 -

$$S_{Y1} = x_1x_2y_2y_3$$
$$R_{Y1} = x_1\bar{x}_2y_2 + x_1y_2\bar{y}_3$$
$$S_{Y2} = \bar{x}_1\bar{x}_2\bar{y}_1y_3$$
$$R_{Y2} = \bar{x}_1\bar{y}_1\bar{y}_3$$
$$S_{Y3} = x_1\bar{x}_2\bar{y}_1\bar{y}_2$$
$$R_{Y3} = x_1x_2\bar{y}_1\bar{y}_2 + \bar{x}_1x_2y_1y_2$$

Figure 3.11: SR-latches implementation for Figure 3.3(c).

(001) to (011). Since both stable states (001) and (011) produce the outputs (00) and (01), respectively, they are distinguishable. However, the entry is a don't care entry as shown in Figure 3.3(b), and is not reachable from the reset state. Thus, the fault is also an invalid-state redundant fault. ♦

Examples 2 and 7 show that the invalid-state redundant fault occurs in the STT-generated or UDC-generated ASLCs with and without SR-latch implementations.

3.2.1.2 State Oscillations

Critical races in a fault-free circuit can always be eliminated with a proper state assignment. However, they may occur as a result of faults. Two types of critical races may occur: *Intrastate races* and *Interstate races.*

When inputs (S_{Yi},R_{Yi}) are changed from (1,1) to (0,0), the output value of the latch is determined by the race between S_{Yi} and R_{Yi}. Due to delays, both S_{Yi} and R_{Yi} are unlikely changed simultaneously. If the change of S_{Yi} is faster than that of R_{Yi}, the output value is 0, otherwise an output 1 is obtained. Such a critical race is referred to as *intrastate race* at Y_i. On the other hand, if a critical race occurs between two or more states, it is called a *interstate race.*

The following examples demonstrate both intrastate and interstate races.

Example 8: Intrastate Races

Consider the UDC-generated ASLC in Figure 3.3(b), but it is realized with SR-latches. Figure 3.12 shows the logic implementations for S_{Yi} and R_{Yi}. Assume that a stuck-at-1 fault occurs at the y_3-input of the AND gate, $x_1x_2y_2y_3$, of S_{Y1}. The fault causes $S_{Y1}=R_{Y1}=1$ and the entries (x_1x_2-$y_1y_2y_3$)=(11-010) and (11-110) to change from (010) to (u10), where u represents the undetermined value. In cases that $S_{Yi}=R_{Yi}=1$ occurs in the presence of fault, four cases can be identified: (1) Y_i=0; (2) Y_i=1; (3) Y_i=unchanged; and (4) Y_i=changed.

For Y_i=0, i.e. (u10)=(010), the faulty circuit has the same behavior as the fault-free one. Thus, the fault is redundant. However, when further changing input from (11) to (01), it results in the intrastate race at Y_1 and the fault detectability is determined by the relative delays between S_{Y1} and R_{Y1}.

For Y_i=1, i.e. (u10)=(110), if the current state is (010) and the input is changed from (10) to (11), the next state will stabilize at (010) in the fault-free circuit and at (110) in the faulty circuit. According to Figure 3.3(b), these two different stable states can be distinguished by further changing the input from (11) to (01). However, when the input changes from (11) to (01), (S_{Y1},R_{Y1}) is changed from (1,1) to (0,0). As a result, it causes an intrastate race at Y_1. Note that the situation can only be verified by the fault simulation, but not by the transition table of the faulty circuit.

For Y_i=unchanged, the entry (x_1x_2-$y_1y_2y_3$)=(11-110) is changed from (010) to (110). Assume that the current state is (110) and the current input is (01). When the input changes from (01) to (11), the next state will stabilize at (010) in the fault-free circuit and at (110) in the faulty circuit. Similar to Case 2, an intrastate race occurs.

For Y_i=changed, the entry (x_1x_2-$y_1y_2y_3$) =(11-010) is changed from (010) to (110). This results in a state oscillation between (x_1x_2-$y_1y_2y_3$)=(11-010) and (11-110). ♦

Example 9: Interstate Races

Consider the same ASLC in Example 8. Assume that a stuck-at-1 fault occurs at the y_2-input of the AND gate, $x_1x_2y_2y_3$, of S_{Y1}. The fault causes the entry (x_1x_2-$y_1y_2y_3$)=(11-001) to change from (000) to (100), and results in a critical race between Y_1 and Y_3. Therefore, state oscillations may occur in this faulty circuit. In addition, the invalid state (100) may cause the fault behavior unpredictable. ♦

The interstate races can be avoided by choosing SR-latch whose output value is unchanged when S_{Yi}=R_{Yi}=1. This type of SR-latch is equivalent to the C-element [1] which is commonly used in STG-modeled ASLCs.

$S_{Y1}R_{Y1}$

Present State $(y_1y_2y_3)$ \ Present Input (x_1x_2)	00	01	11	10
000	0 -	0 -	0 -	0 -
001	0 -	- -	0 -	0 -
011	0 -	0 -	1 0	0 -
010	0 -	0 -	0 -	0 -
110	- 0	- 0	0 1	0 1
111	- -	- 0	- 0	0 1
101	- -	- -	- -	- -
100	- -	- -	- -	- -

$S_{Y2}R_{Y2}$

Present State $(y_1y_2y_3)$ \ Present Input (x_1x_2)	00	01	11	10
000	0 -	0 -	0 -	0 -
001	1 0	- -	0 -	0 -
011	- 0	- 0	- 0	- 0
010	0 1	0 1	- 0	- 0
110	- 0	- 0	- 0	- 0
111	- -	- 0	- 0	- 0
101	- -	- -	- -	- -
100	- -	- -	- -	- -

$S_{Y3}R_{Y3}$

Present State $(y_1y_2y_3)$ \ Present Input (x_1x_2)	00	01	11	10
000	0 -	0 -	0 -	1 0
001	- 0	- -	0 1	- 0
011	- 0	- 0	- 0	- 0
010	0 -	0 -	0 -	0 -
110	0 -	0 -	0 -	0 -
111	- -	0 1	- 0	- 0
101	- -	- -	- -	- -
100	- -	- -	- -	- -

$$S_{Y1} = x_1x_2y_2y_3$$
$$R_{Y1} = x_1\bar{x}_2 + x_1\bar{y}_3$$
$$S_{Y2} = \bar{x}_1y_3$$
$$R_{Y2} = \bar{x}_1\bar{y}_1\bar{y}_3$$
$$S_{Y3} = x_1\bar{x}_2\bar{y}_2$$
$$R_{Y3} = x_2\bar{y}_2 + \bar{x}_1y_1$$

Figure 3.12: SR-latches implementation for Figure 3.3(b).

Property 4: Intrastate and interstate races will never occur in a faulty circuit in the presence of stuck-at-0 fault.

Proof: A stuck-at-0 fault will not cause $S_{Yi}=R_{Yi}=1$, thus no intrastate race occurs at Y_i. Also, the unchanged state variables will remain unchanged in the presence of stuck-at-0 faults. Therefore, no interstate race occurs. *Q.E.D.*

By Property 4, no critical races occur in the presence of stuck-at-0 faults such that state oscillations will not occur. Note that if a circuit is free of intrastate and interstate races, then Properties 1,2 and 3, are also applied for ASLCs implemented with SR-latches.

3.2.2 Testability Synthesis Properties

In this section, the testability synthesis properties for ASLCs implemented with SR-latches are described. If the output functions are synthesized with the prime and irredundant logic, then the output logics are fully testable. Thus, we only consider the stuck-at faults at the circuit for the next-state logics.

The logic implementation of internal state variable, Y_i, is shown in Figure 3.13, where the CLB represents the combinational logic block. First, consider the stuck-at fault at the primary inputs x_i and the feedback line Y_i.

Property 5. A stuck-at fault at the primary input x_i or feedback line Y_i is detectable.

Proof: [By Contradiction] Consider the behaviors of fault-free and faulty circuits, the fault is undetectable if and only if the circuit have the same behavior in both circuits. As a result, the value of primary input x_i has no effect on the behavior of the circuit, therefore this input is redundant and can be removed from the input lines. The same conclusion applies to the stuck-at fault at the feedback line Y_i. *Q.E.D.*

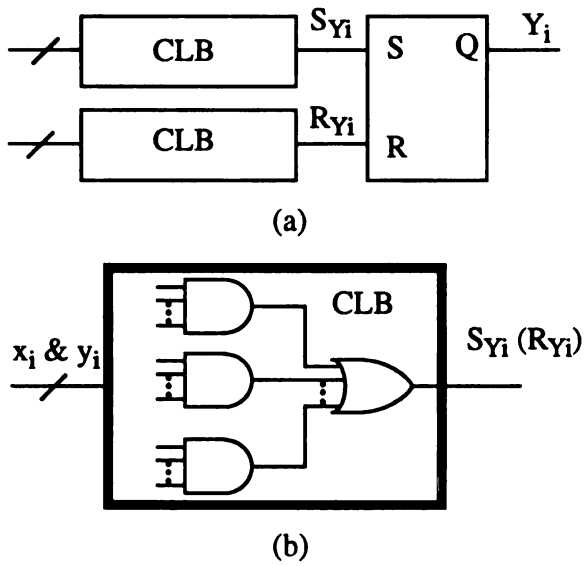


Figure 3.13: Block diagram of next-state logic (a) Y_i-variable; and (b) Combinational logic block (CLB).

By Property 5, only the stuck-at fault within the CLB needs to be considered. A fault in a CLB is testable only if its faulty behavior can be propagated to the output of the CLB. If a CLB is synthesized with prime and irredundant logic, there always exists at least one state which can propagation fault behavior to the output of the CLB. Since each AND gate contains at least one *essential-1* which is not shared by other AND gates, therefore there always exists at least one state q where a stuck-at-0 fault will be propagated to the output of a CLB. Similarly, the effect of a stuck-at-1 fault at the output of AND gate will be seen as long as the expected output of the CLB is 0.

These conditions are necessary but not sufficient to detect the fault. For a fault to be testable, three constraints must be satisfied: (1) *Constraint 1*: the fault effect must be propagated to the output of a SR-latch, i.e., the value of Y_i must be changed. Otherwise, this fault is "unobservable" from the internal state variable and the fault effect cannot be excited. (2) *Constraint 2*: the exciting state q must be reachable from the reset state. (3) *Constraint 3*: the state transition must stabilize at another stable state which is distinguishable from the fault-free circuit. Otherwise, it is a redundant fault as shown in

Examples 6 and 7. Constraints 2 and 3 are the same constraints for ASLCs without latches, but constraint 1 is the condition that only applies to ASLCs with latches. Nevertheless, *Condition C*, described in Section 3.1.2, can still be applied to ASLCs with latches. Further, for SR-latches, only the stuck-at faults at the two inputs S and R and the output are considered.

STT-generated ASLCs with SR-latches

Consider the testability synthesis properties in the presence of stuck-at-0 faults. By Property 4, in the presence of stuck-at-0 faults, no races occur in STT-generated ASLCs with SR-latched and the unchanged state variables will never be affected during the state transition due to the stuck-at-0 fault. Therefore, only the changed bit, y_r in $X_c(S_i,S_k)$, of the $[S_i,S_k]$ in a K-set needs to be considered.

Lemma 5: A stuck-at-0 fault that occurs at y_r in $X_c(S_i,S_k)$ is detectable if (1) the K-set contains an unstable state S_j with $D_H(S_j,S_k)=r$ and $S^p_{fj} \neq S^p_{nj}$; (2) S_j is distinguishable from S_k in the input column I_p; and (3)T^p_{i0} exists.

Proof: Since $D_H(S_j,S_k)=r$ and $S^p_{fj} \neq S^p_{nj}$, S_j satisfies *Condition C.1* and becomes a stable state in the presence of this fault in the input column I_p. By conditions (2) and (3), *Condition C.2* is also satisfied.Thus, this fault is detectable. *Q.E.D.*

For the stuck-at-1 fault, the testability synthesis properties will be much more complicated than that of the stuck-at-0 fault. Since the stable state may also be affected in the presence of stuck-at-1 faults, both the unstable and stable states need to be considered. First, consider the synthesis property for the affected unchanged bit, y_r in $X_u(S_i,S_k)$ of the $[S_i,S_k]$ in a K-set. Similar to Section 3.1.2, when a fault affects y_r in $X_u(S_i,S_k)$, in order to avoid state oscillations, the transition paths of $[S_i,S_k]$ should be directed to stabilize at the same stable state in the faulty circuit. Therefore, Theorem 2, in Section 3.1.2, can also be applied for ASLCs implemented with SR-latches as long as there is no critical race between S_{Yi} and R_{Yi} in the presence of stuck-at-1 faults.

Second, we investigate the testable properties when the stuck-at-1 fault affects the y_r in $X_c(S_i,S_k)$ of the $[S_i,S_k]$ in a K-set. For a stuck-at-1 fault, if it affects the S_{Yr} or R_{Yr}, then it will result in $(S_{Yr},R_{Yr})=(1,1)$. For example, in Figures 3.10(b) and 3.10(c), since the y_1-bit is a changed bit of the transition pair [011,110] in the input (11) column, the term $x_1x_2y_2$ will be covered by at least one prime implicant of S_{Y1} in the prime and irredundant implementations. In other words, it implies $S_{Y1}=1$ for those states in the TP of [011,110]. If a stuck-at-1 fault occurs at y_1-bit and some states in this transition path are affected, $(S_{Y1},R_{Y1})=(1,1)$ will occur in the presence of this fault. Therefore, whether this fault is detectable or not highly depends on the excitation function of SR-latches chosen. The requirement is that the constraint 1 must be satisfied.

As a result, if the changed bit of a transition pair is affected, the condition (S,R)=(1,1) is unavoidable. Similar to Example 8, four cases are investigated when $(S_{Yi},R_{Yi})=(1,1)$. Case 1: For $Y_i=0$, if the stuck-at-1 fault affects the S_{Yi} and the next value of Y_i is 0, it is a redundant fault. This is due to the fact that when $S_{Yi}=R_{Yi}=1$ occurs, the next value of Y_i is still 0 such that the behavior of both fault-free and faulty circuits is the same. (*violating Constraint 1*). Case 2: For $Y_i=1$, it is also a redundant fault, if the stuck-at-1 fault affects the R_{Yi} and the next value of Y_i is 1. (*violating Constraint 1*). Case 3: For Y_i=unchanged, it will always cause the y_r in $X_c(S_i,S_k)$ become unchanged. Therefore, if the affected state is reachable from the reset state and the affected state can stabilize at a distinguishable stable state, then this fault is detectable. Case 4: For Y_i=changed, it is likely to create unstable states, therefore state oscillations may occur as shown in Example 8. Based on above discussions, Case 3 is obvious a good choice for the excitation function of SR-latches.

In addition to the S=R=1 in the presence of stuck-at-1 faults, the testability synthesis property for the stuck-at-1 fault that affects the y_r in $X_c(S_i,S_k)$ is the same as Theorem 1. Based on Theorem 1 and the above discussion for Cases 1 and 2, two corollaries can be derived.

Corollary 1: Assume that the excitation function of SR-latches is assumed Y_j=0 when S_{Yj}=R_{Yj}=1, and a stuck-at-1 fault affects the y_j in $X_c(S_i,S_k)$. Then, this fault is undetectable from those states in TP_{ik}, if the value of y_j-bit of its corresponding stable state S_k is 0.

Corollary 2: Assume that the excitation function of SR-latches is assumed Y_j=1 when S_{Yj}=R_{Yj}=1, and a stuck-at-1 fault affects the y_j in $X_c(S_i,S_k)$. Then, this fault is undetectable from those states in TP_{ik}, if the value of y_j-bit of its corresponding stable state S_k is 1.

UDC-generated ASLCs with SR-latches

Similarly, those synthesis properties for STT-generated ASLCs with SR-latches can be further modified for UDC-generated ASLCs with SR-latches. Based on the UDC state assignment, only one internal state variable is changed at a time, therefore a stuck-at-0 fault can only cause the affected unstable state to become stable. It should be noted that no stable state will be affected in the presence of stuck-at-0 faults.

Lemma 6: A stuck-at-0 fault is detectable if (1) S_u in P_{ab}, where S_u is an unstable state in the input column I_p, and $S^p_{fu} \neq S^p_{nu}$; (2) S_u is distinguishable from S_b in input column I_p; and (3) T^p_{u0} exists.

Proof: A stuck-at-0 fault can only result in an unstable state to become stable. By $S^p_{fu} \neq S^p_{nu}$, S_u satisfies *Condition C.1* and becomes a stable state in the input column I_p. Moreover, no race condition will occur in the presence of this fault. Similar to Lemma 5, this fault is detectable. *Q.E.D.*

By Lemma 6, it is easy to check whether a stuck-at-0 fault is detectable or not. It should be noted that the affected unstable state in the STT-generated ASLCs with latches may be still an unstable state in the presence of stuck-at-0 faults.

Consider the stuck-at-1 fault in UDC-generated ASLCs with SR-latches and a connected path P_{ab} in the input column I_p. If the stuck-at fault affects the y_i in $X_u(S_b,S_{nb})$,

where S_b is a stable state in fault-free circuit, the state S_b will become an unstable state but no critical races occur. On the other hand, if a stuck-at fault affects the y_i in $X_u(S_j,S_{nj})$, where S_j is an unstable state in P_{ab}, then critical races may occur. In general, the affected unstable state will either become a stable state or introduce the race condition in the presence of stuck-at-1 faults. Therefore, in order to reduce the race conditions among internal state variables in UDC-generated ASLCs with SR-latches, the better way is to choose the SR-latches with the characteristic: When $S_{Yi}=R_{Yi}=1$, the output value of this latch is unchanged. However, the *Constraint 1* must also be satisfied in order to excite this fault. Similarly, in the case of $S_{Yi}=R_{Yi}=1$, whether the fault can be excited or not depending on the excitation function of the SR-latches chosen.

It should be mentioned that, based on UDC state assignments, a stuck-at-1 fault that affects the y_i in $X_u(S_x,S_{nx})$ may or may not create the condition $S_{Yi}=R_{Yi}=1$. Moreover, if a stuck-at-1 fault affects the y_j in $X_c(S_x,S_{nx})$, then the condition $S_{Yj}=R_{Yj}=1$ occurs at Y_j. But it does not mean that the other states, within the same connected path, encounter the same condition $S_{Yj}=R_{Yj}=1$. Therefore, the fault may be still detectable from the other states.

In general, a fault may or may not result in race conditions in the faulty circuit. Without race conditions, either the affected unstable state become stable or the affected stable state become unstable. In the former case, if the affected unstable state is reachable from the reset state and is distinguishable from its corresponding stable state in the fault-free circuit, then this fault is detectable. In the latter case, if a fault causes the stable state to become unstable and it finally stabilize at another distinguishable stable state, then this fault is detectable. On the contrary, if race conditions occur, in order to avoid critical races, it should be guaranteed that the race conditions will always stabilize at a unique and distinguishable stable state. Thus, it is concluded that the stable state should be also affected. Therefore, Theorems 3 and 4 are also applied for UDC-generated ASLCs with latches as long as the stuck-at-1 fault satisfies the *Constraint 1*. Theorem 3 is to deal with the affected unstable state without race. When S_t is the stable state S_b in Theorem 4, it is

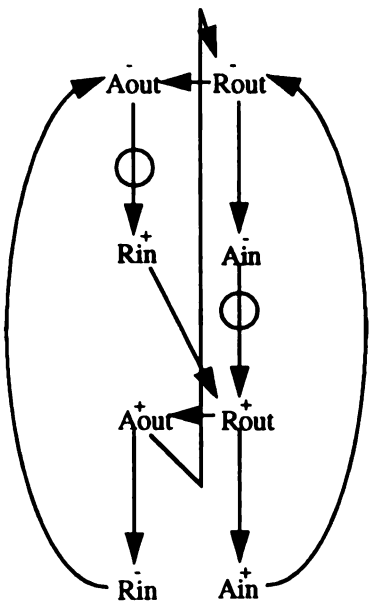
concerned with the affected stable state without race. On the other hand, when S_t is the intermediate state in Theorem 4, it is the testable condition for the faulty circuit with races.

3.3 STG-Modeled ASLCs

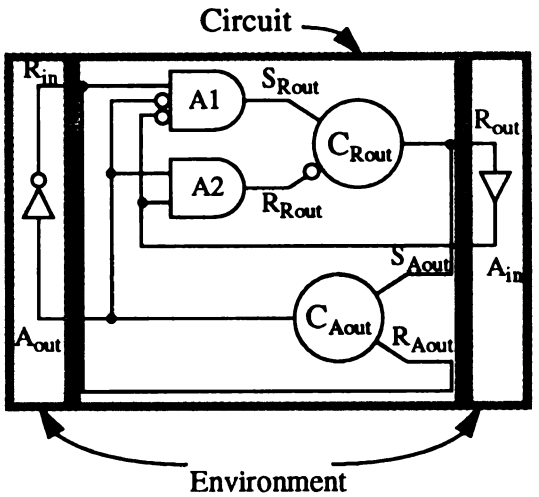
The previous sections have presented the fault effects and synthesis properties of Huffman-modeled ASLCs with and without SR-latches. Prior to the discussion of the fault effects and synthesis properties of STG-modeled ASLCs, the similarities and differences between these two models are described in this section.

In STGs, two or more input/output signal transitions can be enabled simultaneously. Thus, three different types of concurrency can be identified: *input concurrency*, *input/output concurrency*, and *output concurrency*. A STG-modeled circuit without input/output concurrency can be viewed as a special Huffman-modeled circuit. More specifically, as discussed in Section 2.1.1, the output function of a Moore-type Huffman-modeled ASLC is ω:S $\rightarrow$ O. The STG-modeled ASLC is equivalent to the Moore-type Huffman-modeled ASLC with a *unity function*, i.e. ω=1. The unity function implies S=O, i.e. the output variables are the same as state variables. In addition, the signal transition graph is equivalent to the flow table, in Huffman-modeled ASLCs, with concurrency, referred to as a *concurrent flow table* (CFT).

For example, consider a four-phase handshaking circuit [1,19], where its STG representation is shown in Figure 3.14(a) and the logic implementation is illustrated in Figure 3.14(b), where the equations for both S and R terminals in each C-element are also shown. A C-element, as shown in Figure 3.15, is a modified SR-latch which defines the output to be unchanged when (S,R)=(1,1). The STG representation can be transformed to a flow table shown in Figure 3.14(c), where the entries are the next state variables which are the same as the output variables, R_{out} and A_{out}. It is easily to see the input concurrency in STG, as shown in Figure 3.14(a), where the input signal transitions, R_{in}^{+} and A_{in}^{-}, can be fired simultaneously. The concurrency can also be presented in the concurrent flow table.



(a)



$S_{Aout} = R_{out}$ $S_{Rout} = \overline{A}_{in}\overline{A}_{out}R_{in}$

$R_{Aout} = \overline{R}_{in}$ $R_{Rout} = A_{in}A_{out}$

(b)

$R_{out}A_{out}$ \ $R_{in}A_{in}$	00	01	11	10
00	(00)	(00)	(00)	10
01	00	00	(01)	(01)
11	(11)	01	01	(11)
10	--	--	11	11

(c)

Figure 3.14: An STG-modeled ASLC (a) Four-phase handshaking protocol; (b) Logic implementation; and (c) Equivalent concurrent flow table.

As shown in Figure 3.14(c), when (R_{in},A_{in}) is changed from (0,1) to (1,0) with the present state (R_{out},A_{out})=(0,0), it will reach the same destination regardless of the firing sequence. The STG also shows an input/output concurrency, where the input signal transition A_{in}^+ and the output signal transition A_{out}^+ are fired. The same concurrency can be described from the concurrent flow table. Consider the present state is (R_{out},A_{out})=(1,0) and the current input is (R_{in},A_{in})=(1,0), when the A_{in}^+ input is enabled, i.e. the input is changed to (1,1), the next state is (1,1) and further changes to (0,1) when A_{out}^+ is enabled. On the other hand, if the output A_{out}^+ is enabled, the next state is also (1,1) and further changes to (0,1) when A_{in}^+ is enabled. This implies that no matter which signal is enabled earlier or both are enabled at the same time, it will reach the same destination. This STG does not include any output concurrency.

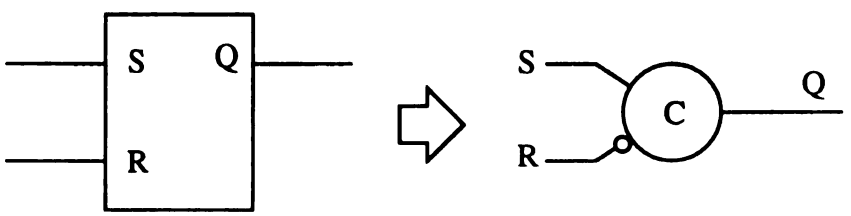


Figure 3.15 The transformation from SR-latch to C-element.

The question that naturally arises is *what is the difference between the Huffman-modeled ASLCs and STG-modeled ASLCs as far as the concurrency is concerned.* For the output concurrency, since the output variables in the STG are the state variables, the output concurrency means that two or more state variables are changed simultaneously. Thus, a STG-modeled ASLC is equivalent to a Huffman-modeled ASLC with the STT state assignment. On the other hand, if the output variables in the STG-modeled ASLC are changed one at a time, then it is equivalent to the Huffman-modeled ASLC with the UDC

state assignment. For input concurrency, a STG-model ASLC is equivalent to a Huffman-modeled ASLC with multiple input changes. For the input/output concurrency, this is the special feature of STGs which resolves the fundamental mode problem in Huffman model. Due to the input/output concurrency in a STG, asynchronous control circuits have been successfully designed and implemented. The next question is *whether or not the STG-modeled ASLCs with input/output concurrency is testable.* If not, such a greater feature will be useless for a practical ASLC design which must be testable.

The following example demonstrates that given a STG-modeled circuit with input/output concurrency, a fault that occurs at the concurrent signals is not testable.

Example 10:

Consider the four-phase handshaking circuit in Figure 3.14. Figures 3.16(a) and 3.16(b) shows the K-maps for the equations presented in Figure 3.14(b). Assume that a stuck-at-1 fault occurs at the input, A_{out}, of Gate A2. The fault causes the K-maps for ($S_{Rout}R_{Rout}$) to change as the one shown in Figure 3.16(c), where the boldfaced entries indicate the faulty bits. As a result, the behavior described in the CFT (Figure 3.14(c)) for the fault-free circuit is changed as that in Figure 3.16(d) for the faulty circuit. Comparing both CFTs, ($R_{in}A_{in}R_{out}A_{out}$)= (1110) is the only entry to test the fault. However, a critical race occurs in the entry, where the state is changed from (10) to (01). If R_{out} is changed faster than A_{out}, then it will stabilize at (00). Therefore, this fault is detectable. However, if A_{out} is changed faster than R_{out}, the circuit will stabilize at (01) which is the same as fault-free circuit. And, this fault is not detectable. Whether or not this fault is detectable is dependent of the relative delays between A_{out} and R_{out}.

Since the internal state variables in STG-modeled ASLCs are also the output variables which are observable, testing of STG-modeled ASLCs is much easier than that of Huffman-modeled ASLCs. However, if a STG-modeled ASLCs does not satisfy the Unique State Coding (USC) property, it is necessary to add the some internal variables

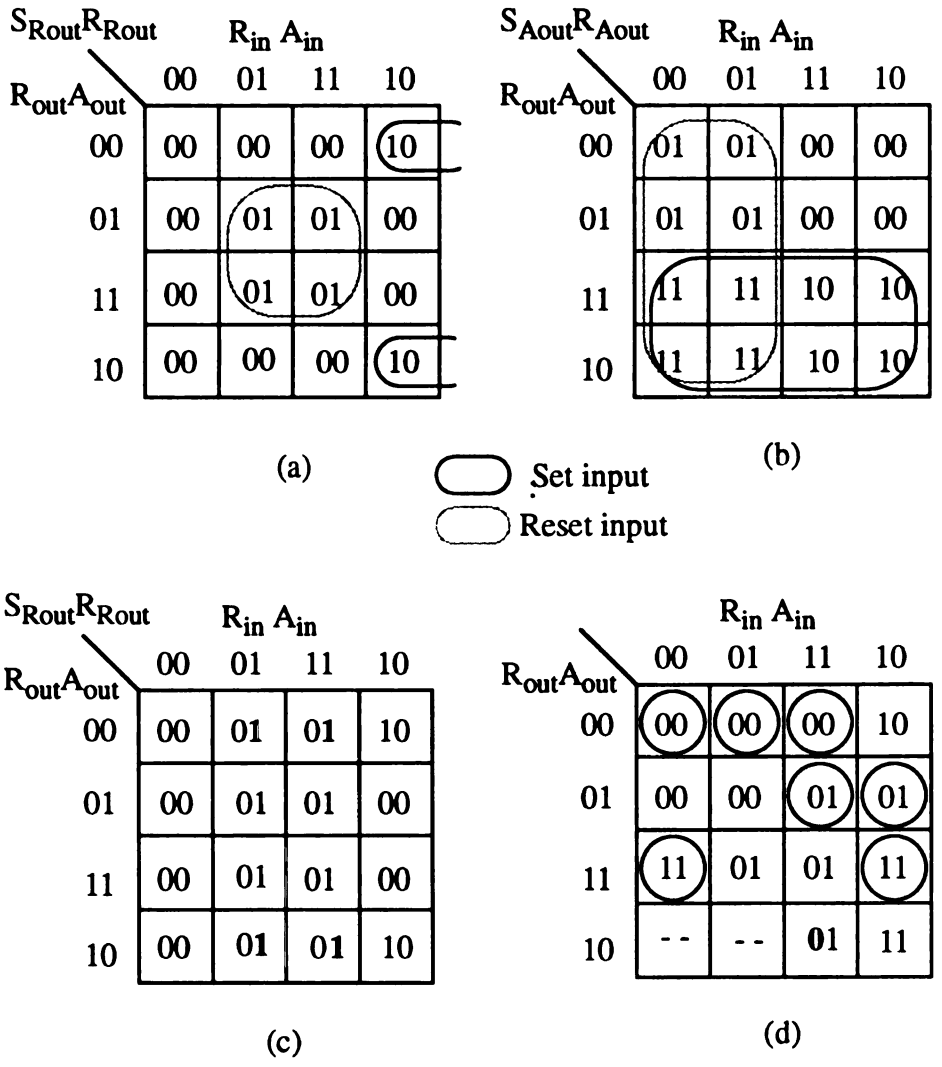


Figure 3.16: Example 10 (a) R_{out}; (b) A_{out}; (c) Map for R_{out} in the presence of a stuck-at-1 fault at the A_{out}-input of Gate A2; and (d) Equivalent concurrent flow table of (c).

which are not the output variables and thus not observable. If so, then the fault effects presented in the previous sections for Huffman-modeled ASLCs will also occur in STG-modeled ASLCs. Similarly, state oscillations in a ASLC can be observed from the output variables if it does not contain any internal variables. However, they may occur in the STG-modeled ASLCs with internal variables. Invalid-state redundant faults may occur in Huffman-modeled ASLCs, but they will never occur in STG-modeled ASLCs. This is simply because the state variables are generally encoded properly when the circuit is synthesized.

Since the input/output concurrency is allowed in STG-modeled ASLCs, applying appropriate test sequence becomes a very difficult problem because the test input must be applied at the same time when the output changes. Obviously, predicting when the output will be changed is not an easy task. Such a problem will not occur in Huffman-modeled ASLCs with the fundamental mode operation. To avoid that problem, STG-modeled ASLCs assume to be operated in the fundamental mode during testing. Note that, under such assumption, the above mentioned fault is not testable due to the fact that the entry (1110) will never be reached because the entry (1010) is an unstable state.

Recently, a design methodology for testable STG-modeled ASLCs has been presented [19] which assumes (1) a wire stuck-at fault model; (2) the circuit is operated in fundamental mode during testing; and (3) a AND gate with inverter(s) in the two-level logic is treated as a gate. Under these assumptions, the STG-modeled circuits can be easily tested. In practical, however, the assumptions are impractical. A gate stuck-at fault is commonly assumed in digital testing instead of the wire stuck-at fault model. The AND gate and inverters should be treated as separated gates. As demonstrated in Example 10, the operation of fundamental mode cannot detect the faults relative to the input/output concurrency.

3.4 Discussion

This chapter has presented the fault effects and testability synthesis properties for Huffman-modeled ASLCs. The fault effects include sequential redundant faults and state oscillations. This study shows that the STT state assignment does not create cycles for avoiding critical races, but it implicitly creates cycles for non-critical races, i.e., it provides multiple transition paths for the circuit to stabilize at the same state for the non-critical races. As a result, the STT-generated ASLCs inherently generate the equivalent-state redundant faults. On the other hand, since the race-free UDC state assignment provides only one connected path for each state transition, the possible equivalent-state redundant faults can be reduced. In addition, a fault in a UDC-generated ASLC can be easily identified by tracing its connected paths. Thus, the UDC-generated ASLCs are better than the STT-generated ASLCs as far as synthesizing testable circuits is concerned. As discussed in Section 3.1, several rules have been concluded for synthesizing state oscillation-free circuits. Those rules assume the race-free faulty circuits. In practice. detecting race conditions in the faulty circuits is not an easy task.

Section 3.3 concludes that testing STG-modeled ASLCs without internal variables is much easier than Huffman-modeled ASLCs. If STG-modeled ASLCs have some internal variables, then the fault effects result in Huffman-modeled ASLCs will also occur. The fault effects include the equivalent-state redundant faults and state oscillations. STGs have a special feature of input/output concurrency, but some faults related to this concurrency are not testable without scan design structure even though the circuits assume to operate in fundamental mode. This has motived the development of a scan design for STG-modeled and Huffman-modeled ASLCs for achieving full testability for all single stuck-at faults.

Chapter 4

ASLCScan - A Scan Design for Asynchronous Sequential Logic Circuits

Due to the lack of controllability and observability, testing of sequential circuits has been recognized as a very difficult task. In order to reduce the complexity of the test generation problem, scan designs [20-24] have been proposed and successfully implemented for CSLCs. With scan structures, all state variables of a sequential circuit are completely controllable and observable from primary inputs and outputs. Thus, the test generation problem is reduced to one of just testing the combinational logic. This chapter presents a scan design, ASLCScan, for ASLCs. The basic concept behind this development is that, with the scan structure, an ASLC is operated in an asynchronous way during the normal operation mode, while it is synchronized with clock signals during the test mode.

In the next section, some design issues on the scan design, LSSD, are addressed. Section 4.2 describes the ASLCScan method including the scan structure and its operation. In order to demonstrate the scan design for ASLCs, two examples with ASLCScan method are illustrated in Section 4.3.

4.1 Level-Sensitive Scan Design

The scan design, LSSD, has been introduced for structural design for testability in many IBM machines. This section discusses the design principle and hazard/race problems.

LSSD Design Principle

There are two fundamental requirements for the LSSD technique. The first requirement is that all changes to the state of the circuit are controlled by the level of a clock control signal, rather than by the edge of the clock. Further, the steady-state response to a change of value on a primary input is independent of the propagation delay of gates and interconnect elements within the circuit. The response is also independent of the order of input-value changes in the event of simultaneous multiple changes. This is the property of "level-sensitivity", designed to reduce the dependency of the circuit on its ac parameters, such as degraded rise and fall times, degraded propagation delays, or other faults that have the potential of introducing race and hazard conditions. Therefore, the potential effect of the failure mechanisms that cause timing faults is reduced. In LSSD design, this property is achieved by using master and slave latches which are controlled by two non-overlapping clocks. The second requirement of LSSD circuits is that the circuit should posses the scan-path property which is achieved by using a special purpose stored-state device called polarity-hold Shift Register Latch (SRL).

Hazard/Race Problems and some Solutions

Let D and C be the data and clock input signals, respectively. In order to avoid races/hazards, both signals must be designed under the following two constraints: (1) D should outlast the ending edge of C sufficiently, i.e. the value of input datum must last for enough time when the pulse of the storing signal returns to its inactive value; and (2) the pulse width of C is larger than the minimum time for the regeneration in the feedback loop to take over control and maintain the new state.

Consider a polarity-hold latch which is designed under these two constraints. The hazard conditions exist only when a "1" is stored. More specifically, as indicated in its Karnaugh-map (K-map), as shown in Figure 4.1 (a), a potential hazard occurs when the signal C is changed from 1 to 0 or from 0 to 1. A glitch caused by the latter case may not cause any serious problem because the Q-output will finally stabilize at a "1". Thus, the following discussion will consider the former case only. Consider the case that Q=1, D=1,

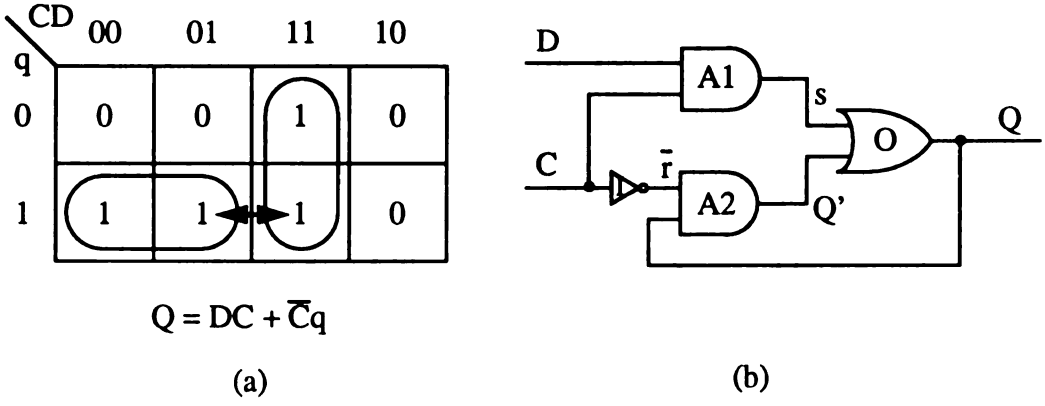
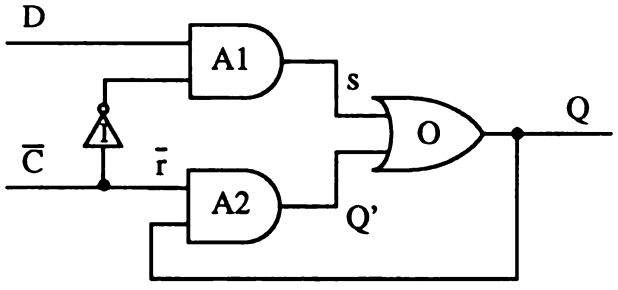


Figure 4.1: Potential hazard condition in the case of storing a "1" (a) K-map representation; and (b) Logic implementation.

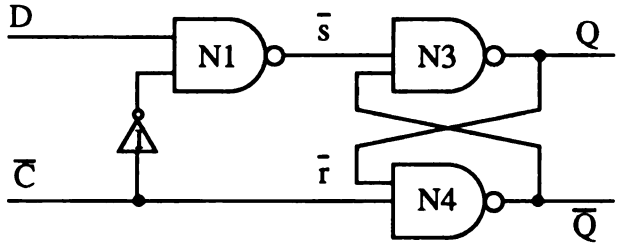
and C is changed from 1 to 0, denoted as 1→0. This results in signal value changes at nodes s and Q' as indicated in Figure 4.1(b) as 1→0 and 0→1, respectively. Let R_a and F_a denote the rising and falling time of a gate a, respectively. Thus, a hazard occurs if $F_{A1} < R_I + R_{A2}$. In general, the combined delay through two gates is usually larger than that through one gate.

To avoid the hazards, numerous design techniques have been presented in [52,53]. Consider the latch design in Figure 4.2(a) [52], where the inverter is connected to the gate A1 instead of A2 in Figure 4.1(b), and the signal $\overline{C}$ is applied. As a result, the hazard will

not occur if $F_I + F_{A1} > R_{A2}$. Figure 4.2(b) shows the NAND gate implementation of Figure 4.2(a).



(a)

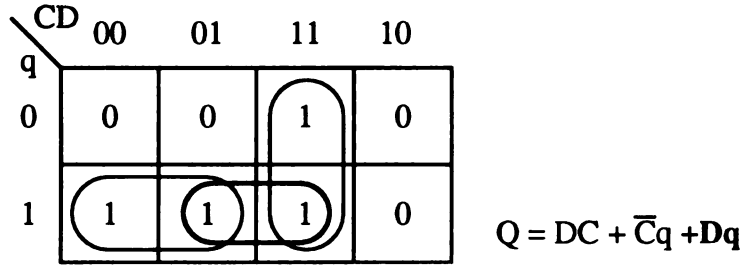


(b)

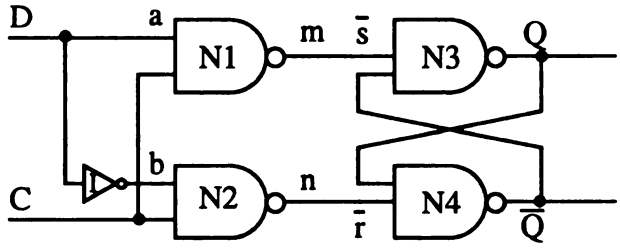
Figure 4.2: Polarity-hold latch in [52].

Enhanced Polarity-Hold Latches

The polarity-hold latch in Figure 4.2(b) is referred to as the *simple* PH. To remove the hazard condition for reliable operation, an enhancement can be made by adding the hazard protection term, Dq, as shown in Figure 4.3(a). The corresponding NAND gate implementation is shown in Figure 4.3(b) which is referred to as the *enhanced* PH. By using the hazard protection term, both $\bar{s}$ and $\bar{r}$ will not be active at the same time, in this case $\bar{s}=\bar{r}=0$, therefore the hazard condition can be avoided.



(a)



(b)

Figure 4.3: Polarity-hold latch with hazard protection (a) K-map representation; and (b) Enhanced PH.

Design Trade-offs

As discussed, both simple and enhanced PHs are hazard-free. The simple PH avoids the hazard by defining the path delays where $F_I + F_{A1} > R_{A2}$, while the enhanced PH employs an extra NAND gate, N2, to become hazard-free. Therefore, the enhanced PH is generally more reliable than the simple PH, but requires an extra NAND gate.

As shown in Figure 4.3, a hazard protection term, Dq, is added for reliable operation. However, hazards and races may occur as a result of faults. Consider the latch in Figure 4.3(b). The s/1 faults at nodes "a" and "b", referred to as *faults Fa* and *Fb*, cause hazards. For example, the logic function of the latch becomes Q=DC+$\overline{C}$q in the presence of fault Fb. This implies that the hazard protection term, Dq, disappears. On the other

hand, when the clock C is turned on and then off, the presence of Fb fault causes the contents of Q and $\overline{Q}$ to be shown as below, where nodes "m" and "n" are indicated in Figure 4.3(b).

D	C	m	n	Q	$\overline{Q}$	
1	1	0	0	1	1	
1	0	1	1	?	?	← Critical Race

When C=1, we obtain (m,n)=(0,0) and (Q,$\overline{Q}$)=(1,1). Then, when clock C turns off, i.e., C changes from 1 to 0, (m,n) is changed from (0,0) to (1,1) which causes a race. If the change of "m" is faster than that of "n", then (Q,$\overline{Q}$)=(0,1). Otherwise, (Q,$\overline{Q}$)=(1,0). Thus, the race is critical. Similarly, the existence of the fault Fa causes the logic function to become Q=Dq+$\overline{C}$q+C, where a static hazard exists. As a result, a critical race occurs as D is 0 and the clock C is turned off. This concludes that the latch in Figure 4.3(b) may not operate properly due to the race caused by the Fa or Fb fault.

As a result, the enhanced PH is generally more reliable than the simple PH in the fault-free circuit. But, enhanced PH becomes more unreliable when the Fb fault occurs in the faulty circuit.

4.2 ASLCScan Method for ASLCs

This section describes a scan design, ASLCScan method, for ASLCs. The fault model considered here is the single stuck-at fault at all the gate inputs and outputs. The ASLC designed with ASLCScan method is operated in two modes: test mode and normal mode. During the test mode, the scan structure is tested in a synchronous way, i.e., two non-overlapping scan clock signals are used to shift the data out for testing. In that case, the ASLC under test is operated in the fundamental mode. The clock rate of the scan clocks and system clocks are determined by the worst case timing in the ASLC under test. On the other hand, during the normal operation mode, there exists no clock signal and the

scan paths are disconnected. The scan structure and the ASLC are operated in an asynchronous way.

In this section, we first describe the scan structure in the ASLCScan method using SR-latches, and then discuss the generation of test pattern for achieving full testability for all single stuck-at faults. Finally, the operation of the ASLCScan method is presented.

4.2.1 Basic Components

Consider the truth table of a regular SR-latch in Figure 3.8(a), where the outputs $(Q,\overline{Q})$ are undetermined when (S,R)=(1,1). Figures 3.8(b) and 3.8(c) show the NOR/ NAND gate implementations, respectively, where the outputs for (S,R)=(1,1) are assigned. Figure 4.4(a) shows the K-map of a clocked SR-latch, or polarity-hold SR-latch, where the entries of (M,S,R)=(-,1,1) denotes "-" for the undetermined values. In our implementation, the undetermined values are assigned as shown in the K-map in Figure 4.4(b) and its next state equation is $Y=\overline{M}y+Sy+\overline{R}y+MS\overline{R}$. The realized circuit is referred to as modified SR-latch (or MSR-latch) shown in Figure 4.4(c). Since the logic function of the MSR-latch is prime and irredundant, the circuit is testable for all single stuck-at faults. The important characteristic of this MSR-latch is that it is free of races and hazards in the presence/absence of fault(s).

Consider the cross-coupled NAND latch in Figure 4.5(a), where $(Q,\overline{Q})$ will never be (0,0). This implies that when the inputs, S and R, are connected to the outputs, Q and $\overline{Q}$, of the cross-coupled NAND latch, the entries of (-00) in Figure 4.4(b) should be don't cares, as indicated in Figure 4.5(b). Thus, the logic function becomes $Y=\overline{M}y+Sy+M\overline{R}$ and the corresponding NAND implementation is shown in Figures 4.5(c). For simplicity, the circuit is referred to as *simplified MSR-latch*, or SMSR-latch

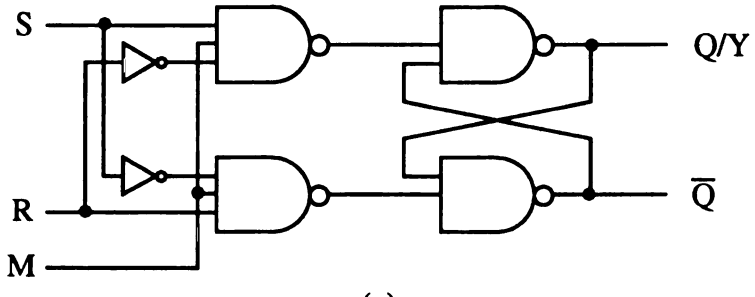
MSR / q	000	001	011	010	110	111	101	100
0	0	0	-	0	1	-	0	0
1	1	1	-	1	1	-	0	1

(a)

MSR / y	000	001	011	010	110	111	101	100
0	0	0	0	0	1	0	0	0
1	1	1	1	1	1	1	0	1

$$Y = \overline{M}y + Sy + \overline{R}y + MS\overline{R}$$

(b)



(c)

Figure 4.4: Clocked SR-latch (a) Truth table; (b) Truth table for modified SR-latch; and (c) NAND gate realization.

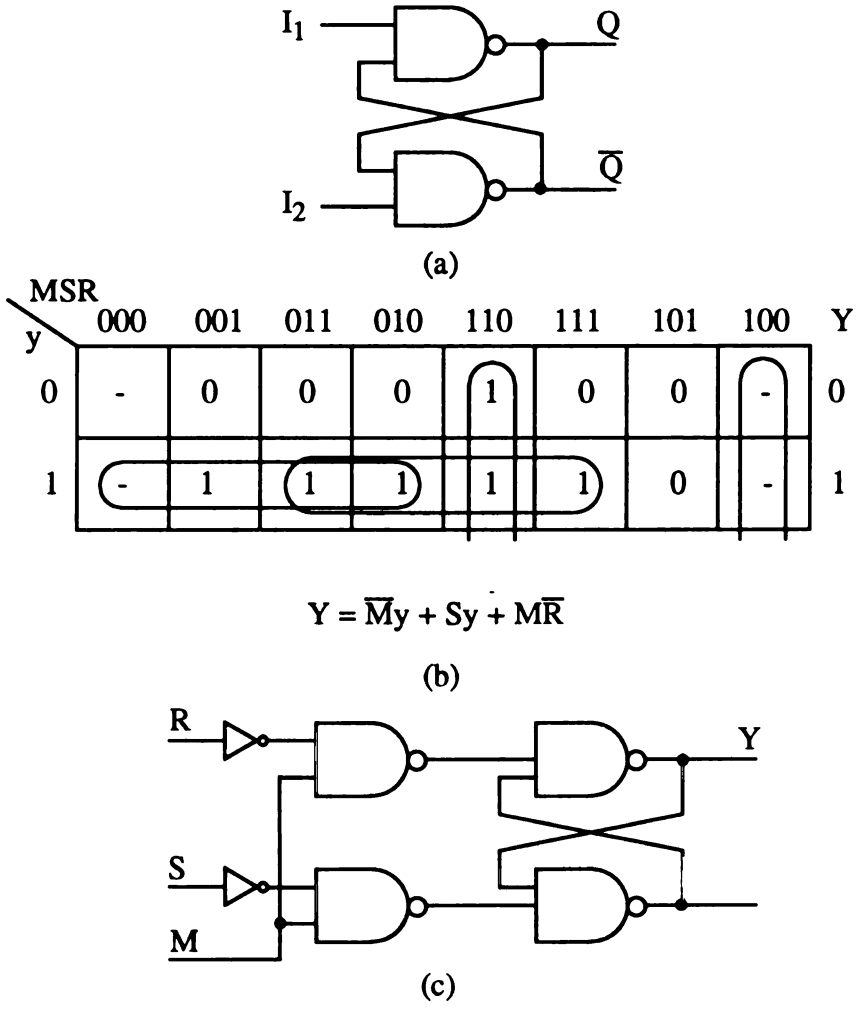


Figure 4.5: (a) Cross-coupled NAND latch; (b) Truth table with (MSR)=(-00) specified as don't cares; and (c) The corresponding NAND gate implementation.

4.2.2 Scan Path

Figure 4.6(a) illustrates two concatenated NAND SMSR-latches. Since the value of the node at "x" is the complement of that at "y", the inverters can be absorbed by the implementation shown in Figure 4.6(b). Note that races may occur in the circuit of Figure

4.6(b). As discussed in Section 2.2.3, the *BFGO test*, consisting of flush test and shift test,

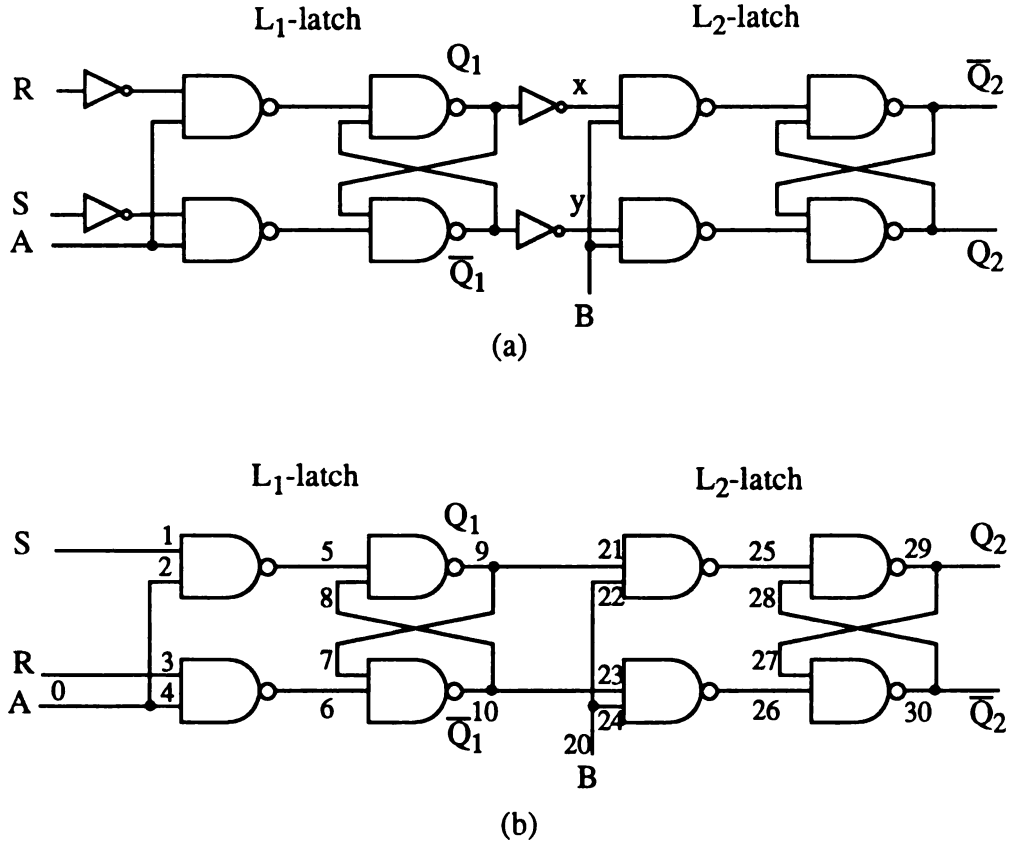


Figure 4.6: (a) Concatenated NAND SMSR-latches; and (b) Simplified implementation of (a).

can be applied for fault detection. As the fault simulation results listed in Table 4.1 show, the BFGO test detects most of faults in that circuit. In Table 4.1, FF means fault-free, a/0 (a/1) means a s/0 (s/1) fault occurs at the node a, and q_i and $\bar{q}_i$, i=1 or 2, represent the previous states of Q_i and $\overline{Q}_i$ in the i-th latch, respectively. However, the BFGO test cannot detect the stuck-at-1 faults at the clock signals. Therefore, a clock test is developed and its fault simulation results are also shown in Table 4.1, where "**" means the signal is ignored as the fault has been detected. Fault simulation results show that the circuit in

Figure 4.6(b) achieves full testability for all single stuck-at faults. However, it should be mentioned that the full testability is achieved based on the assumption that both nodes 29 and 30 are observable. Otherwise, the stuck-at-1 faults at nodes 3 and 23 are not testable, where node 29 has the same value for both fault-free and faulty circuits, as the flush test shown in Table 4.1.

More specifically, in the presence of a stuck-at-1 fault at node 3, a critical race occurs during clock turn off as shown below, where Q_1 and $\overline{Q}_1$ are the outputs of L_1-latch and N5 and N6 represent the nodes 5 and 6, respectively.

S	R	A	N5	N6	Q_1	$\overline{Q}_1$	
1	0	0	1	1	q_1	$\overline{q}_1$	
1	0	1	0	0	1	1	
1	0	0	1	1	?	?	← Critical races

Therefore, it is necessary to have both nodes 29 and 30 be observable.

Figure 4.7(a) illustrates the scan structure in ASLCScan method, where only the first stage needs one inverter, and the inverters in the following stages are absorbed. In addition, two scan-out terminal $+L_2$ and $-L_2$ are used. Thus, the scan path achieves a full testability for all single stuck-at faults at the cost of requiring an extra scan-out terminal, $-L_2$. In practice, if the pin-overhead, extra pins for test purposes, is prohibited, an alternative scan structure, as show in Figure 4.7(b), is suggested, where the latch in Figure 4.2(b) is employed. As mentioned in Section 4.1, there exists a design trade-off between their implementations.

4.2.3 Polarity-Hold Shift Register Latch

Figure 4.8 show the polarity-hold shift register latch (SRL) employed in ASLCScan method. The SRL is comprised of an L_1-latch, an L_2-latch, and two OR gates. It differs from the SRL in Figure 2.11(b) for LSSD in its L_1-latch. For simplicity of this

Table 4.1 Fault Simulation Results for Scan Path

Flush Test

(S,R,A,B) (1,0,1,1)	(S,R,A,B) (0,1,1,1)	Detected Faults
Q_2 $\overline{Q}_2$	Q_2 $\overline{Q}_2$	
1 0	0 1	FF,0/1,2/1,4/1,7/1,8/1,20/1,22/1,24/1,27/1,28/1
1 1	0 1	3/1,6/0,7/0,10/1,23/1,26/0,27/0,30/1
q_1 $\overline{q}_1$	0 1	1/0,2/0,5/1,21/0,22/0,25/1
0 1	0 1	9/0,29/0
1 0	1 0	3/0,4/0,6/1,10/0,23/0,24/0,26/1,30/0
1 0	1 1	1/1,5/0,8/0,9/1,21/1,25/0,28/0,29/1
q_2 $\overline{q}_2$	q_2 $\overline{q}_2$	0/0,20/0

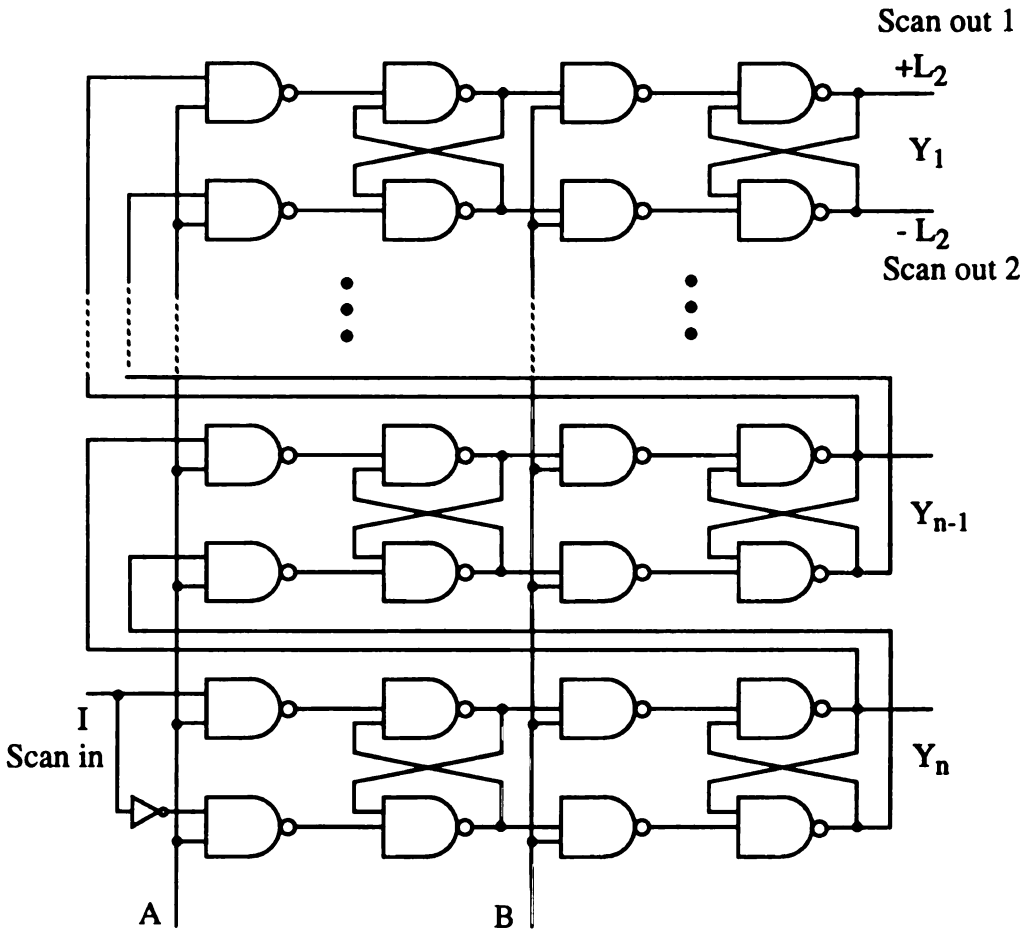
Shift Test

(S,R,A,B) (0,1,1,0)	(S,R,A,B) (0,1,0,1)	(S,R,A,B) (1,0,1,0)	(S,R,A,B) (1,0,0,1)	Detected Faults
Q_2 $\overline{Q}_2$	Q_2 $\overline{Q}_2$	Q_2 $\overline{Q}_2$	Q_2 $\overline{Q}_2$	
0 1	0 1	0 1	1 0	FF,0/1,2/1,4/1,20/1,22/1,24/1
0 1	0 1	0 1	0 1	1/0,2/0,5/1,8/1,21/0,22/0,25/1,28/1
0 1	1 0			7/1
1 0	1 0			27/1

Clock Test

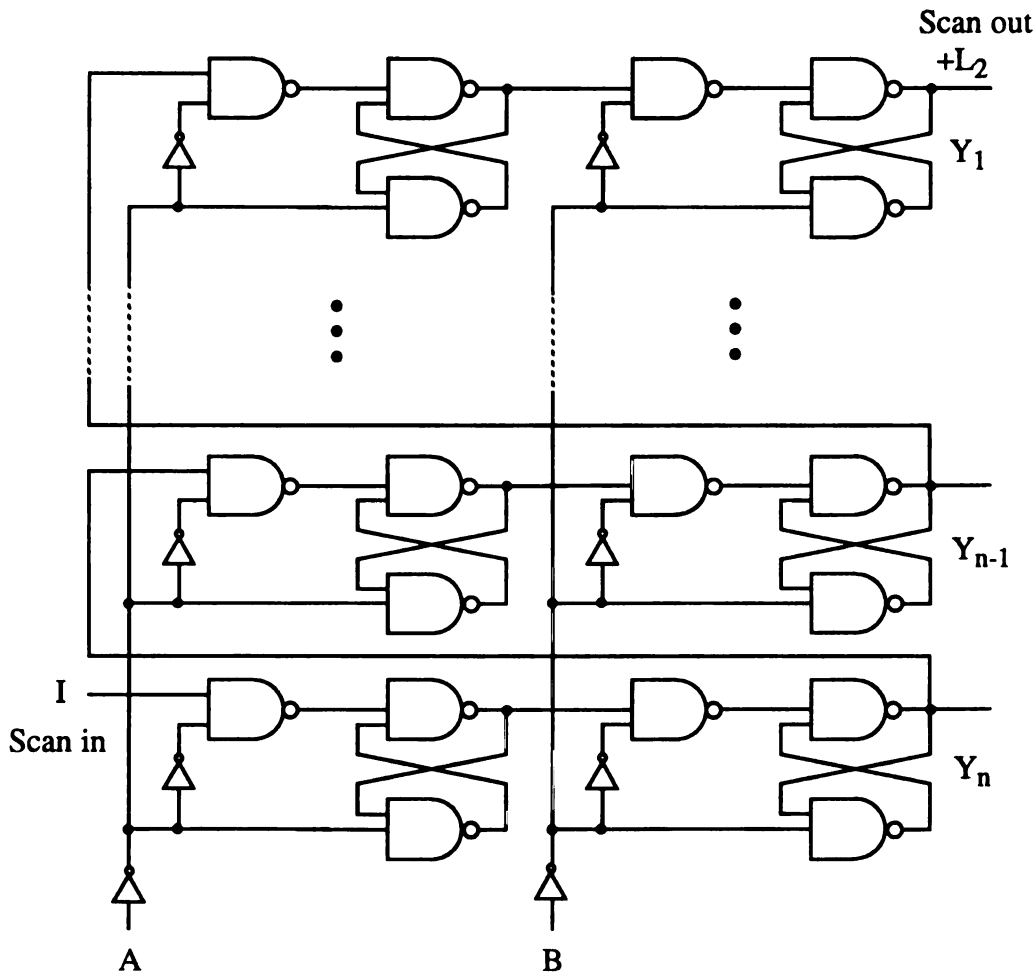
(SRAB)	FF	0/1	2/1	4/1
1010	10	10	10	10
1001	10	10	10	10
0101	10	01	10	01
0110	10	**	10	**
0101	01	**	01	**
1001	01	**	10	**

(SRAB)	FF	20/1	22/1	24/1
1001	10	10	10	10
0110	10	01	10	01
0110	10	01	10	01
0101	01	**	01	**
1010	01	**	10	**
1010	01	**	10	**



(a)

Figure 4.7: (a) ASLCScan scan path; and (b) Alternative scan path.



(b)

Figure 4.7: (cont'd)

discussion, the two pairs of NAND gates in the first stage of the L_1-latch are referred to as the upper pair and the lower pair. The path, referred to as the *normal path*, consisting of the OR gates, the upper pair of NAND gates, the cross-coupled NAND latch, and the L_2-latch, is used for normal operation, while the scan path is comprised of the lower pair of NAND gates, the cross-coupled NAND latch, and the L_2-latch. It has been shown that the scan path is fully testable for all single stuck-at faults. The faults remained undetected so far are the faults in the OR gates and the upper pairs of the NAND gates. Since the MSR-latch is free of hazards and races in the presence/absence of fault(s), it can be tested by the shift test. Also, the flush test and the clock test can detect the remaining faults. However, since the scan design is generally applied for a large circuit, applying the flush test to the normal path is equivalent to the test of a large asynchronous sequential circuit that includes the combinational part and the scan paths for all state variables. The complexity would be considerably high. In order to reduce the complexity, two additional tests are introduced.

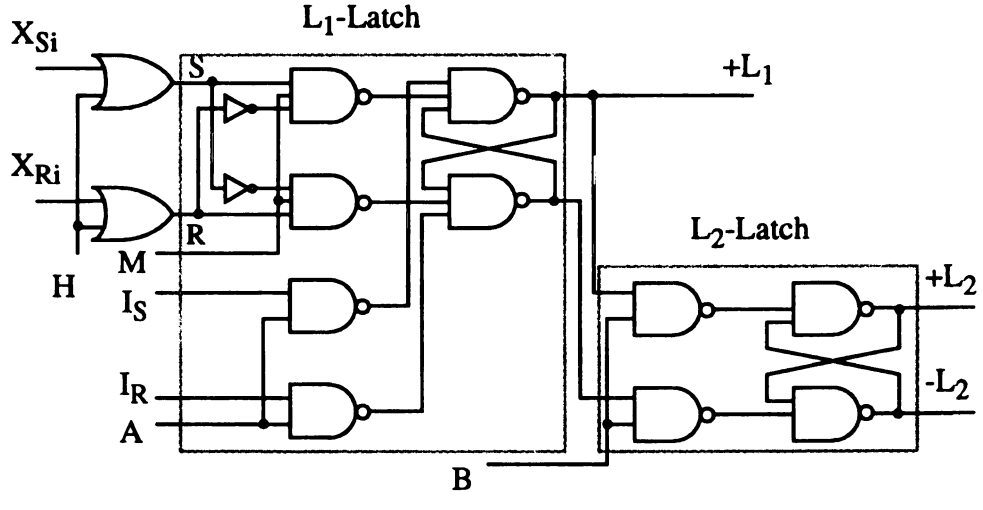


Figure 4.8: Modified polarity-hold SRL.

In Figure 4.8, the two OR gates are added for generating (S,R)=(1,1) to simplify the test process. If the ASLC under test is synthesized such that (S,R)=(1,1) can be obtained, then the two OR gates can be eliminated. In this discussion, we assume that the circuit may not generate such a pattern.

Table 4.2 shows the fault simulation results of the MSR-latch shown in Figure 4.9 with the assigned node numbers. Two tests are introduced: *01-test* and *10-test*. The 01-test is to shift the pattern (01) into the L_1-latch to initialize the contents of Q_1 and $\overline{Q}_1$, i.e., $(Q_1,\overline{Q}_1)=(01)$ and to check the change in these contents for fault detection. More specifically, consider the presence of the 1/0 fault. The 01-test first initializes $(Q_1,\overline{Q}_1)=(01)$ in the L_1-latch. With (S,R)=(1,0), $(Q_1,\overline{Q}_1)$ is unchanged if the fault presents, otherwise, $(Q_1,\overline{Q}_1)$ is changed from (01) to (10). Thus, after shifting the contents, the fault is detected. On the other hand, for Types 2 and 3 faults, with (S,R)=(0,0) or (1,1), $(Q_1,\overline{Q}_1)$ is changed from (01) to (10) if the fault is present, otherwise, $(Q_1,\overline{Q}_1)$ is unchanged. Similarly, the 10-test initializes $(Q_1,\overline{Q}_1)=(1,0)$.

Table 4.2 shows that all single stuck-at faults in the MSR-latch are testable by both the 01-test and the 10-test, where the faults in the cross-coupled NAND gates have been tested during the scan test. However, the full fault coverage is achieved based on the assumptions that (1) both S- and R-inputs can be generated; and (2) the s/1 fault at the clock M has been tested.

Let X_{Si} and X_{Ri} be the outputs of the combinational network for the state variable Y_i. Since both L_1 and L_2 latches are implemented by the MSR-latches and SMSR-latches, respectively, the combinational network N should be synthesized with SR-latches. In general, a irredundant combinational network with SR-implementation can always generate the patterns $(X_{Ri},X_{Si})=(00)$, (01), and (10), but not (11). In order to test Types 3 and 5 faults, two OR gates and an external control signal H are needed as shown in Figure 4.9. The signal X_{Si} and H are ORed to produce the signal S, while R is the output of ORing X_{Ri} and H. Thus, the pattern (11) can be generated by setting H to 1 regardless of

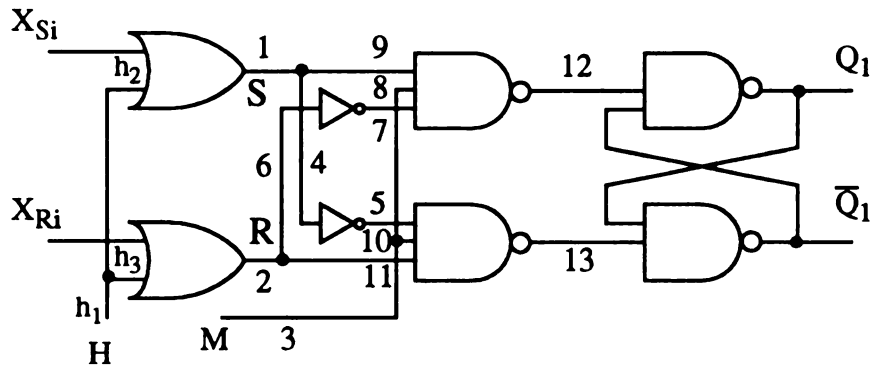


Figure 4.9: Node assignment for the MSR-latch.

Table 4.2 Fault Simulation Results for Modified SR-Latch

01-test

Type	SRM		FF	Faulty	Detected Faults
1	100	101	10	01	1/0,2/1,3/0,6/1,7/0,8/0,9/0,12/1
2	000	001	01	10	1/1,9/1
3	110	111	01	10	2/0,6/0,7/1
	$HX_{Si}X_{Ri}M$		FF	Faulty	Detected Faults
OR-gate	0100	0101	10	01	h1/1,h3/1
	1100	1101	01	10	h1/0,h3/0

10-test

Type	SRM		FF	Faulty	Detected Faults
4	010	011	01	10	4/1,5/0,10/0,11/0,13/1
5	110	111	10	01	4/0,5/1
6	000	001	10	01	11/1
	$HX_{Si}X_{Ri}M$		FF	Faulty	Detected Faults
OR-gate	0010	0011	01	10	h2/1
	1010	1011	10	01	h2/0

the values of the signals X_{Si} and X_{Ri}. As shown in Table 4.2, the OR gates are also testable.

Consider the s/1 fault at the clock M, i.e., s/1 faults at nodes 3, 8, and 10. In this implementation, H=1 and M=0 are set during the test of the scan path so that the combinational network is isolated from the scan path. Since, with H=1 and M=0, the presence of these faults is equivalent to the corresponding input to the NAND gate disappearing, such a fault behavior will not effect the scan test, nor the test of the MSR-latch. Note that, for a given redundant combinational circuit, there always exists a test pattern for each fault. Consider the s/1 faults at nodes 3 and 10. Suppose that a test pattern TP causes the combinational network N to produce $(X_{Si},X_{Ri})=(0,1)$ in Y_i. We first set H=1 and M=0 to initialize $(Q_1,\overline{Q}_1)=(1,0)$ in the L_1-latch for Y_i, and apply the pattern TP, held in L_2-latches, to the combinational network N. Secondly, H is reset to 0, where M=0, we expect to obtain $(X_{Si},X_{Ri})=(0,1)$. This results in changing the contents of $(Q_1,\overline{Q}_1)$ from the initialized values (1,0) to (0,1) if such faults are present, otherwise, no change is made in these contents. Thus, fault behavior can be distinguished. Finally, H is set to 1 to protect the data held in L_1-latch so that the data can be reliably shifted out for fault detection. Similarly, the s/1 fault at node 8 is detected in the same manner, where a test pattern that produces $(X_{Si},X_{Ri})=(1,0)$ at Y_i is applied and the L_1-latch is initialized as $(Q_1,\overline{Q}_1)=(0,1)$.

Let $(Q_j,\overline{Q}_j)_i$, j=1,2, and i=1,2,..,n, denote the data held in L_j-latch for Y_i. Since the shift signals A and B are used to synchronize the shift operation, $(Q_1,\overline{Q}_1)_i$ is either the same as $(Q_2,\overline{Q}_2)_i$ or $(Q_2,\overline{Q}_2)_{i+1}$, or both, but cannot differ from both $(Q_2,\overline{Q}_2)_i$ and $(Q_2,\overline{Q}_2)_{i+1}$ at the same time. If the test patterns in Table 4.2 require that $(Q_1,\overline{Q}_1)_i$ should be different from both $(Q_2,\overline{Q}_2)_i$ and $(Q_2,\overline{Q}_2)_{i+1}$ at the same time, such a problem can be resolved by either rearranging the scan path so that the SRL for Y_i will not follow that for Y_{i+1}, i.e., $SRL_{i+2}\rightarrow SRL_i\rightarrow SRL_{i+1}\rightarrow SRL_{i-1}$, where SRL_i denotes the SRL for Y_i, or adding an dummy SRL between SRL_i and SRL_{i+1} in the scan path, if necessary.

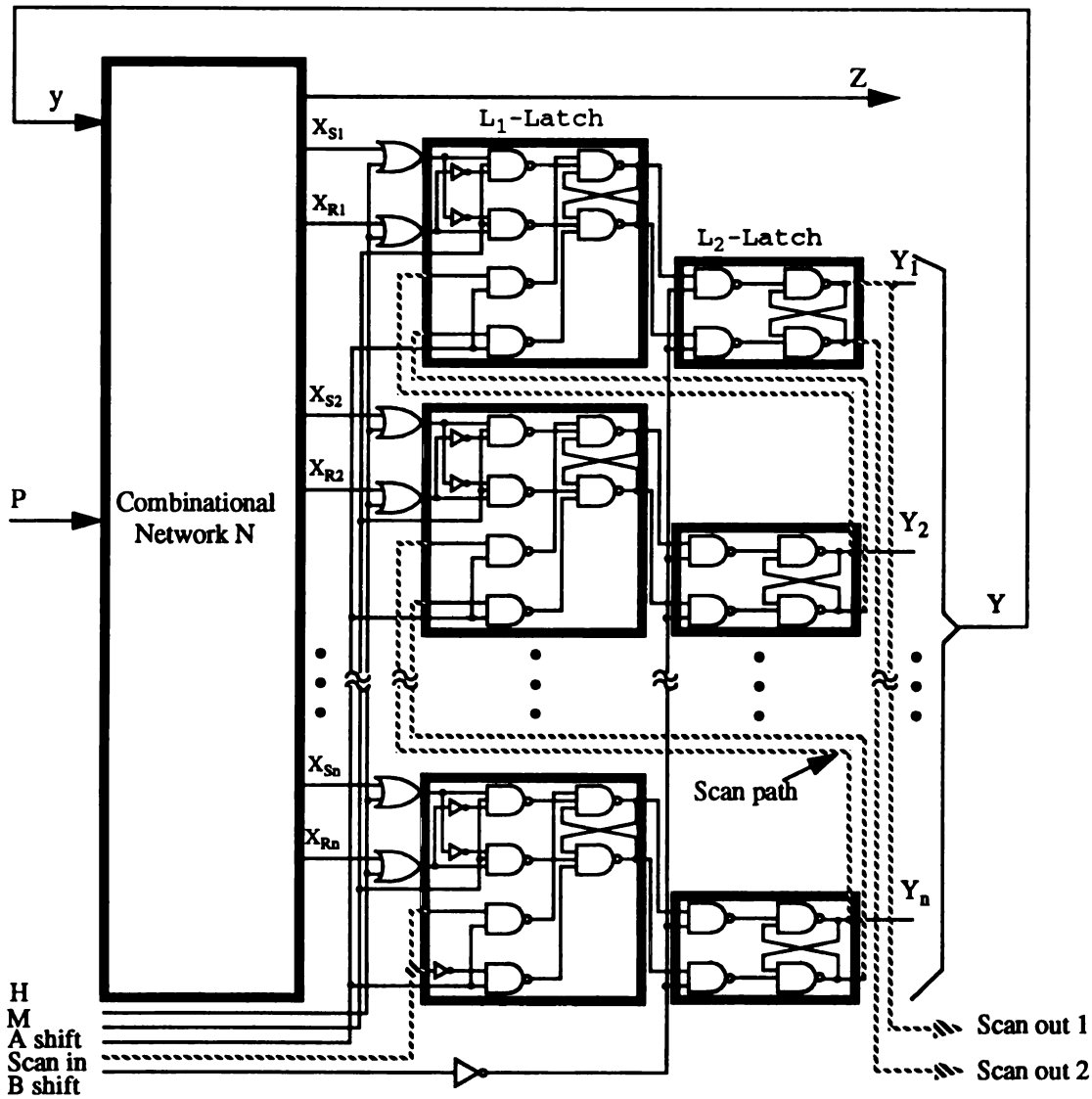


Figure 4.10: Double-latch ASLCScan design.

The double-latch ASLCScan design is shown in Figure 4.10 which generates no critical races during normal operation and test modes.

4.2.4 Operation Modes

The design with ASLCScan method consists of two operation modes: *test mode* and *normal operation mode*. In the test mode, the operation is synchronized in the way

similar to the conventional LSSD [20,35] which can be divided into the following steps.

Step 1: Apply flush test, shift test, and clock test to verify the scan path.

Step 2: Apply 01-test and 10-test for the MSR-latch.

Step 3: Scan in the test vector for internal state variables via scan input pin by applying pulses alternately to two non-overlapping clock A and B.

Step 4: Set the corresponding test pattern on primary inputs.

Step 5: Wait for the worst case delay for the combinational network to become stable and check the values of primary outputs.

Step 6: Apply one clock pulse to M to latch the new internal state variable into the corresponding L_1-latch.

Step 7: Scan out the internal state variable by applying non-overlapping clock signals A and B.

In the normal mode operation, the ASLC is operated in an asynchronous way by setting the clock signals A=0 and B=M=1.

4.3 Examples

In order to demonstrate the circuit design with ASLCScan method, two examples are presented in this section. The first example is a STG-modeled ASLC, while the second example is a Huffman-modeled ASLC.

Example 1: A STG-modeled ASLC

Consider a STG-modeled ASLC, converta.g, in *sis* benchmarks, where its functional behavior is described by a signal transition graph shown in Figure 4.11(a). Using the automated synthesis tool, *astg*, in sis developed by University of California at Berkeley, the synthesized ASLC is shown in Figure 4.11(b), where the C-element is similar to the MSR-latch. For simplicity, the combinational part in Figure 4.11(b) is rearranged as shown in Figure 4.11(c). With the implementation of ASLCScan method, each C-element is replaced by a SRL. Thus, three SRLs are needed in this implementation.

The detailed connection is similar to that in Figure 4.10. ♦

Example 2: A Huffman-modeled ASLC

Consider a Huffman-modeled ASLC, P-SLF, whose functional behavior is described in Figure 4.12(a) [25], where A and B are its inputs and D and E are its outputs. (The upper-case and lower-case letters denote the next state and the present state, respectively.) The output lines change state only on the falling edge of B. Using the automated synthesis system, MSUASLC, the flow tables and next-state and output equations are generated as shown in Figures 4.12(b) and 4.12(c). With the two-level logic implementation, the combinational part and the SR-latches are also shown in Figure 4.12(c). With the implementation of ASLCScan method, each SR-latch is replaced by a SRL. Thus, this implementation requires three SRLs.

4.4 Discussion

Test generation is much more difficult for ASLCs than for CSLCs due to races, hazards, and state oscillations. This chapter presents a scan design, ASLCScan method, using SR-latches for ASLCs. Results show that the scan structure including the normal path and scan path are fully testable for all single stuck-at faults without causing any critical races. The ASLCScan method provides the following salient features: (1) the test generation problem is reduced to one of just testing the combinational logic; (2) the faults that are very difficult to test due to critical races, as discussed in Chapter 3, can be easily identified by checking the scanned data; (3) the use of SR-latches in the scan structure is perfectly implemented to the synthesized STG-modeled ASLCs and Huffman-modeled ASLCs; and (4) the L_2-latch is used to shift out the scanned data during the test mode and it may also be used as a delay element to eliminate possible essential hazards.

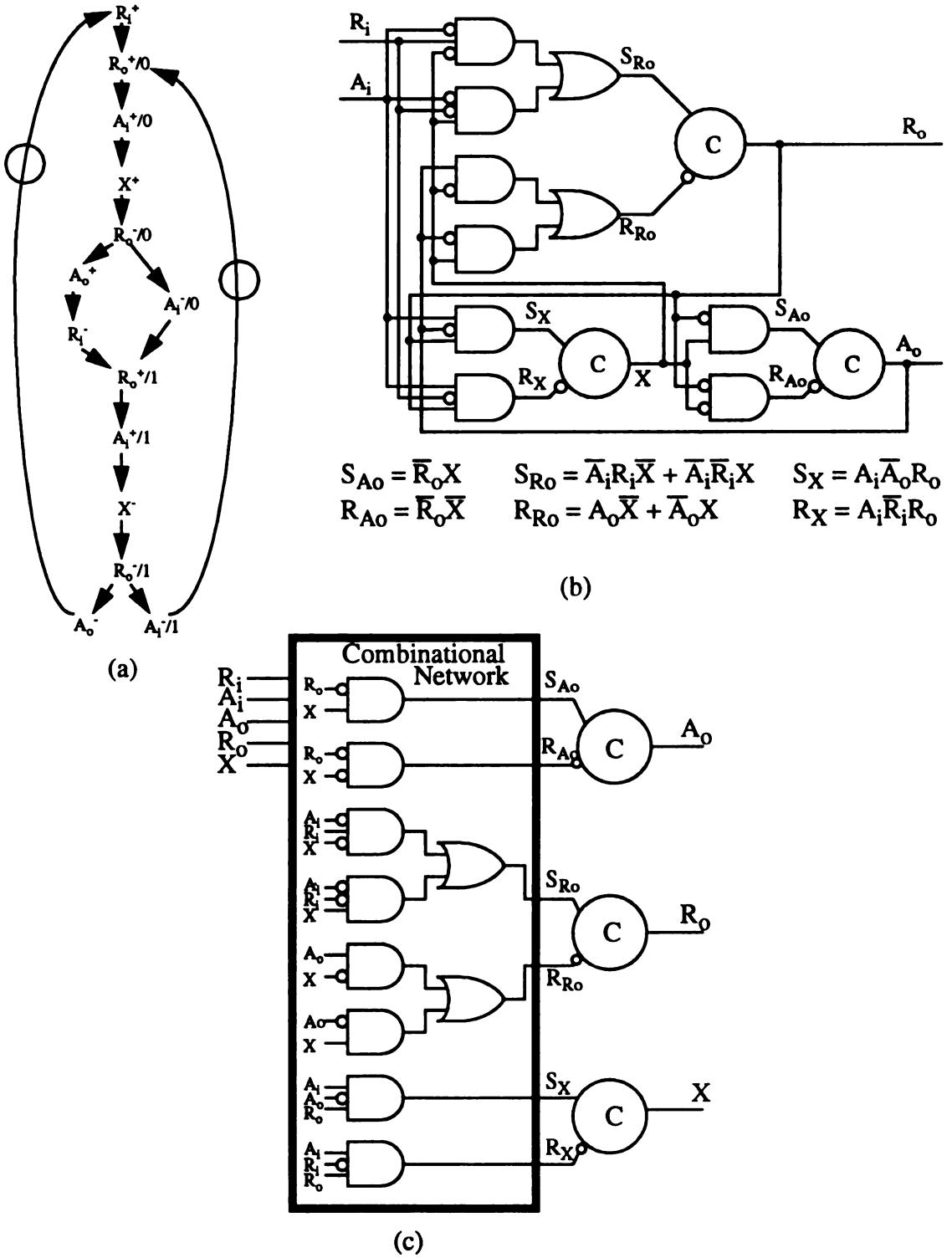


Figure 4.11 Example 1 (a) STG representation; (b) Logic implementation; and (c) Rearrange the combinational part and C-elements.

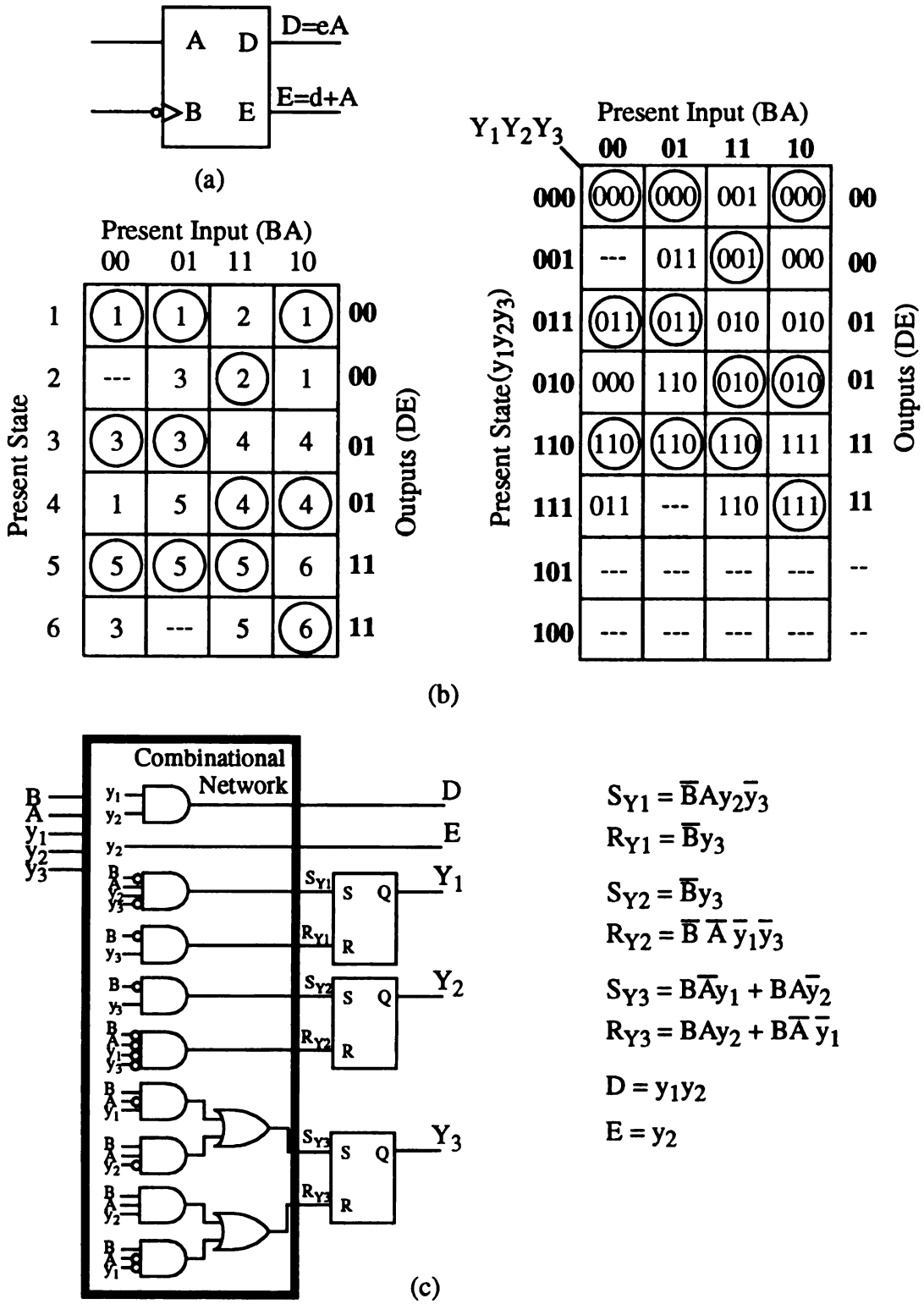


Figure 4.12 Example 2 (a) Behavior description; and (b) Flow table and excitation table; and (c) Logic implementation.

Chapter 5

Conclusions

Test generation is much more difficult for ASLCs than for CSLCs because of races, hazards, and state oscillations. Often a designer does not know whether or not all transitions in a circuit are free of races and hazards. The situation is even more uncertain if a fault is present. Even though fault-free circuits can be designed to be free of races, hazards, and state oscillations, they may occur as a result of faults. And even though the fault-free circuit can be designed to change one state variable at a time during a state transition to avoid races, the condition cannot be guaranteed in the presence of faults. Based on the single stuck-at fault model, two major tasks were expected to be achieved in this study: (1) to investigate the fault effects and synthesis properties of ASLCs; and (2) to develop a scan design for ASLCs.

This chapter summarizes this research work and outlines the major contributions. Finally, several interesting problems based on this research are identified for future research.

5.1 Summary

Fault effects such as redundant faults and state oscillations for both Huffman-modeled and STG-modeled ASLCs have been presented in Chapter 3. Redundancy is sometimes desirable for hazard protections in ASLCs. However, it may also be introduced

unintentionally due in part to the implementation of an improper synthesis procedure, such as improperly encoding state variables or assigning the don't care terms. In Huffman-modeled ASLCs, this study has shown that the STT state assignment does not create cycles for avoiding critical races, but it implicitly provides multiple transition paths for the circuit to stabilize at the same state. As a result, the STT-generated ASLCs inherently generate the equivalent-state redundant faults. On the other hand, since the race-free UDC state assignment provides only one connected path for each state transition, the possible equivalent-state redundant faults can be reduced.

Due to the use of clock signals, state oscillations do not occur in CSLCs. However, they may occur in ASLCs due to hazards and critical races. The state oscillations are generally avoided by eliminating critical races. However, they may occur as a result of faults. In Chapter 3, some rules and properties for synthesizing state oscillation-free Huffman-modeled ASLCs have been presented. Since any fault-free UDC-generated ASLC is race-free, the race conditions that occur in the presence of faults can be easily identified by tracing the corresponding connected paths in the transition table. Thus, this study concludes that UDC-generated ASLCs are better than STT-generated ASLCs as far as testable ASLCs are concerned. Based on the observed fault effects, numerous synthesis properties have also been presented in Chapter 3 for Huffman-modeled ASLCs with or without SR-latches. These properties can be implemented to reduce the complexity of the test generation process.

A STG-modeled ASLC without input/output concurrency can be viewed as a special Huffman-modeled ASLC. A STG-modeled ASLC with output concurrency is equivalent to a Huffman-modeled ASLC with the STT state assignment; a STG-modeled ASLC with output variables which are changed one at a time is equivalent to the Huffman-modeled ASLC with the UDC state assignment; and a STG-modeled ASLC with input concurrency is equivalent to a Huffman-modeled ASLC with multiple input changes. Since the state variables in a STG-modeled circuit are also its output variables which are

generally observable, the testing of STG-modeled ASLCs without having internal variables is much easier than Huffman-modeled ASLCs. However, if a STG-modeled ASLC has some internal variables, then its fault effects are similar to those in Huffman-modeled circuits.

The input/output concurrency is a special feature in STG which resolves the fundamental mode problem in Huffman model. Due to such salient feature, asynchronous control circuits have been successfully designed and implemented using STGs. However, with the input/output concurrency, a test input that must be applied at the same time when the output changes is obvious not an easy task. The results show that the faults due to the input/output concurrency cannot be tested without scan structure.

Due to the lack of controllability and observability of state variables, testing of sequential circuits has been recognized as a very difficult task. Chapter 4 presents the development of the ASLCScan method, a scan design for ASLCs. The scan structure is free of races and hazards in both normal operation and test modes, and achieves full testability for all single stuck-at faults. With the ASLCScan design, the test generation problem of ASLCs is reduced to one of just testing the combinational logic and the introduced races in the combinational circuit of ASLC can be identified easily. In addition, the invalid states and entries in an assigned flow table are generally unreachable if the circuit is fault-free. The fault behavior is unpredictable when these invalid states are reached due to the fault. Such a problem can be easily resolved by using the ASLCScan design.

Since SR-latches or C-elements are commonly used in STG-modeled ASLCs and Huffman-modeled ASLCs, they also can be used as part of the L_1-latch to reduce the hardware overhead. On the other hand, the propagation delay through the L_2-latch can be treated as a delay element during normal operation. The delay value for the L_2-latch may be used to avoid the occurrence of hazards, especially for essential hazards.

5.2 Contributions

The major contributions and impacts of this study can be summarized as follows:

(1) explore fault effects in ASLCs;

(2) identify the similarities and differences between Huffman-modeled and STG-modeled ASLCs;

(3) derive testability synthesis rules and properties; and

(4) develop a first-ever scan design for ASLCs.

This research has explored the fault effects in both Huffman-modeled ASLCs and STG-modeled ASLCs. Based on the single stuck-at fault model, redundant faults and state oscillations have been identified in ASLCs. The ways of identifying and eliminating redundant faults have been presented in Chapter 3, while a set of synthesis rules that generate state oscillation-free ASLCs is also derived. Thus, the test generation process can be simplified considerably and the fault coverage can be increased significantly. The developed scan design, ASLCScan method, further reduces the complexity of test problem. As the examples presented in Chapter 4 have shown, testable design of ASLCs can be achieved based on the developed scan design. In addition, this study has concluded that, for Huffman-modeled ASLCs, the UDC state assignment is better than the STT state assignment as far as testing is concerned. Also, the similarities between the Huffman-modeled ASLCs and STG-modeled ASLCs are identified. Thus, the design, synthesis, and test methodology developed for one model may be implemented for another model.

5.3 Future Research

This research has investigated the fault effects for STG-modeled ASLCs and Huffman-modeled ASLCs. The fault effects are explored based on the single stuck-at fault model. The fault model has identified the fault effects such as redundant faults and state oscillations. Delay faults have been a research topic in CSLCs recently. It would be worth

investigating the fault effects for ASLCs with delay faults.

The fault effects and synthesis properties studied in this research are based on the two-level logic implementation. Due to delay hazards, decomposing two-level logic to multi-level logic is not straightforward. It is believed that the fault effects and synthesis properties may have a significant difference when multi-level logic implementation is concerned. This would be a very interesting problem for future investigation.

To achieve a fully testable ASLC design is not an easy task without using scan structures. This thesis presents a scan design, ASLCScan method, for ASLCs to reduce the complexity of the testing problem. Similar to LSSD in CSLCs, the hardware overhead and speed degradation have always been the critical issues. In ASLCScan method, the SR-latches or C-elements in ASLCs have been used as part of the SRLs in the scan structure to reduce hardware overhead and performance degradation. Further, with a proper design procedure, the L_2-latches may be used as delay elements for avoiding hazards. Thus, it is desirable to develop a design procedure which fully exploits the L_2-latches as the delay elements to simplify the synthesized ASLCs. In addition, in CSLCs, partial scan becomes an effective way to enhance testability while still keeping lower hardware overhead and performance degradation. Based on the fault effects, not all the state variables have to be observed. Only those state variables which may cause critical races condition during testing, need to be observed. Thus, it is necessary to develop a fault simulation tool that can identify the necessarily observed state variables.

LIST OF REFERENCES

LIST OF REFERENCES

[1] T. H.-Y. Meng, R. W. Broderson, and D. G. Messerchmitt, "Automatic Synthesis of Asynchronous Circuits from High-Level Specifications," *IEEE Trans. on Computer-aided Design*, vol. 8, pp. 1185-1205, November 1989.

[2] D. A. Huffman, "The Synthesis of Sequential Switching Circuits," *J. Franklin Institute*, vol. 257, pp. 161-190, March 1954.

[3] S. H. Unger., *Asynchronous Sequential Switching Circuits*, Wiley Interscience, 1969.

[4] R. E. Miller., *Switching Theory, volume II: Sequential Circuits and Machines*, New York, Wiley, 1965.

[5] T. A. Chu, "Synthesis of Self-timed Control Circuits from Graphs: An Example," Proc. IEEE ICCD, pp. 565-571, October 1986.

[6] T. A. Chu, Synthesis of Self-timed VLSI Circuits from Graph-Theoretic Specifications, Ph.D. dissertation, MIT, June 1987.

[7] T. A. Chu, "On the Models for Designing VLSI Asynchronous Digital Systems," *INTEGRATION, the VLSI journal 4*, North-Holland, pp. 99-113, 1986.

[8] G. K. Maki and D. H Sawin, "Real-Time Fault Detection and Fault-Tolerant Implementations for Sequential Circuits," in *Rational Fault Analysis*, ed. by R. Saeks and S.R. Liberty, pp.32-51. Marcel Dekker, 1977.

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

[10] H.-K. T. Ma, S. Devadas, A. R. Newton, and A. Sangiovanni-Vincentelli, "Test Generation for Sequential Circuits," *IEEE Trans. on Computer-aided Design*, vol. 7, pp. 1081-1093, October 1988.

[11] S. Devadas, H.-K.T. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli, "A Synthesis and Optimization Procedure for Fully and Easily Testable Sequential Machines," *IEEE Trans. on Computer-aided Design*, vol. 8, pp. 1100-1107, October 1989.

[12] S. Devadas, H.-K. T. Ma, and A.R. Newton, "Redundancies and Don't Cares in Sequential Logic Synthesis," IEEE International Test Conference, pp. 491-500, 1989.

[13] S. Devadas, H.-K. T. Ma, A. R. Newton, and A. Sangiovanni-Vincelli, "Irredundant Sequential Machines via Optimal Logic Synthesis," *IEEE Trans. on Computer-aided Design*, vol. 9, pp. 8-17, January 1990.

[14] S. Devadas and K. Keutzer, "A Unified Approach to the Synthesis of Fully Testable Sequential Machines," *IEEE Trans. on Computer-aided Design*, vol. 10, pp. 39-50, January 1991.

[15] E. B. Eichelberger, "Hazard Detection in Combinational and Sequential Switching Circuits," *IBM Journal of Research and Development*, pp. 90-99, March 1965.

[16] G. R. Putzolu and J. P. Roth, "A Heuristic Algorithm for the Testing of Asynchronous Circuits," *IEEE Trans. on Computers*, vol. C-20, pp. 639-647, June 1971.

[17] M. A. Breuer, "The Effects of Races, Delays, and Delay Faults on Test Generation," *IEEE Trans. on Computers*, vol. C-23, pp. 1078-1092, October 1974.

[18] K. Keutzer, L. Lavagno, and A. Sangiovanni-Vincentelli, "Synthesis for Testability Techniques for Asynchronous Circuit," IEEE ICCAD Digest of Technical Paper, pp. 326-329, 1991.

[19] P. A. Beerel and T. H.-Y. Meng, "Testability of Asynchronous Timed Control Circuits," 28th ACM/IEEE Design Automation Conference, pp. 446-451, 1991.

[20] E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability," 14th ACM/IEEE Design Automation Conference, pp.462-468, 1977.

[21] T. W. Williams and K. P. Parker, "Design for Testability - A Survey," *IEEE Trans. on Computers*, vol. C-31, pp. 2-15, January 1982.

[22] T. W. Williams and J. B. Angell, "Enhancing Testability of Large Scale Integrated Circuits with Test Points and Additional Logic," *IEEE Trans. on Computers*, vol. C-22, pp. 46-60, January 1983.

[23] S. DasGupta, P. Goel, R. G. Walther, and T. W. Williams, "A Variation of LSSD and its Implications on Design and Test Pattern Generation in VLSI," IEEE International Test Conference, pp. 63-66, 1982.

[24] K. K. Saluja, "An Enhancement of LSSD to Reduce Test Pattern Generation Effect and Increase Fault Coverage," 19th ACM/IEEE Design Automation Conference, pp. 489-494, 1982.

[25] S.-F. Wu and P. D. Fisher, "Automating the Design of Asynchronous Sequential Logic Circuits," *IEEE Journal of Solid-State Circuits*, pp. 364-370, March 1991.

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

[27] B. Hazeltine, "Encoding of Asynchronous Sequential Circuits," *IEEE Trans. on Computers*, vol. EC-14, pp. 727-729, October 1965.

[28] O. G. Langdon, Jr, "Delay-Free Asynchronous Circuits with Constraint Line Delays," *IEEE Trans. on Computers*, pp. 175-181, February 1969.

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

[30] J. H. Tracy, "Internal State Assignments for Asynchronous Sequential Machines," *IEEE Trans. on Electronic Computers*, vol. EC-15, pp. 551-560, August 1966.

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

[32] C. J. Tan, "State Assignments for Asynchronous Sequential Machines," *IEEE Trans. on Computers*, vol. C-20, pp. 382-391, April 1971.

[33] R. J. Smith, "Generation of Internal State Assignment for Large Asynchronous Sequential Machines," *IEEE Trans. on Computers*, vol. C-23, pp. 924-932, September 1974.

[34] D. H. Sawin, III and G. K. Maki, "Asynchronous Sequential Machines Designed for Fault Detection," *IEEE Trans. on Computers*, vol. C-23, pp. 239-248, March 1974.

[35] E. J. McCluskey, *Logic Design Principles with Emphasis on Testable Semicustom Circuit*, Prentice-Hall, New Jersey, 1986.

[36] S. H. Unger, "Hazards and Delays in Asynchronous Sequential Switching Circuits," *IRE Trans. on Circuit Theory*, pp. 12-25, March 1959.

[37] D. B. Armstrong, A. D. Friedman, and P. R. Menon, "Realization of Asynchronous Sequential Circuits Without Inserted Delay Elements," *IEEE Trans. on Computers,* vol. C-17, pp. 129-134, February 1968.

[38] L. Lavagno, K. Keutzer, and A. Sangionvanni-Vincentelli, "Synthesis of Verifiably Hazard-free Asynchronous Control Circuits," Advanced Research in VLSI Conference, pp. 87-102, May 1991.

[39] L. Lavagno, K. Keutzer, and A. Sangionvanni-Vincentelli, "Algorithms for Synthesis of Hazard-free Asynchronous Circuits," 28th ACM/IEEE Design Automation Conference, pp. 302-308, 1991.

[40] K.-J. Lin and C.-S. Lin, "Automatic Synthesis of Asynchronous Circuits," 28th ACM/IEEE Design Automation Conference, pp. 296-301, 1991.

[41] M.-L. Yu and P. A. Subrahmanyam, "A Path-Oriented Approach for Reducing Hazards in Asynchronous Design," 29th ACM/IEEE Design Automation Conference, pp. 239-244, 1992.

[42] D. B. Armstrong, A. D. Friedman, and P. R. Menon, "Design of Asynchronous Circuit Assuming Unbounded Gate Delays," *IEEE Trans. on Computers*, vol. C-18, pp. 1110-1120, December 1969.

[43] C. L. Seitz, "System Timing," in *Introduction to VLSI System*, C. Mead and L. Conway, Eds. Reading, MA. Addison-Wesley, 1980.

[44] A. S. Wojcik and K.-Y. Fang, "On the Design of Three-Valued Asynchronous Modules," *IEEE Trans. on Computers*, vol. C-29, pp. 889-898, October 1980.

[45] I. David, R. Ginosar, and M.Yoeli, "Implementing Sequential Machines as Self-Timed Circuits," *IEEE Trans. on Computers*, vol. 41, pp. 12-17, January 1992.

[46] B. Gilchrist, et. al., "Fast Carry Logic for Digital Computers," *IRE Trans. on Electrical Computers*, vol. EC-4, pp. 133-136, December 1955.

[47] A. D. Friedman and P. R. Menon, "Synthesis of Asynchronous Sequential Circuits with Multiple-Inputs Changes," *IEEE Trans. on Computers*, vol. C-17, pp. 559-565, June 1968.

[48] M.-D. Shieh, C.-L. Wey, and P. D. Fisher, "Model of Asynchronous Finite State Machines and Their Pipelined Structure," 35th Midwest Symposium on Circuits and Systems, pp. 659-662, August 1992.

[49] V. I. Varshavsky, *Self-Timed Control of Concurrent Processes*, Kluwer Academic Publisher, 1990.

[50] B. W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley Publisher, 1989.

[51] H. Fujiwara, *Logic Testing and Design for Testability*, Computer System Series, 1985.

[52] F. F. Tsui, *LSI/VLSI Testability Design*, McGraw-Hill Publisher, 1988.

[53] E. B. Eichelberger, "Latch Design Using Level Sensitive Scan Design," Dig. COMPCON, Spring 83, San Francisco, pp.395-398, 1983.

[54] P. S. Bottorff, R. E. Garges, and E. J. Orosz, "Test Generation for Large Logic Network," 14th ACM/IEEE Design Automation Conference, pp. 479-485, 1977.

[55] H. Ando, "Testing VLSI with Random Access Scan," Dig. COMPCON, pp.50-52, February, 1980.

DATE DUE	DATE DUE	DATE DUE
		JUL 24 1994