1999

Testing asynchronous logic circuits from transistor networks to gate-level designs.

Kaamran. Raahemifar
University of Windsor

Recommended Citation

Raahemifar, Kaamran., "Testing asynchronous logic circuits from transistor networks to gate-level designs." (1999). *Electronic Theses and Dissertations.* Paper 3735.

Testing Asynchronous Logic Circuits from Transistor Networks to Gate-Level Designs

by

Kaamran Raahemifar

A Dissertation
presented to the College of Graduate Studies and Research
through Electrical Engineering
in partial fulfilment of the
requirements for the degree of
Doctor of Philosophy
at
the University of Windsor

Windsor, Ontario, Canada, 1999

0-612-52440-X

I hereby declare that I am the sole author of this thesis.

I authorize the University of Windsor to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Windsor to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Windsor requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

This dissertation is concerned with testing of asynchronous circuits. Asynchronous circuits are attracting increasing interest for future generations of high-speed low-power logic circuits because they facilitate concurrent computation, offer average-case performance and better technology migration potential, and eliminate clock skew.

The research reported in this dissertation is a comprehensive study of testing asynchronous circuits using design-for-testability (DFT) techniques and test generation algorithms.

In the first part of the study we propose an on-line DFT technique for detecting short defects (or I_{DDQ} faults), which create a low-resistance path between the supply lines. It is shown that I_{DDQ} testing, delay testing, and stuck-open testing are necessary in order to achieve a high defect coverage. The second DFT technique presented in this part is a novel circuit for concurrently detecting delay faults and stuck-open faults. In the proposed DFT techniques, in particular, fault detection in CMOS logic family is investigated.

The second half of this study attempts to derive test sequences for sequential circuits. First, initialization phase is studied. Initialization is the process of driving the state signals in the circuit to known states. This dissertation presents an initialization technique for non-initializable asynchronous sequential circuits.

Finally, we proceed by generating test sequences for asynchronous sequential circuits. We assume the presence of all multiple faults of all multiplicities. No faulty machines are generated during these procedures and we do not resort to their explicit enumeration.

Acknowledgements

I would like to express my thanks and gratitude to my supervisor, Professor M. Ahmadi. This work would not have been possible without his continuous support and encouragement.

I am thankful to the members of my defense committee, Professors Michael Miller, William C. Miller, Reza Lashkari, and J. J. Soltis for their valuable comments. I am grateful to Prof. Graham. A. Jullien for his constructive questions and comments. I would also like to thank all my colleagues and friends for their support and useful discussions, in particular, S. Hessabi, M. Shams, and S. Sadeghi-Emamchaie for their valuable consultations, and Phil Regier for his kind assisstance.

Finally, and most of all, I am thankful to my wife for her constant provision of love, friendship, and moral support from which I drew the strength to persevere with this project.

Dedications

To Saeedeh, my wife, and Keyan, my son, who - more than I - endured the hardships of this work.

Contents

List of Tables

List of Figures

Glossary

ALU	*Arithmetic Logic Unit*
ATGP	*Automatic Test Generation Program*
ATPG	*Automatic Test Pattern Generation*
BIST	*Built-In Self-Test*
BMT	*Behavioral Model for Testability*
BiCMOS	*Bipolar CMOS*
CCF	*Combinational Circuit with Forks*
CCVS	*Current-Controlled Voltage-Source*
CLR	*Clear*
CMOS	*Complementary MOS*
CSCD	*Current Sensing Completion Detection*
CSP	*Communicating Sequential Processes*
CTS	*Complete Test Set (Sequence)*
CUT	*Circuit Under Test*
DCC	*Digital Compact Cassette*
DFT	*Design-For-Testability*
DI	*Delay Insensitive*
FF	*Flip-Flop*
FFCC	*Fork-Free Combinational Circuit*
FIFO	*First-In First-Out*

FSM	*Finite State Machine*
GND	*Ground*
IC	*Integrated Circuit*
KCL	*Kirchoff Voltage Law*
KVL	*Kirchoff Current Law*
MUX	*Multiplexer*
NP	*Non-linear Programming*
PCB	*Printed Circuit Board*
PR	*Preset*
PUPD	*Pull-Up Pull-Down*
QDI	*Quasi Delay Insensitive*
ROM	*Read Only Memory*
SI	*Speed Independent*
STG	*Signal Transition Graph*
TTL	*Transistor-Transistol Logic*
UIN	*Up-bounded Inertial Delay*
UUT	*Unit Under Test*
VLSI	*Very-Large Scale Integrated circuit*

Chapter 1

Introduction

The VLSI community has witnessed a renewed and growing interest in asynchronous circuit design and testing during the last decade. Asynchronous circuits are attracting increasing interest for future generations of high-speed low-power logic circuits because they facilitate concurrent computation, offer average-case performance and better technology migration potential, and eliminate clock skew[1].

The potential advantages of asynchronous circuits over their synchronous counterparts are:

1. The power dissipation and the area associated with clock routing has disappeared due to the absence of a global clock in asynchronous circuits. The ac power dissipation on latches is reduced since only those parts of the circuit

[1] The variation in the effective arrival time of the clock at different clocked storage elements is called clock skew [165]. These variations may be due to a combination of several effects: different threshold voltages, signal propagation delays on wires, or variation in element delays such as is found when gated clock signals are used to control register loading.

that are actually performing useful work will be active: Internal node capacitors of other latches would not charge or discharge for each controlling signal transition.

2. Clock skew is eliminated since there is no global clock.

3. Some asynchronous systems have higher modularity. Since there are no global timing requirements, functional blocks may be replaced individually without circuit modification provided that the correct operation of the entire circuit does not depend on the combinations of delays in the new block.

4. Designing asynchronous circuits improves *average* speed performance, since many asynchronous circuits sense when a computation is ended. Therefore, the speed of the entire circuit no longer needs be bounded to the slowest block or the rate of a global signal. In contrast to asynchronous systems, synchronous circuits must wait until *all* possible computations have been completed before latching the results; this yields in worst-case performance.

5. Most asynchronous circuits promise better handling of metastability: a state of unstable equilibrium in which a circuit can remain for an unbounded amount of time. An example of a component where metastability may occur is an arbiter. In a synchronous system a metastable state may last longer than one clock cycle. In this case, incorrect data is sampled and transferred to the next stage, which in turn, might lead to an incorrect output. However, most asynchronous circuits can wait an arbitrarily long time until the circuit element leaves the metastable state.

In spite of their numerous attractive features and the significant work that has been carried out, asynchronous circuits have not been widely used in practice due to the lack of an adequate theory of asynchronous circuits in the past, and the problems with hazards and races. Asynchronous circuits suffer in terms of required area for the following two reasons. First, while hazards are not of concern in synchronous sequential circuits where clock signals exist, any momentary erroneous signal may be troublesome in asynchronous sequential circuits. In order to design hazard-free sequential circuits, we end up using additional gates. Second, there is overhead produced by the handshake circuits which are designed to compensate for the elimination of the global clock signal. The relatively larger components and the handshake circuits both contribute to the overhead, which, in turn, increases density and power consumption. Another disadvantage is that a good portion of the system's time might be spent on producing the handshaking and control signals. As a result, the average speed of an asynchronous circuit designed with previously known methods may be low compared to that of a clocked circuit with a similar function.

Despite the problems listed above, it is unlikely the renewed interest in asynchronous techniques will diminish. It is also unlikely that industry will switch entirely to asynchronous circuit design. It is more likely that asynchronous and synchronous approaches will be developed in areas where they are really advantageous. Asynchronous design is an important research area for the following reasons:

1. When a few synchronous circuits with different clock signals are connected to each other through interface circuits, there will always be a need for asynchronous circuits to synchronize among the clocks. An example of this situation is the handshaking procedure of the IEEE-488 bus.

2. The advantages gained by designing asynchronous circuits are encouraging even if they are comparable with synchronous circuits in terms of area and

speed. Modularity, elimination of clock skew, and handling metastability are not easy to achieve in synchronous design. For this reason, the attitude is changing now. Designing asynchronous circuits to optimize power dissipation, area, and speed has received considerable attention in the past several years and new design methodologies have been developed specifically for asynchronous circuits [79,103,117,137].

3. A clock is an input signal with periodical transitions. From this point of view any synchronous circuit operates asynchronously. Any design or testing methodology for asynchronous circuits should also be applicable to synchronous circuits by applying a constraint on one of the inputs, i.e., 0-to-1 transitions are equally spaced in time, so are 1-to-0 transitions. Note that the opposite concept is not necessarily true.

Some publications [151,188] have pointed out the possibility of eliminating extra hardware due to handshaking signals. Synchronous circuits use a logic structure which allows hazards and races to occur prior to a transition on the clock line. This same logic structure can be used along with an asynchronous clock signal: i.e., the asynchronous clock signals should also allow hazards and races before a change on the clock line. One such logic structure is micropipelines. In general, there have been many attempts to design asynchronous circuits which have better performance in terms of area and speed than their counterparts [43,108]. The results obtained through ongoing research are encouraging [69,70,175].

1.1 Testing Asynchronous Circuits

Fabrication errors, fabrication defects, and physical failures may cause abnormal behavior (or simply malfunction) of IC elements. Such abnormalities are called *physical faults*. Therefore, it is not sufficient just to design and to manufacture integrated circuits, and to fabricate millions of devices using advanced integration circuit technology; semiconductor manufacturers and users must also verify that the circuits work as intended. *Testing* of a system is an experiment in which the system is exercised and its resulting response is analyzed to ascertain whether it behaved correctly [2]. The generation of test vectors for a logic circuit is an $\mathcal{NP}$-complete problem [66,88,93].

A great amount of research has been done to reduce the complexity, the hardware expense, and the execution time of testing synchronous circuits [43,69,70,79, 103,108,117,137,151,175,188]. The testing of combinational circuits has been studied in depth over many years. Testing synchronous sequential circuits is much more difficult, because long test sequences may be needed to excite a given fault and to make it observable. This problem is usually solved in synchronous circuits by using scan design methods [2] which reduce any circuit to a collection of combinational circuits and latches, thus simplifying testing.

Testing techniques for synchronous circuits have failed to be extended to testing asynchronous circuits for the following reasons:

1. The absence of a clock reduces controllability, and testing techniques are often more complex. For example, scan techniques are not easily applicable because one should order scanned logical values to read or to write *serially*. This needs a time constraint on reading from observation points or writing to controlling

points. Adding extra input/output test points is sometimes suggested to ease the controllability and observability of variables inside asynchronous circuits.

2. Although asynchronous circuits must be designed hazard- and race-free, faults may cause oscillations within the circuit, or may change a hazard- and race-free circuit to a hazardous one. These faults are difficult to detect. Therefore, any testing technique should handle hazards and races if they occur.

3. Also, compared to test generation for synchronous circuits, test generation for asynchronous circuits is harder since they tend to have more state holding elements than synchronous circuits.

Therefore, there is a need for a suitable testing method that can address the peculiarities of the asynchronous circuits.

In this dissertation we aim at high *defect coverage* by proposing several design-for-testability (DFT) techniques for digital BiCMOS/CMOS integrated circuits and providing test generation algorithms for asynchronous circuits.

One contribution of this dissertation is a DFT technique for detecting short defects in pull-up/pull-down BiCMOS/CMOS design. We propose an on-line I_{DDQ}-testable design, which uses the concept of virtual ground, and inspects for any undesired connection between the GND and V_{DD} nodes. The occurrence of a fault is announced if an unwanted path exists between these two nodes.

The second contribution of this dissertation is a novel circuit for detecting delay faults. The operation of the delay fault checkers is based on observing the time delay between transitions instead of the normally practiced observation of the outputs *at* a specific sampling time. We provide a fully testable CMOS design technique. We also propose concurrent delay fault checkers for asynchronous circuits by deploying synchronous DFT technique.

The importance of concurrent fault detection is that an asynchronous circuit which has passed the initial tests is prone to some defects which may occur during the normal operation of the circuit. This scheme eliminates the requirement of test-pattern generation, and the circuit can be tested concurrently with the normal operation.

The third contribution of this dissertation is initialization of asynchronous circuits. We provide a new initialization technique which has little overhead and can often be simplified. This technique does not change the complexity and hazard characteristics of the circuit since we do not re-design it.

The fourth contribution of this dissertation is the generation of test sequences for the detection of single & multiple stuck-at faults in asynchronous circuits. We provide a graphic representation of transitions, which occur in this type of circuit depending on the delays of wires and gates. We derive pieces of complete test sequences. We then link these pieces of test sequences to obtain the complete test sequence. We are never concerned with the actual location of the fault(s) in this process.

The organization of this dissertation is as follows. Chapter 2 contains a summary of design terminologies, explains pull-up/pull-down BiCMOS design, and provides the literature review of existing asynchronous designs. Chapter 3 is concerned with the testing terminology and the fault models used in this dissertation, reviews some related DFT techniques, and provides the literature review of testing techniques for asynchronous circuits. In Chapter 4 the proposed DFT method for detecting short and bridging faults in asynchronous CMOS/BiCMOS circuits is described. Fully testable CMOS design and methods for detecting delay faults in asynchronous circuits are presented in Chapter 5. The related DFT method for initialization of asynchronous sequential circuits is dealt with in Chapter 6. A test generation algorithm for asynchronous sequential circuits is discussed in Chapter 7. The summary of the research, conclusions, and the future directions for the problem of designing a fully testable asynchronous circuit are pointed out in Chapter 8.

Chapter 2

Design Overview

The organization of this Chapter is as follows. We present the different levels of abstraction in Section 1. PUPD (pull up/pull down) BiCMOS design, which is used throughout this dissertation, is explained in Section 2. Different structures for asynchronous circuits are explained in Section 3. An appropriate formalism for circuit design and a set of primitive modules are discussed in Section 4. Finally, a brief chronology of design methodologies is given in Section 5.

2.1 Levels of Abstraction

The design representation of digital circuits is separated into *behavioral, structural,* and *physical* domains [126]. *Behavior* refers to the functionality of a system or how its components interact with their environment, i.e., the mapping from inputs to outputs. *Structure* refers to the set of interconnected abstract components that make up the system. Structure is typically described by a netlist. Both behavioral and structural models are realized independently of the implementation technology.

Ultimately, the structure must be mapped into a *physical* design, and each abstract component must be replaced with real components depending on the fabrication technology used.

Behavior, structure, and physical design are usually distinguished as the three domains in which hardware can be described. Within each domain is a set of abstraction levels, which are concerned with specified aspects of the system. Note that the level of detail increases from the system level to the circuit level. Items of specific interest within each level of abstraction are:

- *System level:* The general specification level is often represented by a document written in a natural language. The behavior of a system is described by a set of performance specifications, which define the required operational characteristics for the system. The corresponding structural description contains the components which are required to realize the system: for example, processors, memories, controllers, buses, and switches. In the physical domain, the physical partitions of the system are defined; for example, PCB and chip partitions. Since this level is connected with the overall system structure and information flow, we call it the *system* level.

- *Algorithmic level:* At this level, which is sometimes called *functional* level or *instruction-set* level, the focus is on the computations performed by an individual processor, the way it maps sequences of inputs to sequences of outputs. In the behavioral domain, the level is expressed in an algebraic notation involving variables and operators, while no explicit reference is made to any physical objects or resources, nor to timing constraints. The description would include the algorithm performed by each process, together with its associated data structures and procedures. In the structural domain, hardware subsystems would represent the individual processes. The physical description would contain clusters; i.e., functionally related hardware subsystems.

- *Register-transfer level:* Below the algorithmic level is the *register-transfer level* (RTL), sometimes called *Micro-architecture level.* Here the system is viewed as a set of interconnected storage elements and functional blocks. The behavior is described as a series of data transfers and transformations between storage elements (registers or the *data path*) together with the ordering of the operations and transfers (the *control path*). The corresponding structural description defines the abstract implementation with a set of functional components; for example, ALUs, adders, MUXs, PLAs, ROMs and registers. Separate structural descriptions would be given for the data paths and their corresponding control paths. It may be possible to implement the structural description, or part of it, directly in silicon using library cells or module generators. The physical description would depend on the target implementation; for example, gate array or standard cell.

  The difference between the register-transfer level and the algorithmic level is the level of detail of the specified internal structure. Only the input/output behavior is known at the algorithmic level, but there is no direct relationship between variables and internal registers or between assignment statements and register transfers at this level. The RTL model explicitly involves resource entities (registers, memories) and describes the transfers and operations between these resources and their logic conditions. Busses and structural concepts are not necessarily present.

- *Logic level:* Next comes the *logic level.* The behavior is specified by logic equations. This behavioral description would define *switching circuits*, expressed in terms of *combinatorial logic functions*, together with *finite state machines*. A structural description would consist of a netlist of gates, flip-flops and registers. In the physical domain, the structural description would be realized directly in silicon by predefined library cells. In addition, the chip *floorplan* (a geometrical arrangement of interconnected cells) would be derived.

- *Circuit level*: Below logic level is the *circuit level*, which views the system in terms of the individual transistors of which it is composed. In the behavioral domain, the behavior of a library cell would be given in terms of its d.c. and a.c. electrical characteristics. In the structural domain, transistor networks for each cell, specified to the implementation technology, would be defined. These low level networks are structures of blocks which can be easily mapped to the circuit library of a given technology.

- *Layout level*: Finally, one can go down directly to the layout level where the physical geometry of components is described.

2.2 PUPD BiCMOS Logic Gates

The most common type of VLSI circuits uses CMOS to perform the logic functions, and bipolar transistors to drive the output load. The general structure of a BiCMOS logic gate [11] is shown in Figure 2.1.

It consists of a CMOS logic stage and a bipolar output stage. The CMOS stage performs the logic function and provides proper bias to the output stage. The p-part (of the CMOS stage) consists of parallel/series interconnection of PMOS transistors, determined by the desired logic function. Each of the n-part 1 and n-part 2 is the dual of the p-part and consists of series/parallel interconnection of

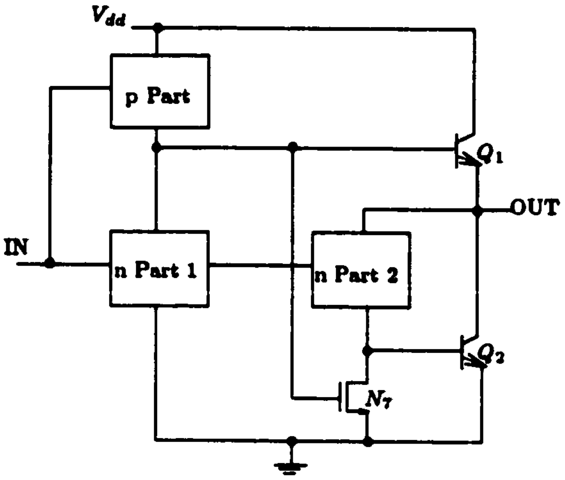


Figure 2.1: *General structure of a complex BiCMOS logic gate.*

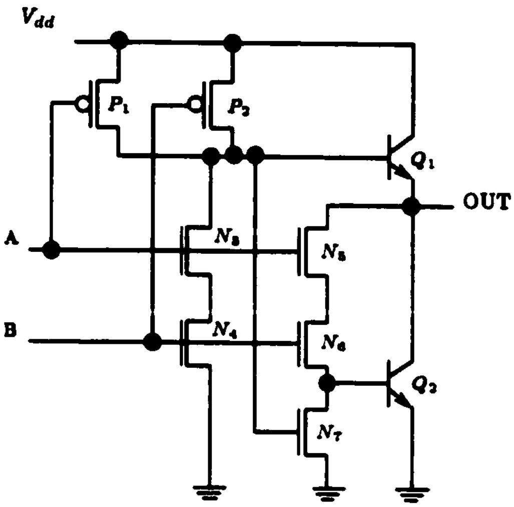


Figure 2.2: *Typical implementation of a conventional BiCMOS NAND gate.*

NMOS transistors. The output stage consists of a pull-up (Q_1), and a pull-down (Q_2) transistor. For normal (fault-free) operation, the CMOS stage ensures that Q_1 and Q_2 do not turn on simultaneously. During a low-to-high output transition, N_7 provides a discharge path for the base of Q_2, causing it to turn off more quickly. Similarly, during a high-to-low output transition, the n-part 1 provides a discharge path for the base of Q_1. A typical implementation of a conventional BiCMOS *NAND* gate is shown in Figure 2.2.

It is expected that in future BiCMOS VLSI systems, the dense logic will be predominantly CMOS [57]. BiCMOS buffers are to be employed for driving the heavily loaded nodes or off-chip load.

2.3 Categories of Digital Circuits

Digital circuits can be classified according to the function used to relate outputs and inputs or the method used to synchronize operations; see Figure 2.3. Digital combinational circuits have their outputs uniquely determined by the current circuit inputs. Circuits without feedback are always combinational.

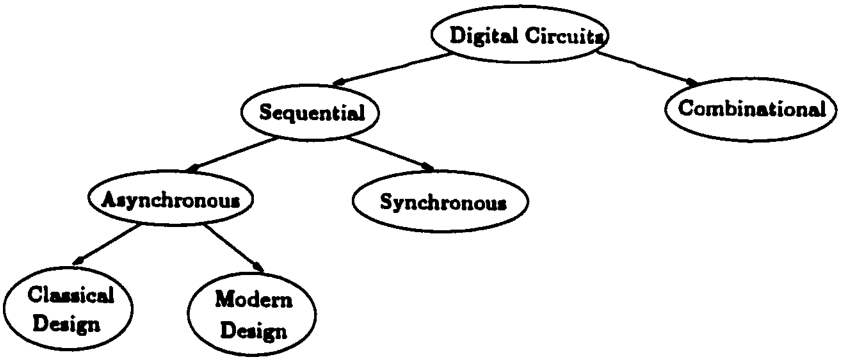


Figure 2.3: *Categories of digital circuits.*

Sequential circuits, on the other hand, contain state variables, which keep the record of input transitions by their present *state*: i.e., the internal state serves to summarize the pertinent history of the circuit. Both the output of a sequential circuit and the next internal state are functions of the current internal states and inputs. Sequential circuits are further classified according to the method used to synchronize operations. Circuits which employ a global clock, a sequence of periodic pulses generated by an independent source, are called *synchronous* while circuits without a clock are called *asynchronous*. Figure 2.4 shows a typical synchronous sequential circuit where the state-holding components are *latches*. All latches are activated simultaneously by a transition on the clock: i.e., between clock pulses, no recognition is taken of the input.

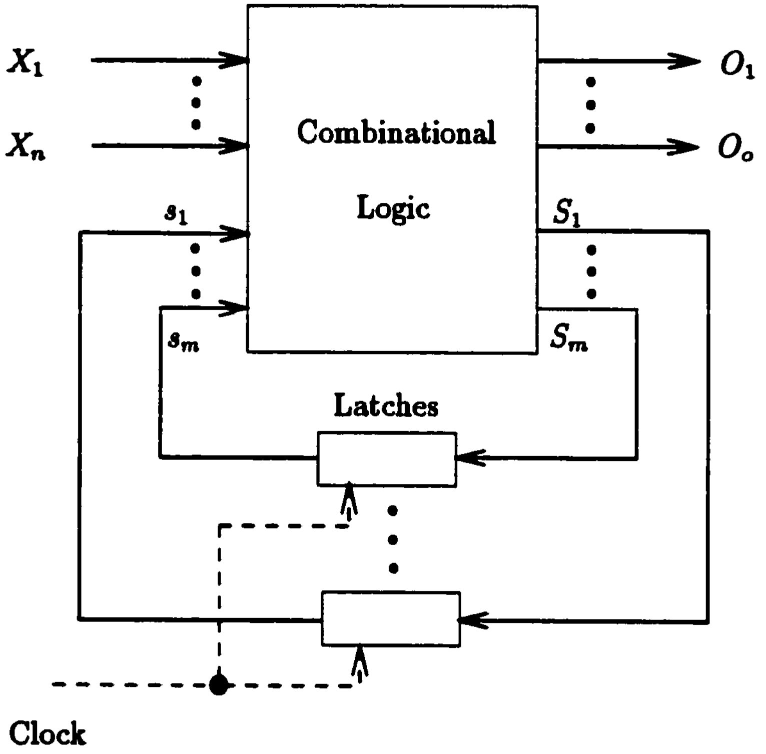


Figure 2.4: *Typical synchronous sequential circuit structure.*

Transitions in asynchronous sequential circuits depend on the delays of components and wires. These delays, although bounded, cannot be precisely determined since they are technology dependent. In this dissertation we address two different asynchronous design methodologies: *classical design* and *modern design.* Boolean algebra constitutes the basic formalism used in designing synchronous and classical asynchronous circuits where only the logic levels of the inputs and outputs are important rather than transitions. Standard basic elements in classical asynchronous design are gates such as AND, OR, XOR, and inverter. A typical classical asynchronous sequential circuit is implemented in Figure 2.5 as a large gate circuit with feedback. Designing classical asynchronous circuits [116] uses the *finite state machine* (FSM) model. The general format used to describe a sequential function,

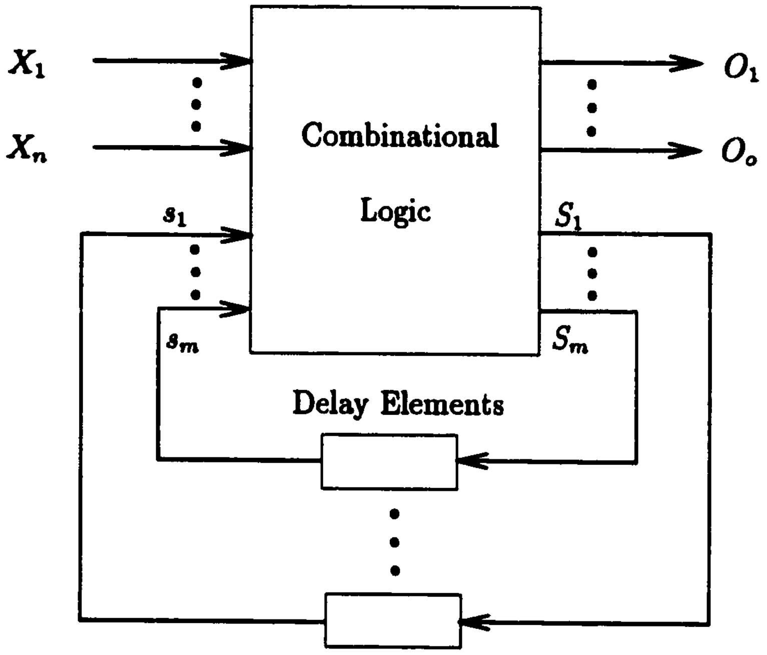


Figure 2.5: *Typical classical asynchronous sequential circuit structure.*

which relates output sequences to input sequences, in classical asynchronous circuits is called a flow table [79,180], which consists of a two-dimensional array. The columns correspond to input states, the rows to internal states, and the entries are ordered pairs representing the next internal state and the current output. A state is stable if the next internal state is the same as the current internal state. Only one stable state is permitted in any row of a primitive flow table [116]. The next step in the synthesis procedure is to find a minimal-state flow table from a primitive flow table. We then select a finite set of binary state variables and assign to each row of the flow table one or more states of these variables. The next step in the synthesis is to obtain truth tables specifying the circuit outputs and next values of the state variables as functions of the current values of the input and state variables. In the last step the truth table is transformed to a Karnaugh-map, from which the logic circuit is designed.

A situation in which more than one state variable must change in the course of a transition is called a race condition. If the behavior of the circuit depends on the outcome of the race, then the race is called critical; otherwise it is noncritical. Critical races are not a problem with clocked systems since the clock pulse will go off before any flip-flop can change its state and have that change pass through the delay element to the primary output. Thus the only requirement for the validity of a row assignment is that no code be assigned to two different rows.

Some of the difficulties with the FSM model [39] are that unrestricted input changes are not handled, concurrent behaviors can not be described directly, and the flow table expands exponentially with the number of input signals. Another problem with FSM synthesis is lack of modularity [49]. Although using clock pulses eases the row-assignment and eliminates hazards, the clock frequency must be set to allow for 'worst case' delays and manufacturing tolerances.

There are two basic signaling conventions for communication, control, and data representation in a digital system: level signaling and transition signaling. In level signaling, the voltage level of a signal is meaningful. In transition signaling, however, only change of a signal is significant; an event can be a transition from low to high (i.e. rising transition) or from high to low (i.e. falling transition). Transition signaling does not distinguish between rising or a falling transition.

The environment in classical asynchronous circuits changes the input signals and holds them fixed until the circuit reaches a stable state; after that, the environment is allowed to apply the next change to the input signals. This is called the fundamental mode operation [125].

Modern asynchronous circuits are sensitive to changes in the inputs in addition to their logic levels. Therefore, new approaches, called event-based formalisms, are used which are capable of dealing with transitions (events) in the input and output signals, allowing the input-output mode of operation [29]. In contrast to fundamental mode, input-output mode allows the input to change upon receiving an appropriate response to a previous input change, even if the circuit has not yet stabilized. Some basic event-based formalisms are trace theory [49, 52, 170] and transition graphs such as Petri nets [4, 39, 134, 145]. While the former has a textual representation form, the later has a graphical representation form. Examples of primitive elements in an event-based formalism are WIRE, JOIN, MERGE, TOGGLE, and ARBITER.

In contemporary asynchronous design, a control signal indicates the validity of the carried data [174]. For instance, each datum, or bundled-data, is accompanied by a control signal which specifies the legitimacy of that bundle of data, see Figure 2.6. When this control signal is activated by a sender, it is regarded as a request to the receiver. The number of data lines is arbitrary, and the request and acknowledge signals can use level (4-phase) signaling or transition (2-phase) signaling. Figure 2.7 shows intervals of data transfer by 2- and 4-phase signaling or handshake. The arrows in the figure indicate the sequence of events.

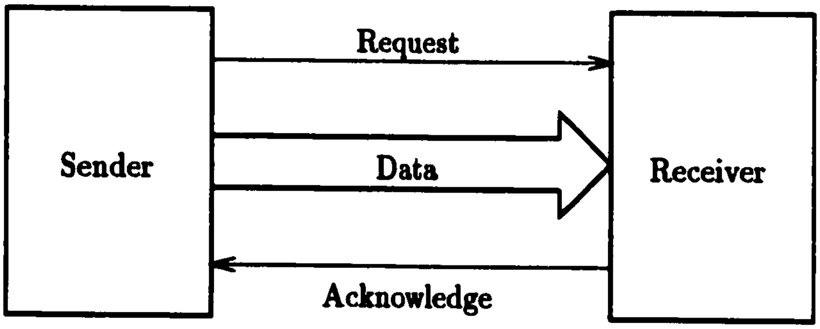


Figure 2.6: *Communication through a bundled data interface.*

For narrow communication channels, one wire is used per data value. This is

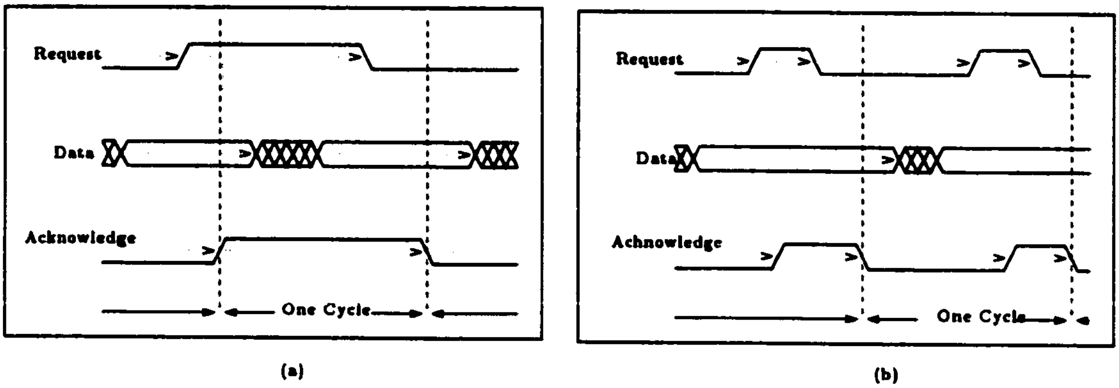


Figure 2.7: *Data transfer in (a) two-phase signaling, and (b) four-phase signaling.*

called *single-rail* encoding. The logic level on data wire indicates which binary value is communicated. Besides the data wire, which indicates *what* value is communicated, there is one wire which is used to indicate *when* the data is communicated. This wire is called the *data valid wire*. This wire can use either two-phase or four phase signaling. In *dual-rail* encoding, which is a widely used coding scheme, two wires are used for each bit in the binary representation of a number, one wire for the value zero and one wire for the value one. Sending a transition (or a logic one) along the '0' wire implements the communication of a zero, and sending a transition (or a logic one) along the '1' wire implements the communication of a one. Consider an n-bit binary number which is about to be transferred to the receiver. If the j-th bit is zero, wire $2j$ is raised, if it is 1, then wire $2j + 1$ is raised. In a four-bit wide dual-rail encoded channel where eight wires are used, "0" is encoded as 01010101, and "1" is encoded as 01010110. Either two-phase or four-phase signaling can be used for dual-rail encoding.

Asynchronous circuits may also be classified on the basis of their dependence on component delays. Widely used categories are Delay-Insensitive (DI), Speed-Independent (SI), and Quasi-Delay-Insensitive (QDI) circuits. DI circuits operate correctly independent of gate and wire delays. SI circuits, however, behave correctly no matter how long it takes for the gates in the circuit to respond, but the wire delays are assumed to be 0. QDI circuits function properly independent of the delays of gates and wires with the assumption that the difference in delays on branches of a fork are less than the minimum gate delay in that circuit. Such forks are named *isochronic* forks.

2.4 Asynchronous Design

This Section is devoted to an appropriate formalism for circuit design, a set of primitive modules, and micropipelines.

Trace Theory (Program Notation):

The high-level specification for circuits is in a language that is based on Hoare's CSP (Communicating Sequential Processes) [83]. A directed trace structure is represented by a *directed command* similar to a regular expression. A command is composed of input and output alphabets, and special symbols which stand for certain operations. We first briefly define each operation, then we illustrate them by using a simple example.

- Inputs and Outputs: In the representations, input symbols are postfixed by ?, and output symbols are postfixed by ! Some variables are neither inputs nor outputs of the element. They are called *internal symbols*, and denoted without any postfix.

- Transitions: A transition from low voltage to high voltage is denoted by "↑". A falling transition is denoted by "↓".

- Prefix operator: "**pref**" stands for *prefix-closure*. A set of string, $S_1S_2\cdots S_n$, is prefix-closed if its initial parts, $S_1S_2\cdots S_i$ for $i < n$, is allowed.

- Sequential transitions: If transition a happens after transition b, it is shown by ";". This is called the *concatenation* operation. This operation is associative.

- Parallel transitions: The parallel execution of two transitions, a and b, is denoted by "||". This is called the *weaving* operation, and it can be considered as the conjunction of two trace structures, i.e. a function that is in accordance with both functions.

- Selection: The selection command consists of a number of subcommands (i.e., a collection of transitions) and is denoted by the symbol "|". It is sometimes called as *union* operation. Only one subcommand in a union operation is executed. The other subcommands are suspended until the running command is completed.

- Repetition: "*[ ]" stands for *repetition* (of the enclosed). The repetition command also consists of a number of subcommands. This command shows that any repetition of the specified behavior can occur. Zero repetition corresponds to the initial state.

As an example, consider the following composed command:

$$\textbf{pref} * [(a?;b!)]$$

This is the WIRE specification. First, a transition, rising or falling, happens on the input (a) of this component. Then, the output (b) sends a transition. The repetition ($*$) and the prefix-closure (**pref**) of this behavior allows the following sequences of transitions: ε (empty set), a, ab, aba, $abab$, etc.

Petri Nets:

A Petri net is composed of *places*, represented by circles, *transitions*, represented by bars, and *arcs*, represented by arrows directed from places to transitions or from transitions to places. A transition in a Petri net corresponds to an event, and *firing* the transition corresponds to the occurrence of the event. A transition is enabled if all of its input places contain a token. To fire a transition, tokens are moved from each of its input places to each of its output places. Initial states are shown by assigning tokens to places on the Petri net.

2.4.1 Basic Asynchronous Element: An Example

To give an example of how an asynchronous module is specified in terms of commands, trace structures [52], and Petri nets, we describe the function JOIN in this Section.

The JOIN component produces an output transition only after both of its inputs have a transition. Whenever both inputs are "0" or "1", the output will also have the same logic value. If the inputs have different states (mismatch), the output will retain its previous state. Figure 2.8 shows the command, Petri net, and schematic of JOIN.

A JOIN operated in a more liberal environment is a C-element. The specification of the C-element is given in the graph of Figure 2.9. The nodes of the graph denote the external states of the C-element. Thus, each node is labeled with a triple $ab \cdot c$ of binary values, the first two of which are the input values and the third is the output value. The centered dot $\cdot$ is used for convenience to separate the input and

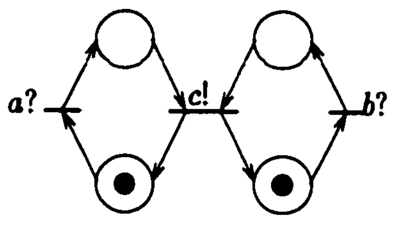
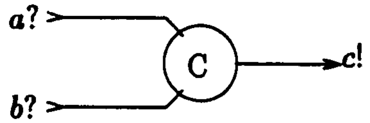
C-Element		
Trace Command	Petri Net	Schematic
pref $*[a?;c!]$ || **pref** $*[b?;c!]$		

Figure 2.8: *Representations of JOIN.*

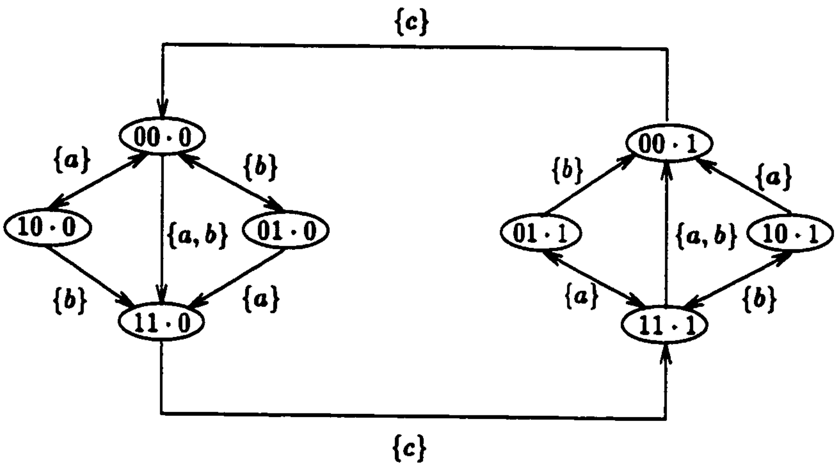


Figure 2.9: *Specification of the C-element.*

output parts of the state. A directed edge indicates a transition from one state to another. An edge with two arrowheads is a shorthand for two edges, one in each direction.

The C-element behaves as follows: When the two inputs agree, the output has the common input value. Otherwise, the C-element remembers its previous state.

The specification of Figure 2.9 defines the behavior not only of the C-element but also of its environment. For example, consider state $00 \cdot 0$; this state is stable. Here, the environment is allowed to change input a, or input b, or both. If only one input is changed, the new state is still stable. The environment is then allowed to "withdraw" the input and return to the initial state. Once both inputs change, however, the environment must wait for the C-element to respond by changing c. Thus, the C-element is operated in the input/output mode [29]. Once c changes, the environment is again permitted to change a, or b, or both, etc.

2.4.2 Micropipelines

This section describes an efficient implementation of asynchronous pipelines modules developed by Sutherland. [174]. This approach can lead to very efficient and fast implementation of arithmetic units.

Figure 2.10 shows a micropipeline. Consider that all latches are empty, and all wires, except the inverter outputs, are 0. While the environment puts data on the interface, a transition from 0 to 1 occurs at R_{in}, passing through all C-elements in series, and emerging on R_{out} as follows. First, the input data is captured by the first latch as soon as a rising transition happens on C (capture) input. After an appropriate delay to latch the data safely, a rising transition on C_d is observed, followed by a rising transition on C of the next latch. Assuming the initial state, the empty data on the input of the second register is taken, showing a rising transition on C_d of the second register, or P_d of the previous register. This means passing the data latched from the input of the first register to its output. Then, nothing will happen to the first register while transitions can travel on C_i, C_{d_i}, P_i, P_{d_i} to R_{out}. Note that the C_d signal produced by stage i plays the role of *ack* for stage $i-1$

and of *Req* (after an appropriate delay) for stage $i + 1$. The input data of each register (empty or not) is first latched, then passed to its output, processed by the combinational logic, and is ready at the input of the next stage register. In fact, the data moves one block forward. Note also that the data latched is not allowed to pass from the input of the first register to its output until a transition occurs on P. As mentioned earlier, C_d and P_d are copies of the control signals C and P, delayed so that the register completes its response to the control signal transitions.

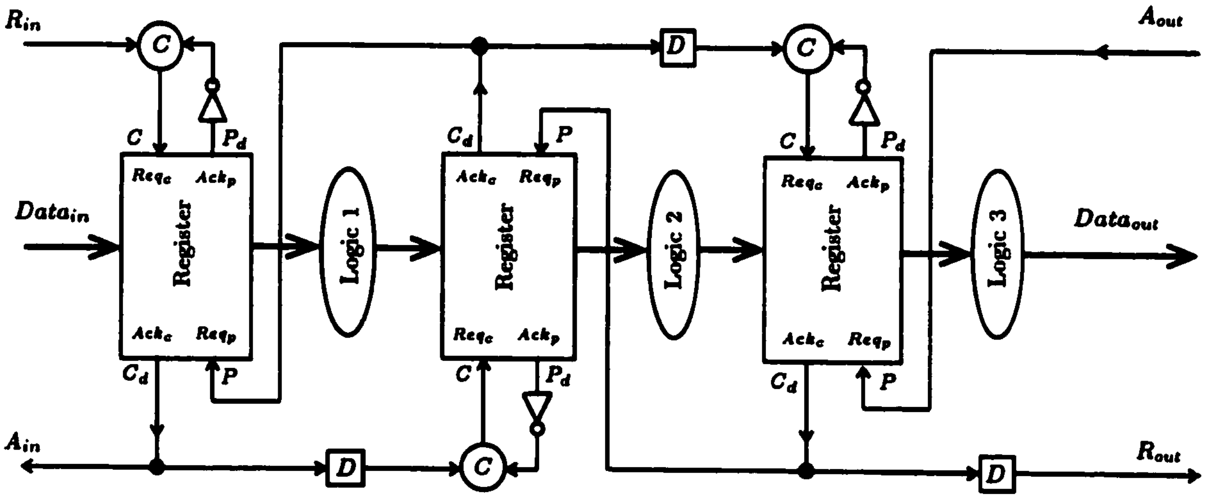


Figure 2.10: *Control and computational parts in Micropipeline structure.*

The next transition on R_{in} is from 1 to 0 which passes through the first two C-elements, but not through the third one. This allows one more data to advance ahead in the micropipeline. Consider an instant when the micropipeline is filled with data. The output side of the micropipeline will not be ready to accept new processed data unless it raises A_{out}. Until then, the micropipeline will not accept any more data. When a transition on A_{out} appears, a space in the micropipeline is freed, and a transition on A_{in} is generated.

The data stored in a micropipeline is processed through the logic blocks between register stages, which are usually assumed to be combinational. Since these blocks slow down the data moving through them, the accompanying control transition must also be delayed, here shown as D. This delay on the control path must be adjusted to allow the combinational logic outputs to settle and the latch setup time to be satisfied. Therefore, it must be at least as large as the worst-case delay of the logic block. However, the overall processing in the pipeline occurs at rates closer to the average processing time than the worst-case time of the entire circuit. This can be best understood by looking at the controlling transitions on C or P as they resemble a non-periodical asynchronous clock. The period of such a clock need not be limited by the maximum worst-case delay of the logic blocks. If we ignore the logic blocks and associated delay elements, we have a simple data FIFO (First-In, First-Out) queue.

Micropipelines are based on two ideas. First, transition signaling or two-phase handshaking is used. Such an approach permits much faster operation than the classical four-phase handshaking at the expense of slightly more complex circuitry. Secondly, bundled data communication protocol is used where control signals define the validity of the data signals. Only the control part must be designed with a delay-insensitive signaling protocol coupled with delay matching for data communication.

There are advantages and disadvantages in designing micropipelines. Due to the registers, hazards of logic blocks in the data path do not interfere with the circuit operation and its effects do not propagate any further. This allows the use of any synchronous structures in micropipelines by replacing the latches and the clock with micropipeline latches and control structures. Micropipelines also benefit from some asynchronous design advantages. For example, they are elastic, in that data can be sent to them and received from them at arbitrary times. The disadvantages are as follows: they do not yet deliver average-case performance for each logic unit, and the delay of the control part must be guaranteed to be larger than the delay in the data part, demanding delay fault testing.

2.5 Chronology

In this section we give a brief survey of classical, speed-independent (SI), delay-insensitive (DI), Self-timed (ST), quasi-delay-insensitive (QDI), and micropipeline asynchronous design methodologies.

Classical Asynchronous Design:

Huffman [85] was the first to develop the traditional way of designing asynchronous circuits using flow tables. Classical asynchronous circuit design as described by Unger [180] generally places restrictions on the allowed input changes, such as the requirement for fundamental-mode operation that only one input can change at a time, and succeeding input changes must be delayed until the circuit is stable. His method did not allow simultaneous input changes. Synchronizer and arbiter circuits cannot, in general, satisfy these restrictions. Later in 1979 [181], Unger studied possible way of easing this restriction. However, concurrent behaviors cannot be described directly in classical asynchronous circuit design.

It appears at first that the realized circuit should behave as specified in the flow table. However, this is true only under ideal conditions: There are no delays in the logic elements or in the wiring, but only in the feedback lines. If this unrealistic assumption is removed, then several kinds of malfunctions such as hazards become possible. Such transient errors can lead to steady-state errors. There are circuits which neither altering the logic circuit nor altering the state assignment eliminates these malfunctions. The only remedy is to insert adequate delay elements at the output nodes of some gates.

The state assignment is by no means a simple one. Not only must each row be assigned a unique code, but the codes must be so interrelated that no transition involves a non-removable critical race. The difficulties become more evident when we try to minimize the transition time, the number of state variables, and the complexity of the resulting logic circuit. Besides, the flow table expands exponentially with the number of input signals. Another problem with FSM synthesis is lack of modularity.

In 1993, Dill used *burst-mode* specifications, a class of asynchronous finite-state machine specifications which allows multiple-input changes [194]. Using asynchronous specification style, Nowick *et al.* [139,140] presented an automated design methodology for locally-clocked, globally asynchronous state machine controllers. They then described a heuristic technique for state minimization and optimization to improve implementations and demonstrated the feasibility of their method by designing a cache controller [141,142]. Their technique tends to provide hazard-free logic with multiple-input changes [143].

Speed-Independent Circuits:

The concept of SI switching circuits was first studied by D. Muller more than four decades ago [133,136]. SI design makes it possible to guarantee correct operation of a circuit at speeds limited only by actual delays present in the physical devices of the circuit assuming wire delays are zero. A Muller-circuit can generate a ready signal as soon as it has "digested" the previous input and is ready for a new input. Muller also introduced the basic component of parallelism, the C-element. Utilizing C-elements, Muller designed various blocks to implement functions such as delay, AND, XOR, switch, and memory components.

Dill [50,51] was the first to develop an automatic verifier of speed-independent circuits based on a formalism called *trace theory.*

In 1987, Chu presented a synthesis technique for speed-independent circuits based on a high-level graph model of finite automata called STGs (Signal Transition Graphs) [39], a form of *interpreted Petri nets.* STGs can represent ordering and concurrency.

Delay-Insensitive Circuits:

Clark [40] and Molnar *et al.* [41,135,159] considered *module design* and *system design using modules* as two separate issues. They assumed that the timing behavior of the physical realization of each module is left up to the module designer. They proposed a set of relatively simple, general-purpose delay-insensitive components, called *Macromodules,* from the system level point of view, rather than the circuit level. Working systems can be readily assembled for evaluation and studied from these components without creating any logical problems such as waveform deterioration or power supply interactions. These modules have been successfully used to design input-output devices, registers, and memory units.

Although the concept of DI circuits has existed since the 1960's, it was not formalized until the mid 1980's by Udding [179]. He proved that a set of five conditions are necessary and sufficient for a delay-insensitive specification. Ebergen developed theorems for designing and decomposing a component into delay-insensitive elements and showed that by interconnecting a small set of primitive components delay-insensitive operation for a large class of behaviors can be achieved [52]. In his approach, the behaviors of components are specified by programs written in a notation based on trace theory.

Self-Timed Circuits:

The term *Self-timed* circuit was first used by Seitz [165]. By pointing out the problems of system timing, Seitz mentioned that all system events should occur in proper *sequence* in a self-timed circuit, but nothing ever has to occur at a particular *time*. A self-timed system is a legal interconnection of *elements* which may have an internal clock and some delays to perform computations on their input signals. Modules in a self-timed system, however, communicate asynchronously; i.e., signal events at their inputs are regarded as *request* signals and signal events at their output are *completion* signals, and each element is allowed to impose an *arbitrary delay* between the occurrence of input and output signal events. He then noted that a self-timed circuit can use either *equipotential* (what we call speed-independent today) or *delay-insensitive* self-timed signaling conventions. The design of self-timed systems was further studied in [105, 176, 193]. Self-checking circuits were also studied by Varshavsky [182] and more recently by Beerel and Meng [15, 16]. The former have presented a formal proof that DI and SI are self-timed circuits. Meng incorporated high-level specifications using STGs to automate the design of asynchronous self-timed circuits [127, 128].

Quasi-Delay-Insensitive Circuits:

Martin [32, 118, 122, 123] proposed a novel design approach for compiling QDI circuits from CSP-like specifications. In this approach, modules are specified as sequential processes which can be refined into more detailed descriptions. In the final step, these descriptions are replaced with hardware templates which implement the actions of the circuit. Many circuits, including a complete asynchronous microprocessor, were actually implemented using Martin's approach. He also pointed out some of the limitations to the delay-insensitivity of asynchronous circuits [119].

Compiler-based styles have also been successfully used for certain applications such as pipeline controllers by Philips researchers [28].

Micropipelines:

Special types of modern asynchronous circuits called *micropipelines* were discussed by Sutherland [174]. Micropipelines are simple, elastic, event-driven pipeline structures which consist of a control circuit and a data path structure that may or may not have internal data processing. The communication protocol used in micropipelines is known as the two-phase bundled data convention. Sutherland gave examples of simple micropipelines like a FIFO queue circuit and more complicated ones such as a multiplier, binary to one-out-of-N decoder, and memory controller chips.

Furber *et al.* adopted the micropipeline approach in October 1990, and completed an asynchronous ARM processor within the AMULET project [69, 70]. Although their design did not outperform its clocked counterpart, its performance, area, and power dissipation were within a factor of two compared to the synchronous ARM. This is encouraging, though, if one considers the modularity a designer can achieve by asynchronous logic architecture. Work is still being continued to enhance the design. It is not yet clear whether such techniques will be cost-effective.

Finally, Sproull *et al.* [171] have advanced micropipeline architecture to counterflow pipeline processor architecture which uses a bi-directional pipeline in which instructions and results flow in opposite directions as they pass each other. Although this work is still in its early stages of development and some limitations are not fully understood, it promises several advantages. The structure has a fast communication since pipeline stages communicate primarily with their nearest neighbors. Since each pipeline stage can be designed separately for a different purpose, the design structure is highly modular. The processor design procedure is simpler due to local control over each item in the counterflow pipeline.

Chapter 3

Testing Overview

The rapid advance of integrated circuit technology has made possible the fabrication of ICs containing literally millions of devices. However, it is not sufficient just to design and manufacture integrated circuits; semiconductor manufacturers and users must also verify that the circuits work as intended. Semiconductor manufacturing processes are so complex that this verification cannot be done on a sampling basis; rather, each VLSI circuit must be individually tested [63]. In this environment, testing costs are now a significant portion of the costs of fabricating ICs. To justify this extra cost, one must keep in mind that a significant portion of manufactured integrated circuits are faulty due to manufacturing defects. Even a well-fabricated IC may get damaged while in service, so testing is essential through all phases of fabrication, assembly, and service as a means of achieving high reliability.

The well-known "rule-of-10" [1] states that detecting and repairing a fault at the printed-circuit board level is ten times more expensive than detecting the fault in the supply of chips before assembly, and detecting and repairing a fault at the system level is ten times more expensive than detecting the fault in the supply of boards. Once a system is in service with a customer, this cost increases ten-fold again with respect to detecting faults before shipment of the new system. It is therefore clear that the costs are minimized by early detection of the faulty ICs. Hence, IC testing is the concern, not only of semiconductor manufacturers, but also of the larger digital systems industry.

The organization of this Chapter is as follows. Section 1 defines the terms used in literature. Section 2 provides proposed fault models, which are used throughout this dissertation. The rest of this chapter gives a brief overview of known testing approaches. These techniques are investigated in two parts. Section 3 provides a survey of test generation algorithms, while Section 4 presents Design-for-testability (DFT) techniques for detecting I_{DDQ} faults, delay faults, and testing sequential logic circuits.

3.1 Defects and Faults

Testing of a system is an experiment in which the system is exercised and its resulting response is analyzed to ascertain whether it behaved correctly [2]. If incorrect behavior is detected, a second goal of a testing experiment may be to *diagnose*, or locate, the cause of the misbehavior. Diagnosis assumes knowledge of the internal structure of the system under test.

An instance of an incorrect operation of the system being tested (or UUT for *unit under test*) is referred to as an *(observed) error*. The causes of errors and their corresponding examples are summarized in Table 3.1 [2].

The term *"defect"* is applied to any permanent deformation in the structure of fabricated ICs, caused by process-induced disturbances and contaminations [114].

Design errors: (human errors)	incomplete or inconsistent specifications, violations of design rules, incorrect mapping between different levels of design
Fabrication errors: (human errors)	wrong components, incorrect wiring, shorts caused by improper soldering
Fabrication defects: (imperfect process)	shorts, opens, improper doping profiles, mask alignment errors, poor encapsulation
Physical failures:	component wear-out, environmental factors

Table 3.1: *Causes of observed errors [2].*

A test can have a major impact on improving the process by detecting these defects.

The last cause of errors, mentioned in Table 3.1, is any physical failure which occurs during the lifetime of a system due to component wear-out and/or environmental factors. For example, aluminum connectors inside an IC package thin out with time and may break because of electron migration or corrosion. Environmental factors, such as temperature, humidity, and vibration accelerate the aging of components.

Fabrication errors, fabrication defects, and physical failures may cause abnormal behavior (or simply malfunction) of the ICs elements. Such abnormalities are called *physical faults.*

The integrated circuit failure problem can be divided into two categories: those failures that occur during the fabrication process (*process yield*), and those failures that occur during the device's in-service period (*reliability*). However, the process yield and reliability of a given process can both be affected by the same nature of failure.

3.1.1 Types of Testing

Table 3.2 summarizes the most important attributes of the testing methods and the associated terminology [2].

3.2 Fault Modeling

A large number of different physical defects can potentially occur in integrated circuits. In general, physical faults do not allow a direct mathematical treatment of testing and diagnosis. The solution is to deal with *logical faults* (or simply, *faults*) which are a convenient representation of the effect of the physical faults on the operation of the system. A fault is *detected* by observing an error caused by it. The basic assumptions regarding the nature of logical faults are referred to as a *fault model*. Through a process of abstraction, a fault model *maps* this relatively large number of defects into a small number of modeled faults. The selection of adequate fault models is crucial in achieving a high capacity of testing since the quality of a test set, and the usefulness of a DFT method are strongly correlated with the quality of the fault model used. In this section, we review the fundamentals of fault modeling for digital circuits.

3.2.1 Types of Faults

Depending on how we approach the fault model, there are *logical* vs. *parametric* faults, or *single* vs. *multiple* faults.

A defect that causes a change to the logical function of the circuit can be represented by a logical level abstraction known as a *logical fault*. Similarly, a

Criterion	Attribute of testing method	Terminology
When is testing performed?	• Concurrently with the normal system operation	• On-line testing (Concurrent testing)
	• As a separate activity	• Off-line testing
Where is the source of stimuli?	• Within the system itself	• Self-testing
	• Applied by a tester	• External testing
What is the physical object being tested?	• IC	• Component-level testing
	• Board	• Board-level testing
	• System	• System-level testing
Which attribute of the object is being tested?	• Logical behavior	• Functional testing
	• Analog qualities of the input/output interface	• Parametric testing
	• Timing specifications	• Dynamic (Delay) testing
How fast are the stimuli applied?	• Much slower than the normal operation speed	• DC (static) testing
	• At the normal operation speed	• AC testing
What are the observed results?	• The entire output pattern	N/A
	• Some functions of the output pattern	• Compact testing
What lines are accessible for testing?	• Only the I/O lines	• Edge-pin testing
	• I/O and internal lines	• Guided-probe testing • Bed-of-nails testing • Electron-beam testing • In-Circuit testing
Who checks the results?	• The system itself	• Self-testing
	• An external device (tester)	• External testing

Table 3.2: *Types of testing [2].*

defect that causes a change in a continuous parameter (e.g., delay, voltage levels) of the circuit can be represented by an abstraction called a *parametric fault* [62].

Logical Faults:

The most common logical fault model used in digital circuit testing is the *stuck-at* fault model, originally proposed by Eldred in 1959 [56]. This model assumes that failures are manifested by fixed logical values at certain nodes, i.e., each circuit node may have one of the two possible faults: permanently stuck at logic value "1" (SA1) fault, or permanently stuck at logic value "0" (SA0) fault.

1. **The Single Fault Model**: It is usually assumed that only a single fault is present in the circuit at any time. This is done to avoid the complexity of considering the multiple fault case.

2. **The Multiple Fault Model**: In the *multiple stuck-at* (MSA) fault model, it is assumed that two or more stuck-at faults may exist at the same time. A circuit with n lines can potentially contain $3^n - 2n - 1$ distinct [1] *multiple* stuck-at faults.

Parametric Faults:

Physical defects in integrated circuits can sometimes degrade circuit performance without altering the logic functionality. These faults can be detected by parametric or dynamic testings (cf. Table 3.2). The following parametric faults are addressed in this dissertation.

[1] Each line can be either stuck-at zero, stuck-at one, or fault-free, so there will be 3^n different cases. $2n$ of them, however, correspond to *single* stuck-at faults and one case corresponds to fault-free circuit; therefore the remainder will be $3^n - 2n - 1$.

1. I_{DDQ} **Fault::** A low-impedance path between V_{DD} and GND happens when two nodes of the circuit are shorted, leading to an excessive leakage current. Such a fault, which increases quiescent power supply current, is called I_{DDQ} fault.

2. **Delay Fault::** A *delay fault* causes the propagation delay of the circuit to be longer than the specified limit. Delay faults are modeled as either the *gate delay model* [112] or the *path delay model* [169]. The former assumes that the delay faults are lumped at a *faulty gate*, while the latter models isolated failures as well as distributed failures.

3.2.2 Levels of Faults

Fault models can be classified according to the level of the circuit representation to which they are applied. Classes of fault models in increasing levels of abstraction are [33]:

- **Transistor Level::** These models assume that individual transistors in the circuit are permanently fixed in either a stuck-open (*off*) or a stuck-closed (*on*) state.

- **Gate Level::** Faults are considered at the input or output nodes of each component.

At the expense of the test generation, lower levels of abstraction represent the fault more precisely, because they are closer to the physical failures.

3.2.3 Fault Models

At a time when digital circuits were constructed from discrete components, the stuck-at fault model seemed adequate for fault modeling because actual failures often behaved in this manner. With the rapid growth of IC industry, VLSI circuits have replaced discrete devices. Furthermore, the technology has evolved from bipolar to MOS, and from multi-micron to sub-micron line-widths. The imbalance of these two trends has caused researchers to question the adequacy of stuck-at fault model for fault modeling [20, 61, 168]. The following fault models have been used in this dissertation.

- Line *stuck-open* fault:

  A *stuck-open* fault is said to occur when a circuit node becomes floating, and exhibits a high impedance, so that the signal value at the node becomes dependent on its previous value, see Figure 3.1. Open-circuits are modeled as a large resistor inserted between the affected node and the node to which it would normally have been connected.

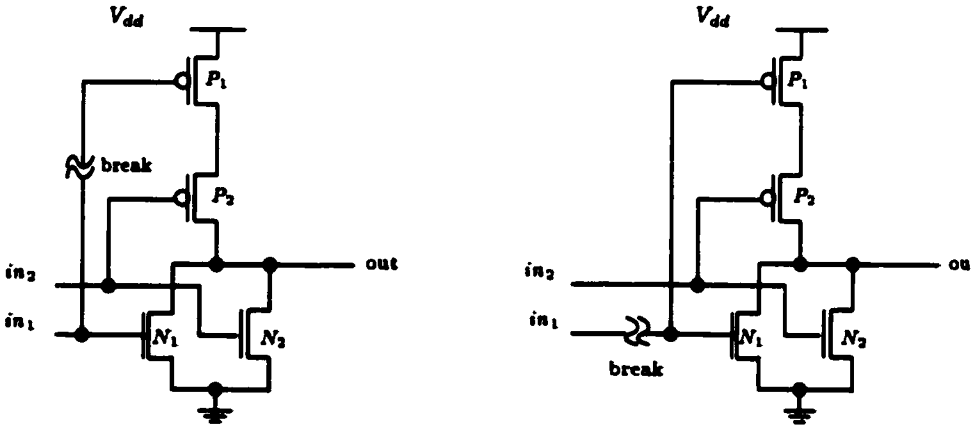


Figure 3.1: *Two possible open defects in a CMOS NOR gate.*

- Line *stuck-at 0/1* fault:

There are different line stuck-at models. Figure 3.2 shows four possibilities. Consider a fault-free line originating from *source* and sinking into *destination*. Stuck-at models assume that the line is connected to V_{DD} (GND) somewhere in the middle in the event of stuck-at 1 (0). However, they differ as to whether a discontinuity occurs. If the wire is not broken between source and destination, its behavior exhibits two different faults. There is a set of inputs which forces the source node to its opposite value at which it is stuck. This usually shows itself as an I_{DDQ} fault. On the other hand, if the stuck-at fault is detectable, it is likely that there exists a test vector by which the fault behaves as a logical fault. Therefore, the probability of detecting such faults is doubled if one employs the I_{DDQ} testing technique as well as logical testing techniques. It is wise to choose the fault model in which there exists a short

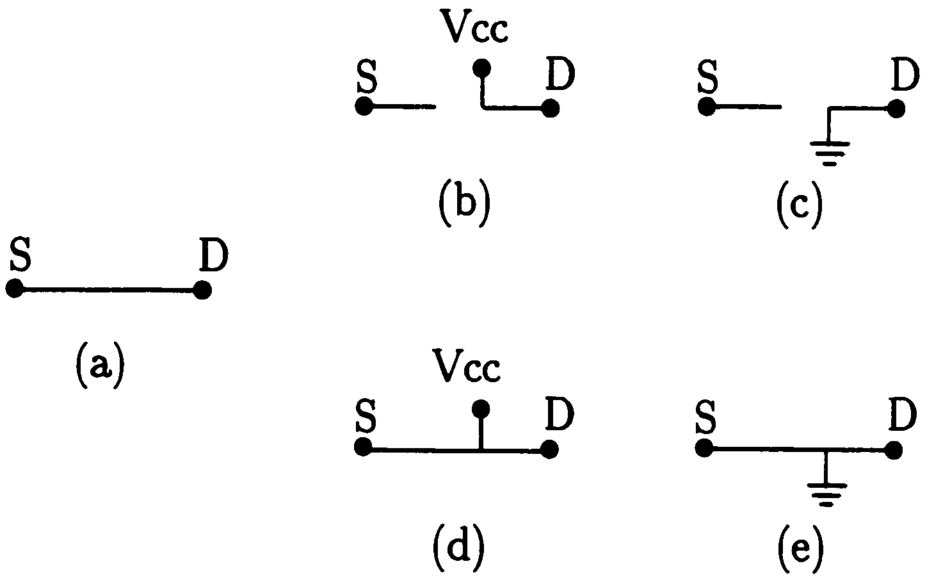


Figure 3.2: *Line stuck-at faults: (a) Fault-free line, (b) and (d) stuck-at 1 fault models, (c) and (e) stuck-at 0 fault models.*

between the wire to either V_{DD} or GND, and stuck-open fault, see Figure 3.2. In the case of a complex circuit in which making a decision as to what model should be used becomes difficult, one may simply adopt the former.

- Transistor stuck-on:

  A transistor stuck-on is a transistor that is always in the conducting mode. The (drain, source)/(emitter, collector) nodes of stuck-on MOS/Bipolar transistors are connected through a small resistor at the time of simulation. Study [2, 55, 67, 92, 99, 154] shows that transistor stuck-on usually leads to an excessive leakage current between V_{DD} and GND for some combination of input logic values. Therefore, they may be detectable using I_{DDQ} fault testing.

- Transistor stuck-open:

  A stuck-open transistor acts as if one of its terminals is stuck-open. The open-circuit in the gate of an MOS transistor is treated as a floating point, and for simulation, the gate terminal is tied to the corresponding bulk terminal. Study shows that a delay fault or a logical fault will usually occur at the presence of transistor stuck-open.

A transistor which is stuck-on or stuck-open behaves as if its gate is stuck-at 0/1 in many cases. For example, a PMOS stuck-open acts as if its gate is stuck-at 1. Figure 3.3 shows stuck-at 1 (0), stuck-on, and stuck-open transistors for conventional CMOS NAND gate.

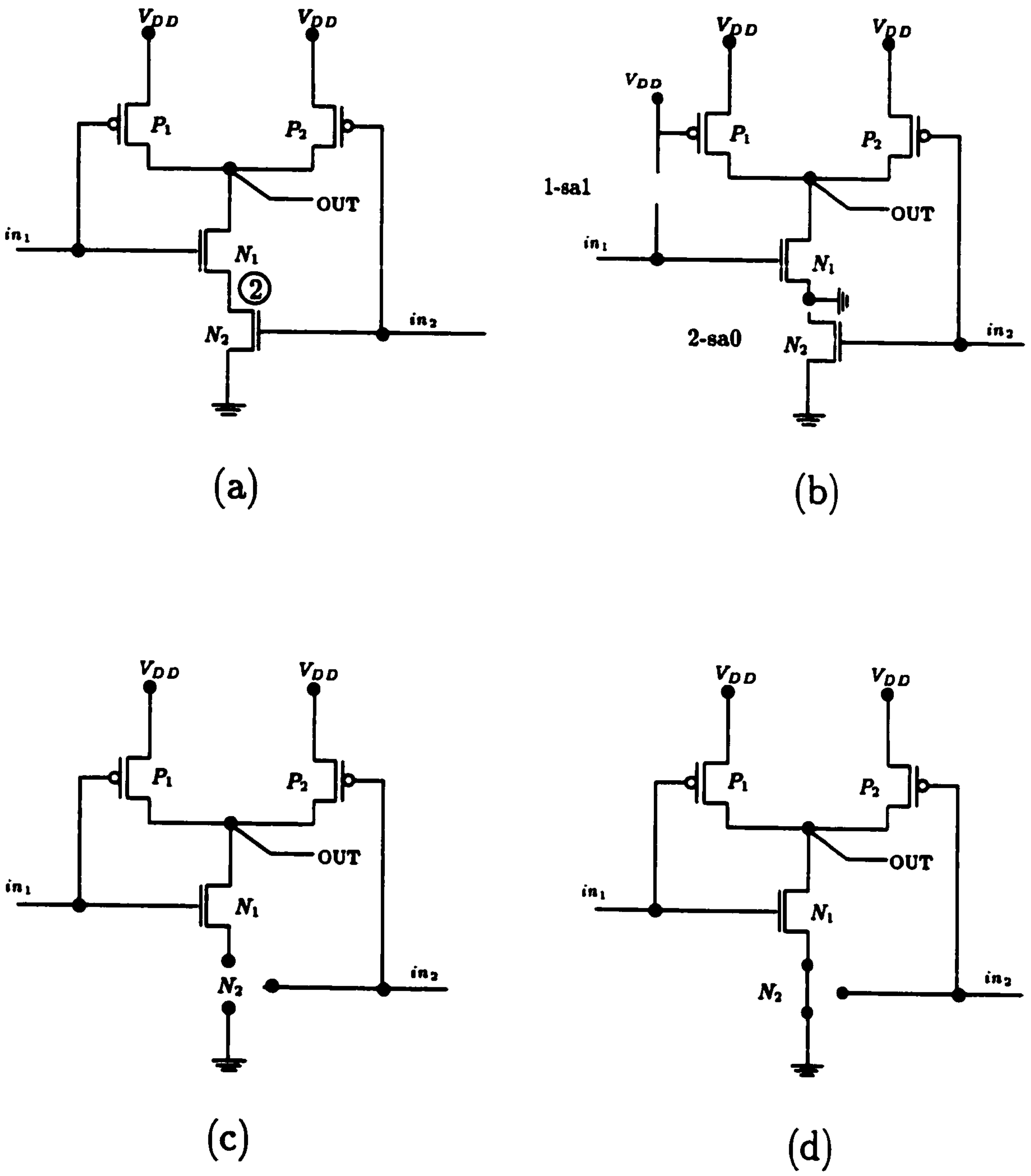


Figure 3.3: *Transistor stuck-at faults: (a) Fault-free NAND gate, (b) Faulty NAND gate with line stuck-at 1 (0), (c) N_2 stuck-open (delay fault), (d) N_2 stuck-on (I_{DDQ} fault).*

The values of the resistors, modeling shorts and opens, should be varied in a wide range to accommodate the effects of short and open defects of variable strengths. Due to the difficulty of this approach, most researchers [2, 55, 67, 92, 99, 154] have assumed fixed values for shorts and opens (e.g., less than 10Ω for shorts, and more than 10MΩ for opens).

Fault characterization for the conventional family has been reported in the literature [106, 110, 160, 163, 172]. According to these fault models, physical defects in conventional BiCMOS logic gates manifest themselves as one or more of the following categories of logical and/or performance degradation faults:

1. Output is stuck-at 1 (*SA1*), or is stuck-at 0 (*SA0*).
2. Output is stuck-open (i.e., the output is floating and presents a high impedance).
3. Output follows some of the inputs, but not all.
4. Output has a *soft* [2] logic 1 or a soft logic 0.
5. Output is logically indeterminate (at intermediate level).
6. Abnormal increase in quiescent power supply current (I_{DDQ}).
7. Output is slow to rise (*STR*) or is slow to fall (*STF*).

Table 3.3 [163] shows the results of simulating physical defects in a conventional BiCMOS two-input NAND gate (Figure 2.2).

To make it easier to locate each fault in the circuit, we use the following notation: If a fault is caused by a short between two nodes, we refer to it by the first letter of each node, followed by the name of the transistor; e.g. a fault caused by a short

[2] A soft logic level is a degraded, yet logically valid, voltage level.

Fault Characterization	Simulated Physical Defect
Normal logic operation	Fault-free case, $gsN_3, dsN_3, gN_3 = dN_3 = sN_3,$ $dgN_4, dsN_4, gN_4 = dN_4 = sN_4, gsN_5, dN_5,$ $dsN_7 = beQ_2, gN_7 = dN_7 = sN_7, beQ_1, cQ_1, cQ_2$
Output SA1 or input SA0	$dsP_1 = dsP_2, gsN_4, cbQ_1, ceQ_1$
Output SA0	dgN_7, cbQ_2, ceQ_2
Output stuck open	$gP_1 = dP_1 = sP_1, gP_2 = dP_2 = sP_2,$ $gN_5 = dN_5 = sN_5, gN_6 = dN_6 = sN_6,$ $gsN_7, bQ_1 = eQ_1, bQ_2 = eQ_2$
Output at intermediate level	$dsN_5, dgN_6, gsN_6, dsN_6$
Output follows one of the inputs	$dgP_1 = dgN_3, dgP_2, dgN_5$
Output has a *soft* logic level	$gN_3, dsN_3, dgN_4, dsN_4$
Output is slow to rise/fall	$gN_3 = dN_3 = sN_3, gN_4 = dN_4 = sN_4,$ $dsN_7 = beQ_2, gN_7 = dN_7 = sN_7, beQ_1, cQ_1, cQ_2$
An increase in I_{DDQ} for certain test vectors	$dgP_1 = dgN_3, gsP_1, dsP_1 = dsP_2, dgP_2, gsP_2,$ $gsN_3, dsN_3, dgN_4, gsN_4, dsN_4, dgN_5, gsN_5, dsN_5,$ $dgN_6, gsN_6, dsN_6, dgN_7, gsN_7, cbQ_1, ceQ_1, cbQ_2, ceQ_2$

Table 3.3: *Fault characterization of the conventional BiCMOS NAND gate shown in Figure 2.2 [163].*

between the base and emitter nodes of Q_2 will be referred to as beQ_2, while a short between the drain and source of N_3 will be called dsN_3. A fault caused by an open in a node will be referred to by the first letter of that node, followed by the name of the transistor; e.g. an open in the collector of Q_1 is called cQ_1, while one in the gate of N_4 is gN_4.

The results of this simulation and other reports [59,106,110,160,172] suggest:

- Stuck-at fault test sets are not sufficient for testing BiCMOS circuits.
- Most open defects manifest themselves as either stuck-open or delay fault.
- Many physical failures will not affect the logical behavior of the gate. Therefore, functional testing alone will not be sufficient, because the inherent redundancy of BiCMOS circuits can mask the effect of some of the faults on the functional behavior of the circuit. Such faults, however, are manifested as performance degradation (delay, voltage degradation, or I_{DDQ}) faults.
- Short defects are usually accompanied by much higher currents than in the CMOS case. These defects can be detected by a technique based on observing leakage currents.

So, to achieve a high level of defect coverage, BiCMOS circuits should be tested for stuck-at, stuck-open, delay, and I_{DDQ} faults.

3.3 Test Generation Algorithm

The problem of fault test generation can be formulated as follows [71]. Given a description of the circuit and a list of faults, one must derive a sequence of input vectors, as short as possible, enabling the detection of some of the listed faults in the circuit. Finding a sequence of input vectors is an $\mathcal{NP}$-complete problem. This Section explains fault simulation, demonstrates the D-algorithm using an example, discusses $\mathcal{NP}$-completeness, and gives an overview of test generation algorithms for asynchronous circuits. The latter includes discussion on initialization, testing self-timed circuits, and illustration of a test generation algorithm inspired by Hazewindus [82].

3.3.1 Fault Simulation

Fault simulation consists of simulating a circuit in the presence of faults. Comparing the simulation results with those of the fault-free simulation of the same circuit simulated with the same applied test T, we can determine the faults detected by T.

One use of fault simulation is to *evaluate (grade) a test* T. Usually the grade of T is given by its *fault coverage*, which is the ratio of the number of faults detected by T to the total number of simulated faults [2]. This figure is directly relevant only to the faults processed by the simulator; hence, a test with 100% fault coverage may still fail to detect faults outside the considered fault model. *Defect coverage* is the probability that T detects any physical fault in the circuit.

The D-algorithm [161] was introduced in 1967. Subsequently, more efficient test generation algorithms such as 9-V algorithm [35], PODEM [75], and FAN [65] were developed. The D-algorithm is explained next for the following reasons. First, only the D-algorithm was utilized by Hazewindus [82] in his test generation algorithm for asynchronous circuits. Second, it presents common concepts to the class of deterministic test generation algorithms. These concepts are explained in this Section as forward and backward operations. These test generation algorithms produce test vectors by processing a model of the circuit. Third, it also represents a class of test generation algorithms which use a *fault-oriented* process: i.e., tests are generated for specified faults, namely, *single* stuck-at faults.

Let us simulate the circuit of Figure 3.4 for the test t=1001, both with and without the fault " G_2 SA1" present. The values of the signals are shown in the figure. The results that are different in the two cases have the form v/v_f, where v and v_f are corresponding signal values in the fault-free and in the faulty circuits, respectively. The fault is detected since the output values in the two cases are different. Test vector t that detects fault f *excites* f, i.e., *generates an error* by creating different v and v_f values at the site of the fault (G_2). Then, vector t *propagates the error* to a primary output (Z), i.e., makes all the lines along at least one path between the fault site and Z have different v and v_f values. In Figure 3.4 the error propagates along the path (G_2, G_4, G_5). Line G_2 is said to be *sensitized to the fault f by the test t*. The path (G_2, G_4, G_5) composed of sensitized lines is called a *sensitized path*.

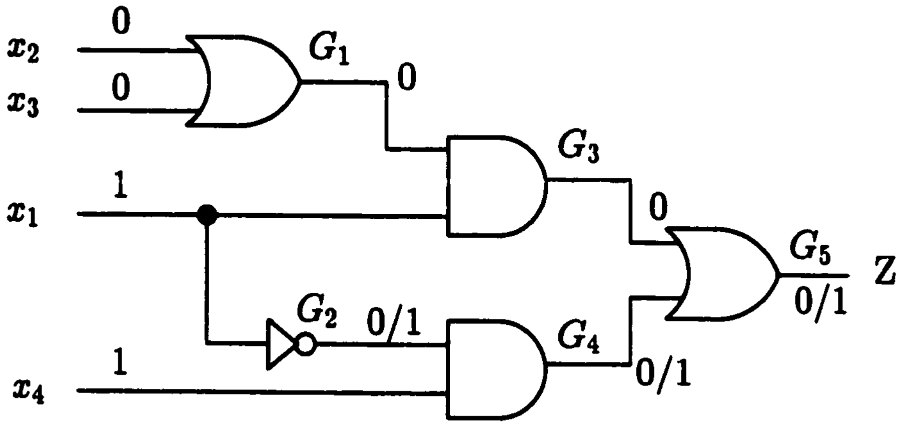


Figure 3.4: *A sample gate-level representation*

Test generation consists of two basic steps. First, a logical fault must be excited to its opposite logic value by applying primary inputs in the fault-free circuit. Second, the fault site must be made observable by at least one circuit output. Definitions for detectability, controllability, and observability are given below [33]:

Definition 3.1 *The detectability of a fault is the fraction of all possible input vectors which detect the fault.*

For an undetectable fault *f*, no test exists that can simultaneously excite *f* and create a sensitized path to a primary output.

Definition 3.2 *The controllability of a fault is the fraction of all possible input vectors which create a difference in the good circuit value and the faulty circuit value at the fault site.*

Definition 3.3 *The observability of a fault is the fraction of all possible input vectors which sensitize the fault site to at least one observable circuit output.*

3.3.2 $\mathcal{NP}$-Completeness

The class of problems solvable in polynomial time is usually denoted by $\mathcal{P}$. The letter $\mathcal{NP}$ stands for "non-deterministic polynomial-time" [9, 72]. In particular, $\mathcal{NP}$ does not mean "non polynomial time". The class of $\mathcal{NP}$ includes most combinatorial optimization problems, including all problems that are in $\mathcal{P}$. Checking the correctness of the answer in polynomial time means checking it in time bounded by a polynomial in the original input.

Within the class $\mathcal{NP}$ there are "$\mathcal{NP}$-Complete" problems. These are by definition the hardest problems in the class $\mathcal{NP}$. This implies that if one $\mathcal{NP}$-complete problem can be proved to be solvable in polynomial time, then *every* $\mathcal{NP}$-complete problem can be solved in polynomial time.

The theory of $\mathcal{NP}$-completeness does not provide a method for obtaining polynomial-time algorithms for $\mathcal{NP}$-complete problems. Neither does it say that algorithms of this complexity do not exist. A problem that is $\mathcal{NP}$-complete has the property that it can be solved in polynomial time if all other $\mathcal{NP}$-complete problems can also be solved in polynomial time. (It has not been proved or disproved that $\mathcal{NP}$-completeness implies exponential complexity. Hence, a polynomial time solution to an $\mathcal{NP}$-complete problem has not been found. Consequently, if such a problem is attacked with a method of less-than-exponential complexity, it is extremely likely that some penalty will be paid. For example, some irredundant faults are not detected.)

Surprisingly, there are a great many prominent combinatorial optimization problems that are $\mathcal{NP}$-complete, like the traveling salesman problem. One generally distinguishes between the polynomially solvable problems and the $\mathcal{NP}$-complete problems. For almost every combinatorial optimization problem one has been able either to prove that it is solvable in polynomial time, or that it is $\mathcal{NP}$-complete.

Test generation is a complex problem with many interacting aspects. The most important are:

- the cost of test generation;
- the quality of the general test;
- the cost of applying the test.

It is well known that the fault detection problem is $\mathcal{NP}$-complete [66, 67, 88, 90, 93, 157]. Ibarra and Sahni [88] have shown that the problem of generating tests to detect single stuck-at faults in a combinational circuit modeled at the gate level is an $\mathcal{NP}$-complete problem. The complexity of this problem depends on the topology of the network. Moreover, if the circuit is sequential, the problem can become even more difficult depending on how many state-holding components (such as Flip-Flops) are used.

There are two possible solutions for test generation problems, namely *heuristic* algorithms and *branch-and-bound* methods. Branch-and-bound is a general scheme that requires two main decisions: How to branch and how to bound. The following items describe a branch-and-bound algorithm.

- The algorithm finds optimal solutions theoretically.
- Because of backtrack operations in such algorithms, they are prohibitive due to execution time.
- It is possible to improve execution time using the upper-bound on objective function(s).
- Branch-and-bound algorithms are advised for small size samples.

The search for a solution involves a decision tree. A decision node denotes a problem that the algorithm is attempting to solve. A branch leaving a decision node corresponds to a decision, i.e., trying one of the available alternative ways to solve the problem. However, a decision may have been selected which leads to an inconsistency or it encounters a state that precludes further error propagation. Such terminal node is called *failure*. A *success* terminal node represents finding a test. An important property of this algorithm is that it is exhaustive, that is, it is guaranteed to find a solution (test) if one exists because it enumerates all possible solutions. Thus, if the algorithm fails to generate a test for a specified fault, then the fault is undetectable. Because of the exhaustive nature of the search process, the worst-case complexity of such algorithms is *exponential*: i.e., the number of operations performed is an exponential function of the number of gates in the circuit.

This happens when many attempts have failed and the test vector is found after much searching is done. The best-case behavior occurs when the result is obtained without backtracking. Then the number of operations is a linear function of the number of gates. From analyzing the worst-case and the best-case behaviors of an algorithm we conclude that the key factor in controlling the complexity of a test generation algorithm is to minimize the number of incorrect decisions. Heuristic algorithms have been devised to attempt such an approach.

Heuristics are methods which cannot be guaranteed to produce optimal solutions, but which, hopefully, produce nearly-optimal solutions for most practical applications of that specific problem. The following items describe a heuristic algorithm.

- At each step, the next move is selected based on the available information.
- A test is performed after each selection. This test is performed using an objective function.
- Such an algorithm does not carry a thorough search of the solution space.
- The effectiveness of heuristic algorithms is based on the selection rules.

Selection criteria are helpful in speeding up the search process. These selection criteria are based on the following principles:

- Among different unsolved problems, first attack the most difficult one.
- Among different solutions of a problem, first try the easiest one.

Selection criteria differ mainly by the cost functions they use to measure "difficulty." Typically, cost functions are of two types:

- controllability measures, which indicates the relative difficulty of setting a line to a value;
- observability measures, which indicates the relative difficulty of propagating an error from a line to a primary output.

The saving realized by using cost functions should be greater than the effort required to compute them. In general, cost functions are computed by a preprocessing step and are not modified during test generation process.

Any cost function should show that primary inputs are the easiest to control and the difficulty of controlling a line should increase with its distance from the primary inputs. Similarly, primary outputs are the easiest to observe and the difficulty of observing a line should increase with its distance from the primary outputs.

Branch-and-Bound techniques provide high quality sequences of test vectors, but the run time makes its use prohibitive for some applications. On the other hand, heuristic algorithms provide results in a run time proportional to the size of the circuit to the power of N, $(1 \leq N \leq 4)$. However, the results obtained may be nearly-optimal.

The D-algorithm [161], the 9-V algorithm [35], PODEM [75], and FAN [65] are branch-and-bound solutions to test generation problem of single stuck-at faults. Algorithms presented in [3, 5, 24, 37, 76, 162] are heuristic solutions to test generation problem. Algorithms presented in [89, 153] combine the computation of cost functions with a partial test generation using branch-and-bound technique.

Applying a sequence of test vectors is *not* an $\mathcal{NP}$-complete problem. However, the test time has direct relationship with the size of the test set.

3.3.3 Initialization

Both fault-free and faulty sequential circuits may start in an arbitrary state during the powering up and testing of circuits. Initialization is the process of driving the state signals in the circuit to known states.

Testing of sequential circuits is usually divided into two distinct phases. In the first phase we apply an initialization sequence such that the fault-free and faulty-circuits are brought to known states. If the faulty circuit fails to be initialized then a fault is already detected. The output of the circuit is not important and can be ignored until the initialization phase is completed. In the second phase we apply a test sequence to both faulty and fault-free circuits and observe the primary outputs. If the responses are different then the presence of a fault in the faulty circuit is detected. The second phase of testing is based on the assumption that the first phase for initializing the circuit is successfully completed.

For many designs initialization can most easily be achieved by providing explicit asynchronous preset (PR) or clear (CLR) inputs to a flip-flop (FF). This needs one extra primary input and some additional logic. This method, although inherently asynchronous, is applicable only to circuits which use state-holding elements such as FF's. If the resetting process is performed at the same time as powering up, a race may occur and the flip-flops may go to either 0 or 1 state unpredictably [2,25]. For this reason, additional hardware is used to properly delay the resetting process. Also, if the circuitry associated with the initialization of a circuit fails, test results are usually unpredictable, and the possible failures are hard to diagnose.

Several algorithms, such as the D-algorithm, have been adapted to find the initialization sequence. Unfortunately, there are designs for which no initialization sequence exists. Such circuits[3] are sometimes tested by employing a homing sequence, i.e., a sequence such that the state of the circuit is deduced by observing the output and comparing it with the input sequence, which is not an easy method to use in general cases. Finding the initialization sequence is not always practical since the initialization sequence can itself be lengthy. In addition, a large amount of computation may be needed to discover such a sequence.

Chakradhar *et al.* [36] have shown that automatic synthesis of asynchronous circuits may result in a non-initializable realization of the circuit since they intend to minimize a logic or hazard-free implementation. They have illustrated a method which involves re-designing a given flow table if some "don't cares" are available. They have presented an implicit enumeration technique to search the entire space and to selectively assign "don't cares" to obtain an initializable implementation. The method works as follows. First, Karnaugh maps representing the next-state functions of the non-input signals are derived from signal transition graphs (STGs). Their algorithm, then, begins by marking all primary inputs and all non-input signals as unknown. An initialization tree is constructed as follows. At the top level, this tree has as many nodes as the non-input signals. A node corresponds to a partial assignment of primary inputs and "don't cares" on the non-input signals. A node has as many branches as initialization vectors based on the present node assignment and currently non-initialized non-input signals. If a particular partial assignment of primary inputs and "don't cares" is unable to initialize the non-input signal at the node under consideration, then the node is marked as a *failure* node, the search after that node is terminated, and the partial assignment is discarded. The process may also lead to a node where all non-input signals are initialized.

[3] Ring counters and T flip-flops are simple examples of this type of circuits. Note that a ring counter goes through a periodic state sequence.

Such a node is marked as a *success* node and every path from the top level node to such node corresponds to an initialization vector.

3.3.4 Testing Classical Asynchronous Circuits

One way to generate test vectors for classical asynchronous sequential circuits is to conceptually (i.e., not physically) transform the sequential circuit into a combinational circuit and then to apply combinational circuit test generation algorithm [87]. In this transformation, all state-holding elements are considered as combinational gates with an extra input (Q_{in}), representing the present state, and an extra output (Q_{out}), representing the next state of the element. Then, the sequential circuit is transformed into a combinational one by making several copies of the circuit called *time frames*, connecting the next state-holding element in one time frame to the present state input of the same element in the succeeding time frame, forming an *iterative array*. We can then apply any test generation algorithm for combinational circuits to the iterative array to generate a test vector for the sequential circuit. The number of time frames that are necessary to detect a given fault determines the number of test vectors needed to test for the fault. The major problem for this test generation approach is that the number of time frames necessary in the worst case grows exponentially when the number of state-holding elements increases.

3.3.5 Self-Checking circuits

Self-timed circuits are *self-checking circuits* [17, 42, 87, 182] or *self-diagnostic* [15, 44], and are easier to test. They promise several advantages over other circuits, including on-line testing. In particular, some stuck-at faults are detectable because the circuit simply halts at the presence of such a fault. The fundamental idea behind self-checking asynchronous circuits is that if a node performs both positive and negative transitions, any static stuck-at fault will inhibit one of these transitions. Conceptually, a static stuck-at fault corresponds to a circuit node with an infinite delay. If nodes of a circuit are related in such a way that the transition of one node depends successively on a transition of the previous node, a static stuck-at fault will cause the circuit to halt. In an asynchronous circuit using handshaking, the fault is detected if the expected acknowledge signal does not appear at a finite time after the request has been sent.

3.3.6 Testing Quasi-Delay-Insensitive (QDI) circuits

The gate-level realizations of DI circuits are limited, and their transistor-level realizations, although they are feasible [107], are quite difficult to achieve. SI circuits, on the other hand, are practical if the wire delays are negligible compared to the gate delays. For VLSI purposes where the length of wires is comparable to the size of a gate, SI circuits fail to function properly. Quasi-Delay-Insensitive (QDI) design methodology, however, is the most practical method to date since they have been used to design small components such as D-element [119], medium size circuits such as DCC error corrector by Philips research laboratory, and large circuits such as microprocessors [121, 177] successfully. The circuit operates independently of gate and wire delays except for some forks which are assumed to be isochronic[4]. Such an assumption is very practical because isochronic forks are easy to implement locally, and the wire length on branches of isochronic forks can be made negligible compared to the length of other wires and the size of gates. Other parametric variations such as threshold voltages of transistors at the end of branches of the isochronic

[4] The isochronic fork assumption states that a transition at the input to a fork arrives at the end of the branches of the fork at the same time or with a delay less than one gate delay.

forks can be unified or controlled [19] so that the difference in arrival times is less than one gate delay.

Recently, Hazewindus [82] has studied QDI circuits. Stuck-at faults in non-redundant QDI circuits will either halt the circuit or cause a *premature firing.* A transition that is supposed to occur but does not because of a stuck-at fault is *inhibited.* A fault that causes an inhibited transition will eventually cause the circuit to halt. A premature firing is a signal that changes too early according to the specification. In [82], the conditions for inhibiting (INH) and premature (PRE) firing faults are derived. Then an input sequence is found to satisfy the conditions. Consider an AND gate, with inputs a and b and output c, which is specified as follows[5]:

$$a \wedge b \rightarrow c\uparrow,$$

$$\neg a \vee \neg b \rightarrow c\downarrow.$$

Consider the fault a-stuck-at-1 (or briefly, a-sa1). This fault can inhibit $c\downarrow$ if c and b are high, and a is low. In this state c will go down in a fault-free circuit, but will remain high in the presence of the fault. Therefore, the condition to halt the circuit in the presence of a-sa1 is:

$$INH_{c\downarrow} = \neg a \wedge b \wedge c.$$

The same fault can cause a premature firing of $c\uparrow$ if a and c are low, and b is high. A condition for the premature firing of c is derived as

$$PRE_{c\uparrow} = \neg a \wedge b \wedge \neg c.$$

The next step is to determine a test sequence that will put the circuit into a state where the conditions hold, and the effect of the fault must also be propagated to

[5] This notation is called *production rules.* A production rule consists of a Boolean guard and an assignment: *guard* → *assignment.* When the guard is true the assignment can be performed.

a primary output. Since these procedures are not always possible, not all faults are testable. Adding extra test points is suggested to ease the observability and testability of such circuits.

3.4 Design-for-Testability

Through-the-pins [147] testing has some difficulties. The memory allocated, the time for generating test patterns, and the cost of equipment for applying them may be immense. One way to reduce the testing time is to design circuits to be easily testable. Several Design-for-testability (DFT) techniques have been proposed in the literature [2,67,99,189].

The design for testability techniques are divided into two categories [190]. The first category is the *ad hoc* techniques which solve a problem for a given design and are not generally applicable to all designs. The second category is the *structured approaches* which are generally applicable and involve a set of design rules.

One important ad hoc design for testability is circuit partitioning [10,80]. It involves augmenting a circuit and effectively dividing it into smaller circuits. This technique is very useful since the task of test pattern generation and fault simulation for single stuck-at faults is polynomially proportional to the number of logic gates[6] [74]. Partitioning reduces the number of required inputs for fault coverage. Thus,

[6] A quick rationale goes as follows: with a linear increase k in circuit size comes a linear increase in the number of failure mechanisms. (Since each line can be either stuck-at zero or stuck-at one, the number of single stuck-at faults is $2n$ for a circuit which has n lines.) This yields k squared increase in work. Also, as circuits become larger, they tend to become strongly connected such that a given block is affected more by other blocks and even itself. This causes more work to be done in a range, yielding k cubed. This fairly nebulous concept of connectivity seems to be the cause for debate on whether the exponent should be 3 or some other value. This relationship does

the objective of partitioning techniques is to minimize the required test space at the expense of additional test points. There are a number of ways to implement the partitioning approach. To name a few, mechanical partitioning by dividing a network in half, using jumper wires, and degating. Optimal circuit partitioning is an $\mathcal{NP}$-complete problem. Typically optimization techniques coupled with heuristic algorithms are used unless some design rules are forced to choose the partitioned nodes.

An example of partitioning where some rules are applied is to break any feedback variable, insert a controllability element, and convert sequential circuits into combinational ones, see Figure 3.5. The controllability element in test mode effectively breaks the feedback loop, allowing the circuit to be tested as a combinational circuit. However, the circuit operates as intended while in normal mode of operation.

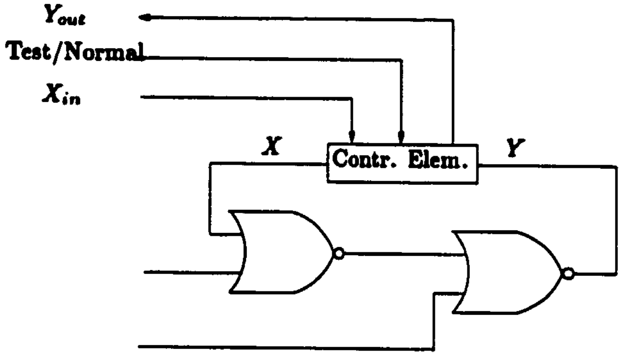


Figure 3.5: *Partitioning Logic Circuits.*

Another easy ad hoc technique is to add extra test points [80,81] or additional circuit inputs and outputs to be used during testing. If the test point is used as a

not take into account the sequential complexity of the network. This rationale is best visualized with the aid of a combinational circuit with *only a few* fanout points.

primary input, then it can enhance controllability. If it is used as primary output then it enhances the observability of the network. There is always a cost associated with adding test points. For circuit boards, the cost of test points is often well justified. On the other hand, for ICs the cost can be prohibitive because of IC pin limitation.

Built-in self test (BIST) [6,7] is a DFT technique in which (pseudo-random) test pattern generation and output response analysis are both accomplished through built-in (added) hardware features. One such technique was proposed by Hewlett Packard and is known as *signature analysis* [101]. The output patterns produced by a pseudo-random number generator are applied to the circuit under test. The output data is fed into an n-bit *linear feedback shift register* (LFSR). Selected taps of the LFSR are fed back to the input via XOR gates. After the data stream has been clocked through, a residue is left in LFSR. This residue is intended to be unique to the data stream and is known as its signature. The fault is detected if the circuit signature is different than that of the fault-free circuit. The problem with signature analysis is explained as follows. An n-stage signature generator can generate 2^n signatures. However, many input sequences can map into one signature. In general, if the length of an input sequence is m and the signature generator has n stages, then 2^n input sequences map into 2^n signatures. In other words, 2^{m-n} input sequences map into each signature. Only one out of 2^m possible input sequences is error-free and produces the correct signature. However, any one of the remaining $2^{m-n} - 1$ sequences may also map into correct signature. This mapping gives rise to ***aliasing***, that is, the signature generated from the faulty output response of a circuit may be identical to the signature obtained from the fault-free response. In other words, the presence of a fault in the circuit is masked.

There are synthesis tools that produce testable designs [18, 46, 48, 91, 99, 156, 158, 167, 187].

The most structured method is the scan-design technique [2, 67] which is presented in Section 3.4.2.

3.4.1 Design-for-Testability of CMOS/BiCMOS Circuits

Several design for testability (DFT) techniques [2, 55, 67, 92, 99, 154] have been developed for bipolar and CMOS technologies to reduce the complexity and cost of testing. In this Section we describe some ad hoc I_{DDQ} DFT techniques and delay fault testing techniques of conventional CMOS/BiCMOS logic circuits, see Figure 2.1.

I_{DDQ} Fault Testing Techniques

The first DFT technique for conventional BiCMOS circuits is proposed in [163]. It is a built-in current sensor (BICS) for detecting excessive I_{DDQ} current. The technique is illustrated in Figure 3.6 for BiCMOS inverters, although it could be employed for any conventional BiCMOS circuit.

BiCMOS gates are modified with the addition of transistor Q_t, whose base and emitter nodes are connected to the base and emitter nodes of the pull-down transistor Q_2. The collector of Q_t is connected to the multi-emitter transistor Q_M. The multi-emitter transistor is the input of a TTL-like NAND gate. For a circuit, consisting of n different BiCMOS gates, a multi-emitter transistor with n emitters will be connected to n different BiCMOS gates, through the added current-mirror like transistors Q_t.

Qt and Q2 carry proportional currents when a fault causes an abnormal current between VDD and GND. This is due to the mirror effect. This current will discharge

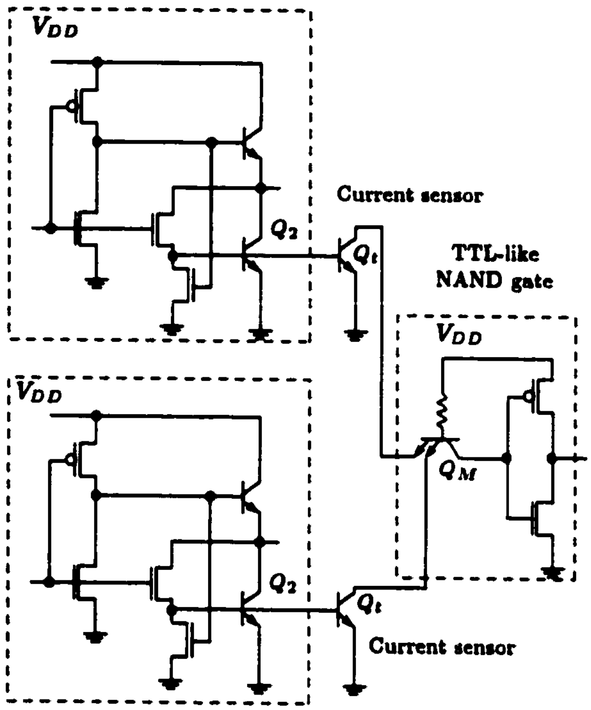


Figure 3.6: *Built-in Current Testing Circuit.*

the input capacitor of the CMOS inverter and its output will be at logic 1. On the other hand, for fault-free circuit operation, the steady state current that passes in Q_2 (and the proportion of it that passes in Q_t) is very small and the output of the CMOS inverter will be at logic 0, indicating fault-free operation. Thus, any fault that causes an abnormal current can be detected by checking the output of the added TTL-like NAND gate.

The method explained in this Section is capable of detecting all the defects that increase the leakage current in a pull-up/pull-down (PUPD) BiCMOS circuit, provided that the excessive leakage current is reflected in the output (bipolar) stage. This assumption, however, might not always be true. Defects can exist that establish a path between V_{DD} and GND at intermediate stages of BiCMOS, leading to an increase in I_{DDQ}, without increasing the output stage current. Also, some bridging faults may not be detected by this technique.

Increased DC current is also detected through monitoring the current flowing from V_{DD} to ground. Several design schemes for testing have been proposed [77, 109,154] for CMOS circuits. Figure 3.7 shows the basic concepts of these schemes. Since all the circuits operate on a similar basis, we explain the circuit shown in (a).

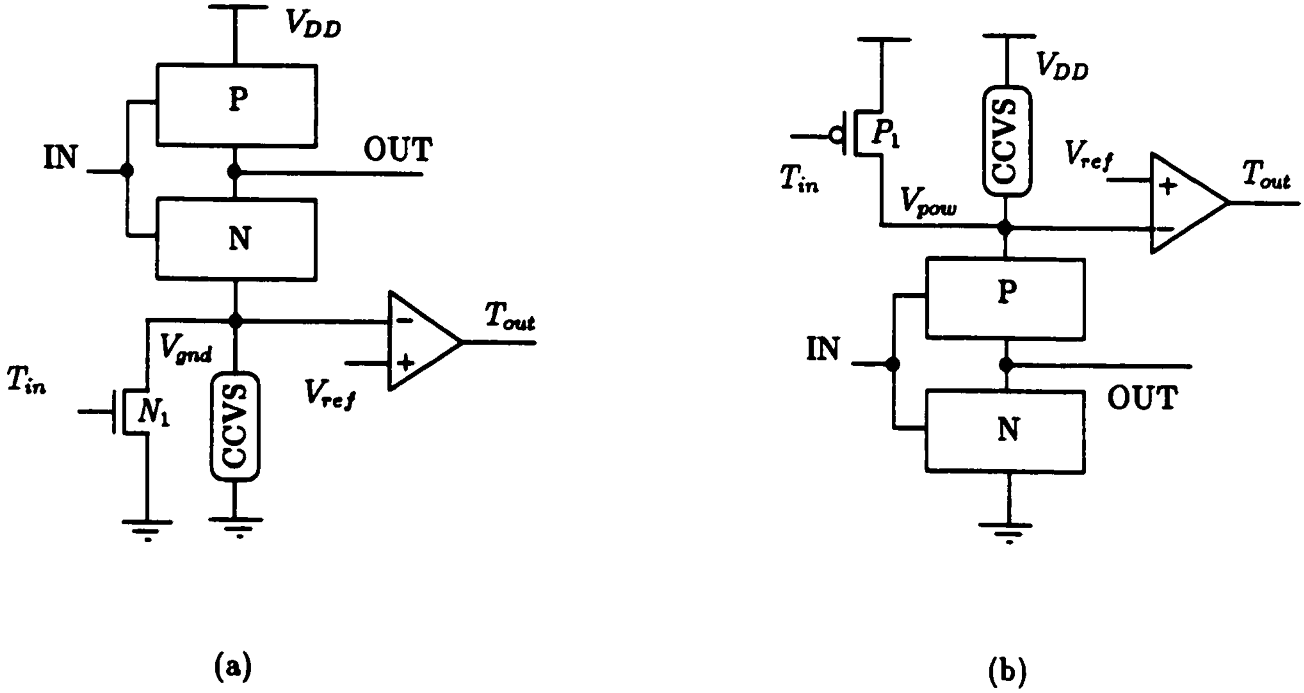


Figure 3.7: *I_{DDQ} testable CMOS gates proposed in literature by checking the increased DC current through: (a) V_{DD}, (b) Ground .*

In this method, the system ground, V_{gnd}, is separated from actual ground by an NMOS transistor, N_1. The gate of this transistor is controlled by an external input signal. In normal operation, T_{in} is set to logic 1 so that the added components for test purposes do not affect the normal operation of the circuit. Furthermore, N_1 is designed such that its *on resistance*, (R_{on}), is much smaller than the *off resistor* of the test circuit, realizing a strong short between system ground, V_{gnd}, and actual ground. In test mode T_{in} is low and N_1 is off. Its *off resistor* is much larger than the resistor introduced by the test circuit at V_{gnd} and ground. The inputs are applied such that the I_{DDQ} fault is excited. The DC current flowing from V_{gnd} to actual ground will increase if the circuit is faulty, while it will remain unchanged for the fault-free circuit. Current-Controlled Voltage-Source (CCVS) or the current sensor will translate current to a voltage. As the current changes from virtually zero (almost nanoampers) to a few microampers, the voltage at the upper node of CCVS changes from zero to a voltage higher than V_{ref}. This excessive voltage will then be detected by an analog comparator.

The circuit shown in Figure 3.8 works on a similar basis. This technique was proposed by Favalli *et al.* [58, 59, 104]. This test circuit works the same way as explained above in normal mode of operation. In test mode, N_i' plays the rule of current sensor and the analog comparator at the same time. The drain of the transistor N_i' is connected to V_{DD} by PMOS type resistor. If the circuit is fault-free, there is no excessive stable current, hence, the gate voltage or the system ground is theoretically "0". When a fault realizing a short path between V_{DD} and system ground exists, the gate of this transistor is charged to a voltage high enough to turn the NMOS transistor on.

A disadvantage of this built-in current sensor is the impact of the test circuit on the operation of the system in test mode. The test circuit introduces an inevitable impedance in the ground path of the IC device. The residual voltage drop between system ground and actual ground would cause the circuit to malfunction even if it is fault-free. The tester introduces substantial parasitic capacitance increasing the time constant at the system ground, and slowing the rate of I_{DDQ} testing.

A circuit idea for solving these problems is introduced in Chapter 4. We will

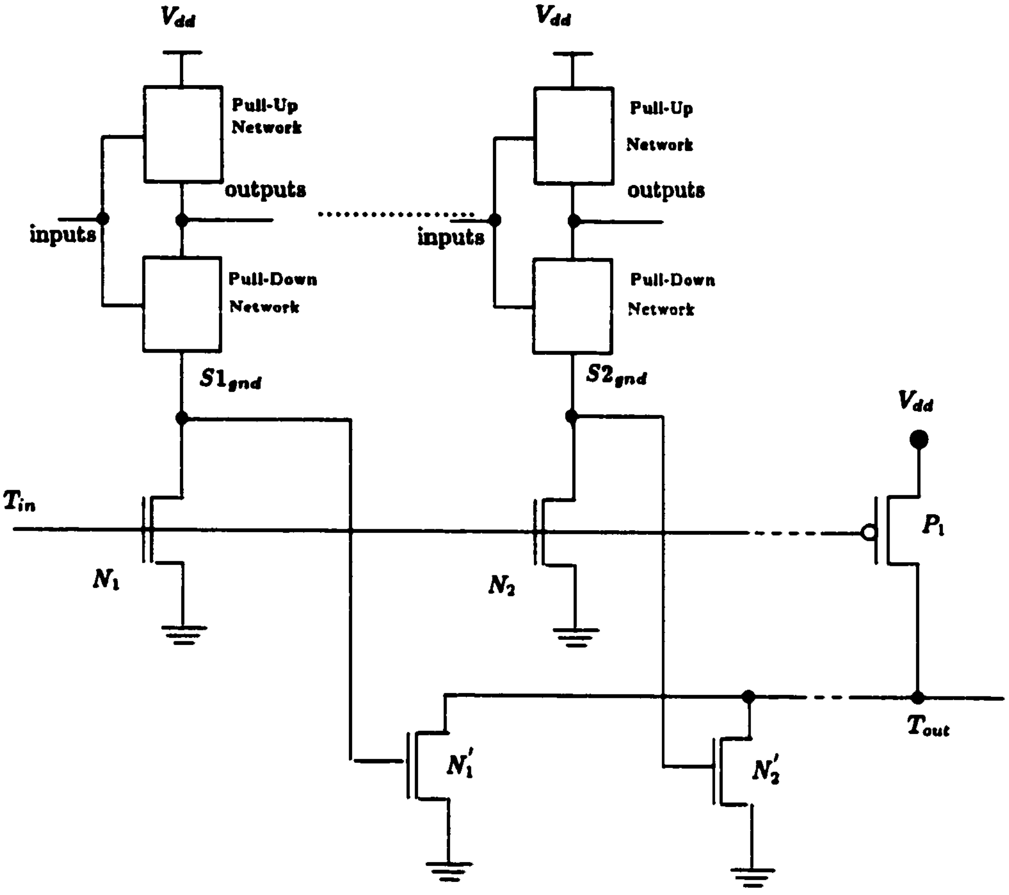


Figure 3.8: *Built-in Current Testing Circuit.*

propose a test circuit which works on-line.

Delay Fault Testing Techniques

The simplest form of delay testing is to apply pseudo-random patterns to the circuit under test (CUT) at desired speed. Consider the simple combinational circuit in Figure 3.9 implementing the function $F = a\bar{b} + \bar{b}c + b\bar{c}$. The delay (for a rising transition) of the path from a to F can be tested by the two vectors $V_1 = (a, b, c) = (0, 0, 0)$, and $V_2 = (1, 0, 0)$.

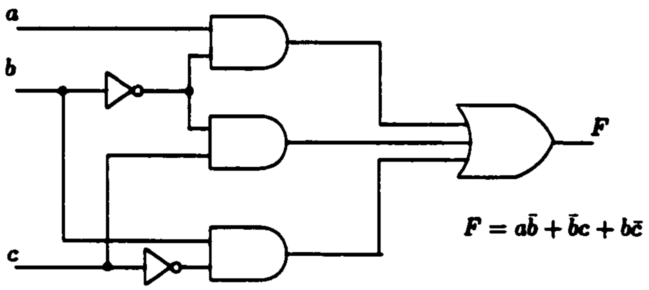


Figure 3.9: *Two-pattern test for delay faults in logic circuits.*

Two-pattern tests have been widely used for testing stuck-open and delay faults in logic circuits. There are a number of drawbacks, however. This method requires very long test lengths to achieve an acceptable fault coverage [164]. The reason is that faults need to be sensitized through long paths. The other drawback of this method is that tests can be invalidated by delays in other parts of the circuit, hazards, and timing skews [92, 154]. A test that detects a delay fault of a certain size does not necessarily detect faults that have longer or shorter delays [60, 64]. As a result, there have been efforts to eliminate the need for them. One such effort is the design-for-testability technique proposed in [64] for detecting delay faults in CMOS circuits. The authors discuss the importance of observing the output waveform between the samples, instead of only latching the output at the sampling time. Figure 3.10 shows how a delay test, based on latching the output at specific sampling times can be invalidated due to a dynamic hazard.

Since there is significant information in the output waveform between samples, they propose to perform delay testing by applying test patterns and sampling the output in the conventional manner, together with a circuit that monitors the output waveform between samples. In order to observe any changes after the sampling time, they have proposed the DFT technique, shown in Figure 3.11. This circuit uses the

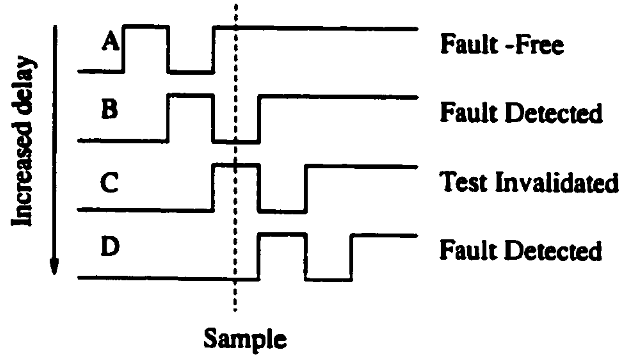


Figure 3.10: *Delay test invalidation by dynamic hazard.*

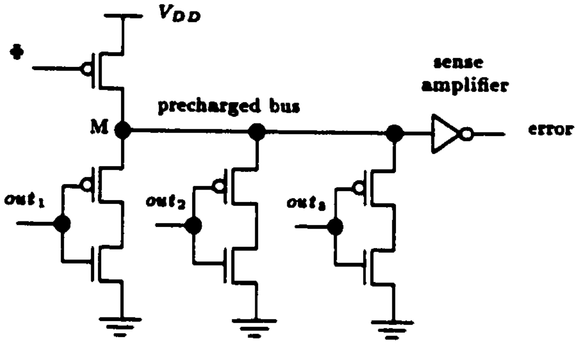


Figure 3.11: *DFT technique proposed in [64] for post-sampling analysis.*

transient switching current in CMOS inverters to detect signal changes. While Φ is low, the bus is kept at V_{DD} by the precharging transistor, so the inverters operate normally. Φ changes to logic high, when all the nodes of interest in the CUT (circuit under test) are supposed to have stabilized in the fault-free case. Once Φ is high, the bus is left floating. If out_i does not change, the corresponding inverter will draw negligible static current and the bus will remain high. If any out_i changes, the transient current while the inverter changes state will partially discharge the bus, which can be detected with a sense amplifier.

Figure 3.12 shows another DFT technique proposed in [58]. Here, there are two controlling signals, ϕ_1 and $\phi_2 = \bar{\phi}_1$. The rising transition on ϕ_1 and the falling transition on ϕ_2 signals announce the end of the valid reading time, t_{sample}, for the circuit signal, S. T_{out} is generated through an intermediate logic circuit. T_{out} is "1" if S does not show a transition after $t > t_{sample}$. If S changes after the transition on ϕ_1 and ϕ_2 due to a delay fault, the extra logic circuit will set T_{out} to zero and the fault is detected.

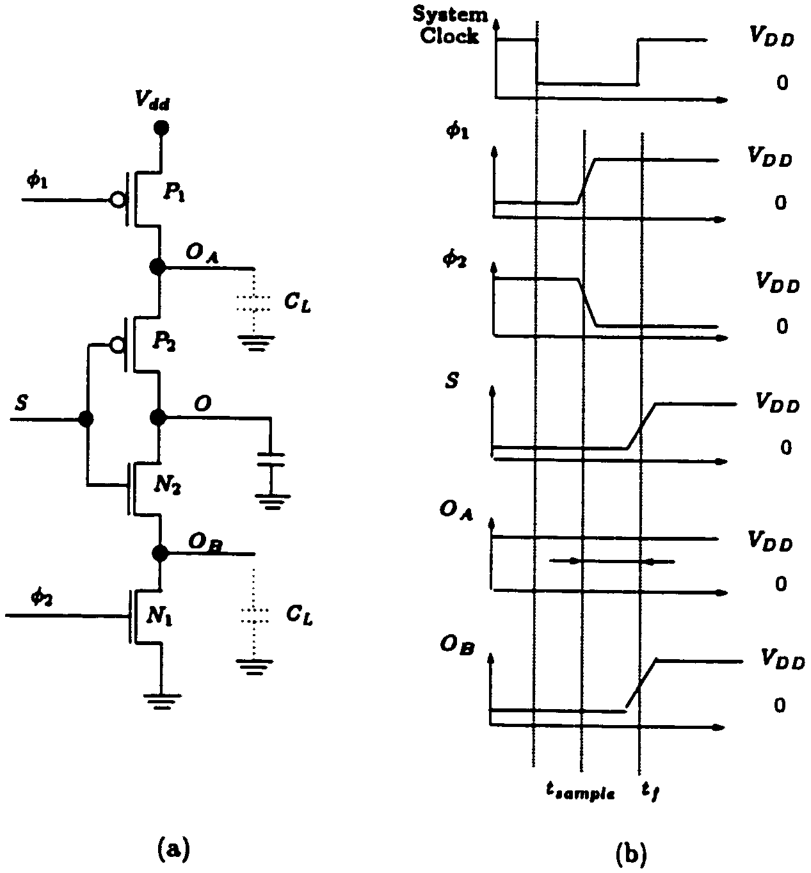


Figure 3.12: *(a) Delay testing circuit, (b) Schematic timing diagram as presented in [58].*

The DFT techniques explained for delay fault testing use controlling signal(s)

ϕ. If this signal is generated for each path, the result would be added circuitry and significant area overhead. For this reason, this signal is generated from the main system clock with simple circuitry. ϕ_i is generated regardless of whether the fault exists or not, or the circuit under test is active or not. These techniques are, therefore, synchronous.

The last transistor-level DFT method is a charge/discharge test path, proposed in [38, 77, 109, 144, 154] to facilitate testing of stuck-open faults in conventional CMOS/BiCMOS circuits. Two-pattern tests have been widely used for testing stuck-open faults in CMOS logic circuits. However, they can be invalidated due to circuit delays, hazards, and timing skews [92, 154]. Several design schemes for *robust* stuck-open test have been proposed for CMOS circuits. Figure 3.13 shows the basic concepts of these schemes. In one particular case, the signals C_p and C_n are tied together, and are called C_1. In the left-hand side circuit, in order to test for a stuck-open in the p-part, C_1 is first set to ONE and the inputs are applied such that the fault-free output is ONE, but the stuck-open output is floating. Since C_1 is high, the p-part is blocked by the PMOS transistor. On the other hand, the NMOS transistor (parallel with the n-part) forces the output to a low value. Next, C_1 is changed to ZERO. In the fault-free circuit, the output will charge to V_{DD}

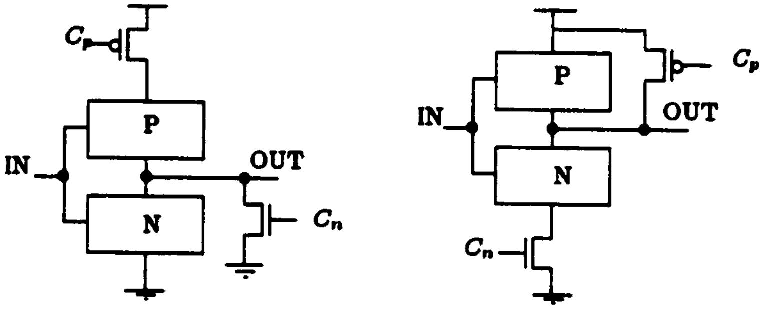


Figure 3.13: *Testable CMOS gates proposed in [38, 77, 109, 144, 154].*

via the p-part and the PMOS transistor, while for the circuit with a stuck-open fault, the output will keep its previous low value. For detecting a stuck-open in the n-part, the C_1 signal will change from ZERO to ONE, and the input test vector should be chosen such that stuck-open in the n-part can be tested. For the circuit in the right-hand side, the values of C_1 in the two cases are opposite to the values described above.

The test generation for the circuit shown in Figure 3.13 is less complex. In this circuit, a two- or multi-pattern robust sequence can be obtained by appropriately controlling the signals C_P and C_n. However, because the circuit shown in Figure 3.13 requires two-pattern test sequences, glitches caused by the delays in the prior logic may invalidate the tests [150].

A circuit idea for solving this problem was first introduced in [92] for CMOS circuits, and was later adopted for BiCMOS circuits in [130, 144]. It uses a single test vector and avoids the high-impedance state during testing. As a result, the test cannot be invalidated due to glitches. The circuit modification for a two-input BiCMOS NAND gate is shown in Figure 3.14.

Suppose that a defect occurs that forces the output of the gate to be stuck-open. According to Table 3.3, this defect can be any of the following opens or shorts:

- Opens at the source, drain, or gate terminals of the MOS transistors in the p-part (P_1, P_2) and the second n-part of Figure 2.1 (N_5, N_6),
- Opens at the base or emitter terminals of Q_1 and Q_2,
- Gate to source short in N_7.

Let us assume that P_2 is stuck-open. Then for the input pattern AB = 10, the output floats and assumes a value of 0 or 1 depending on its previous state. This

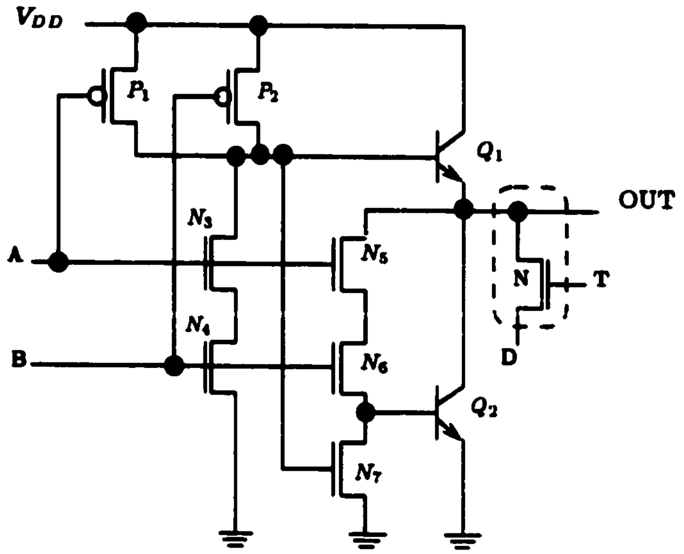


Figure 3.14: *Modified circuit for detecting stuck-open faults.*

fault can be detected by the two-pattern test sequence AB = {11, 10} [7], since the fault-free gate output will be 01, whereas the faulty gate output is 00.

The proposed circuit modification provides, during testing, an additional path for charging or discharging the output of the gate whenever it is floating. Thus, it removes the initialization requirement and allows stuck-open faults in BiCMOS logic circuits to be detected by single test patterns. The additional path is only excited during testing and does not affect normal circuit operation. In a complex circuit consisting of several BiCMOS gates, each gate must be augmented by an additional NMOS transistor, N, as shown in Figure 3.14.

The augmented BiCMOS gate facilitates testing of stuck-open faults using single test vectors, as explained below.

In normal operation, T is set to logic 0 so that the added transistor N is off and does not affect the normal operation of the circuit. Furthermore, N is designed such

[7] The output is 0 for the input AB=11, then it changes to 1 for AB=10.

that its *on resistance*, (R_{on}), is much higher than that of n-part 2, see Figure 2.1, Q_1, and Q_2. In addition, its *on resistance* must also be much smaller than the *off resistance*, (R_{off}), of n-part 2 (Figure 2.1), Q_1, and Q_2.

For detecting a stuck-open fault in the pull-up section (e.g., P_1, P_2, or Q_1 permanently off), the following single test vector is applied: T=1, D=0, inputs=such that output is set to 1 in fault-free circuit (e.g., at least one input equal to 0 for the NAND gate). Since T=1, N conducts and provides a pull down path for the output. In the fault-free circuit, Q_1 is on and Q_2 is off. The output is, therefore, being pulled up through Q_1 and pulled down through N. But since the *on* resistance of N is much higher than that of Q_1, the output is pulled up and attains logic 1. In the faulty circuit, however, Q_1 is off (due to the fault) as well as Q_2, so the output is pulled down through N and attains logic 0. Hence, this test vector produces an output of 1 in the fault-free circuit and an output of 0 if any transistor in the pull-up section is stuck-open. Similarly, a stuck-open in the pull-down section (e.g., N_5, N_6, or Q_2 permanently off) can be detected by the single test pattern: T=1, D=1, inputs=such that output is set to 0 in fault-free circuit (e.g., both inputs equal to 1 for the NAND gate).

Transistor N can be replaced by a PMOS transistor. In that case, T is set to logic 1 in normal operation, and changes to logic 0 in test mode.

3.4.2 Design-For-Testability of Asynchronous Circuits

We discuss scan-design technique in detail. Its use in testing micropipelines is studied in depth.

Scan technique

The most important structured technique is the *scan* technique [34, 54, 94, 191]. Scan techniques attempt to add either additional observability, controllability, or both. They have the property of using some sort of scan method to permit access to the internal nodes of a circuit without requiring a separate external connection for each node accessed. Very few (typically from one to four) additional external connections are used to access many internal nodes at the expense of increased time for testing. This is made possible at the cost of additional internal logic circuitry used primarily for testing.

Besides increased accessibility, the scan-path technique has another very important benefit: it is possible to test the entire circuit outside the scan path by generating test patterns for only its combinational portions. The scan-path itself is tested by a simple procedure, independent of the specific circuit in which the scan path is embedded, and therefore does not require any explicit ATPG. In this technique the circuit is designed so that it has two modes of operation: Normal mode and test mode. With the circuit in test mode, it is possible to shift an arbitrary test pattern into the flip-flops. By returning the circuit to normal mode for one clock period, the combinational circuit can act upon the flip-flop contents and primary input signals, then stores the results in the flip-flops. If the circuit is then placed into test mode, it is possible to shift out the contents of the flip-flops and compare these contents with the correct response.

Typical observability scan hardware consists of n latches whose data come from required observation points in the circuit, see Figure 3.15. Data is latched from the circuit by the *Latch Data signal.* Once the data is latched, it is shifted out serially by clocking *Shift Data Out* signal. A designer can also add registers for controlling

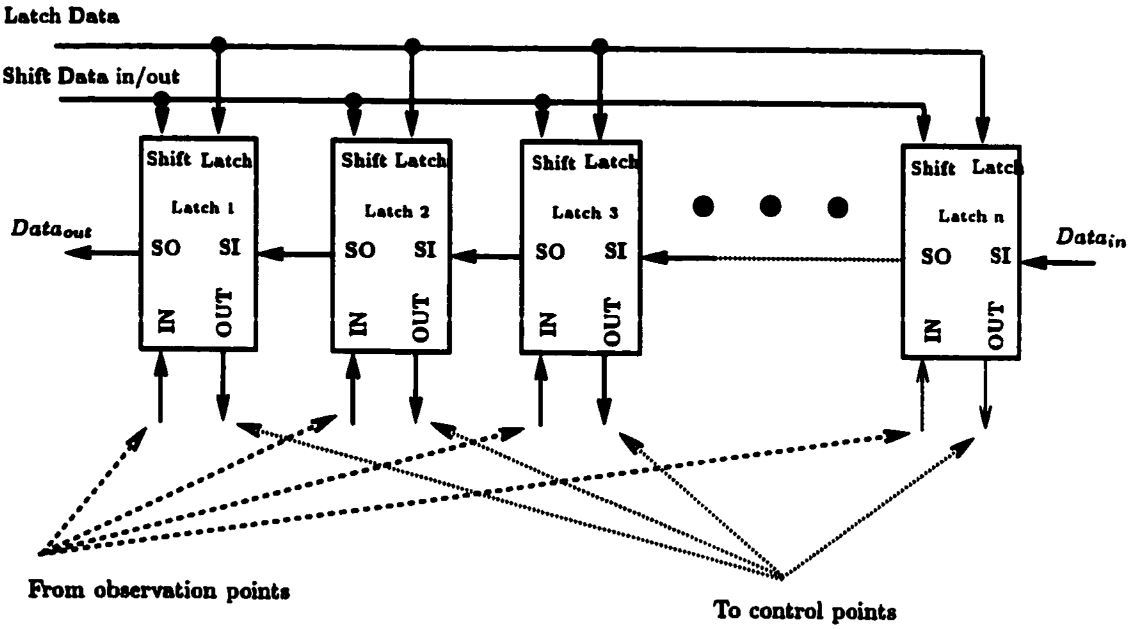


Figure 3.15: *General scan-path technique.*

some points. To decrease the area associated with the latches, some have proposed techniques where these latches are the same as the originally designed latches, see Figure 3.16.

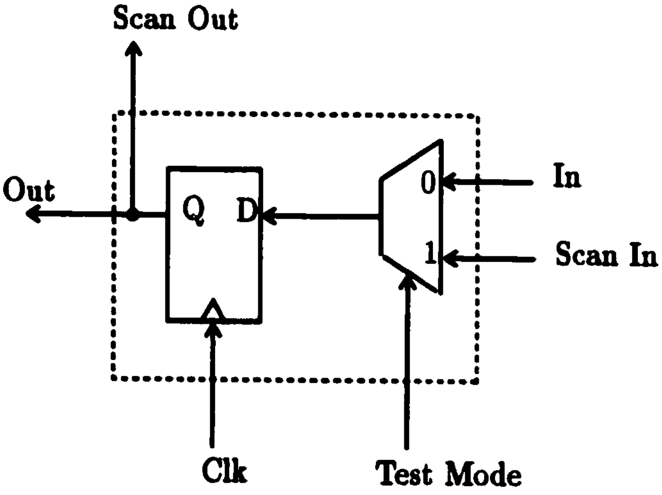


Figure 3.16: *Scan-path technique using circuit latches.*

In considering the cost performance, there are a number of negative impacts. An apparent disadvantage is the serialization of the test, potentially costing more time for actually running a test. Assuming m controllability latches and n observability latches, each test takes at least in the order of $O(m+n)$ time. Secondly, the shift register latches are logically two or three times as complex as simple latches. The overhead from experience has been in the range of 4 to 20 percent of the total area before using scan design. Thirdly, up to four additional primary inputs/outputs are required at each package level for control of the shift registers. This can be reduced significantly by making functional use of some of the pins. For example, the scan-out pin could also be used as the functional output.

It has been reported by the Nippon Electric Company [68] that they have used the scan-path approach for the FLT-700 system, which is a large processor system with 100,000 blocks. Motorola has come forth with a chip which is a gate array in T^2L and shift data serial bit in I^2L logic integrated on that same chip. It is up to the customer to use the bit serial logic if he chooses so. IBM, UNIVAC, Fujitsu, and Amdahl are now currently using scan-path techniques with some modifications in their commercial products. The feasibility of the scan approach for asynchronous circuits has been demonstrated in [152] with a 144-bit scan-path in a systolic array. Asynchronous dual-rail combinational logic, implemented as PLAs, can be made fully testable by introducing a dual-rail scan-path [78]. However, Hazewindus [82] showed that dual-rail combinational logic can be tested using standard test generation techniques, e.g., the D-algorithm [161].

The scan-path technique can be used on any circuit which has state-holding elements. Recently it has been applied to asynchronous circuits such as micropipelines [95, 183]. As another example, Susskind [173] describes a scan technique whereby the states of a classical asynchronous system are scanned in and out. Scan-in allows placement of the state machine in any state and scan-out reveals the machine's next state. This technique offers the advantage that it does not implicitly assume a fault model and allows sequential circuits to be tested by sampling their states. The major disadvantage of this approach is that circuits need to be stable before their state can be sampled, limiting testable circuits to the fundamental mode of operation. Also, some care must be taken that a race-free state assignment is used. Besides, testing time is proportional to the number of states in the machine. This limits the approach to small state machines.

Wey *et al.* [186] has addressed the shortcoming of the fundamental mode of operation presented by Susskind. By proposing special scannable latches, see Figure 3.17, that automatically store the circuit's state as soon as it changes, input-output mode circuits can be tested. This latch consists of two paths: Normal path and scan path. 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 in normal

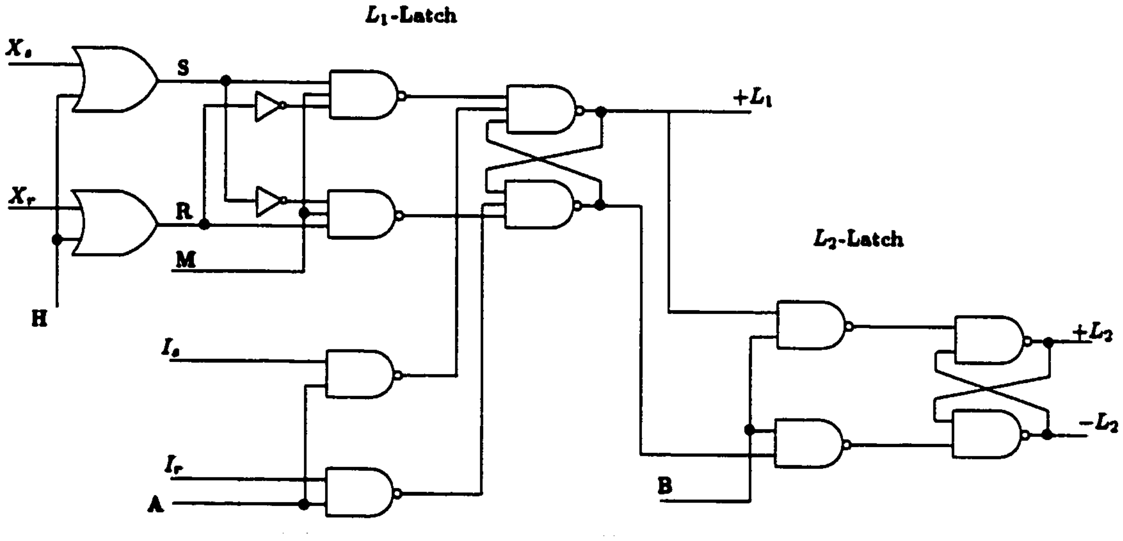


Figure 3.17: *Polarity-hold shift register latch.*

operation, while the scan path, including the lower pair of NAND gates, the cross-coupled NAND latch, and L_2 latch, is used in test mode. In normal operational mode, each output pair (X_S, X_R) of the combinational network is connected to the normal path of the latch where no clock signals are applied, i.e., the clock signals are set as A=0 and B=M=1. Hence, the circuit need not be stable in order to sample the data out of the storage components. In test mode, the scan structure that cascades the scan paths of all latches is formed. Both the scan structure and combinational network are tested in a similar manner as in the well-known scan-path technique. Note that the circuit is operated asynchronously in normal mode, and is tested in a synchronous way using clock signals in test mode. The major difference between this approach and that described by Susskind is that the latches update the values of state continuously and not in test mode only.

Testing Micropipelines

As for any integrated circuits, testing is necessary for micropipelines. A micropipeline must be tested in three parts: Control circuit, combinational logic circuit, and the latches. Besides, the delay constraints must be checked; hence, delay fault testing is necessary. Synchronous testing techniques can easily be adapted for micropipelines since they resemble synchronous circuits in latches and combinational logic. In 1994, the scan-path technique was applied to micropipelines [95]. In doing this, one must take care of the asynchronous operation of the control unit. In the synchronous approach, registers are loaded and observed at discrete times defined by a global clock. Here, there must be a constraint of time on reading from and writing to latches. One method is to halt the entire circuit and scan in and out. Other possibility is to design the latches so that they take or put data on the bus through a handshaking protocol.

To date, there are two published works on testing micropipelines [95, 183]. In [183] the processing logic between the stages in the micropipeline is tested altogether. To test the latches, enough elements are pushed in the pipeline so that it is completely full from stage $i+1$ to the last stage. Then two test vectors are applied. The first vector initializes the latch in stage i, and the second vector is applied for a possible stuck-at-pass fault on the target register. Then the output of the latch is propagated to the end to be compared with the fault-free output. There are several drawbacks to this approach. The test generation execution time is much larger by assuming all combinational logic lumped together than the execution time by testing each stage of combinational logic at a time. Testing latches also takes a lot of time for asynchronously clocking the data to the final stage. Testing micropipelines with and without logic # 1 to # n will take almost the same time if the entire network is treated as one network at the time of testing. Note that the response of the micropipeline at the output is delayed at each stage due to the combinational logic response and latching time. Besides, some of the faults in combinational logic are not testable by this approach, and latches are not properly tested in the presence of faults in the combinational logic. Also, this work does not address the delay fault.

A second attempt on testing micropipelines was presented in [95]. In this work, the control unit, the combinational logic, and the latches are tested separately. Besides, delay fault testing is performed to check whether the bundling constraint is violated or not. Their test methodology now follows.

First, latches are modified, the same way as scan-path registers in synchronous designs, so that they can receive input data, whenever they are in scan mode, by bypassing the combinational logics and previous registers. These latches function much like master/slave flip-flops after adjustment. Secondly, the C-elements in the control unit are altered so that they simply transport their negated input to the output whenever they are in the scan mode. This way, the control path forms a clocking network for the scan path with A_{out} being the clock input. Therefore, the clock moves in the opposite direction to that of data allowing data to march forward. Besides, the bundling delays are no longer on the clock path, see Figure 3.18.

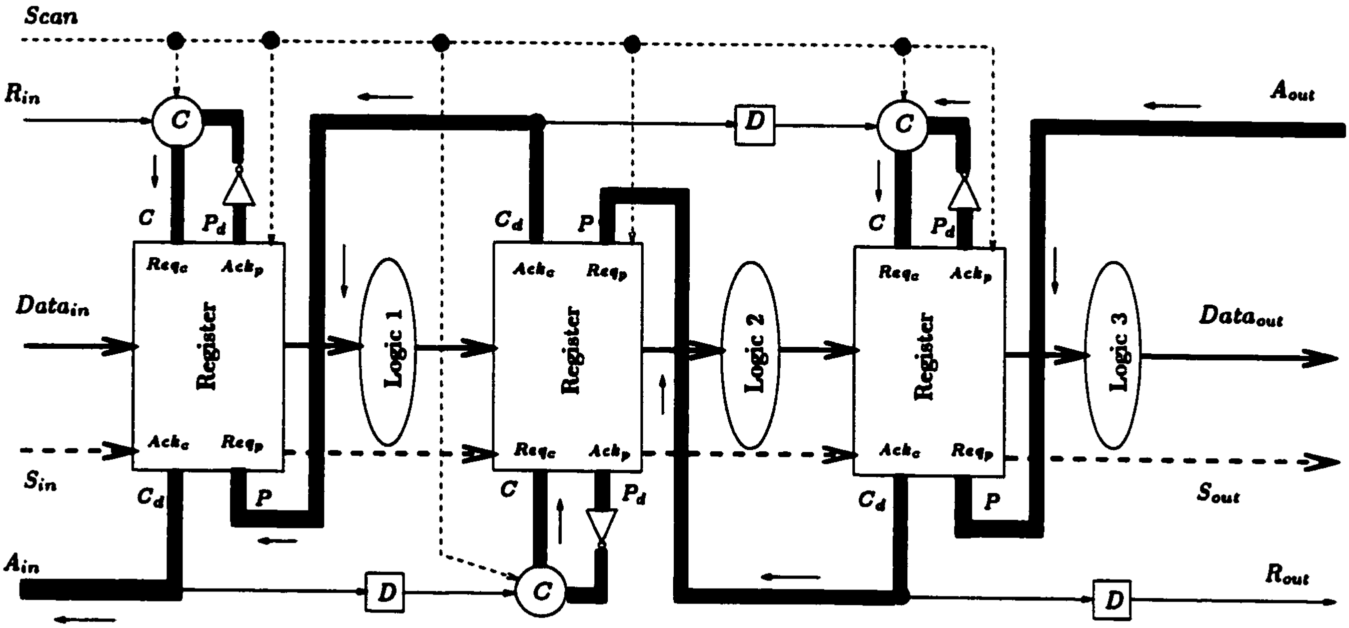


Figure 3.18: *Scan-path testing technique for micropipelines.*

The processing logic between the stages is tested as follows. First, the circuit is put into scan mode. The necessary test vectors for combinational logic blocks are shifted into the latches by clocking A_{out}. These test vectors can be obtained using any conventional automatic test generation program (ATGP). The circuit is then returned to normal mode and the data in each latch is advanced one block ahead, passed through combinational logic, and latched in the next stage register. Finally, the circuit is returned to scan mode, and the data is shifted out of the registers. As seen for each combinational logic block, the previous stage register is used as a controlling point, and the next stage register is used as an observation point. Since each latch is a master-slave register, it can operate as both observation point for the next stage, and controllability point for the previous stage.

The latches are tested more like memory elements. Two types of faults are considered for the latches: stuck-at-pass and stuck-at-capture. In the former case, the latch fails to hold any data and is always transparent. This can be tested by an alternating 0-1 pattern in the control unit while in scan-mode. In the latter case, the storage element holds a particular value forever. This is detected if the latch is written with the opposite value of the fault and read while in scan-mode.

The control unit testing is very simple since the type of component is very limited. Each node on the control path is responsible for an action on the latches, and other nodes of the control path. For simplicity, we have not discussed these faults.

Any delay fault testing requires two test vectors: V_1, and V_2. If the second test vector is applied after the first one, a transition occurs on a sensitized path including the node where the physical defect exists, and propagates to the primary output. Although the output transition is correct in combinational circuits, the undesired delay fault would cause a transition at the output node late enough to be detectable. In synchronous design, the circuit is initialized by applying the first test vector. The second test vector is applied at time t, and the output response is latched at time $t + \delta$, where δ is the desired delay. If the sampled value is different than the desired value, the delay fault is detected. In micropipelines, the delay elements are assumed fault-free at the beginning. As soon as the circuit goes into normal mode, the control unit must operate until the latches are sampled; i.e., it cannot go into scan-mode afterward. Assume we wish to test block i for a delay fault. Knowing V_2 and the function of the combinational block at stage $i - 1$, B, a third vector, V_3, is derived such that $B(V_3) = V_2$. Now the test procedure involves putting the circuit in scan-mode, supplying latches i and $i-1$ with V_1 and V_3, returning the circuit to normal mode, and finally putting the circuit in scan-mode again and reading latch $i+1$ which is in front of the logic block i. If the values captured are not correct, then a delay fault exists.

Chapter 4

I_{DDQ} Faults in Asynchronous Circuits

It was mentioned in Section 3.2.3 that most of the defects in integrated circuits can be modeled as transistor-level shorts and opens. Transistor-level testing provides a higher coverage of faults compared to that of the gate-level. Thus, it is necessary to study the effects of failures at the transistor-level. In this chapter, we present the behavior of BiCMOS logic families under short and open defects, and derive a complete fault model for various families. Using the results of simulations, we investigate the suitability of different fault models for BiCMOS families, and develop a new DFT technique for detecting shorts.

Various modes of failure can occur in VLSI devices, where bridging faults have been shown to be major contributors [61, 168]. Bridging faults are particularly important in modern technology because the high density of integration reduces the distance between lines and/or contacts. Menon *et al.* [132] studied bridging faults in BiCMOS circuits and showed that several bridging faults can be detected by quiescent power supply current (I_{DDQ}) monitoring. In [163], a design-for-testability (DFT) technique has been proposed for detecting I_{DDQ} faults, as explained in Section 3.4.1. This technique can detect all the defects which cause an increase in the quiescent current passing through the bipolar transistor. It cannot be used for detection of a few defects which increase the current of the CMOS section of the structure without increasing the current of the output bipolar transistors. This technique is not suitable for CMOS circuits.

The second DFT technique was proposed by Favalli *et al.* [58, 59, 104], as explained in Section 3.4.1. The residual voltage drop between system ground and actual ground in test mode would cause the circuit to malfunction even if it is fault-free. The increased voltage level on the system ground may have two effects. First, it can trigger the test circuit resulting faulty T_{out} while the circuit is fault-free. Second, the voltage level of internal nodes of the logic circuit depends on the system ground. Consequently, logic "0" on some internal nodes may be interpreted as soft logic "1", although the circuit is fault-free. The tester introduces substantial parasitic capacitance increasing the time constant at the system ground, and slowing the rate of I_{DDQ} testing.

In functional testing, the test vector must satisfy fault excitation, as well as fault propagation. While the DFT technique proposed by Favalli *et al.* does not need input test vectors for fault propagation, it suffers from the common problem that the test vectors at the input nodes must excite the fault. This would, in turn, rise the need of off-line testing, as well as the use of a test generation algorithm.

A circuit idea for solving these problems is introduced in this Chapter. We propose a test circuit which works on-line and eliminates the need for test generation algorithms. Primitive components are the subject of our test. Here, we use different transistor-level realizations of a NAND gate.

This Chapter provides improvements over the results reported in [149]. The modifications include the most up-to-date literature review, an enumerative proof of a general claim which points to fault characteristics of CMOS/BiCMOS circuits, revised fault characterization tables, and the enhanced performance of the proposed DFT technique.

The rest of this chapter is organized as follows. Section 4.1 provides typical cases which are used for simulation in our test analysis. Section 4.2 studies the fault characterization of different BiCMOS structures. It will be shown that many defects may cause parametric faults, which cannot be detected with usual test methods. In Section 4.3, we introduce a DFT technique which enables the detection of bridging faults in these structures. The area and delay overheads resulting from this circuit modification will also be discussed. Conclusions are provided in Section 4.5.

4.1 Typical Examples

In this chapter, we focus on the pull-up/pull-down family. General description of this structure and a conventional BiCMOS circuit have been discussed in Section 2.2. From the pull-up/pull-down family, we also study three structures called BiNMOS, and full-swing BiCMOS, and CMOS circuits. For consistency, a NAND gate was implemented in each of these structures. Figures 4.1 to 4.4 show the implementations of a NAND gate in the mentioned structures. Simulations results have been presented in this Chapter with and without test circuit for each of the structures respectively.

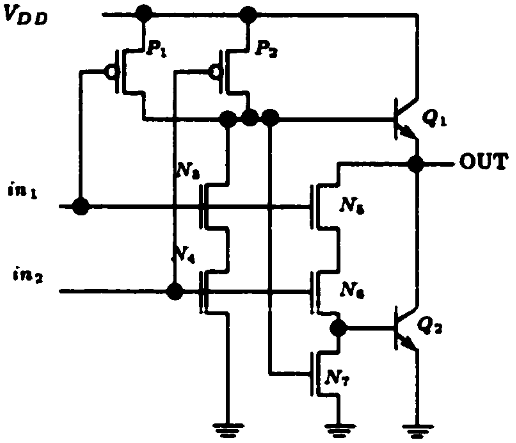


Figure 4.1: *Conventional BiCMOS*

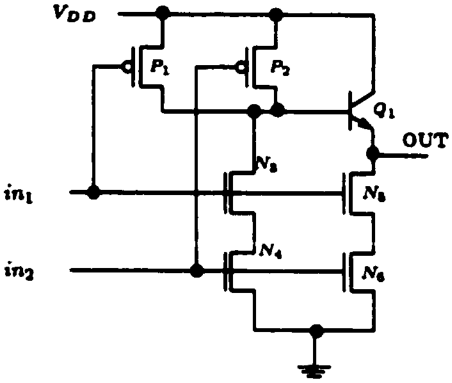


Figure 4.2: *BiNMOS*

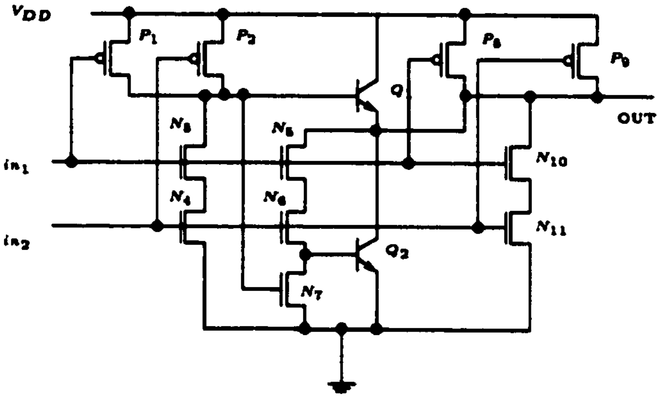


Figure 4.3: *Full-swing BiCMOS*

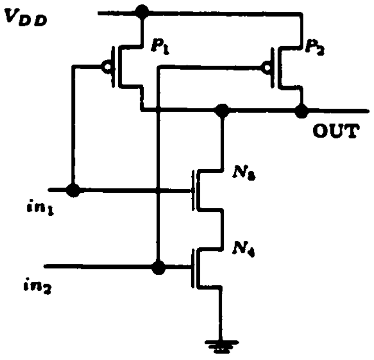


Figure 4.4: *CMOS*

4.2 Fault Characterization

4.2.1 Defect Modeling

As described in Section 3.2.3, we model a short defect as a small resistor (10Ω) between the two nodes. Open-circuits are modeled as a large resistor (10 MΩ) inserted between the affected node and the node to which it would normally have been connected. The open-circuit in the gate of a MOS transistor is treated as a floating point. A floating gate may result in transistor stuck-open (permanently *off*) or may cause the transistor to conduct (stuck-on), as reported in [166]. In this study, we assumed that a floating gate will cause a stuck-open transistor. We have considered the former case for the following reason. We have considered stuck-at faults at the terminals of a transistor. A floating gate which results in transistor stuck-on behaves as if the gate terminal is stuck-at 1 (0) for NMOS (PMOS) transistor. Therefore, we consider floating gates which cause transistor stuck-open to prevent multiple copy of a situation and to provide a more realistic fault coverage. If the latter case is taken into account, the fault detection ratio will increase.

The notation introduced in Section 3.2.3 is used in this Chapter. To avoid the complexity of dealing with the multiple-defect case, we assume that not more than one defect can occur at a time.

4.2.2 Conventional BiCMOS NAND Gate

The conventional BiCMOS NAND gate shown in Figure 4.1 was implemented in 0.8μ BiCMOS technology and simulated with HSPICE. The lengths of all MOS transistors used for this study were 0.8μ. The widths of the transistors were 4μ, except for N_5, N_6, and N_7, which had W $= 2\mu$. The supply voltage was 5V, and the output was connected to a capacitive load of 2pF. In order to account for the limited current drive capability of the inputs, the inputs to the circuit were taken from CMOS inverters.

The DC voltage transfer characteristics of the gate for each of the inputs was examined to determine the logic levels. Figure 4.5 shows the DC transfer function for the output vs. in_2, when in_1 is set at 5V.

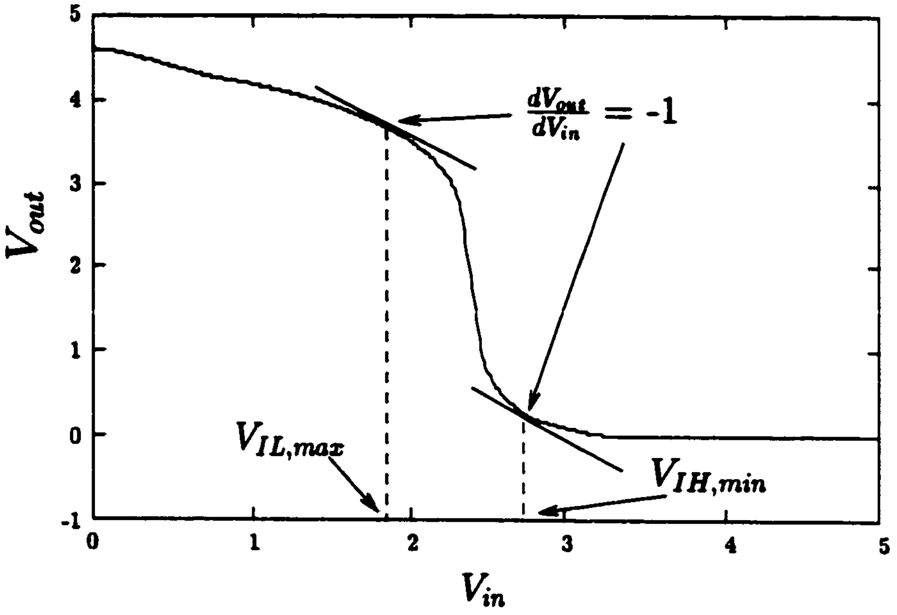


Figure 4.5: *DC transfer function of BiCMOS NAND gate.*

$V_{IL,max}$ and $V_{IH,min}$ were determined by finding the $\dfrac{dV_{out}}{dV_{in}} = -1$ points [84] on the voltage characteristics. The values which satisfy the requirements for both inputs were determined to be 1.9V and 2.8V, respectively. Therefore, the input logic levels for the BiCMOS NAND gate are 0 to 1.9V for logic level "0" and 2.8V to 5V for logic level "1". Any voltage between 1.9V and 2.8V is considered indeterminate (intermediate state).

A comprehensive simulation study was done for all possible single defects in the circuit. Simultaneous current monitoring was performed during simulations to measure I_{DDQ}. The results of the simulations show that each injected defect may result in one or more logical fault(s). These logic-level faults belong to one of the two major fault classes, listed below:

1. *Parametric faults* - Defects may alter any one of delay time, quiescent supply current, or output voltage magnitude. For example, either of beQ_2 or cQ_2

turns Q_2 *off* . As a result, the output must discharge solely through the NMOS transistors N_5, N_6, and N_7. This results in a *delay fault.*

Some defects (e.g., dsN_4) cause an increase in the steady-state supply current. This happens if a low-impedance path is created between V_{DD} and GND due to a defect, leading to an excessive leakage current. This is an I_{DDQ} fault.

Some defects degrade the output voltage. The output may take an intermediate voltage level (e.g., dsN_4 drives the output to an intermediate voltage for $in_1 in_2 = 10$). For this NAND gate, all the intermediate voltage faults occur in conjunction with I_{DDQ} faults. Some defects change the output voltage level, and create a *soft* ZERO or a *soft* ONE. The *soft* ZERO has somewhat higher voltage than the normal ZERO logic level, and the *soft* ONE has somewhat lower voltage level than the normal ONE logic level. The result is a reduced noise margin. For example, the dN_5, sN_5, and gN_5 defects turn N_5 *off* and create a *soft* ZERO fault, as well as a delay fault.

2. *Functional faults* - A functional fault occurs whenever the output takes a wrong, yet legal logic level (ONE instead of ZERO, or vice-versa) for certain input combinations. Some defects cause the output to be *stuck-at* a logic level for all input combinations (like dsP_2 which creates a stuck-at-one fault).

   It is also possible that the output take a wrong logic level for certain input combinations. For example, dgP_1 causes the output to follow in_1. These faults are called *truth-table faults* and can be detected similarly to stuck-at faults. Functional faults in CMOS/BiCMOS circuits, however, exhibit parametric fault behaviors as will be discussed in Section 4.2.7.

   Another fault in this class is the stuck-open fault, which occurs when a circuit node becomes floating and exhibits a high impedance, so that the signal value at the node becomes dependent on its previous value. For example, each of dP_1, sP_1, or gP_1 creates a stuck-open fault.

The fault characterization of the conventional BiCMOS NAND gate is shown in Table 4.1. It shows that all short defects (except dsN_7, beQ_1, and beQ_2) result in I_{DDQ} fault. Open defects normally manifest themselves as stuck-open, or delay faults. This result is true for other BiCMOS families, as well.

Fault Characterization	Simulated Physical Defect
Delay fault	$gsN_3, gsN_4, dN_5, sN_5, gN_5, dN_6, sN_6, gN_6, dsN_7, dN_7, sN_7, gN_7, beQ_1, cQ_1, beQ_2, cQ_2$
I_{DDQ} fault	$dsP_1, dgP_1, gsP_1, dsP_2, dgP_2, gsP_2, dsN_3, dgN_3, gsN_3, dsN_4, dgN_4, gsN_4, dsN_5, dgN_5, gsN_5, dsN_6, dgN_6, gsN_6, dgN_7, gsN_7, cbQ_1, ceQ_1, cbQ_2, ceQ_2$
Output indeterminate	$dsN_3, dgN_3, gsN_3, dsN_4, dgN_4, gsN_4, dsN_5, dgN_5, dgN_6, cbQ_2$
Output stuck open	$dP_1, gP_1, sP_1, dP_2, sP_2, gP_2, dN_3, gN_3, sN_3, dN_4, gN_4, sN_4, gsN_6, bQ_1, eQ_1, bQ_2, eQ_2$
Soft logic level	$dN_5, sN_5, gN_5, dN_6, sN_6, gN_6, dN_7, sN_7, gN_7$

Table 4.1: *Fault characterization of the conventional BiCMOS NAND gate.*

4.2.3 BiNMOS NAND Gate

The procedure described above was repeated for the BiNMOS NAND gate (Figure 4.2). The result of simulations is shown in Table 4.2. Great similarity can be seen between the fault characteristics of BiNMOS and conventional BiCMOS NAND gates.

Fault Characterization	Simulated Physical Defect
Delay fault	$dN_3, sN_3, gN_3, dN_4, sN_4, gN_4, gsN_5, gsN_6, beQ_1, cQ_1$
I_{DDQ} fault	$dsP_1, dgP_1, gsP_1, dsP_2, dgP_2, gsP_2, dsN_3, dgN_3, gsN_3, dsN_4,$ $dgN_4, gsN_4, dsN_5, dgN_5, gsN_5, dsN_6, dgN_6, gsN_6, cbQ_1, ceQ_1$
Output indeterminate	$dsN_3, dgN_3, dgN_4, dsN_5, dgN_5, gsN_5, dsN_6, dgN_6, gsN_6$
Output stuck open	$dP_1, gP_1, sP_1, dP_2, sP_2, gP_2, gsN_4, dN_5, gN_5, sN_5, dN_6, gN_6,$ $sN_6, gsN_4, bQ_1, eQ_1, bQ_2, eQ_2$
Soft logic level	$dN_3, sN_3, gN_3, dN_4, sN_4, gN_4$

Table 4.2: *Fault characterization of the BiNMOS NAND gate.*

4.2.4 Full-Swing BiCMOS NAND Gate

Table 4.3 shows the effects of hard shorts and opens on the behavior of the full-swing BiCMOS NAND gate (Figure 4.3).

It can be seen that defects create different faults, compared to the previous two structures. This is mainly due to the great redundancy in this structure. For example, most of the defects which create a stuck-open fault for the conventional BiCMOS structure cannot affect the functional validity of the output in this structure, because there is an alternative path from the inputs to the output. Therefore, the output will switch correctly. However, delay faults will occur for most of these defects.

4.2.5 CMOS NAND Gate

Table 4.4 presents the results of the simulations for the CMOS NAND gate (Figure 4.4).

Fault Characterization	Simulated Physical Defect
Delay fault	$dP_1, sP_1, dP_2, sP_2, dgN_3, gsN_3, dN_3, sN_3, gN_3, dN_4, sN_4, gN_4, dN_5, sN_5, gN_5, gsN_6, dN_6, sN_6, gN_6, dgN_3, dsN_7, gsN_7, beQ_1, cQ_1, eQ_1, bQ_1, beQ_2, cQ_2, eQ_2, bQ_2$
I_{DDQ} fault	$dsP_1, dgP_1, gsP_1, dsP_2, dgP_2, gsP_2, dgN_3, dsN_4, dgN_4, gsN_4, dsN_5, dgN_5, gsN_5, dsN_6, dgN_6, gsN_6, dgN_7, gsN_7, dsP_8, dgP_8, gsP_8, dsP_9, dgP_9, gsP_9, dsN_{10}, dgN_{10}, gsN_{10}, dsN_{11}, dgN_{11}, gsN_{11}, cbQ_1, ceQ_1, cbQ_2, ceQ_2, eQ_2, bQ_2$
Indeterminate	$dgP_1, gP_1, dgP_2, gP_2, dsN_3, dgN_3, dgN_4, dsN_5, dgN_5, dsN_6, dgN_6, gsN_6, dgN_3, dgN_4, gsN_7, dgP_8, dgP_9, dgN_{10}, dgN_{11}, cbQ_2$
Output stuck open	gsN_4
Soft logic level	$dN_3, sN_3, gN_3, dN_4, sN_4, gN_4, dN_7, sN_7, gN_7, dsN_{10}, dsN_{11}$

Table 4.3: *Fault characterization of the full-swing BiCMOS NAND gate.*

Fault Characterization	Simulated Physical Defect
Delay fault	$dN_3, sN_3, gN_3, dN_4, sN_4, gN_4$
I_{DDQ} fault	$dsP_1, dgP_1, gsP_1, dsP_2, dgP_2, gsP_2, dsN_3, dgN_3, gsN_3, dsN_4, dgN_4$
Output indeterminate	dsN_3, dgN_3, dgN_4
Output stuck open	$dP_1, gP_1, sP_1, dP_2, sP_2, gP_2, gsN_4$
Soft logic level	$dN_3, sN_3, gN_3, dN_4, sN_4, gN_4$

Table 4.4: *Fault characterization of the CMOS NAND gate.*

4.2.6 Analysis of Results

Table 4.5 summarizes the fault distributions corresponding to solid short/open defects for the four mentioned structures. The numerical values of the table are

Fault	Percentage			
	Conventional	BiNMOS	Full-swing	CMOS
Delay	30	22	43	25
I_{DDQ}	44	43	53	46
Intermediate level	19	20	26	13
Stuck open	31	40	0	29
Soft level	17	13	16	25

Table 4.5: *Fault distribution for BiCMOS/CMOS NAND gates.*

obtained as follows. First, the number of faults for each defect is obtained. Second, the total number of faults has been obtained. It is important to note that there are faults which fall into more than one category. Third, the number of faults for each defect is divided by the total number of faults. The result is multiplied by 100.

An important result derived from the fault characterization tables (Tables 4.1 to 4.4) is that most of the short defects in CMOS/BiCMOS circuits cause an I_{DDQ} fault, while most of the open defects result in delay or stuck-open faults.

These tables, as well as Table 4.5, further reveal that as the redundancy in a BiCMOS structure increases, less defects can cause a stuck-open fault. For example, in the full-swing structure— which has 100% redundancy— none of the open defects create a stuck-open fault. This is due to the fact that an alternative path to the output exists due to the redundancy (the full-swing structure is actually a CMOS gate, connected in parallel with a conventional BiCMOS gate. A single open defect in this structure is, therefore, masked by the parallel redundancy).

As can be expected, our simulations show that the most important effect of bridging faults is an increase in the quiescent current.

Another important result is that the sum of the fault percentages for each family is more than 100%. This fact shows that there are faults which behave differently depending on what the input excitations are. The following Section generalizes these results.

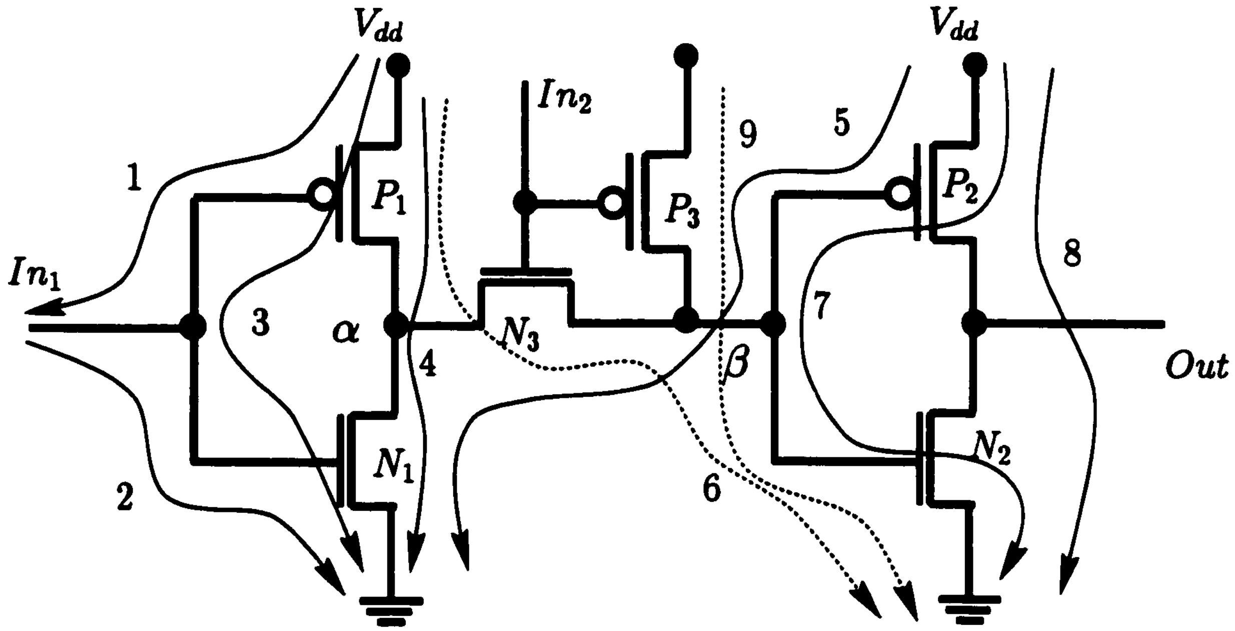
4.2.7 General Fault Characterization: A proof

The results obtained from simulations can be generalized by the following theorems. We show that any fault in CMOS/BiCMOS circuits is a delay fault, an I_{DDQ} fault, or a fault which inhibits transitions at the primary output. We investigate each case and present an enumerative proof of such claim. The following definitions are used in this Section.

An *output function* realized by a circuit is represented by a 3-tuple (V, I, t), where V represents its voltage specifications and is normally expressed using Boolean functions, I shows the power supply current characteristics, and t gives the time frame for the output function to be correct. A fault is *detectable* if it changes any of the parameters V, I, or t. If the fault changes V characteristics, the fault is usually regarded as logic fault. If it changes the I parameter, it is often considered an I_{DDQ} fault in CMOS circuits. We assume that CMOS/BiCMOS circuits are not intentionally designed to create excessive DC current in fault-free circuits. Delay faults degrade the time specification of the output response.

A transistor is *redundant* if removing it produces no change in the output function. A node is *redundant* if it can be permanently connected to either '0' or '1' without altering the output function. Supply nodes, primary inputs, and the primary output are called *external* nodes. Primary inputs can adopt *high* or *low* voltages freely in a fault-free circuit. The polarity of primary inputs are defined by their present logic value which are set by previous CMOS stages. The primary output is the result of mapping the inputs, and is to be '0' and '1' frequently. The polarity of the primary output is defined by its present logic value. Its behavior is often affected by the next stage CMOS circuit.

A *path* is an ordered chain of nodes connected together via the terminals of transistors. Different paths in CMOS/BiCMOS circuits are shown in Figure 4.6.



I-paths: 4, 6, 8

V-paths: 1, 2, 3, 5, 6, 7, 9

V_{PN}-path: 6, 9

V_{NP}-path: 5

Figure 4.6: *Different paths in CMOS inverter circuit.*

Consider two voltage levels v_1 and v_2. If v_1 is greater than v_2, we consider v_1 as positive voltage and v_2 as negative voltage. V_{DD} is a positive voltage in regard to any voltage levels in the circuit, and GND is a negative voltage level.

Definition 4.1 *An* **i**-*path is a directed edge. Drain-to-source of an NMOS transistor, source-to-drain of a PMOS transistor, collector-to-emitter of an NPN bipolar transistor, and emitter-to-collector of a PNP bipolar transistor are* **i**-*paths.*

Definition 4.2 *A* **v**-*path is a directed edge. The gate-to-source of an NMOS transistor, source-to-gate of a PMOS transistor, base-to-emitter of an NPN bipolar transistor, and emitter-to-base of a PNP bipolar transistor are* **v**-*paths.*

Definition 4.3 *An* **I**-*path is a directed path which starts with a positive supply node, continues through* **i**-*paths and ends in the negative supply node.*

An **I**-path provides a charging path to positive supply, referred to as the **p**-part, and a discharging path to the negative supply, referred to as the **n**-part.

Definition 4.4 *A* **V**-*path is a directed path which starts with a positive external node, continues through* **i**-*paths and/or* **v**-*paths, and ends in a negative external node. A* **V**-*path contains at least one* **v**-*path.*

Definition 4.5 *A* $\mathbf{V}_{\mathbf{PN}}$-*path is a directed* **V**-*path which contains at least one* **p**-*part of an* **I**-*path.*

Definition 4.6 *A* $\mathbf{V}_{\mathbf{NP}}$-*path is a directed* **V**-*path which contains at least one* **n**-*part of an* **I**-*path.*

Definition 4.7 *A path is active if the transistors which are on the path are active. The collection of input combinations for which a path is active is called the* **on**-*set.*

Definition 4.8 *A path is off if there is at least one transistor which is on the path and it is off. A set composed of input vectors which turn off the path is called the* **off**-*set.*

There should not exist a V- path between two inputs. Since any V-path or I-path follows Kirchoff Voltage Law (KVL), it cannot start from a positive (negative) node and end in a positive (negative) node. A path is *redundant* if it can be permanently *off* or permanently *active*.

An I-path in a fault-free CMOS circuit is active only temporarily to accommodate the transitions. In other words, its **on**-set is empty while its **off**-set contains every possible input combination. Either the **p**-part or the **n**-part is *on* to prevent the output node from floating. Floating nodes charge or discharge eventually through parasitic paths and their logic values are not known if they keep floating for a long period. The intersection of the **on**-set of the **p**-part of an I-path and that of the **n**-part of the same path is empty set. This is because these two parts should not be active at the same time.

There are interactions among different paths. There is at least one active V-path to the primary output at any time. A **p**-part or an **n**-part of an I-path is active at the same time. While a V-path arranges for the output node to change, parts of the I-path provide a short path from the output node to either of the supply nodes for a fast transition.

Any node of a CMOS/BiCMOS circuit should charge to the positive supply and discharge to the negative supply at least once. The reason which supports this idea is as follows. If a node does not have a path to the positive supply at any time, it cannot charge to positive supply through an *active* path. It retains its *low* logic value unless it is charged through an *off*-path after a long time has elapsed. An output response is expected by a designer much faster than the time interval calculated by parasitic capacitances and resistors connected between such a node and the supply nodes. (This process takes much longer than the period predicted by the frequency of operation. The circuit is not supposed to stay still until a node is charged or discharged through parasitic capacitances.) The node never charges to the positive supply in practice. Therefore, it is the objective of the designer to provide a path to the positive supply for each node. A similar argument is valid for a path from any node to negative supply through an *active* path at some time.

A V-path which charges a node to the positive supply is not simultaneously active with the V-path which discharges the same node to the negative supply. This is due to the assumption that the on-set of an I-path should be the empty set. Such V-paths may coexist only temporarily while an output transition happens. However, the intersection of their on-sets should be empty.

Many nodes of a CMOS circuit are charged to V_{DD} through the p-part, or discharged to GND through the n-part to facilitate a fast transition. Therefore, nodes are usually on an I-path. Any node of the circuit is either on an I-path or on at least two V-paths. It should be on a $\mathrm{V_{PN}}$-path to charge toward the positive supply. It should also be on a $\mathrm{V_{NP}}$-path to discharge to the negative supply. However, the node should not be pulled up and down simultaneously. Therefore, the intersection of the on-set of $\mathrm{V_{PN}}$-path and that of $\mathrm{V_{NP}}$-path should be the empty set.

Each transistor is a shared component among different paths. An NMOS (NPN) transistor is shared between two paths: One which goes through the drain-source (collector-emitter), and the other which passes through the gate-source (base-emitter). As an example, P_1 in Figure 4.6 is shared among paths # 1, # 3, # 4, and # 6. A similar statement is true for the PMOS (PNP) transistor. Shared components in fault-free circuits are frequently turned *on* and *off* by their associated V-paths: i.e., a V-path cannot permanently turn *on* or *off* the shared components. Two paths which share at least one component are considered *neighbors*. Each path has at least one neighbor.

The resistivity introduced by a transistor among its terminals plays a significant role in the time response of the circuit. While the transistor is *off* (*on*), it provides a huge (small) resistor between its drain-source or collector-base terminals. The RC time constant of one of the transistor terminals changes considerably in either of the states or both.

Any fault which inhibits a transition at the primary output is called a *non-transitional* fault in this dissertation.

Theorem 4.1 *An internal, detectable stuck-open fault in an irredundant CMOS/BiCMOS circuit is equivalent to a delay fault, an I_{DDQ} fault, or a non-transitional fault.*

Proof:

1. Any path which includes a break or a stuck-open fault is permanently *off*. Such a path which is supposed to be active for some input combination and to provide a short path for the output node to either of the supplies will no longer perform its task. Nodes of the path will either charge or discharge through parasitic paths to either of the power supplies. Since the RC of such nodes are defined by the resistivity and capacitivity of the elements connected to it, the time response of the **I**-path or **V**-path is not within the acceptable range for which the circuit is designed. The fault shows itself as a delay fault.

2. If a multiple breaks or stuck-open faults happen so that both the $\mathbf{V_{PN}}$-path and the $\mathbf{V_{NP}}$-path are disabled, an internal node retains its previous logic value or it has an intermediate voltage which has never been fully charged or discharged to any of the power supplies. Such fault(s) will inhibit at least one transition at the primary output. The floating node can accept any voltage

value employed by the environment. If they do not inhibit a transition, they are redundant nodes and can be permanently connected to either '0' or '1'.

3. If a break happens on a V-path which is neither a V_{PN}-path nor a V_{NP}-path, it would force the I-path with which it shares components to be *off* for at least one test vector. Therefore, a node at the I-path is floating, showing itself like a non-transitional fault. For example, if the gate of P_2 is disconnected from β, see Figure 4.6, and retains a positive voltage such that P_2 is *off*, the output node would be floating when N_2 is also *off*. This is detectable when an outside voltage is applied to the node. The faulty node tends to follow whatever voltage that is applied to it regardless of the input vectors.

If the broken node has a voltage value which keeps a transistor *on* permanently, its faulty behavior is detectable using items # 1, # 2, # 3, and # 5 of the following theorem. QED.

Theorem 4.2 *An internal, detectable stuck-on fault in an irredundant CMOS/BiCMOS circuit is equivalent to an I_{DDQ} fault, a delay fault, or a non-transitional fault.*

Proof: A transistor that is stuck-on operates as if it is permanently conducting. There are different conditions for which the on-sets or the off-sets are changed. We consider them separately.

1. The stuck-on fault happens on an I-path. The on-set of the I-path is not empty set. There is an input combination which excites the I-path, and increases the DC current supply through the I-path. This fault is detectable using I_{DDQ} testing technique. For example, if source-emitter of P_2 is permanently conducting, see Figure 4.6, its effect is known when N_2 is also

conducting. This would create a short path between supplies on path # 8, and it is detectable using I_{DDQ} testing technique.

2. The i-path(s) on a V-path is replaced by a linear resistor due to stuck-on fault so that no v-path has been affected. There are two different cases. First case involves the stuck-on fault which has happened on a V_{PN}-path. The intersection of the on-set of the V_{PN}-path and that of the V_{NP}-path is no longer empty. The non-empty set provides the test vector for which the p-part of an I-path is active simultaneously with the n-part of the same path. The fault is, therefore, detectable using the I_{DDQ} testing technique. For example, if the drain-source of N_3 is stuck-on, an I_{DDQ} fault is announced through P_3, N_3, N_1 for the input combination ($In_1 = 1$, $In_2 = 0$). Similar argument is valid for the second case which happens when the stuck-on fault is on a V_{NP}-path. If there is no change in the on-sets of different paths, the transistor is redundant and it is replaced with a short wire between its two terminals.

3. A V-path is changed to an I-path due to a stuck-on fault. The fault is detectable using the I_{DDQ} testing technique. For example, if the source-gate of P_2 is replaced by a linear resistor due to a fault, see Figure 4.6, it provides a short path between supply lines through path # 5.

4. A v-path is always *off* because of a stuck-on fault. The fault may or may not have happened on the same path as the v-path. A transistor whose v-path is always *off* will act like a stuck-open fault. This fault is detectable as theorem 4.1 states. For example, consider that source-gate of P_2 (see Figure 4.6) is connected by a short wire. P_2 is always *off*. This shows itself as stuck-open fault.

5. A v-path is always *on* because of a stuck-on fault. The fault may or may not

have happened on the same path as the v-path. A transistor whose v-path is always *on* will act like a stuck-on fault. This fault, however, is present on a different path. The faulty behavior of this transistor is detectable depending on how it affects its own paths or its neighboring paths. If the same situation repeatedly happens so that a shared transistor is finally on an I-path, the fault is detected using I_{DDQ} fault testing technique. Otherwise, the fault is detected as explained in the items mentioned above.

QED.

Theorem 4.3 *An internal, detectable bridging fault which happens between two nodes is equivalent to an I_{DDQ} fault, a delay fault, or a non-transitional fault.*

Proof: There are different bridging faults.

1. We can imitate much of the arguments explained in theorem 4.2 for the bridging faults which happen on a single path. Such bridging faults behave as if one or two transistors are stuck-on.

2. There are bridging faults which happen between a node of the circuit and either of the supply lines. The faulty node is present on different paths, and its behavior is detectable by either delay fault or I_{DDQ} fault testing techniques as expressed in theorem 4.2.

3. A bridging fault which occurs between two internal nodes, namely A and B. Node B follows any transition on node A due to this fault. However, each node belongs to different paths. Both nodes are connected to supply nodes through charging and discharging paths. If we force node A to charge to positive voltage, and apply inputs such that node B has to discharge to

negative voltage, we have created an I-path which is observed as I_{DDQ} fault. Similar argument is true for the opposite voltage value.

4. If the resistivity introduced by the bridging fault is relatively large, both nodes are free to follow any voltage value they are forced to by their own paths. The fault is detectable as a delay fault if the RC time constant of either of the nodes is changed. This will show itself as a slow-to-rise or slow-to-fall transition at these nodes.

5. If the bridging fault does not force any of the nodes to follow the other, nor does it change the RC constant of the nodes, the fault is not detectable.

QED.

Theorem 4.4 *An internal, detectable stuck-at 0/1 fault in a CMOS/BiCMOS circuit is equivalent to an I_{DDQ} fault, delay fault, or a non-transitional fault.*

Proof: We present the proof by enumerating the possibilities. We refer to the above theorems in many cases.

1. If the gate of an NMOS (NPN) transistor is stuck-at 0, the transistor is inactive, and it behaves like a stuck-open fault.

2. If the gate of an NMOS (NPN) transistor is stuck-at 1, the transistor is active, and it behaves like a stuck-on fault.

3. If the gate of a PMOS (PNP) transistor is stuck-at 0, the transistor is active, and it behaves like a stuck-on fault.

4. If the gate of a PMOS (PNP) transistor is stuck-at 1, the transistor is inactive, and it behaves like a stuck-open fault.

5. If a stuck-at 0 fault happens at the drain (collector) terminal of a MOS (bipolar) transistor which is on an I-path, it would result in an I_{DDQ} fault. In this case the n-part behaves as if it is always *on*.

6. If a stuck-at 1 fault happens at the drain (collector) terminal of a MOS (bipolar) transistor which is on an I-path, it would result in an I_{DDQ} fault. In this case the p-part behaves as if it is always *on*.

7. A stuck-at 0/1 fault which happens on any node of a V-path behaves as if a bridging fault has happened between the node and either of the supply nodes. Theorem 4.3 shows that the fault is detectable through delay fault testing, I_{DDQ} fault testing, or non-transitional fault testing techniques.

If the on-set and the off-set of all paths are not changed, the fault is undetectable. If there exists no test vector which can be produced through the inputs, the fault is considered undetectable. QED.

The proof suggests the existence of a test vector for which the fault is detected. Obtaining such a test vector is relatively difficult in practice. In this dissertation we show, however, that there is no need to obtain test vectors if our proposed DFT techniques are used. The objective of the theorems mentioned above is to show that 100% fault coverage is gained if we detect I_{DDQ} faults, delay faults, and non-transitional faults and there is no need to test the CMOS/BiCMOS circuits for any other faults.

In the next section, we propose a DFT method which can detect most of the faults which create an I_{DDQ} fault. Detection of delay faults and non-transitional faults is the subject of the next Chapter.

4.3 DFT Technique

Bridging between two nodes (including terminals of transistors) may create a low-impedance path between V_{DD} and GND, thus increasing the quiescent supply current. Another effect of these defects is to drive the output to an intermediate level, which is an undetectable fault in functional testing.

In this Section, we show a circuit modification that can be applied to CMOS/ BiCMOS circuits to improve their testability. The technique presented here is schematically illustrated in Figure 4.7. The basic idea is to introduce a virtual GND (V_{gnd}), which is forced to GND in the initialization mode. To achieve this, the control signal *init* must be ONE in initialization mode, thus connecting V_{gnd} to GND via the bipolar transistor Q_{t1}. In normal mode, the control line *init* is switched to ZERO. If no fault is present, no conducting path is established between

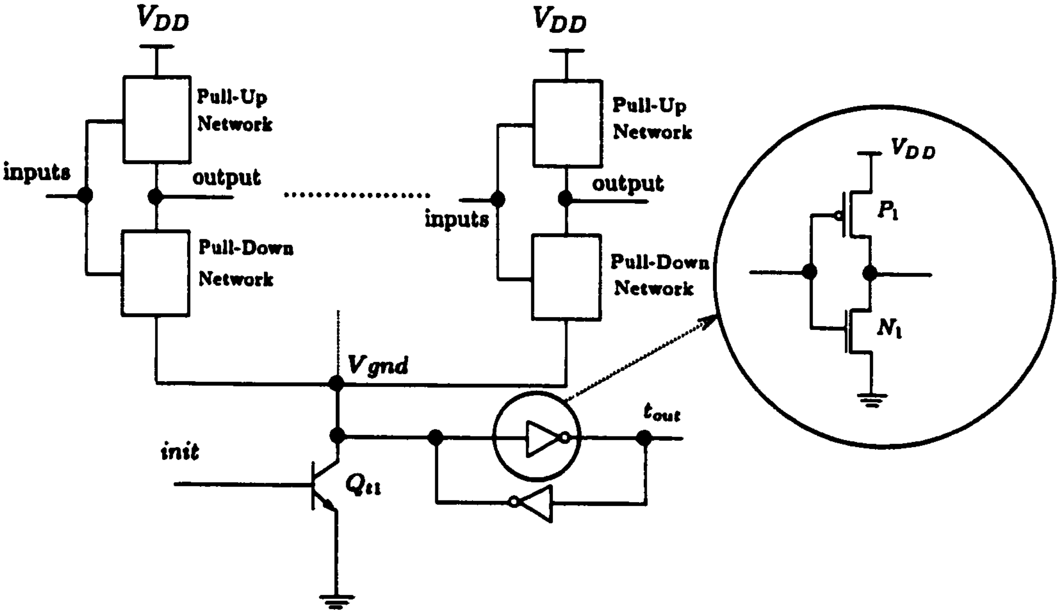


Figure 4.7: *Introducing virtual GND for detecting shorts.*

V_{DD} and V_{gnd}. So, this node remains at logic ZERO. As a result, the output of the CMOS inverter (node t_{out}) will be logic ONE. On the contrary, if a path exists between V_{DD} and V_{gnd} due to a bridging, V_{gnd} can be charged to a high value, thus t_{out} is lowered to logic ZERO.

The main advantage of this technique over the proposed technique by Favalli *et al.* [58,59,104] is as follows. Since V_{gnd} is floating in test mode under fault-free condition in their proposed technique, its voltage is not guaranteed to be ZERO, and the normal leakage current may drive it to a voltage which is high enough to lower t_{out}. To overcome this problem, we have added another inverter in a feedback path. This extra inverter stabilizes t_{out} against temporary variations during transitions and keeps V_{gnd} at ZERO in the fault-free case.

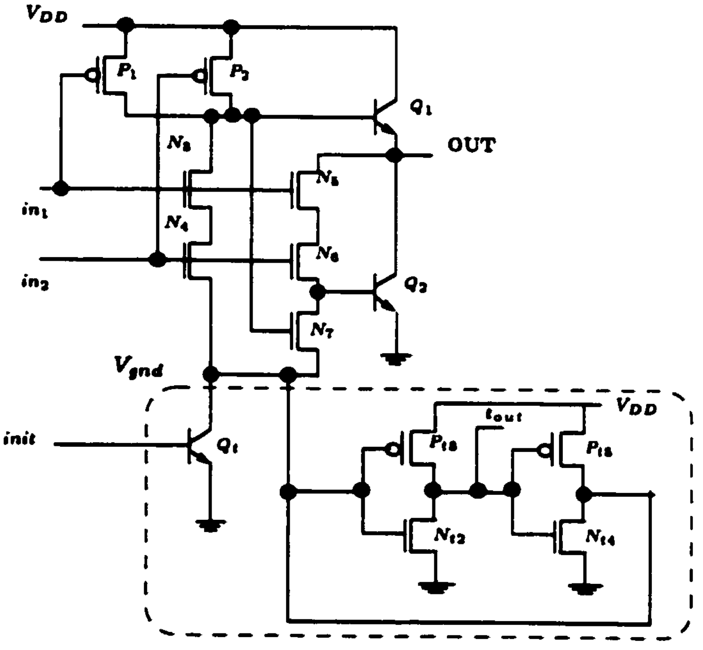


Figure 4.8: *Conventional BiCMOS NAND gate, modified for detecting bridgings.*

An important feature of this technique is that there is no need for the input test sequence to excite the fault or propagate its faulty behavior to the primary output. Simulation is thus straightforward, as is its detection.

Figure 4.8 shows a conventional BiCMOS NAND gate, modified for detecting shorts and bridges.

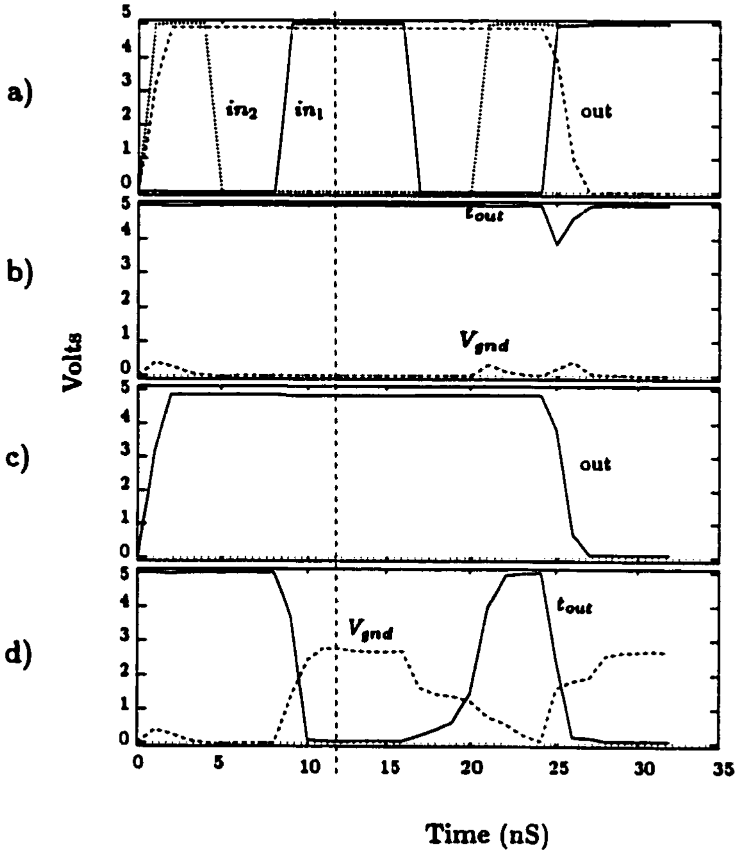


Figure 4.9: *Detection of an I_{DDQ} fault in the modified BiCMOS NAND: a) inputs and output for the fault-free case b) t_{out} and V_{gnd} for the fault-free case (normal mode) c) output for dsN_6 defect (I_{DDQ} fault, only) d) t_{out} and V_{gnd} for dsm6 defect (normal mode)*

To show the operation of the test circuit, we compare the behavior of a fault-free circuit with a gate which is affected by dsN_6 defect, see Figure 4.9. The only effect of the chosen defect on the behavior of the circuit is an increase in I_{DDQ}, and it does not affect the logical behavior of the gate. Therefore, it is among those defects which cannot be detected by functional testing. Graph "a" shows the inputs and the output for the fault-free case. The test output t_{out} is at a high logic level, as shown in graph "b", representing the fault-free case. As depicted in graph "c", the logical behavior of the output does not change under the injected defect. The only consequence of this defect is a substantial increase in quiescent current, because a low impedance path is created between V_{DD} and GND. This fault is excited by the input combination $in_1 in_2 = 1\ 0$. Graph "d" shows how this low-impedance path causes t_{out} to assume a low voltage level. Therefore, the I_{DDQ} fault can be detected by this method.

Simulations were done for all possible defects mentioned in Tables 4.1, 4.2, 4.3, and 4.4, including the possible defects in the test circuit. The input CMOS inverters, mentioned in Section 4.2.2, were also modified by using V_{gnd} instead of GND.

As expected, not all open defects are detectable with this method. However, the following open defects can be detected:

- eQ_2 and bQ_2, because they cause the output capacitor to be discharged via V_{gnd} in a long period, enough to activate the test circuit.

- Open defects in P_{t3} and N_{t4}, because they cause the corresponding transistors to be stuck-open, driving t_{out} to a low voltage level, independent of the rest of the circuit.

In total, 67% of all possible shorts and bridging faults mentioned in Tables 4.1, 4.2, 4.3, and 4.4 are detectable at t_{out}. All of the bridgings in the BiCMOS NAND gate of

Figure 4.8 which are not detectable are among those that create a conducting path which includes the base of Q_2. Since this point is kept at a low voltage ($V_{BE,on} \leq$ 0.8 V), V_{gnd} cannot reach a voltage high enough to turn N_{t2} *on*. Since the BiNMOS and CMOS structures do not have a bipolar transistor in the pull-down path, their detection ratios increase significantly.

The DFT scheme of Figure 4.8 presents a number of undetectable faults at the test circuit. For the BiCMOS NAND gate, ten defects out of 30 defects (33%) at the test circuit are undetectable. They can produce masking problems only in the case of multiple faults. There are 33 defects in the gate shown in Figure 4.8 which cause an I_{DDQ} fault. 24 of these faults are mentioned in Table 4.1 and there are 9 faults inside the test circuit. It should be noted that among 33 defects which cause an I_{DDQ} fault in the gate shown in Figure 4.8, approximately 78% are detectable by this method. All of those which are not detectable create the excessive current only through the emitter node of Q_2. We will propose DFT methods in Chapter 5 which will detect these faults. If the proposed technique in this Chapter is combined with the DFT method presented in next Chapter, complete I_{DDQ} fault coverage will be achieved. Another method for achieving complete I_{DDQ} fault coverage is to connect the emitter of Q_2 to V_{gnd}. This, however, may degrade the noise margin as well as the delay performance, unless the area of N_{t4} is chosen properly.

It should be noted that several stages in a VLSI circuit can share a common V_{gnd}, thus, using a single test circuit. However, as the number of the stages increases, the required size for N_{t4} increases, because it must conduct the current for all stages. This constrain suggests that as shown in Figure 4.8, the output stage of the circuit (consisting of bipolar drivers) which carries high current should not be connected to V$_{gnd}$. The other advantage of using GND instead of V$_{gnd}$ in the final stage is that the output low level will not increase by the slight voltage drop across drain-source of N_{t4} transistor.

When applying the technique to a CMOS structure, transistor Q_t is replaced by an NMOS transistor. This is necessary in order to keep the circuit compatible with CMOS technology. Another advantage of using this technique in CMOS structure is the elimination of t_{out}. The reason is as follows. Figure 4.9 shows that V_{gnd} discharges to ZERO when the inputs do not excite the I_{DDQ} fault, through base-emitter of Q_2. This would set t_{out} to ONE. The primary output, however, shows correct transition from ONE to ZERO. In the absence of Q_2 in CMOS structure, there is hardly a path to the actual ground through the internal nodes of the logic circuit. This means that t_{out} would remain low, V_{gnd} would charge to V_{DD}, and the primary output stops functioning properly. The logic block to which the test circuit is connected behaves as if there is single or multiple stuck-at faults, and the output halts. This would convert the parametric fault into a logic fault. In this case, the I_{DDQ} fault is easier to detect since the output(s) stops functioning. This would, in turn, eliminate the need of t_{out}, and the area overhead associated with it.

It should also be noted that as the number of the stages in the circuit increases, the detection ratio will increase because there will be more paths to V_{gnd}. In order to verify this, a fully-CMOS master-slave JK flip-flop was implemented using two-input NAND gates, as shown in Figure 4.10. Simulations showed that 98% of all the short defects in the functional part of the circuit are detectable by the proposed DFT technique. If the shorts in the test circuit are also taken into consideration, the figure drops to 92%, which is still a very high coverage[1].

[1] We distinguish between fault coverage produced by test generation programs, and the transistor-level fault coverage. Fault coverage provided by a simulator is the ratio of the number of faults detected by test set to the total number of simulated faults. This figure is directly relevant only to the faults processed by the simulator; hence, a test with 100% fault coverage may still fail to detect faults outside the considered fault model by the simulator. Here, we have provided the

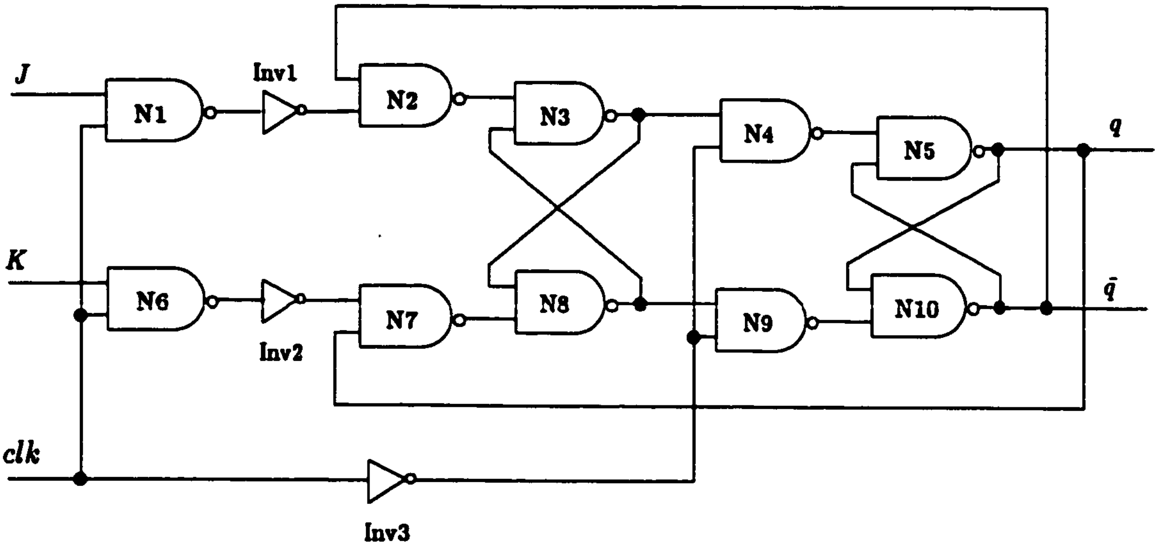


Figure 4.10: *A master-slave JK flip-flop with 2-input NAND gates*

Table 4.6 shows the effectiveness of the proposed technique for the four studied structures.

Circuit structure	Conventional	BiNMOS	Full-swing	CMOS
Detectable I_{DDQ} faults (%) including the test circuit	78	88	84	83

Table 4.6: *Fault detection comparison for 4 structures.*

Simulations show that if we model the short defects with larger resistances (e.g., 1K-10K) this technique is still effective.

In the test circuit, the sizes of N_{t2} and P_{t3} transistors (the first inverter) determine the required voltage at V_{gnd} which can lead to a fault detection, while the size of N_{t4} determines the maximum allowable I_{DDQ}. Thus, depending on the com-

actual fault coverage. We have counted every possible fault. Our fault coverage report is the first of its kind.

plexity of the main circuit, the widths of N_{t2} and N_{t4} can be adjusted for correct operation of the test circuit.

Applying the proposed technique has the following effects on the normal behavior of the circuit:

- The output voltage, corresponding to the ZERO level (V_{OL}) increases by only 80 mV. The $V_{IH,min}$ and $V_{IL,max}$ values remain unchanged. As a result, there is not a noticeable degradation in noise margin.

- In BiCMOS structures employing two bipolar transistors (i.e. conventional and full-swing), there is no observable increase in the delay characteristics because the bipolar transistors which drive the output capacitance remain unchanged. For BiNMOS and CMOS structures, there is a small delay overhead which can be minimized if N_{t4} is chosen appropriately.

- There is an area penalty due to the added devices and the two extra pins required for *init* and t_{out}. However, the penalty associated with this test circuit is not significant for a VLSI circuit, because a test circuit can be used for several independent stages. If a circuit is large enough to need more than one test circuit, a single *init* can be used for all test circuits, and the local t_{out} outputs can be applied to a CMOS NAND gate to make a single test output.

- No DC power is dissipated in the test circuit, because it is fully CMOS. Also, the transistor Q_t which is *on* in initialization mode does not contribute excessive power, because it is in the saturation region at steady-state situations (for CMOS structure, the NMOS transistor which replaces Q_t will be in linear region, with no power dissipation). For the conventional BiCMOS NAND gate, the steady-state power dissipation increased from 0.3nW to 0.5 nW. For a more complex system with a higher power dissipation, the increase will be quite insignificant, because only one test circuit will be used for the circuit. It is also possible to use an NMOS transistor instead of Q_t for all structures. This will significantly reduce the additional power, by compromising for either delay or area overheads.

These overheads can be well justified by the enhanced testability of the circuit.

4.4 Medium Size Logic Circuits

We have explained how the proposed I_{DDQ} test circuit works and compared its performance with those available in the literature. Advantages and disadvantages have been outlined. This comparison is limited to the functionality of the test circuit for the following reasons.

Automated Test Pattern Generation programs have been developed to find test vectors for I_{DDQ} fault detection purposes. Such test patterns have been graded 92% [146], 97% [155], and 98% [138]. Fault coverage reported by ATPGs is not the same as fault coverage reported for actual hardware[2].

Many published results on I_{DDQ} faults do not provide exact transistor sizes or the schematic diagram of the circuit under test. Moreover, there is no benchmark for I_{DDQ} fault testing.

Injecting I_{DDQ} faults on a fabricated chip and testing the chip for I_{DDQ} fault using DFT techniques is not helpful since each injection requires one chip being

[2] We noted in Section 3.3.1 that 100% fault coverage obtained using a ATGP may not be 100% actual fault coverage.

fabricated. This is not practical, and for this reason, there is no fault coverage reported in the literature today on this basis.

Obtaining fault coverage using I_{DDQ} fault injection on a circuit at the simulation level is also difficult. This means that a faulty circuit should be simulated using SPICE. The fault coverage is obtained after simulation is performed for each I_{DDQ} fault. This is tedious as it has to be done manually. For this reason, there is no fault coverage reported in the literature for I_{DDQ} fault testing using DFT testing techniques. To the best knowledge of the author, DFT fault coverage for I_{DDQ} faults has been reported in this dissertation for the first time.

We have selected three circuits for fault simulation which satisfy the following conditions.

- *Medium Size Circuits:* Since HSPICE simulation should be carried out for each line stuck-at fault which causes increased I_{DDQ}, we have selected medium size circuits.

- *Modularity:* In order to simplify data entry for HSPICE, we have used modular designs. Such designs are easily duplicated in HSPICE entry file giving us an opportunity to increase the size of the circuit as we wish. This will allow us to observe relationships between test circuit and circuit under test in terms of area overhead and fault coverage.

- *Input Criterion:* We have proposed I_{DDQ} testing technique which works online, and there is no need for ATPGs. Since we have simulated faulty machines for each input combination to test the design exhaustively, we have selected designs which have only a few inputs.

An 8-bit parallel binary full-adder [116], an 8-bit digital multiplier [116], an

ALU (74181), and an 8-to-256 decoder[3] [116] are the subjects for our I_{DDQ} testing.

The simulation has been carried out as the following steps describe.

- The logic designs have been changed to include only 2-input NAND gates and inverters.
- Gate-level designs have been transformed to transistor-level designs using CMOS NAND gates and CMOS inverters. NMOS and PMOS transistor sizes are the same as the ones we considered in this Chapter.
- We have considered fault-free input signals. All possible input combinations have been employed at the inputs using phase shift technique[4] similar to the inputs we applied for NAND gate simulations. This allows us to manage an exhaustive testing.
- We have injected single line stuck-at faults and short faults and enumerated I_{DDQ} faults by monitoring the power supply current. This is performed without a test circuit.
- We have connected our proposed I_{DDQ} test circuit between virtual ground and actual ground, and simulated the circuit with single stuck-at faults. The only output monitored was t_{out}. (The primary outputs have been ignored.) A fault is detected if t_{out} is zero, and the fault is not detected if t_{out} is one.

Table 4.7 provides simulation results.

We have been able to make the following observations through simulations. *Observation 1:* Our I_{DDQ} test circuit works based on the observation of supply

[3] An 8-to-256 decoder is constructed using seventeen 4-to-16 decoders.

[4] Other possible techniques are *frequency division* and counter.

Circuit structure	8-bit Adder	8-bit Multiplier	ALU 74181	Decoder
Detectable I_{DDQ} faults (%) including the test circuit	98.6	99.4	98.7	99.6

Table 4.7: *Fault detection comparison for four MSI structures.*

current and it is independent of the voltage levels of internal nodes or primary outputs.

Observation 2: The area overhead is linearly proportional to the size of the circuit. We have used one proposed I_{DDQ} test circuit (two CMOS inverters) per nineteen 2-input CMOS NAND-gates. This is roughly 5.2% increase in area overhead. The transistor sizes for the test circuit are selected by trial and error, and we speculate that this area overhead could be reduced. However, this area overhead is little price to pay to gain high I_{DDQ} fault coverage without the price of test generation attached.

Observation 3: The structure of a group of nineteen NAND gates have the same static and dynamic supply currents when the same number of transistors in the circuit under test is turned *on* or *off*. That is why we speculate that the fault coverage obtained would be roughly 99.5% for VLSI circuits. This speculation can best be visualized using a dual adder (multiplier) where the corresponding inputs of 8-bit adders (multipliers) are tied together. Any change that happens in one adder (multiplier) is duplicated and detected in the other part as well. Like other proposed I_{DDQ} testing techniques, partitioning should be used to deal with VLSI circuits. However, our proposed I_{DDQ} test circuit has only four transistors which is very small compared to 23 NMOS and PMOS transistors proposed in [115] or 12 NMOS and PMOS transistors in the I_{DDQ} test circuit proposed in [45].

4.5 Conclusions

In this chapter, we have characterized the effects of physical defects on different PUPD BiCMOS families. Our study confirms that the traditional stuck-at and delay fault models are not adequate for modeling the fault behavior of BiCMOS families. In order to achieve a high defect coverage, I_{DDQ} testing must also be employed. We presented a new design-for-testability technique for BiCMOS/BiNMOS/CMOS logic gates that results in a high coverage for shorts and bridging faults. It eliminates the need for fault excitation or propagation. The voltage levels of internal nodes remain in their defined logic level range of ZERO or ONE in fault-free circuit. The impact of this circuit modification on the behavior of the circuit in normal mode has been shown to be insignificant.

Chapter 5

Delay Faults in Asynchronous Circuits

Theorems 4.1, 4.2, 4.3, and 4.4 in Chapter 4 state that the set of "delay faults", "stuck-open faults", and "I$_{DDQ}$ faults" is an adequate fault set to detect faults (stuck-at faults included) in a CMOS circuit. The fault characterization, reported in Chapter 4, showed that most open defects result in either a delay or a stuck-open fault. Short defects, on the other hand, most frequently result in an I_{DDQ} fault and/or intermediate output. Short defects and their effects on BiCMOS circuits were studied in Chapter 4, and a DFT technique was proposed for facilitating the detection of these defects. This chapter deals with delay and stuck-open faults, and presents a novel DFT technique for detecting them. We show that the modified version of this technique is capable of detecting short defects too.

We distinguish *concurrent* and *on-line* testing techniques using the following definitions.

Definition 5.1 *The previous or current value of the output need not be known for* *detecting the fault in a concurrent testing technique. However, the circuit may or may not operate in normal mode while testing is performed.*

Definition 5.2 *We call a testing technique on-line if the testing is performed while the circuit is in normal mode of operation.*

Occasionally, the faulty node should be excited by a test vector, and/or its faulty behavior should be propagated to the primary output. To attain this goal, one or two test vectors are devised using test generation algorithms, and applied to the primary inputs of the circuit. The testing is achieved if it begins after the normal operation of the circuit is halted. Such a testing technique is neither concurrent nor on-line. On the other hand, there are testing techniques [58, 60, 64] which operate using the physical characteristics of the faulty node, and do not rely on exciting (or propagating) the fault. However, these techniques may detect the fault if the circuit is not functioning in normal mode. This class of testing technique is considered to be concurrent in this Chapter. If the faults in the circuit are detected as they occur in normal mode of operation, the testing technique is on-line.

The rest of this chapter is organized as follows: Section 5.1 reviews delay faults and the importance of detecting them. It also discusses the drawbacks of the techniques proposed in [58, 60, 64] for detecting this fault. In Section 5.2, we present a DFT technique which is free from similar shortcomings. Stuck-open and intermediate output faults are the subject of Section 5.3. Fully testable CMOS designs are discussed in Section 5.4. HSPICE simulation results are given in Section 5.5. Generation of controlling signals for asynchronous circuits are discussed in Section 5.6. Their faulty behaviors are explained in Section 5.7. The test results for medium size circuits are reported in Section 5.8. Conclusions are provided in Section 5.9.

5.1 Delay Faults in BiCMOS Circuits

It was mentioned in Chapter 3 that some defects do not change the functional behavior of the circuit, and only increase the propagation delay of the circuit from the specified limit. The result is a delay fault, which may manifest itself as a "slow to rise" or a "slow to fall" transition. It is important in high-speed applications to detect delay faults to ensure that the circuits work as intended. Also, it has been shown [111] that even those delay faults which are not important in a particular application may cause functional problems and create logical faults in BiCMOS circuits. Thus, delay fault testing is of great importance. The simplest form of delay testing is to apply pseudo-random patterns to the circuit under test (CUT) at a desired speed. This method requires very long test lengths to achieve an acceptable fault coverage [164]. The other drawback of this method is that tests can be invalidated by delays in other parts of the circuit. A test that detects a delay fault of a certain size does not necessarily detect faults that have longer or shorter delays [64].

The analyses described in Chapter 4 supported the results reported in the literature [106, 111, 129–131, 160] that most open defects manifest themselves as delay faults. However, a few short defects also result in a delay fault. These analyses further revealed that as the redundancy of BiCMOS structures increases, less open defects can be detected by stuck-open tests. In order to detect these defects, delay testing must be performed.

Delay faults appear as one of the seven alternatives shown in Figure 5.1. Graph "a" shows a fault-free output signal, in a high-to-low transition. The waveform in graph "b" is slow-to-fall. Although it starts its transition as early as the fault-free signal, it requires much more time to reach its final value. The waveform depicted

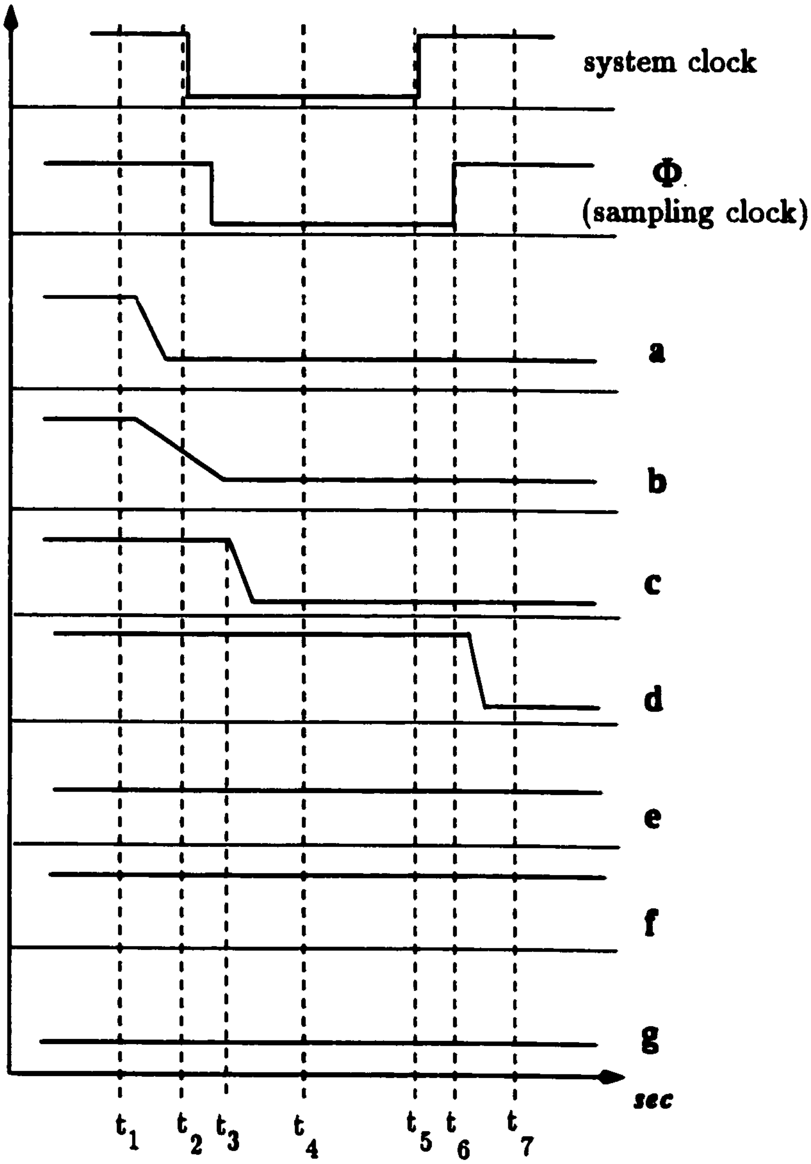


Figure 5.1: *Different appearances of delay fault*

in graph "c" does its high-to-low transition as fast as the fault-free signal. However, there is a time delay (t_3 - t_1) before it starts the transition. The waveforms in graphs "a", "b", "c", and "d" show high-to-low transitions. Similar malfunctionings can occur in a low-to-high transition. (t_3 - t_1) in case "d" is too late. The sampling clock has changed its logic value and the testing phase is terminated already. The waveform in graph "e" shows an intermediate voltage value which cannot be verified as either '1' or '0'. While graph "f" shows stuck-at 1, graph "g" shows stuck-at 0. The waveforms in graphs "e", "f", and "g" can be interpreted as a delay fault with infinite delay. Although test pattern generation for stuck-at faults is different than that of delay faults but similar in many ways, graphs "e", "f", and "g" suggest that these faults can be detected by the test patterns employed for delay fault detection. We show that a delay fault testing technique that detects fault type "b", may not detect fault type "c". There are cases where the delay fault technique detects types "b" and "c", but not "e", "f", or "g".

In either of the cases "b" and "c" as depicted in Figure 5.1, the faulty output finally reaches the correct value. However, if the output is observed at an *appropriate* time (at the maximum acceptable normal delay), the fault can be detected. To detect a delay fault, a two-pattern test is applied to create and propagate signal transitions along the path to be tested. The first input vector initializes the output to a known state. The circuit is allowed to stabilize under this input vector. At time t_1, the second input vector is applied such that the output changes to the opposite logical level in the fault-free circuit. The output is sampled at time t_2, where $\delta = (t_2 - t_1)$ corresponds to the maximum acceptable delay for the fault-free operation. The clock frequency is proportional to the inverse of $max(\delta)$. If the output does not reach the opposite logical level in the presence of any of the two types of faults, then the fault can be detected.

Two-pattern tests have been widely used for testing stuck-open and delay faults in logic circuits. However, they can be invalidated by circuit delays, hazards, and timing skews [92, 154]. As a result, there have been some efforts to eliminate the need for them. One such effort, is the design-for-testability technique proposed in [60,64], described in Chapter 3. This technique suffers from the following shortcomings:

- This method is capable of detecting delay faults only in the presence of hazards in the CUT response. If all the outputs have single transitions, the delay fault remains undetected. We simulated the proposed circuit shown in Figure 3.11 and observed that with careful selection of transistors width-to-length ratios, it is possible to detect slow-to-fall signals even for single transition outputs. However, slow-to-rise signals cannot be detected because output M in Figure 3.11 can be discharged to 0 only if out_i makes a transition from 1 to 0.

- Node M of Figure 3.11 does not go all the way down to 0 V, because PMOS transistors are imperfect switches when passing a 0. This is due to the fact that a PMOS transistor with gate node at voltage 0, begins to turn *off* when the output voltage reaches V_{tp} (threshold voltage). Similarly, NMOS transistors are imperfect switches when passing a *ONE*, because an NMOS transistor with gate at V_{DD} ceases conducting when its output reaches $V_{DD} - V_{tn}$. Therefore, the output does not go lower than V_{tp}. Our simulations show that M in Figure 3.11 does not discharge below 1.5 V. This, reduces the noise margin and may cause difficulties in fault detection.

Another design-for-testability technique is proposed in [58] and is shown in Figure 3.12. The method has several drawbacks.

- The authors have pointed out that their delay test circuit fails to detect an existing fault if skews happen in the transitions of Φ_1 and Φ_2. Such behavior is expected either due to the presence of clock skews, or the delay associated with the inverter which generates $\Phi_2 = \bar{\Phi}_1$. The authors have also shown that glitches at the test circuit nodes can invalidate the results.

- The presented method also needs ratioed capacitors.
- The method can detect one type of faults which causes a delayed transition (see Figure 5.1 (c)). Type "b" is not detected by this method.

5.2 Proposed DFT Technique for Delay Fault Detection

We propose a new DFT technique for detecting delay faults, which does not suffer from the shortcomings of the techniques presented in [58, 60, 64]. We adopt their approach of observing the output waveform between the samples instead of only latching the output at sampling time. This will ensure that our method will not be invalidated by excessive delays. However, in the proposed circuit, the generation of the error signal is not based on the current drawn during the transition. On the contrary, it is based on charging or discharging an internal node to a logic level. Like the techniques proposed in [58, 60, 64], we monitor any changes in logic value of the internal node. The proposed technique is graphically depicted in Figure 5.2.

Figure 5.2 shows the similarities and differences between our proposed technique and those proposed in [58, 60, 64]. Here, the control signal Φ_1 or Φ_2 is positioned in the middle and the circuit signal is distributed to the uppermost PMOS and the lowest NMOS transistors. The circuit in Figure 5.2 (a) operates as follows. Φ_1 is *high* and Φ_2 is *low* in normal mode of operation. This would create a short

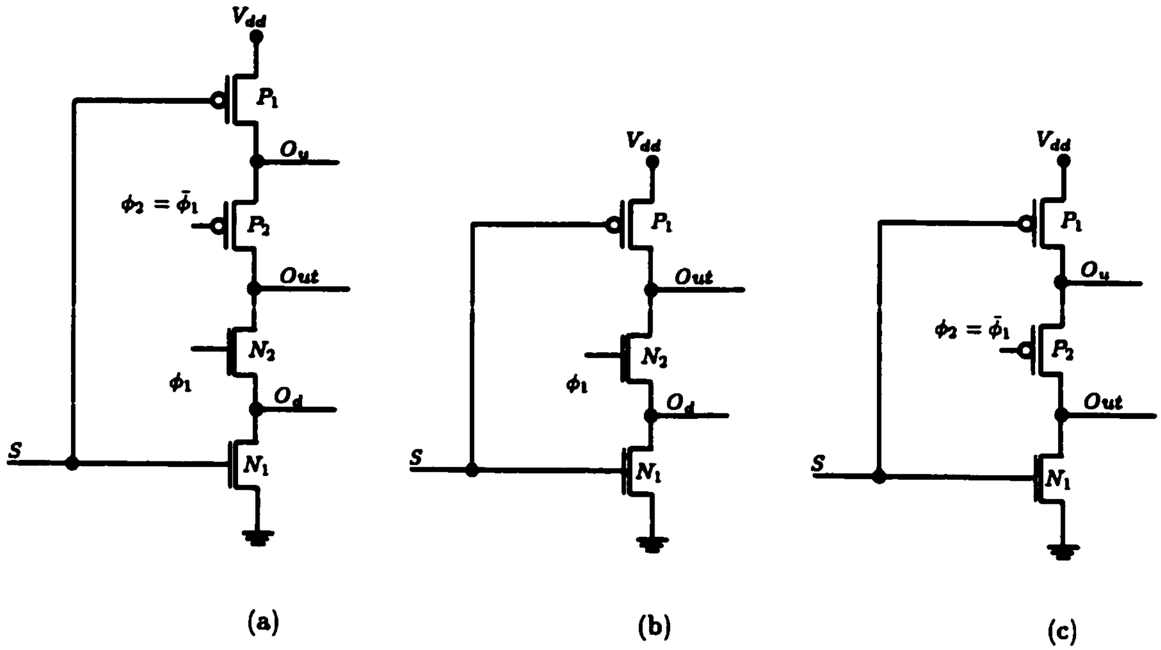


Figure 5.2: *Proposed concurrent checkers for delay faults: (a) using two control signals Φ_1 and Φ_2, (b) using Φ_1 only, (c) using Φ_2 only.*

path between O_u and O_d, and the the test circuit operates like a CMOS inverter. Regardless of what S is, O_d and O_u have the same logic values, which is the inverted value of signal S. The rising transition of Φ_1 or the falling transition of Φ_2 marks the end of the maximum delay in the fault-free circuit (t_2 in Figure 5.1). Therefore, when Φ_1 changes to *low* or Φ_2 changes to *high*, all the output signals of the fault-free circuit have already stabilized to their final values. At t_2, both transistors P_2 and N_2 are turned *off*. At this time, one of the nodes O_d or O_u becomes floating, and tends to maintain its previous value. Since there is no delayed transition of the signal S after time t_2 in the fault-free case, O_d and O_u keep their previous values. In a circuit with excessive delay, however, one of the outputs will change after the expected time, t_2. As a result, the state of the P_1 or N_1 transistor will change. This will cause the logic value to change in one of the nodes O_d or O_u, i.e., either O_d or

O_u will assume a value other than its previous value. As a result, O_dO_u will either be 01 or 10. The difference in logic values of O_d and O_u is detected by an XOR circuit.

Figures 5.2 (b) and (c) operate on a similar basis. They intend to provide a short path between *Out* and either of O_d or O_u in normal mode, and disconnect P_1 from N_1 in test mode. None of these techniques need ratioed capacitors, and glitches on Φ_1 or Φ_2 would not have any effect on testing results. As an example, consider Figure 5.2 (a). As soon as either of Φ_1 or Φ_2 changes its normal mode logic value, the uppermost and the lowest transistors are disconnected and the circuit is ready for fault inspection. Figures 5.2 (b) and (c) offer less area overhead compared to that of Figure 5.2 (a) because they have one transistor less, and the generation of the inverted control signal value is prevented. We recommend Figure 5.2 (b) since NMOS transistors tend to operate faster and introduce little voltage drop while they are *on*.

We propose test circuits for CMOS circuits which work on the same principles, see Figure 5.3. We have adopted the structure in Figure 5.3 (b) for the rest of this Chapter since it introduces less area overhead, and has less loading effect on normal operation of the CMOS circuit.

When input patterns in CMOS circuits are such that the output tends to become ONE, the n-part is turned *off*, creating a high impedance between the output node and the ground. This permits the output node to take any new voltage value or keep its previous voltage level. At the same time, the p-part provides a short path between the output node and V_{DD} charging the output capacitor to logic ONE. Similar events happen when the output tends to become ZERO. While the p-part is presently *off*, a discharging path for the output node is provided through the n-part to the ground. Although both parts are *on* for a short time while the output is

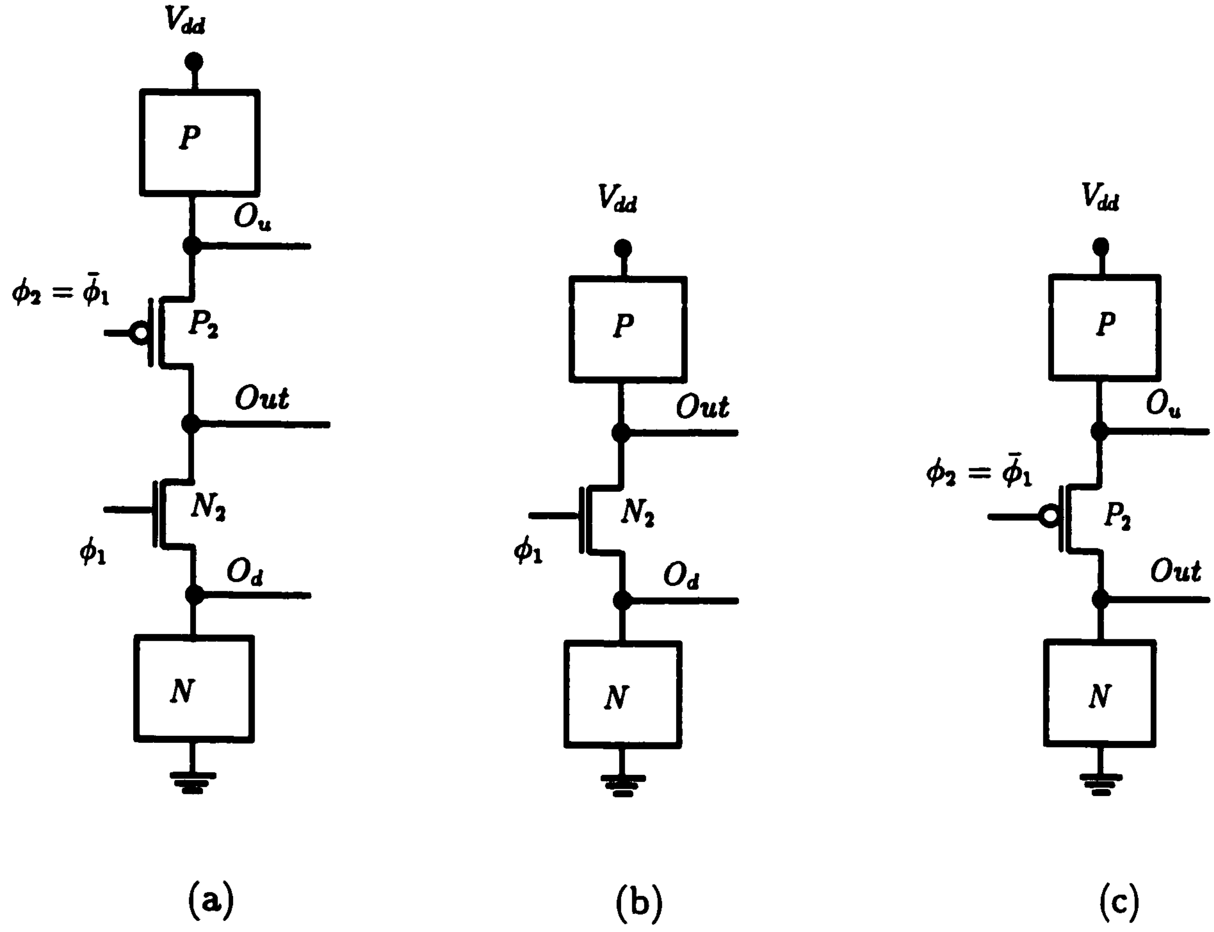


Figure 5.3: *Proposed concurrent checkers for delay faults: (a) using two controlling signals Φ_1 and Φ_2, (b) using Φ_1 only, (c) using Φ_2 only.*

in transition, it should not last long for a fault-free circuit. We will investigate the performance of the circuit shown in Figure 5.3 (b) in Section 5.4 where we design a fully testable CMOS design. Let us illustrate the circuit operation for delay fault testing using an example.

Let us assume that at $t < t_3$, the voltages at nodes *Out* and O_d are at V_{DD} and Φ_1 is high. We further assume that the inputs are applied long ago so that *Out* and O_d are supposed to fall (see Figure 5.1.c) in fault-free circuit *at* or before Φ_1

changes to ZERO, i.e., the p-part is *off* and the n-part is *on* at t_2. Since there is no delayed transition of the output signal after time t_2 in the fault-free case, *Out* and O_d will keep their previous values. The p-part turns *off* at $t_3 > t_2$ if a delay fault exists. At this time, node *Out* becomes floating, and tends to maintain its previous value, which is ONE in this example. The n-part, on the other hand, turns *on* and discharges O_d to ZERO. This provides a difference in the logic values of *Out* and O_d which is detected by an XOR and a delay fault is announced.

A stuck-open fault can be interpreted as a delay fault with infinite delay. This, suggests that it can be detected by the means employed for delay fault detection. However, the concurrent delay checker presented in this Section is not capable of detecting stuck-open faults, because it responds to the delayed *transition* of the signal under test. If the signal does not make any transition for a test vector (which is the case for stuck-open faults), it will remain undetected with this method. Therefore, it is necessary to devise a DFT technique for detecting stuck-open faults.

The following Sections demonstrate that the technique mentioned above, if modified, is capable of detecting stuck-open, stuck-on, inconclusive voltages, and I_{DDQ} fault. We prove that the proposed technique also provides single and multiple fault detection at the expense of a little area overhead.

5.3 Detecting Stuck-Open Faults

The DFT technique presented in Chapter 4 results in a high coverage for short defects and bridging faults in BiCMOS circuits. Its inability to detect most open defects is compensated by the concurrent BIST technique presented in Section 5.2 for delay fault detection. While the first technique targets the short defects, the second technique attempts to detect the open defects which normally result in delay faults in CMOS/BiCMOS circuits. The stuck-open fault occurs when the output becomes floating (not connected to either V_{DD} or GND via a conducting path). Therefore, the output remains at its previous value. So, for an input vector that excites a stuck-open fault, the faulty output is different from the fault-free output. Thus, the standard method for detecting stuck-open faults is to apply two-pattern test vectors and observe the output. However, generation of robust two-pattern tests (which are not invalidated by timing skews) is a complex process. The requirement of large CPU times makes the test generation very costly. It is also possible that a combinational circuit may not have any two-pattern robust sequence [154].

5.3.1 Proposed DFT Technique for Stuck-open Fault Detection

The basic idea of the proposed technique is to apply a test signal to the output such that it undergoes a consequent transition in case of a stuck-open fault. The concept of applying an external signal to the output is an adaptation of the design method presented in [92,130,144]. Similarly, our proposed circuit does not need two-pattern test vectors. Therefore, it cannot be invalidated by timing skews, delays, glitches, or charge sharing among the internal nodes. Figure 5.4 shows the concurrent circuit for stuck-open fault detection. It consists of two NMOS transistors N_3 and N_4, which are used for generating a transition on an output affected by a stuck-open fault. The operation of the circuit is based on the fact that Out and O_d have equal logic values in the fault-free circuit. These outputs will not be affected if they are momentarily connected to a weak signal. However, if the circuit has a stuck-open fault, both the p-part and the n-part show high-impedance states if the input

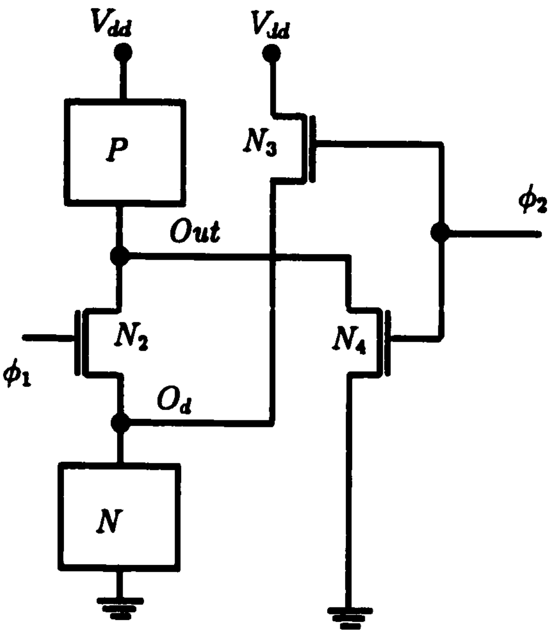


Figure 5.4: *Proposed concurrent stuck-open detector.*

vector excites the fault and propagates it to the output. Therefore, a weak signal can overdrive the output. These weak signals are applied via N_3 and N_4 which are controlled by the sampling clock, Φ_2.

The dimension of N_3 and N_4 should be chosen such that the *on*-resistance of these transistors are considerably higher than that of the pull-up or that of the pull-down network. In general, minimum-size transistors are sufficient for this purpose.

O_d does not charge to full V_{DD} when Φ_2 is *high* because an NMOS transistor does not transfer ONE efficiently. Although it is possible to avoid this performance degradation by using a PMOS transistor instead of NMOS, we do not recommend it, because the voltage degradation in O_d does not create any major problem in the operation of the circuit. On the contrary, it is helpful as Table 5.3 shows. If a PMOS transistor is used instead of the NMOS transistor, the complementary signal $\overline{\Phi}_2$ will be required. However, this is the same as Φ_1.

In normal operation, Φ_1 is *high* and Φ_2 is *low*; therefore, the main output is not affected by the test circuit. In test mode, Φ_1 falls to ZERO and turns off N_2. Then, Φ_2 goes to high value and turns on both the N_3 and N_4 transistors. A floating *Out* (O_d) will change to ZERO (ONE). There are two scenarios in the fault-free circuits. If *Out* and O_d are both ONE, the n-part is *off*. This will leave O_d floating. N_3 will charge O_d to high logic value. At the same time, the p-part is *on*. This provides a short path between *Out* and V_{DD}. Although N_4 is *on*, it cannot overdrive *Out* to ZERO since its *on*-resistance is much higher that that of the p-part. Therefore, both *Out* and O_d will tend to be high in the fault-free circuit. If there is a stuck-open fault which is excited by the input patterns, then both pull-up and pull-down networks are *off*, showing high impedance between their two terminals. In this situation, N_3 will drive O_d to ONE and N_4 will drive *Out* to ZERO. Therefore, the fault is detected using a logic gate. Similar behavior is explained if *Out* and O_d were originally ZEROs. This analysis also suggests that the output value need not be known. *This eliminates the requirement of the test-pattern generation procedure.* For each input signal applied to the circuit in normal mode, the circuit will switch to test mode after the output has had enough time to stabilize in the fault-free case, and any occurrences of stuck-open faults will be detected.

5.4 Fully Testable CMOS Designs

In this Section we further improve our test circuit to provide a fully testable CMOS design. Table 5.1 summarizes all possible faulty and fault-free functional behaviors of *Out* and O_d if the structure of Figure 5.4 is used. Transistors N_3 and N_4 have no effect on normal operation of the fault-free circuit regardless of whether Φ_1 is high or low. Therefore, the controlling signal Φ_2 may have any logic value at any time.

Φ_1	Φ_2	p-part	n-part	*Out*	O_d	Observation
0	x	on	off	1	1	Fault-free
0	x	off	on	0	0	Fault-free
0	0	off→ on	on	0→1	0	Delay fault
0	0	on	off → on	1	1→0	Delay fault
0	1	off	off	0	1	Stuck-open
0	1	on	on	1	0	I_{DDQ} fault

Table 5.1: *Faulty and fault-free behaviors of CMOS circuits at* $t_3 > t_2$.

However, to minimize the power dissipation, we have chosen Φ_2 to be low when Φ_1 is high. This will further facilitate the use of on-line I_{DDQ} test circuit proposed in Chapter 4. On the other hand, Φ_2 must be low for delay fault testing. We can choose Φ_2 so that $\Phi_2 = \bar{\Phi}_1$ for stuck-open and I_{DDQ} fault testing. We can generate this signal through an odd number of inverters to create a time gap long enough for the purpose of the delay fault testing.

We have discussed the behavior of the test circuit proposed in Figure 5.4 at the presence of delay and stuck-open faults in Sections 5.2 and 5.3.1. This test circuit is also capable of detecting I_{DDQ} faults which may not be detectable through supply lines. Only one of the p-part or the n-part is *on* and the other part is *off* in the fault-free circuit. However, both parts tend to create a short path between their two terminals in the case of an I_{DDQ} fault: i.e., the p-part tends to charge *Out* to V_{DD}, and the n-part tends to discharge O_d to ZERO. The I_{DDQ} fault is announced at the combination $(Out = 1, O_d = 0)$.

Our fully testable CMOS design is schematically illustrated in Figure 5.5. Signal *Out* is connected to N_5 and P_6. Similarly, signal O_d is connected to P_5 and N_6.

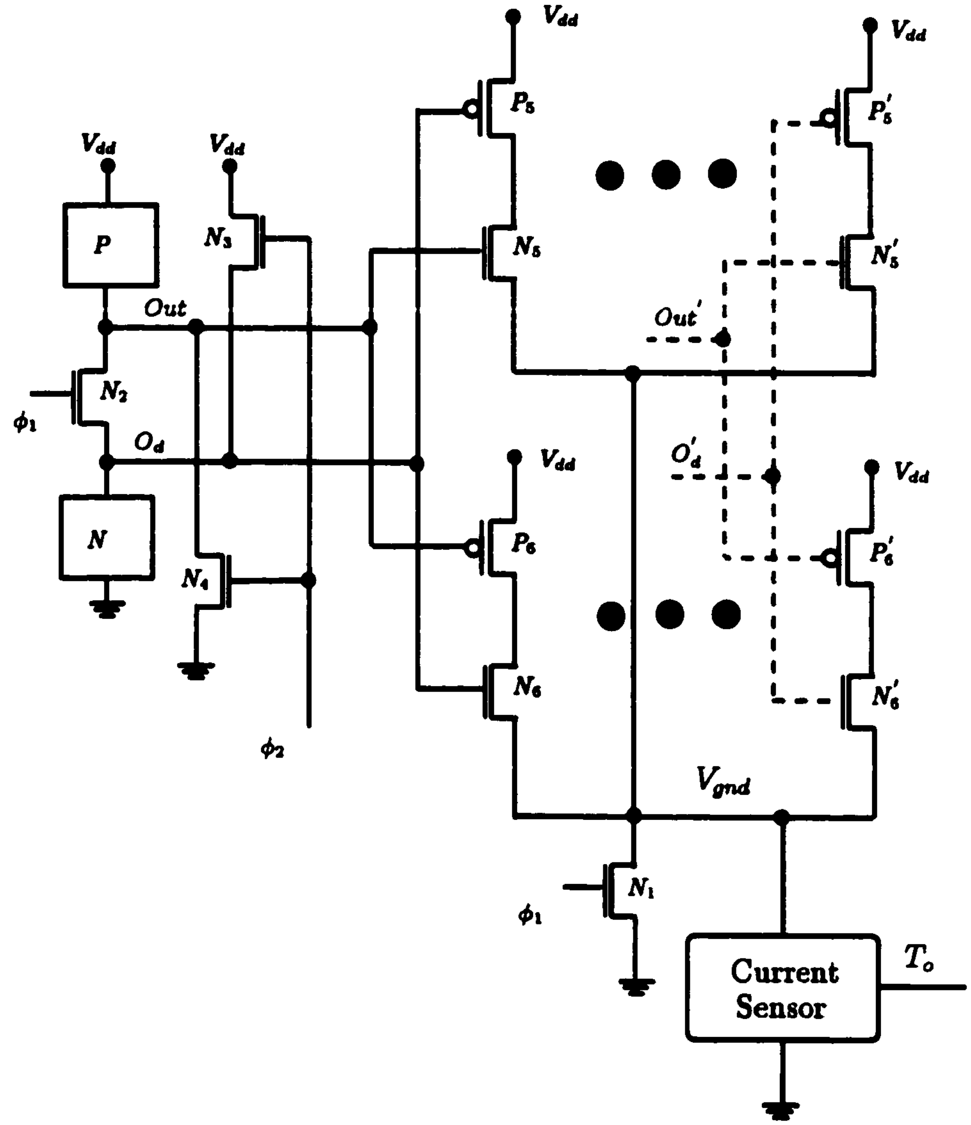


Figure 5.5: *Proposed fully testable CMOS design.*

Figure 5.5 shows only two set of signals (Out, O_d) and (Out', O_d'), while in a complex sequential circuit any set of outputs (Out, O_d) representing the states of the primary outputs of each CMOS block can be tested by the above circuit. One NMOS transistor N_1 is used to create a short path between V_{gnd} and the actual ground, and to reset the current sensor during the normal operation. The current sensor which is a combination of two inverters as explained in Chapter 4 is employed for generating the error signal. One transistor N_1 and one current sensor are used to serve many CMOS blocks for testing purposes. The test circuit as shown in Figure 5.5 does not require the complement of any of the output signals.

The circuit operates as follows. Φ_1 is *high* and Φ_2 is *low* in normal mode. N_1 is *on* and V_{gnd} is initialized to ZERO. Φ_1 switches to *low* in test mode after the output signal has had enough time to stabilize. If the circuit is fault-free, *Out* and O_d have the same logic value, and they tend to preserve it. P_5 (P_6) is not *on* concurrently with N_5 (N_6). The leakage current is negligible, and the output of the

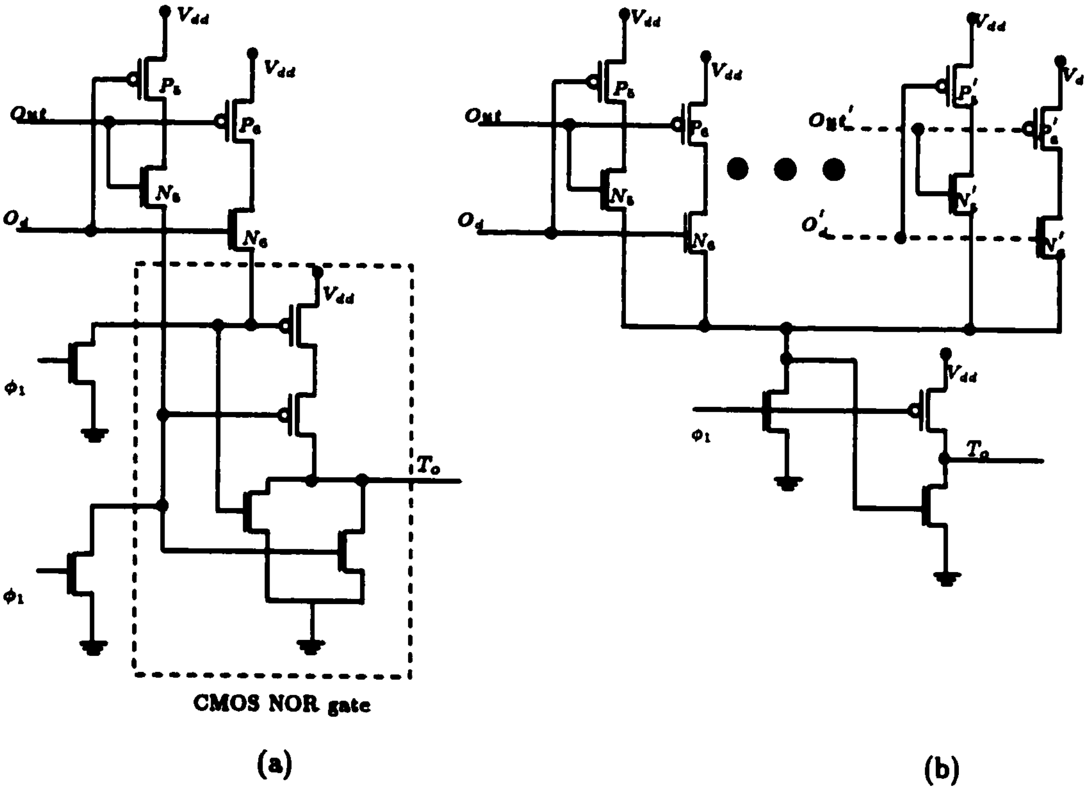


Figure 5.6: *Alternative solutions for the current sensor: (a) using a NOR gate, (b) using one inverter.*

current sensor is high. At the presence of any fault, *Out* and O_d will have opposing logic values. This, in turn, will provide a short path between V_{DD} and V_{gnd} through either (P_5, N_5) or (P_6, N_6). The excessive current is, then, detected by the current sensor, and the error signal becomes *low*. This will, in turn, disconnect V_{gnd} from actual ground preventing excessive power dissipation.

Alternative solutions are presented in Figure 5.6. The current sensor is replaced in Figure 5.6 (a) by a CMOS NOR gate. This will increase area overhead by almost 100% as the NOR gate is used for each CMOS stage separately, or a multiple input NOR gate is used. Favalli's technique [58] is adopted in Figure 5.6 (b). This, however, suffers from the setbacks outlined earlier in Chapter 4.

Table 5.2 summarizes the operation of the test circuit for $t_3 > t_2$. An intermediate output is produced when both pull-up and pull-down networks conduct, and neither can overdrive the other. As a result, the output lies in a region close to (but outside) the two defined logic level. In this faulty case, both sets of (P_5, N_5) and (P_6, N_6) are conducting. This would produce excessive current which is detected

Φ_1	Φ_2	p-part	n-part	*Out*	O_d	P_5	N_5	P_6	N_6	Observation
0	x	on	off	1	1	off	on	off	on	Fault-free
0	x	off	on	0	0	on	off	on	off	Fault-free
0	0	off→ on	on	0→1	0	on	on	off	off	Delay fault
0	0	on	off → on	1	1→0	on	on	off	off	Delay fault
0	1	off	off	0	1	off	off	on	on	Stuck-open
0	1	on	on	1	0	on	on	off	off	I_{DDQ} fault
0	x	on	on	X	X	on	on	on	on	Intermediate

Table 5.2: *Faulty and fault-free behaviors of CMOS circuits at* $t_3 > t_2$.

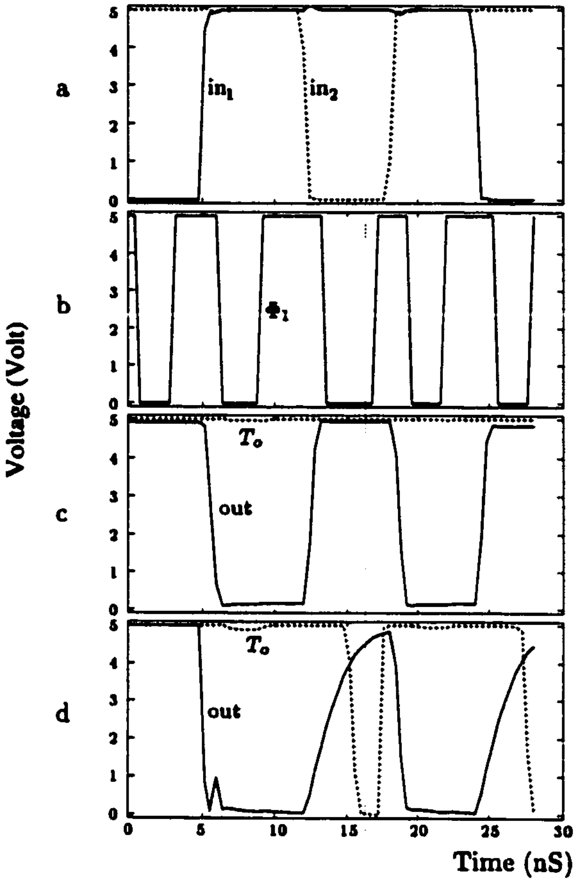
by the current sensor. The fault is detected regardless of whether N_3 and N_4 would overdrive the output or not. This implies that the proposed circuit can be used for detecting the defects which cause an intermediate output.

5.5 Evaluation

5.5.1 Delay Fault Testing

In order to evaluate the capability of the circuit in detecting delay faults, we used a conventional BiCMOS NAND gate (Figure 4.1) as the CUT. HSPICE simulations were done for all defects which cause a delay fault. Figure 5.7 shows the performance of the delay tester for the fault-free operation, as well as for beQ_1 (base-emitter short in transistor Q_1) defect. This defect, as described in Chapter 4, results in a slow-to-rise output. The fault is excited by any input pattern which causes a low-to-high output transition. Graph "a" shows the inputs in_1 and in_2. The sampling clock Φ_1 is shown in graph "b". It is *high* for all the period when the fault-free output (graph "c") is allowed to have its changes, and becomes *low* after the maximum acceptable delay. For the fault-free output, the T_o signal constantly remains *high*, because there is no excessive current. A delayed output (graph "d"), however, makes all or part of its transition after Φ_1 has become *low*. As a result, the T_o signal assumes a low level, and the delay fault is detected, as depicted in graph "d".

As depicted in Figure 5.1, there are different types of delay faults. The delay fault shown in Figure 5.7-d is from type "b" in Figure 5.1. In order to observe a delay fault of type "c", the inputs of a conventional BiCMOS NAND gate as shown in Figure 4.1 were tied together to form an inverter, and five such inverters were cascaded immediately after one conventional BiCMOS NAND gate, see Figure 4.1.



delay fault detected in "d"

Figure 5.7: *Delay fault detection using the concurrent checker. a) inputs, b) sampling clock, c) output and delay signal for the fault-free circuit, d) output and delay signal for the circuit with beq$_1$ defect. "T_o" assumes a low voltage level for the delayed signal.*

The last inverter was augmented with the proposed DFT circuitry. A defect was introduced in one of the gates to create a delay fault. Figure 5.8 shows the results of a simulation. As graph "c" shows, this type of delay fault is also detectable by the proposed DFT technique.

We repeated the simulations for other families (BiNMOS, full-swing BiCMOS,

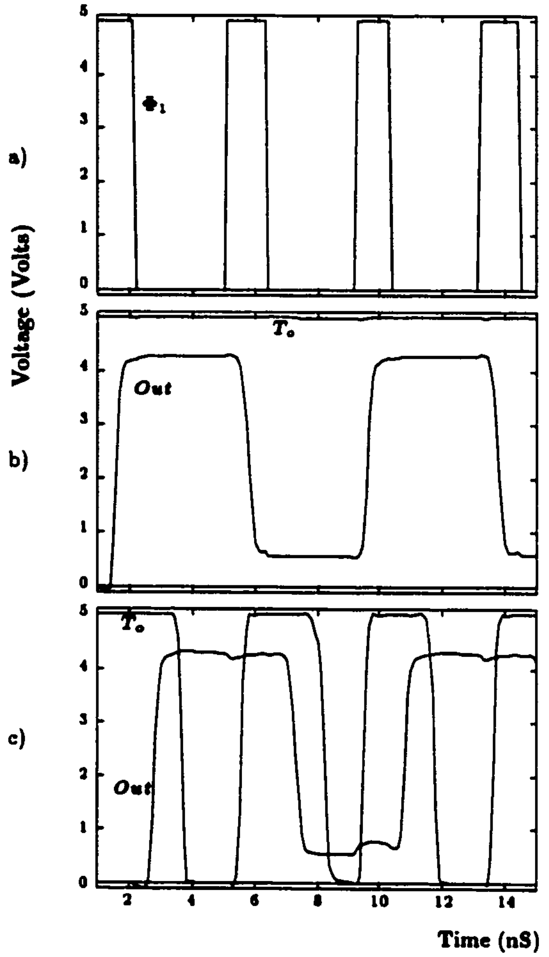


Figure 5.8: *Detecting delay fault (type "c") using the concurrent checker. a) sampling clock b) output and delay signal for the fault-free circuit, c) output and delay signal for the faulty circuit. "T_o" assumes a low level for the delayed signal.*

and CMOS as shown in Figures 4.1, 4.2, 4.3, and 4.4). Simulations show that the proposed concurrent delay checker circuit is applicable to all mentioned families, and is capable of detecting *all* the delay faults as given in Tables 4.1, 4.2, 4.3, and 4.4.

5.5.2 Stuck-Open Faults

In order to evaluate the performance of the proposed scheme for detecting stuck-open faults, a conventional BiCMOS NAND gate is augmented with the mentioned structure. For each possible single defect (three shorts and three opens for each transistor), simulation was performed and the error signal was monitored. Figure 5.9 shows the result of HSPICE simulation for an sP_1 defect (source of P_1 is open).

The fault-free output changes from *low* to *high* when in_1in_2 change from 11 to 01. The error signal remains *high* for all input vectors. The faulty output, however, assumes a high-impedance state and stays at its previous value or accepts new logic value. As a result, the fault is detected as shown in graph "d" of Figure 5.9. All stuck-open and inconclusive voltage values provided in Table 3.3 are detectable by this method.

Similar simulations were performed for other families (BiNMOS, full-rail BiCMOS, CMOS). The simulations showed that all the stuck-open and intermediate output faults are detectable by the proposed technique. The results obtained from simulations can be generalized by the following theorem:

Theorem 5.1 *By augmenting a CMOS/BiCMOS gate as shown in Figures 5.5 and 4.7, any single or multiple fault in the functional part can be detected concurrently if the test circuit is not faulty. The error signal is not invalidated by timing skews/delays, glitches, or charge redistribution.*

Proof: In Section 5.5, it was shown that the test circuit, shown in Figures 5.2 or 5.3, has no effect on logic values of *Out* and O_d if the circuit is fault-free. This would allow excitation of the fault in the next stage CMOS blocks through the primary

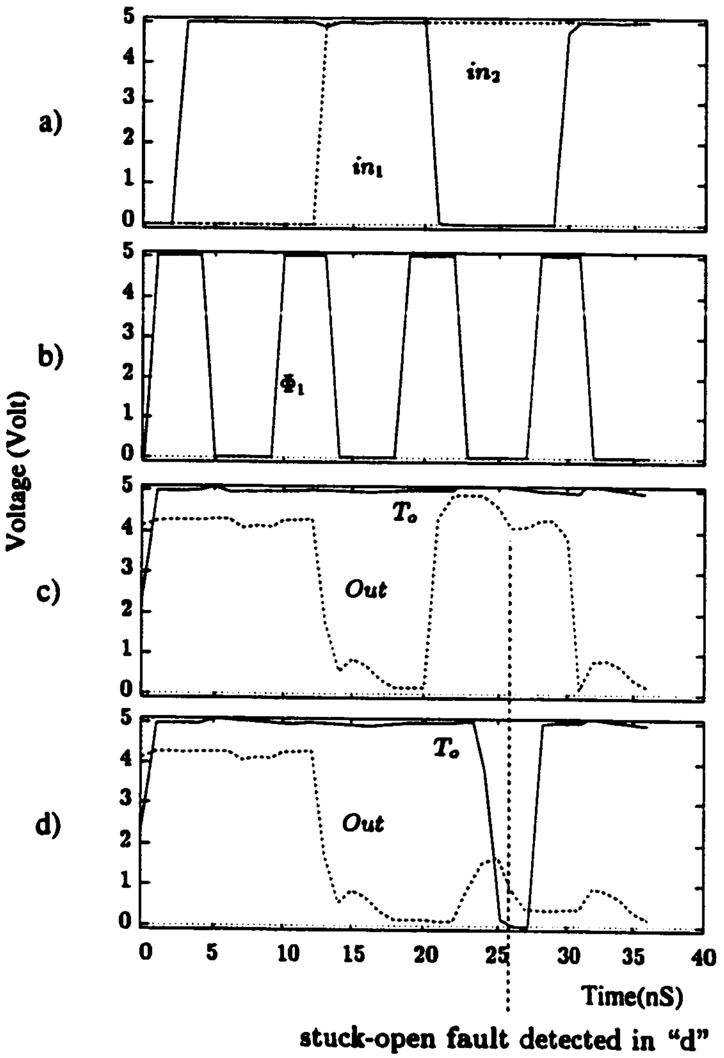


Figure 5.9: *Stuck-open fault detection using the proposed BIST technique. a) inputs, b) control signal, c) output and error signal in the fault-free case, d) output and error signal for a stuck-open fault.*

inputs. It was also shown that any single fault in the functional part is detectable. Similarly, if a multiple fault exists within one CMOS block, its faulty behavior is detected through the test circuit, or the fault is announced undetectable due to redundancy. There are two possible connections of CMOS blocks. They are either connected serially, or they operate in parallel. In the case of serial connection, the first faulty CMOS block would switch the error signal. At the same time, the values of Out and O_d are changed due to the fault, see Figure 5.5. Whether this combination of (Out, O_d) excites the fault in the next CMOS block is not important since the fault is already detected. In the case of parallel connection of two CMOS block, the multiple fault is detected by both of the test circuits connected to them.

The previous or current value of the outputs need not be known for detecting the fault. A fault is announced if the output changes while Φ_1 is ZERO. Hence, one does not need to set the circuit output to ONE or ZERO by applying input patterns generated by an ATGP. This eliminates the need for the generation of test vectors in CMOS structures. According to our definition, the testing is performed concurrently.

Since Φ_1 goes to ZERO after all the internal signals are normally stabilized, no glitch should appear at the output. However, if some signals experience excessive delay faults, they may create a glitch at the output during testing (while Φ=0). This glitch may change the output momentarily, and a delay fault or stuck-open fault is announced. The outputs will change at the presence of other types of faults as well. The circuit is announced fault-free if, and only if, the outputs have equal logic values. QED.

It should be noted that an open defect in any of the additional transistors N_3, N_4, P_5, N_5, P_6, or N_6 of Figure 5.5 may or may not be detected in this design. They may also invalidate our test results. It can be shown that such a defect does not affect the normal circuit operation. The short defects in the test circuit, however, are detectable.

Applying the proposed technique has the following effects on the normal behavior of the circuit: the output voltage levels of the circuit do not change for the fault-free circuit under normal-mode of operation. Speed performance is not degraded noticeably. The power dissipation of the circuit increases because either N_3 or N_4 try to overdrive one of the outputs in the fault-free circuit. This increase is in an acceptable range ($\leq 15\%$) as shown in Table 5.3. Although it may seem as if the only disadvantage of this technique, compared to the previous ones ([92,130,144]), is its area overhead, this is not the case if we compare the area overhead attributed to this technique with that of all previously proposed counterparts combined for detecting different types of faults. Our test circuit shares many transistors to perform fault detections, and the increase in testability is quite significant.

Table 5.3 shows the relative amounts of the mentioned overheads for two conventional BiCMOS circuits. The first is a two-input NAND gate. The second is a ring-oscillator, consisting of eleven cascaded NAND gates. The last NAND gate in the ring-oscillator is augmented by the circuit shown in Figure 5.5.

It can be seen that the change in the output voltage levels in the normal mode is quite insignificant. Since the test circuit tries to drive the output to the opposite level, the output will deteriorate. The area increase for the simple NAND gate is not in an acceptable range, but it should be noted that for a circuit which is more complex than a simple NAND gate, this overhead decreases significantly. Table 5.3 shows that there is a very small penalty associated with the proposed technique, especially as the complexity of the CUT increases. However, the fault coverage obtained is quite significant.

Parameter		Change	
		NAND	Ring-oscillator
V_{OL}	Normal mode	+ 60 mV	+ 60 mV
	Test mode	+ 200 mV	+ 200 mV
V_{OH}	Normal mode	- 60 mV	- 60 mV
	Test mode	- 250 mV	- 250 mV
Delay		+ 2%	+ 0.2%
Device count (area)		+ 200%	+ 16%
Average power (@160 MHz)	Normal mode	+ 8%	+ 3%
	Test mode	+ 15%	+ 12%

Table 5.3: *Effects of the DFT technique for delay/stuck-open fault detection on the circuit parameters.*

5.6 Generation of Φ_1

Φ_1 and/or Φ_2 in the proposed checker can be created in synchronous circuits from the master clock with a simple circuit. For asynchronous circuits, on the other hand, it is generated using the handshaking signals, which control the maximum limit for acceptable delay as well. Figure 5.10 (a) provides the controlling signals based on two-phase protocol explained in Chapter 2, and demonstrated in Figure 2.7. Figure 5.10 (b), on the other hand, replaces Φ_1 and Φ_2 for four-phase protocols. Ack_{c_i} in Figure 5.10 (c) is the acknowledge signal in stage i of the micropipeline, whereas $Req_{c_{i-1}}$ is the the request signal in stage $(i-1)$.

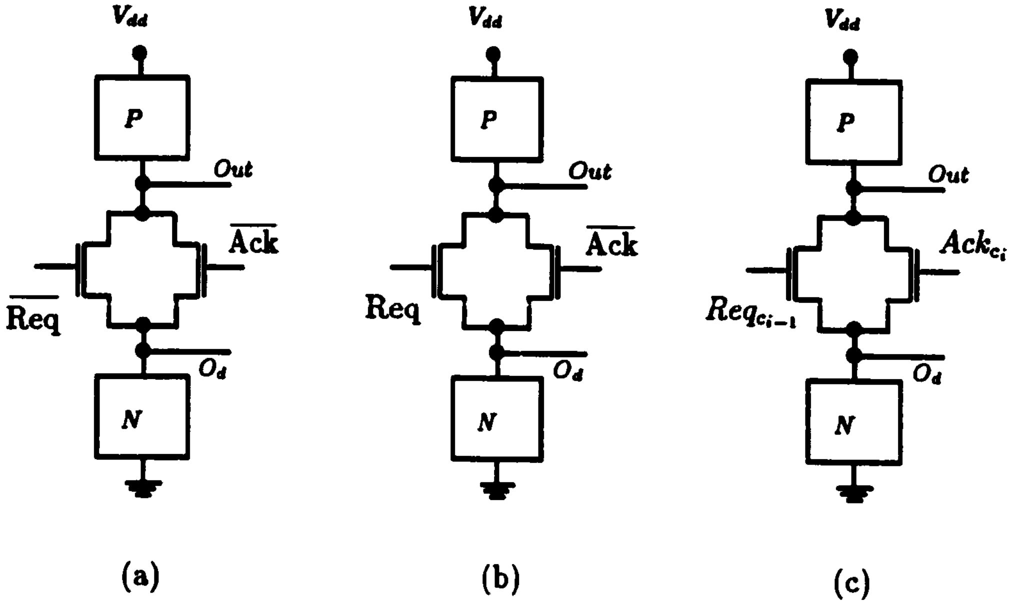


Figure 5.10: *Availability of controlling signals in asynchronous circuits: (a) in two-phase protocol, (b) in four-phase protocols, (c) in micropipelines.*

5.7 Faults on Φ_1 and Φ_2

Either Φ_1 or Φ_2 can be faulty. The following faulty behaviors have been studied.

1. If both Φ_1 and Φ_2 are stuck-at zero, transistors N_2, N_3, and N_4 in Figure 5.5 are *off*. The p-part is separated from the n-part even though the circuit under test is fault-free. *Out* and O_d in Figure 5.5 will have no effect on each other. Therefore, *Out* will charge to V_{dd} and O_d will discharge to GND at some point of time. Since their logic values are different, the remaining parts of the test circuit would detect the fault.

2. The fault Φ_2 stuck-at zero is detectable when we turn off Φ_1. The same situation as described by item #1 would happen.

3. If Φ_2 is stuck-at one and Φ_1 is stuck-at zero, N_3 and N_4 are *on* while N_2 is *off*. When the p-part is active, there is a short path between V_{DD} and the ground. This causes an increase in the static current which is provided by the power supply. Similar situation happens when the n-part is active. This fault, therefore, is detectable through I_{DDQ} fault testing discussed in Chapter 4.

4. If Φ_2 is stuck-at one, but Φ_1 is fault-free, the fault is detectable as previous item describes when we set Φ_1 to zero.

5. If both Φ_1 and Φ_2 are stuck-at one, the fault is detected as an I_{DDQ} fault.

6. Transistor N_2 is always *on* if Φ_1 is stuck-at one. This fault is not detectable.

The time accuracy of the sensing circuit and its sensitivity to possible hazards in the transitions of Φ_1 and Φ_2 have been studied by means of SPICE simulations. There are a number of scenarios as described next.

- If the hazards (static or dynamic) happen after the circuit under test is stabilized, the test circuit works as intended and the faults in the circuit under test would be detected.

- If the hazards happen before the circuit under test is stabilized, changes at the circuit output would be detected by the test circuit and the presence of fault(s) would be known.

Similar arguments are valid if Φ_1 and/or Φ_2 have delay faults. Hazards in the circuit outputs would also be detected in a similar fashion. (If hazards on S occurs before Φ_1 and Φ_2 change states, the circuit is considered fault-free. Such hazards would be harmless to the functionality of the circuit under test. If hazards on S occurs after Φ_1 and Φ_2 change states, the hazards would be detected by the test circuit. Note that we consider one fault at a time in this study.)

5.8 Medium Size Logic Circuits

Published papers on DFT techniques for testing delay faults and stuck-open faults include description of functional behavior of the test circuit. In this Chapter we have described our test circuit in detail and we have made comparisons based on the functionality of our test circuit with those available techniques. No fault simulation comparison is made since there are no benchmark circuits for the purpose of delay fault and stuck-open fault testing. Neither are there transistor sizes available in the literature to make a comparison based on experimental results possible. This is mainly because reporting fault coverage requires manually injecting the fault and simulating the response of the faulty circuit using SPICE.

For reasons mentioned in Section 4.4, we have selected an 8-bit parallel binary adder [116], 8-bit multiplier [116], and the ALU 74181. Simulations steps are the same as those explained in Section 4.4. The differences are outlined as follows.

- The final stage of the adder and multiplier circuits generate the final carry (C_9). The longest path delay is from the primary inputs to C_9. This stage has been augmented using our test circuit for delay fault testing.

- The critical path in a decoder is relatively small compared to other samples we considered. We have considered the Carry Generate [116] (CG) circuit for an 8-bit Carry-Look Ahead adder [116]. The circuit function is described using the following equations.

  $$C_{i+1} = G_i + C_iP_i, \text{ where } P_i = A_i \oplus B_i \text{ and } G_i = A_iB_i$$

  Only one test circuit is used at the output of CG circuit.

- The test circuit is added to only one output of the ALU 74181.

- Transistors N_3 and N_4 are removed from the test circuit during delay fault testing, and only Φ_1 is applied.
- The maximum delay of the critical path has been considered to estimate transitions on Φ_1 signal. We assume that transition on Φ_1 arrives *as soon as* the fault-free outputs are stabilized in the worst case scenario.
- Only T_o is observed.

Table 5.4 provides simulation results.

Circuit structure	8-bit Adder	8-bit Multiplier	ALU 74181	8-bit CG
Detectable delay faults (%)	100	100	100	100

Table 5.4: *Fault detection comparison for four MSI structures.*

The reasons for this high coverage are explained in *observations* one and two.

Observation 1: The minimum delay caused by a delay fault (a transistor stuck-open or a stuck-at 0/1 which causes a transistor to be off during the simulation) is greater than the time response of the fault-free test circuit.

Observation 2: The maximum delay caused by a delay fault is as much as 20% of the time response of the fault-free circuit.

Observation 3: Hazards before and after transitions on Φ_1 do not invalidate the test results. If the hazard occurs before a transition on Φ_1 arrives, the circuit is considered fault-free. However, if the hazard happens after the specified delay for the fault-free circuit, it would cause the test circuit to respond, and a fault is detected.

Observation 4: Although delay faults are less pronounced in 8-bit Carry Generate circuit that those of decoder, the fault coverage is still 100%.

The area overhead for delay fault testing is small if a single output is considered. There are ten transistors in the test circuit. This area overhead is equal to the area of 2.5 NAND gate. Since one current sensor circuit and one NMOS transistor (to reset the sensor) are used for multiple output testing, the area overhead would be roughly $\{1.25 * (\text{number of outputs}) + 1.25\}$ NAND gates. This figure does not take into account the circuit which generates Φ_1. In practice, multiple output circuits interface with next-stage logic circuit through state-holding elements. Therefore, only one circuit is necessary to generate Φ_1. If each output under test is handshaking with next-stage logic separately, there would be separate *ack* and *req* signals. Different Φ_1s are, therefore, generated using these controlling signals as described in Section 5.6.

The delay faults considered in our simulation include stuck-at faults which permanently turn off (or on) a transistor, and transistor stuck-open faults. Further simulation should be carried out for parasitic capacitors and the variations on transistor parameters which would change the time response of the circuit beyond acceptable range.

We have carried out simulations for stuck-open faults as well. Φ_2 $(= \bar{\Phi}_1)$ is generated using a CMOS inverter. Table 5.5 shows the fault coverage for each design.

Circuit structure	8-bit Adder	8-bit Multiplier	ALU 74181	decoder
Detectable delay faults (%)	100	100	100	100

Table 5.5: *Fault detection comparison for four MSI structures.*

We have observed that many stuck-open faults manifest as delay faults. The area overhead for stuck-open fault testing is approximately three NAND gates for single output design.

Delay fault test circuit shares many transistors with stuck-open fault test circuit. If one test circuit is used to perform testing of both fault types, the area overhead is $\{1.75 * (\text{number of outputs under test}) + 1.25\}$ NAND gates. Φ_2 must be generated from Φ_1 through an odd number of CMOS inverters to allow delay fault testing performed before stuck-open fault testing begins.

5.9 Conclusions

In this chapter, we have proposed a DFT technique for detecting faults in BiCMOS, BiNMOS, and CMOS circuits (combinational and sequential).

The CUT will be tested for faults concurrently with the normal operation. This is due to the fact that the error signal is created if any of the observed outputs makes a transition after Φ_1 becomes low and/or Φ_2 becomes high[1]. It is not important whether the transition is from low to high, or vice versa. The checker does not need to know the fault-free output voltage level for detecting the fault. Therefore, *special test-pattern generation is not required for this method.* The important consequence of these points is that the proposed technique can be used *concurrently* with the normal operation of the circuit to detect faults as they occur. This eliminates the difficult task of generating robust test patterns for fault detection.

In order to increase the range of the detectable faults, Φ_1 should change to *ZERO* as soon as possible (after the maximum acceptable delay), and should remain at this level for as long as possible.

There is an area penalty due to the added devices and due to the circuitry needed for generation of the main sampling clock Φ_1.

[1] The transition on either Φ_1 or Φ_2 happens after the maximum specified delay is elapsed.

Simulations results of different realizations of NAND gates show the output voltage levels of the circuit remain unaffected, but the power dissipation increases slightly. The increase in power dissipation or the area overhead is negligible if the test circuit is used for a complex circuit in normal-mode operation.

Chapter 6

Initialization

6.1 Introduction

Two fundamental processes must be followed for testing sequential circuits. The first one is that vectors must be applied in the proper sequence in order to detect specific faults. If a sequence of input vectors $X(0), X(1), \cdots, X(n)$ detects a particular fault, then it is a test sequence for that fault. This order in the sequence has to be preserved in actual testing. The second one is that such a test sequence is effective if it is applied after a specific state, called the *reference state.*

The process of test vector generation for sequential circuits involves fault excitation and fault propagation using forward and backward processings. Fault propagation is to determine a state into which the circuit must be driven so that additional vectors can be applied to make the fault observable at a primary output.

There are two different techniques to find test vectors depending on how the reference state is considered. The first method is as follows. At the beginning stage of test generation, the ATPG needs to know the power-up state of the CUT so that it can search for a sequence of vectors. When the power is applied, however, flip-flops, counters, shift registers, and other memory elements assume unpredictable states, and they must be set to known states before testing can begin. We can either initialize memory elements using an external point, see Figure 6.1 (a), or a power-up reset may be added to provide internal initialization, see Figure 6.1 (b). Relying on the power-up state for setting up its initial state requires power to be cycled before testing each fault.

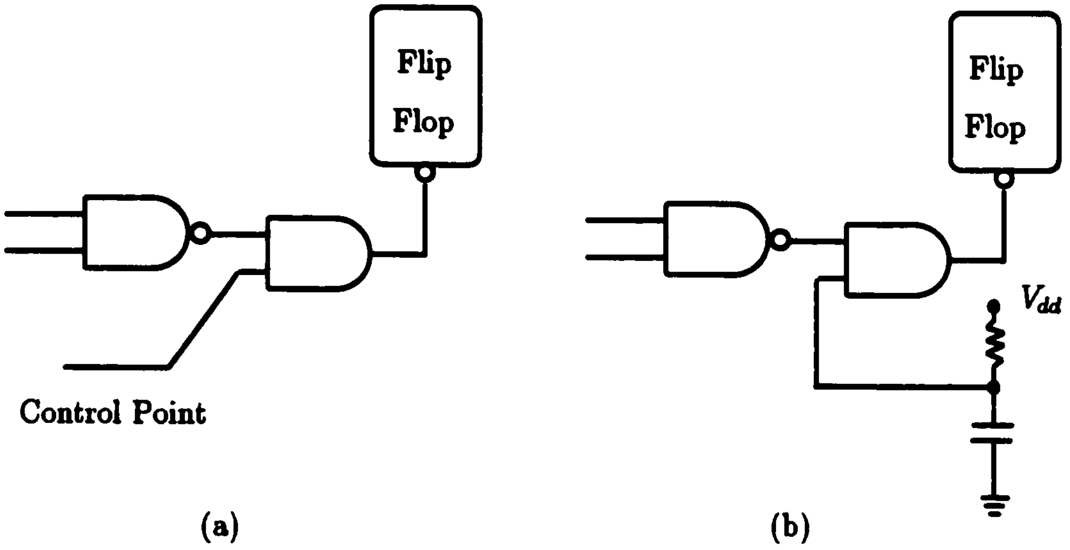


Figure 6.1: *Typical reset techniques: (a) Externally, (b) Internally.*

In order to ensure that the machine starts at some proper state in synchronous circuits, most designers incorporate a reset signal for all memory elements. This general conception, however, does not necessarily guarantee that the state is known. Figure 6.2 shows a sequential circuit with one memory element. We assume the combinational circuit is simplified for a specific set of input logic values such that Q' is connected to the input of the memory component which sets the output of the Flip-Flop to one. It is clear from this circuit that as soon as the *reset* phase is terminated, Q' becomes one, and forces Q to change from '0' to '1'. Therefore, it is not sufficient to provide memory elements with a reset signal, and assume that the state value is known.

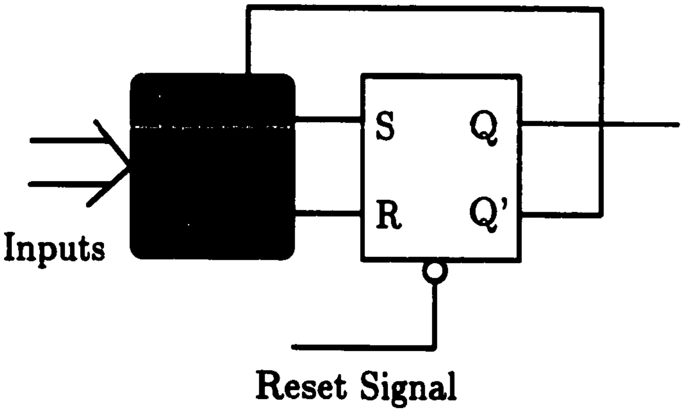


Figure 6.2: *An example where* reset *signal may fail.*

The second method to produce the reference sate is applicable if the information about the power-up state is not available. The ATPG has to search for a sequence of vectors (called the *synchronizing sequence*) in order to apply to the sequential machine, and force it, irrespective of its starting state, to a known state. The starting state, in this method, acts as a reference state from which all test vectors for fault propagation are applied. Figure 6.3 shows a typical test vector set for a sequential circuit that contains many test sequences, each of which used for a specific fault(s) so that a complete testing of the circuit is carried out. It is, therefore, of great importance to make the synchronizing sequence as small as possible to achieve a shorter test sequence.

Testing of sequential circuits is usually divided into two distinct phases. In the first phase we apply an initialization sequence such that the fault-free and faulty-circuits are brought to known states. If the faulty circuit fails to be initialized then

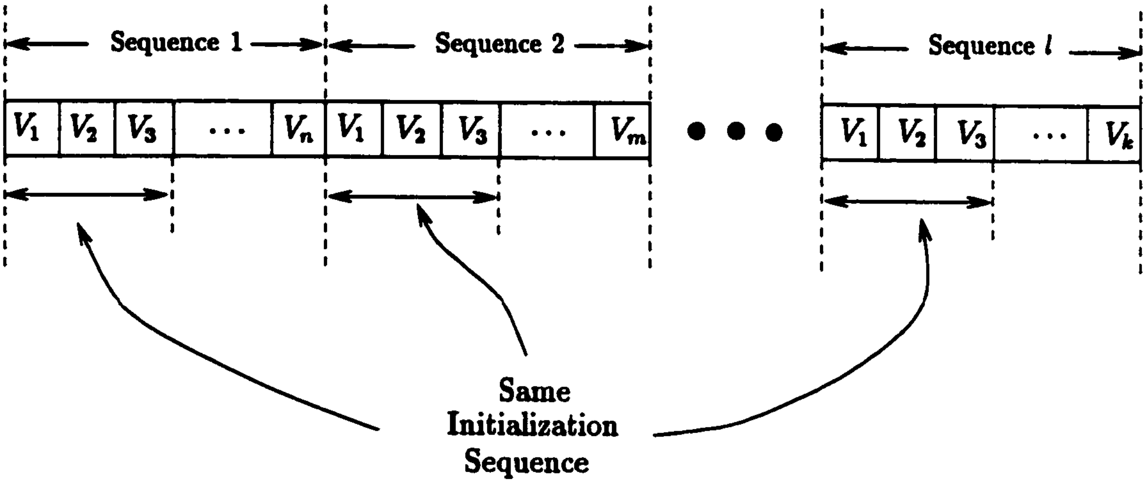


Figure 6.3: *A typical test vector set.*

a fault is already detected. The output of the circuit is not important and can be ignored till the initialization phase is completed. In the second phase we apply a test sequence to both faulty and fault-free circuits and observe the primary outputs. If the responses are different then the presence of a fault in the circuit is detected. The second phase of testing is based on the assumption that the first phase for initializing the circuit is successfully completed.

Synchronous initialization techniques are not generally extended to asynchronous sequential circuits. This is because of the absence of a global clock which would cause races in the circuit when the initialization phase is terminated.

A sequence of test vectors is needed to drive the sequential circuit from any unknown states into a state required for propagating the fault effect to a primary output of the circuit. Depending on the circuit structure, such a sequence of vectors may exist for a circuit. If so, we call the circuit *initializable*. On the other hand, some circuit structures preclude the existence of such a sequence; we then call such circuits *non-initializable*.

Serious testing problems can occur for circuits that are not initialized from any arbitrary state into a known state. The most effective ATPG tool fails to generate tests for a non-initializable circuit. It is, therefore, important to produce initializable circuits.

Re-designing a logic circuit is proposed in [36], also discussed in Section 3.3.3, for a limited class of asynchronous circuits which have "don't cares" in their flow tables. The method suffers from several shortcomings. First, re-designing by reassigning "don't cares" is not a reliable solution especially when there are no "don't cares" to reassign. Secondly, the proposed re-designing technique may sacrifice minimal area or hazard- and race-free implementations. Finally, the enumeration technique is very time consuming and costly.

This Chapter provides a design technique which initializes a fault-free asynchronous sequential circuit to a known state regardless of whether memory elements exist or not, see Figures 2.4 and 2.5. The technique does not require an optimal circuit to be re-designed.

6.2 Background

A typical classical asynchronous sequential circuit is implemented in Figure 6.4 as a large gate circuit with feedback. Designing classical asynchronous circuits [116] uses the *finite state machine* (FSM) model. The general format used to describe a sequential function, which relates output sequences to input sequences, in classical asynchronous circuits is called a flow table [79, 180], which consists of a two-dimensional array. The columns correspond to input states, the rows to internal states, and the entries are ordered pairs representing the next internal state and

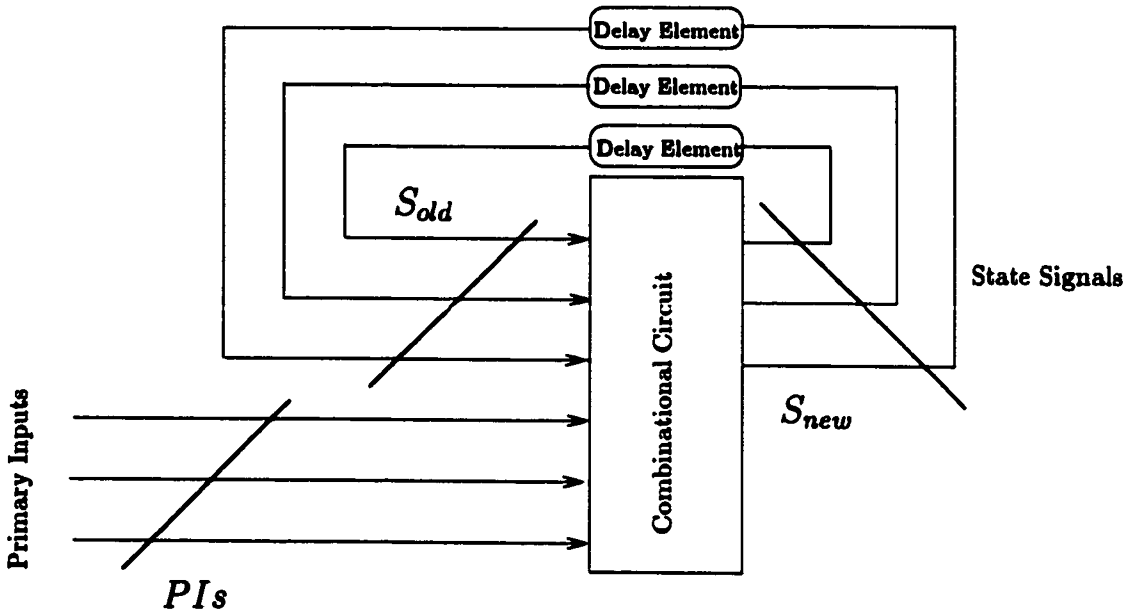
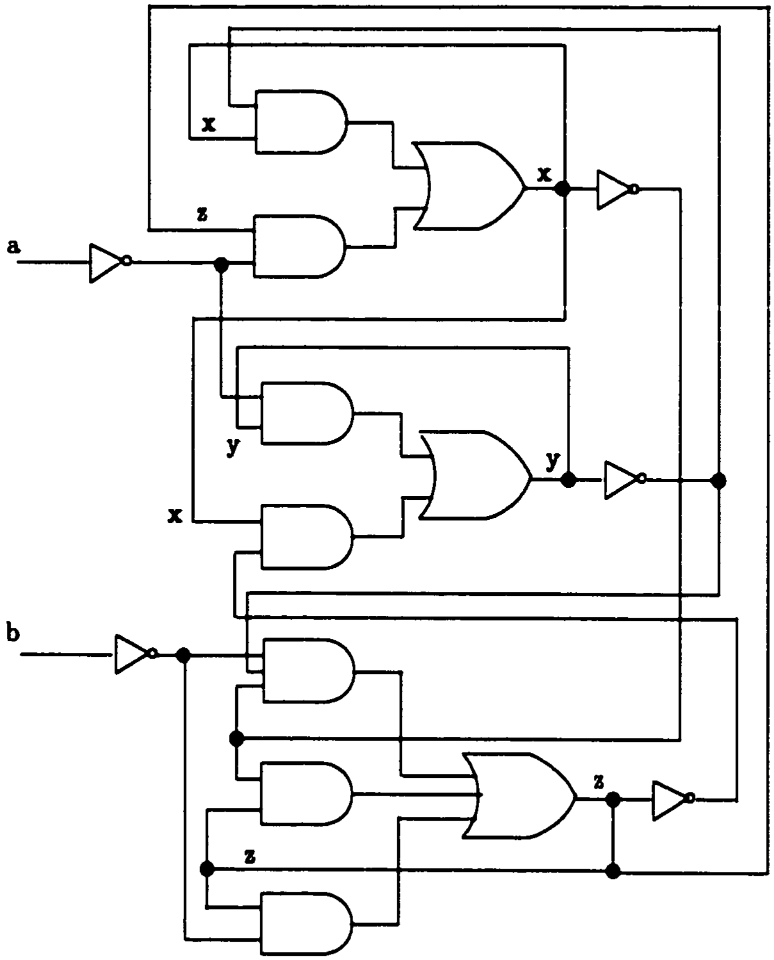


Figure 6.4: *Typical classical asynchronous sequential circuit structure.*

the current output. A state is stable if the next internal state is the same as the current internal state.

Figure 6.5 shows a non-initializable logic circuit presented in [36]. This circuit is designed using the minimum number of logic gates. It generates multiple outputs based on the given Boolean functions. As the flow table shows, the circuit has more than one stable state for a selected logical values of the primary inputs. Therefore, it is impossible to initialize this circuit using the primary inputs only. As explained in [36], this circuit is re-designed by re-assigning the "don't cares" using an enumeration technique.

The following solution seems tempting. Adding an extra input which provides an *initialization* signal, one can force all gates to produce an output '0'. The problem with this proposed technique is that when the initialization signal is dis-activated, the state variables tend to respond to the present logical values of the



$$x = \bar{a}z + x\bar{y}$$

$$y = x\bar{z} + y\bar{a}$$

$$z = \bar{b}\bar{x}\bar{y} + z\bar{x} + z\bar{b}$$

	ab			
xyz	00	01	10	11
000	001	000	001	000
001	101	101	001	001
010	010	010	000	000
011	111	111	001	001
100	110	110	110	110
101	101	100	101	100
110	010	010	010	010
111	111	110	001	000

Figure 6.5: *Original logic circuit along with its flow table and its Boolean functions.*

primary inputs. This may cause the circuit to go to an unknown state which is not desirable.

6.3 The Proposed Initialization Technique

Consider a general sequential circuit as shown in Figure 6.4. Our proposed method uses the following two facts.

- Combinational circuits do not need any initialization sequence since any node of the circuit can be initialized to a known logical value by a single input vector. Therefore, we physically transform a sequential circuit to a combinational circuit and initialize it by applying a single input vector instead of applying an initialization sequence.
- A stable state in a sequential circuit does not change if the primary inputs and the state logical values are unchanged.

Our proposed technique, therefore, requires the following.

- At least one stable state;
- The nodes or the physical places where these state variables reside;
- The corresponding logic values of primary inputs.

The availability of these items is realistic. There is no need to analyze the logic circuit in the time domain since the stable states are known in the design phase.

We first set the primary input lines to the selected logic values. We then cut feedback lines, and supply each with its corresponding stable logic value using additional logic gates. Such hardware should set the state variable node to the desired stable value in initialization mode. However, the additional hardware should not interfere with the normal operation when the circuit is not under initialization mode. Figure 6.6 shows all the possibilities.

- An AND (NAND) gate is suitable for generating 0 (1) if one of its inputs is set to 0 by initialization signal.
- An OR (NOR) gate can be used to generate 1 (0) on the feedback line if one of its inputs is set to 1 for the duration of the initialization process

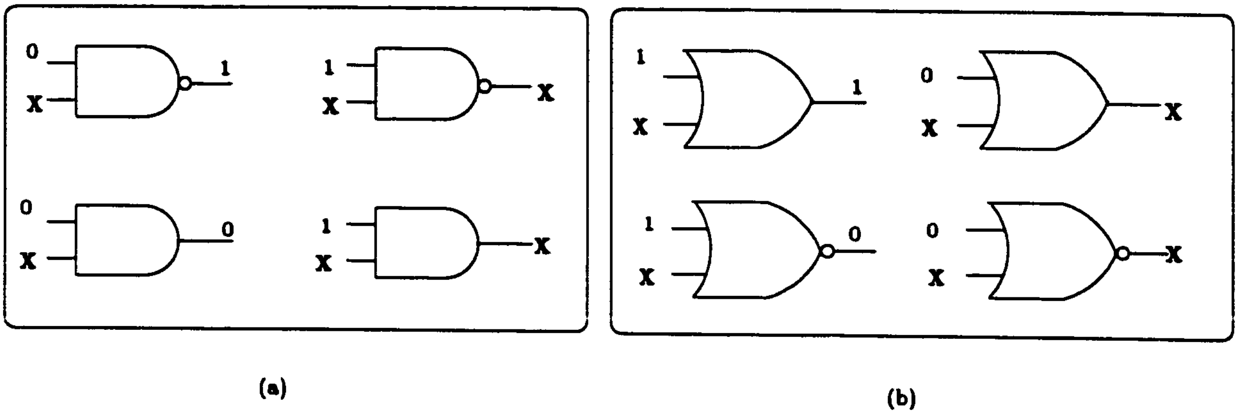


Figure 6.6: *Initializing a node to one or zero using a) AND/NAND, or b) OR/NOR.*

Suppose that the asynchronous circuit of Figure 6.4 has a stable state, "010", if its primary inputs are set to "110". Figure 6.7 shows our initialization technique where we have included AND/OR gates to generate "010" on the feedback lines. The circuit operates in two phases: Initialization mode and normal mode. Originally, the initialization signal is logic ONE.

The initialization phase begins when a falling transition on initialization line appears. In this mode, the circuit behaves like a combinational circuit. Since state "010" is stable at the presence of "110" on primary inputs, the next state variables would be "010", which are the same as the present state values after a minimum unit of time is elapsed. The initialization phase is ended by a rising transition on initialization line. The normal mode operation begins.

In normal mode, the inserted gates would act like a buffer. This allows the next state signals to be fed into the combinational part as the present state signals. The circuit is then sequential.

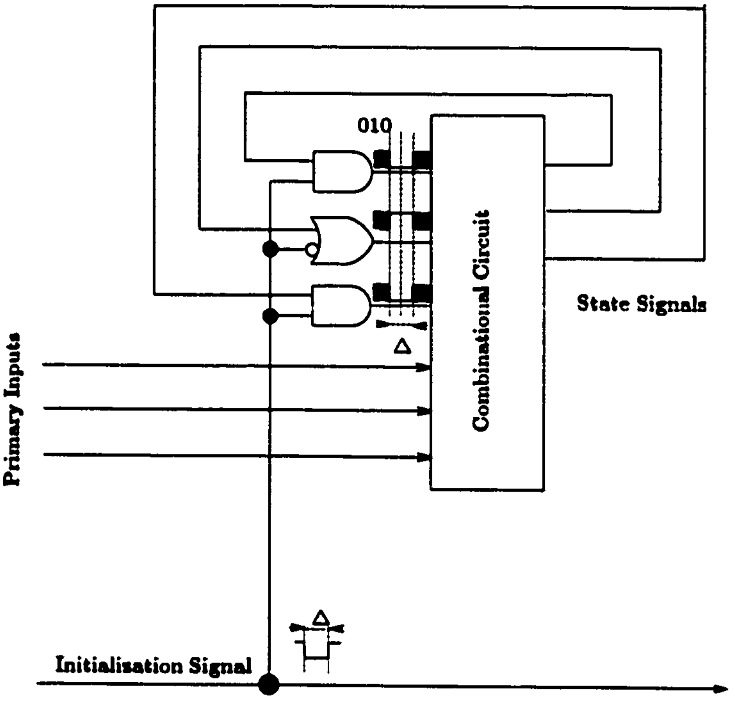


Figure 6.7: *DFT: The proposed initialization technique.*

6.4 An Example

In contrast to the technique presented in [36], we do not need to search the entire solution space in our technique. We need to know, however, the logical values of one stable state, its corresponding nodes , and the corresponding logical values for primary inputs. We have selected state "001" for $ab = 10$, see Figure 6.8.

The hardware overhead is roughly 20% based on the gate count. Such an overhead can be avoided if the circuit gates are used to force logical values on feedback lines. Figure 6.9 shows an initializable circuit with 7% device count overhead.

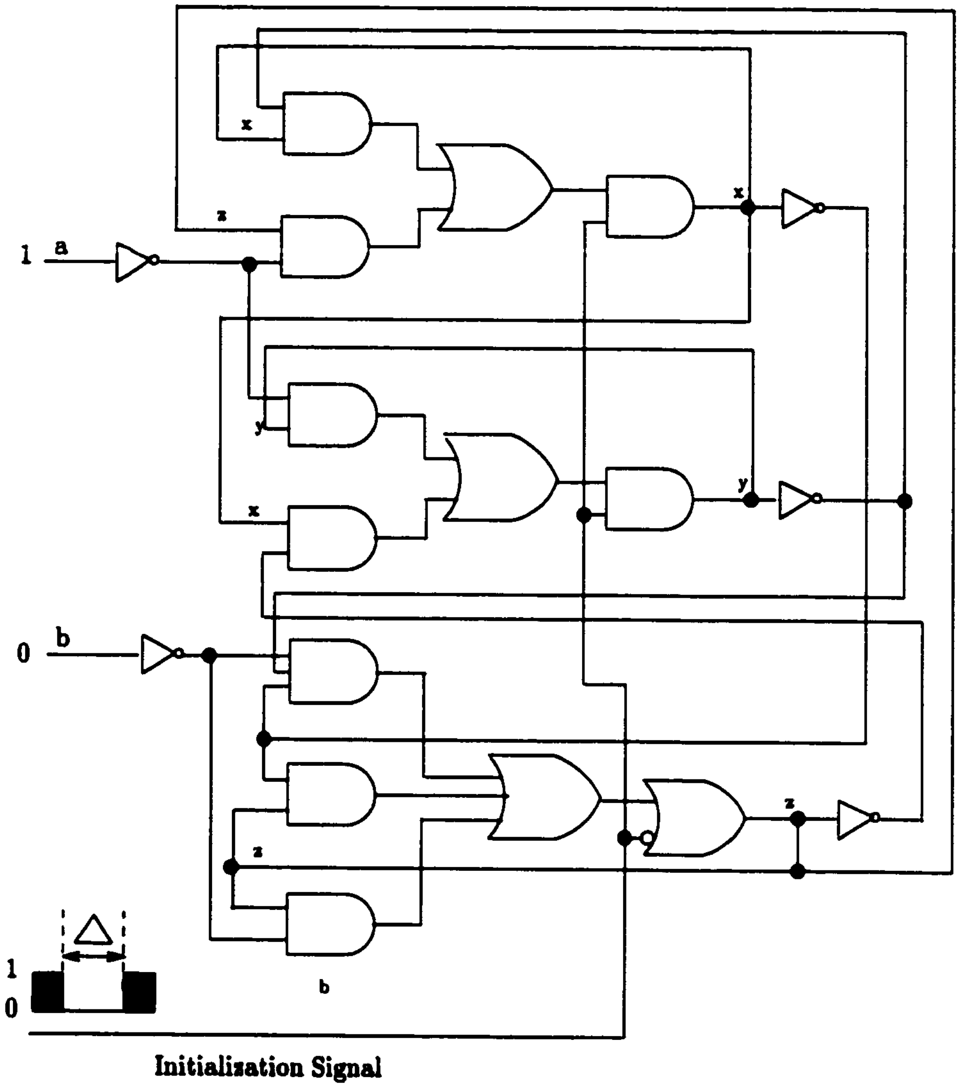


Figure 6.8: *The initializable logic circuit.*

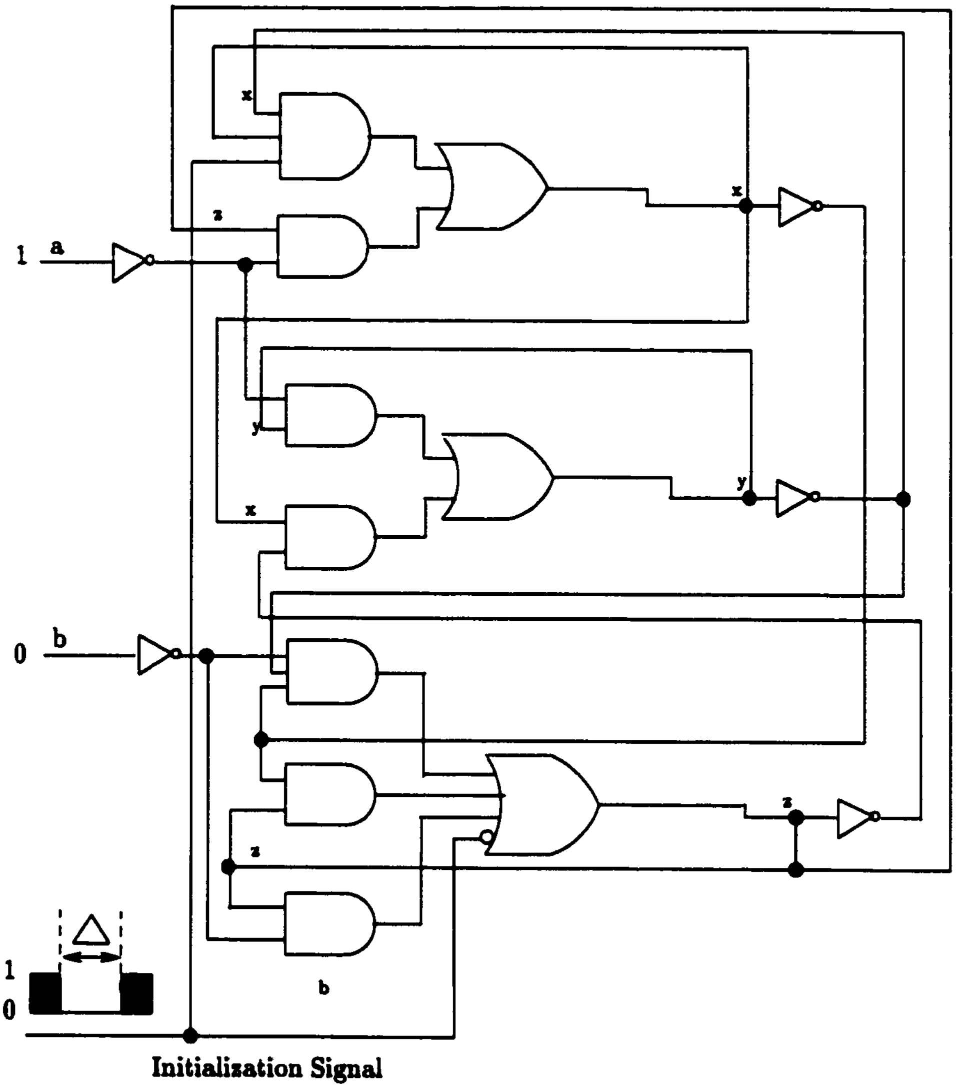


Figure 6.9: *The initializable logic circuit with reduced area overhead.*

6.5 Advantages

The method discussed in this Section has several advantages.

- The method proposed does not re-design a given gate circuit which is normally optimized in terms of area and speed and is usually race- and hazard-free.
- Although it is not important at the initialization time, the method itself does not introduce hazards or races.
- The method introduces little delay on the paths from primary inputs to primary outputs.
- There is a trade-off between applying extra gates on the feedback lines or applying the initialization signal to the inputs of the pre-stage gates. We believe that the outcome of final decision can introduce very little hardware overhead depending on the application.
- There is no execution time for applying this initialization technique to the given circuit if at least one stable state is known at the presence of one set of primary inputs.
- Any stable state can be selected in case there are multiple stable states for a given set of primary inputs. This can further reduce the hardware overhead.

Adding extra input/output test points is sometimes suggested to ease the controllability and observability of variables inside asynchronous circuits. Adding test signals is also recommended to ease computation in test mode. That is why we speculate that many logic circuits tend to have *test signal* as an additional input signal. The initialization signal can be generated internally using the test signal.

We can generate the initialization signal using a product term of inputs if the stable state selected is unique. For example, if '010' were the only possible stable state for $ab = 10$, then we can generate the initialization signal using an OR gate which realizes $\bar{a} + b$.

6.6 Medium Size Circuits

Up to date, there are only a few non-initializable logic circuits provided with schematics or well-known pinouts. Table 6.1 shows a few non-initializable designs from ISCAS'89 [26] benchmark. Since we did not have software tools to verify as to whether they are initializable or not, we relied on a report published in [184]. There was no symbolic analysis tool to derive the logic function of the circuits before and after the initialization process was completed. Therefore, we applied our technique on the circuits and simulated them using available logic simulators. The results showed that the modified circuits are initializable.

Circuit Name	# of Inputs	# of FFs	Area Overhead (%)
s208.1	10	8	5.2
s420.1	18	16	4.2
s510.1	19	6	5.7

Table 6.1: *Initializability Results.*

Table 6.1 shows that our initialization technique is efficient, and introduces little area overhead. Further work should be carried out to determine minimum number of gates which, if modified, would initialize the circuit with minimum area overhead.

6.7 Conclusion

We have proposed a resetting technique to initialize any fault-free sequential circuit which may or may not have storage component such as flip-flops in its design. There is no need for "don't cares" in the design specification. Although we modify the circuit, we do not re-design an existing logic circuit which meets the area/speed/power specifications.

Our method involves physically transforming the sequential circuit to a combinational one by cutting the state-holding variables and inserting a minimum number of gates to bring the circuit to a known stable state. The following steps show how to initialize an existing non-initializable circuit.

1. Determine the state the device is to be initialized to.

2. Based on the initialized state, determine the corresponding logic value of each input pin.

3. Determine the physical places of state variables.

4. In case the stable state is unique for the set of input logic values, use the product terms of the inputs in order to produce the initialization signal.

This technique is always applicable, and has very little overhead. The additional hardware does not interfere with normal operation. The additional logic gates are tested along with the rest of the circuit. In initialization mode the circuit is combinational. Having known logic values of state variables in the fault-free circuit, the output(s) should depend solely on the logic values of the primary inputs.

Chapter 7

Asynchronous Sequential Circuit Testing

7.1 Introduction

Finding test vectors for a logic circuit is a difficult problem. A great amount of research has been done to reduce the execution time of testing combinational circuits [2,101,102]. Testing sequential circuits has received less attention and is much more difficult because long test sequences may be needed to excite a given fault and to make it observable. This problem is usually solved by using scan design methods [95,97,158,183,186] which reduce any sequential circuit to a collection of combinational circuits and latches, thus simplifying testing. Self-timed asynchronous circuits are easier to test [17,42,87,182] if the faults are considered only at the output nodes of gates since they would halt at the presence of stuck-at faults. This, however,may not be true if the test analysis is performed at the transistor-level. So far, general testing methodology for asynchronous sequential circuits is not fully developed. The reason may be their reduced controllability due to the absence of a global clock.

One available technique to find the complete test sequence is to generate faulty machines due to all single and multiple faults and use **observer**[1]. This technique is not cost efficient since generating faulty machines is tedious even for simple circuits and **observer** uses branch-and-bound techniques and may take a long time before it responds. A counter example is provided in this Chapter for which **observer** fails to produce a minimum length test sequence.

Using Martin's design methodology [118], Hazewindus [82] has described a testing methodology which works as follows. First, a fault is considered and the necessary conditions to detect the fault are derived. Then, the D-algorithm [161] is used to find the logical values of inputs or a sequence of inputs which put the circuit in the condition derived and propagate the effects of the fault to the primary output. This technique has some drawbacks. It fails to produce the test sequence for some detectable faults. The computation time to find the test sequence for circuits which have a large number of state variables is long. Its usefulness is only limited to a class of asynchronous circuits. Adding extra test points is suggested to ease the observability and testability of such circuits.

The mainstream test analysis of sequential circuits is explained as follows. Consider two sequential circuits which have identical topologies, but one of the circuits is fault-free and the other circuit has a single or multiple fault, see Figure 7.1. Let us assume that the initial state of both circuits are the same. The fault is detectable if there exists a test sequence which, if it is applied to both circuits, would produce

[1] **observer** is a program for diagnosing and testing sequential machines. It is based on the theory developed in [31] and was written at the University of Waterloo by C.-J. Richard Shi; some additional features were later added by Paul Kwiatkowski.

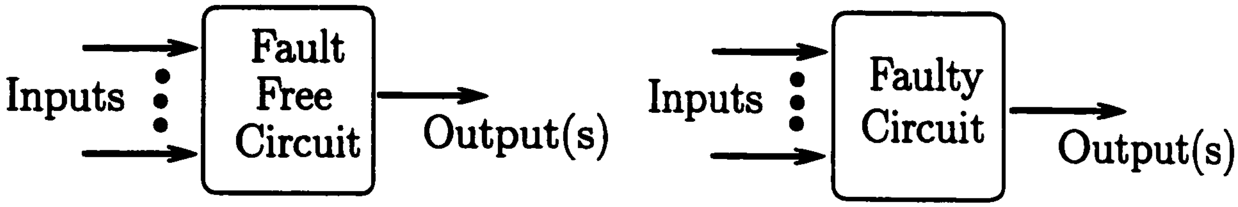


Figure 7.1: *Two identical circuits, (a) The fault-free circuit, (b) The faulty circuit.*

different outputs at the end of testing phase. Each test sequence is obtained by studying the faulty circuit. These test sequences are linked together to form the complete test set.

The number of faulty machines depends on the number of wires and gates of the circuit. If there are 10,000 non-equal single stuck-at faults, 10,000 copies of the good machine containing one, and only one, of the single stuck-at faults should be simulated. Thus, fault simulation, with respect to run time, is similar to performing 10,001 good machine simulations. Therefore, generating faulty machines is tedious if single faults are considered and it is impractical if multiple faults are taken into account.

Our technique in testing sequential circuits is different in essence from the mainstream test analysis in the following sense. Unlike well-known test generation algorithms, we study the correct circuit behavior instead of the faulty one. If the fault-free circuit is precisely defined, any behavior shown by the circuit which is out of its specification would confirm the existence of a fault. This simplifies test analysis as there exists only *one* fault-free circuit. Like the **observer**, we produce test sequences, and link them together to generate the complete test set. In this process we eliminate redundant sequences.

This Chapter is concerned with testing of asynchronous sequential circuits and it is organized as follows. The terminologies used and the assumptions made in this Chapter are explained in Section 7.2 and Section 7.3. Our proposed algorithm is based on a number of steps. First, limited but unknown delay elements are assigned to wires and gates, and the network model is precisely defined using minimal feedback set in Section 7.5. Sequential circuits are transformed to combinational ones. Then, inputs which contribute to a transition in either primary output or internal states are considered. To study these transitions, the collection of states and transitions is shown by the Behavioral Model for Testability (BMT) graph in Section 7.6. Mapping available correct circuit behavior provided by Karnaugh map or state table on to the BMT graph, input sequences which correspond to correct circuit behavior are derived in Section 7.7. Redundant sequences are discussed in Section 7.8. Pieces of irredundant sequences are linked together in series to generate the complete test sequence in Section 7.9. No faulty machines are generated during these procedures. We show that the circuit is fault-free if it is initialized from any stable state to another using the valid input sequences devised in Section 7.7. A comparison of the computational costs is given in Section 7.10. An underlying reason for our algorithm is provided in Section 7.11. Important conclusions are explained in Section 7.12. Limitations we have faced to generate test sequences for relatively large circuits are discussed in Section 7.13. A summary is provided in Section 7.14.

We have used the majority C-element to illustrate details of our algorithm. However, the technique is applied on other asynchronous sequential circuits and the results show that the technique is cost effective.

7.2 Definitions

In this Section we define the terminologies we have used in our algorithm.

7.2.1 Components

Combinational components and state-holding elements are explained and their functions are defined in this section. Consider a component with n inputs, $a_1, a_2, \cdots, a_n$, and one output C. Let $A = \{X = (a_1, a_2, \cdots, a_n) | X \in \{0,1\}^n\}$. Let there be a relation between set A and set $\{0, 1\}$ such that *each* member of set A is associated with one or more members of the set $\{0,1\}$. This relation is called a *mapping* from set A to $\{0,1\}$, written $\mathcal{M}$: $A \rightarrow \{0,1\}$. A digital component is either a *combinational* or a *state-holding* component depending on how A is mapped onto $\{0,1\}$.

A component is combinational if for every $X \in A$, there exists a *unique* element $c \in \{0,1\}$. Such a mapping is called a *function*, written $f : A \rightarrow \{0,1\}$. AND, OR, NAND, NOR, and XOR are examples of combinational components.

A component is called a state-holding element if there exists at least one $X \in A$ which is mapped to both 0 and 1. The logical values of the outputs for state-holding elements depend on the history of the input signals. Examples of state-holding elements are the C-element, Flip-Flop, and Counter.

We study combinational circuits for two reasons. First, as we will show in Chapter 7, a sequential circuit is considered as a combinational circuit between each two states. Second, a combinational circuit operates asynchronously since its output is defined purely by the delays of wires and gates.

7.2.2 Red & Green Edges

Let $I = \{a_1, a_2, \cdots, a_m, \cdots, a_M\}$ show the set of inputs for a combinational circuit. The number of elements in a set, say I, is called its cardinality and shown as

Card $(I) = M$. A vector $v = (a_1, a_2, \cdots, a_m, \cdots, a_M)$ is obtained when each input is assigned a logic value. The output is a logic function which is given by $F = f(a_1, a_2, \cdots, a_m, \cdots, a_M) = F(v) \in \{0, 1\}$. Let us consider two input vectors and their corresponding outputs:

$$\begin{array}{lcl} v_i = (a_1, a_2, \cdots, a_m, \cdots, a_M) & \Rightarrow & F_i \in \{0,1\} \\ v_j = (a_1', a_2', \cdots, a_m', \cdots, a_M') & \Rightarrow & F_j \in \{0,1\} \end{array}$$

Vectors v_i and v_j are connected by a *red* edge (or *solid* line) if $F_i \neq F_j$. We mathematically show this relationship as $(v_i \Re v_j) = Red$. On the other hand, the two vectors are connected by a *green* edge (or *dashed* line) if $F_i = F_j$. We show this relationship as $(v_i \Re v_j) = Green$. Figure 7.2 shows a 3-input AND gate and some of its red and green edges.

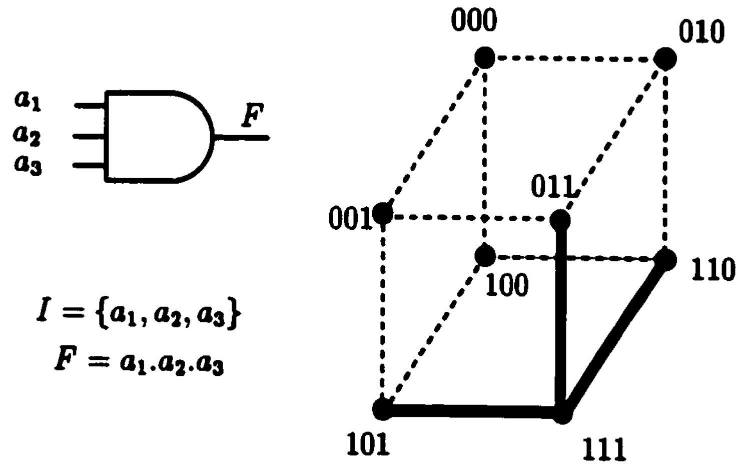


Figure 7.2: *Demonstration of Red (Solid) and Green (Dashed) edges.*

If the input vectors v_i and v_j differ in only one variable, i.e., $a_p = a_m'$ for $p \neq m$ and $a_p = \overline{a_m'}$ for $p = m$, then these two input vectors are called *first-degree* neighbors. We define the *transition set* as $\Upsilon_1(v_i, v_j) = \{a_m\}$, where a_m is the input variable for which the two input vectors differ. We also show this relationship using $v_j = N_{v_i}(\Upsilon_1)$.

The K-th degree neighbor of v_i, denoted by $v_j = N_{v_i}(\Upsilon_k)$, is obtained by complementing K input variables from the transition set $\Upsilon_k(v_i, v_j)$.

Let us group a number of input vectors in a set denoted by $V = \{v_1, v_2, \cdots, v_L\}$. Such a set is considered to be a *Complete Test Set* if it has two properties:

1. the set detects any single & multiple fault: i.e., Fault Coverage(V)=%100,
2. and the fault coverage of the reduced set $(V - \{v_i\})$ is less than % 100, for any i between 1 and L.

An input vector v_i is *redundant* if the fault Coverage(V) = Fault Coverage($V - \{v_i\}$).

7.3 Assumptions

In this Section we provide the assumptions under which our algorithm works. The assumptions are outlined as follows.

1. The circuit has only one primary output.
2. The outputs of any two gates or components should not be connected to each other: i.e., no wired output.
3. Only digital elements are allowed. For example, no transistor[2], capacitor, resistor, inductor, or Op-Amp is allowed. Permissible components are AND,

[2] A transistor can be used as a logic element. Its input changes from GND to V_{DD} or vice versa. However, a transistor behaves differently than a logic gate. A logic gate has one output and usually more than one input. A transistor has only one input and two outputs. As a matter of fact, the behaviors of the terminals of a transistor are highly correlated. For example, if the voltage value of an emitter increases toward V_{DD}, the transistor may turn off. Therefore, the relationships among the terminals of a transistor are not described by Boolean function: They are

NAND, OR, NOR, NOT, buffer, wire, fork, and any component-level combinational circuit. A fork is a fanout point which has one input, called *stem*, and two or more outgoing lines. It is considered a component in asynchronous circuits since it has characteristics of any logic component. Any component-level combinational circuit is also included in our algorithm. Such components have a number of inputs and one output, and its functionality is defined as described in the previous Section. For example, an XOR gate is a component-level circuit if we disregard any knowledge of how this gate is realized.

4. Only deterministic components are considered in this dissertation. Arbiters[3], for example, are not considered.

5. The correct circuit behavior is available in either fundamental [125] or input/output mode of operation as the Karnaugh map or state table of the circuit.

6. The delay element associated with each state variable, which are defined in Section 7.5.1, obeys the Up-bounded Inertial Delay (UIN) model's rules. It is inertial since short input pulses may not propagate to the output of the delay element and are ignored.

Let D be the maximum time constant for a delay element to respond to its

described by curves or analog functions. Due to these relationship, faulty transistors behave much differently than faulty logic gates. For this reason, we exclude transistors from our discussions. Similar arguments are valid for capacitors or inductors.

[3] When two or more processors attempt to simultaneously use a functional unit (memory, multiplier, etc.) an arbiter module must be employed to insure that processor requests are honored in sequence. During the time that one request is being handled, more requests may appear. The design of asynchronous arbiters is complicated because multiple input changes are allowed, and because inputs may change even if the circuit is not in a stable state.

Let D be the maximum time constant for a delay element to respond to its input pulses, $x(t)$ be the output signal, and $X(t)$ be the input signal. The following two rules define the behavior of this element:

- If $x(t)$ changes from α to $\bar{\alpha}$ at time τ, then there must exist $\sigma > 0$ such that $X(t) = \bar{\alpha}$ for $\tau - \sigma \leq t < \tau$.
- If $X(t) = \alpha$ for $\tau \leq t < \tau + D$, then there must exist a time $\tilde{\tau}$, $\tau \leq \tilde{\tau} < \tau + D$ such that $x(t) = \alpha$ for $\tilde{\tau} \leq t < \tau + D$.

The exact size of the delay, D, is not necessarily known.

7. No redundancy is allowed in the circuit. A wire is said to be *redundant* if it can be permanently connected to logic '1' or '0'. A component is *redundant* if all of its input wires or its output wire is redundant.

8. The correct circuit behavior is (static & dynamic) hazard-free. This assumption is necessary for sequential circuits only. Whether there is static or dynamic hazard in combinational circuits is not important since the output is analyzed when the circuit is stabilized, and the there is only one stable output per input vector.

9. The logic values of inputs and outputs belong to $\{0, 1\}$, i.e., binary inputs/output.

10. The system is assumed to be *Causal.* If the output is changed in a causal system, one of its inputs must have changed first.

11. We consider the stuck-at 0/1 fault model.

12. Single or multiple faults are considered at the input or output nodes of components.

13. Faults are assumed to be detectable. This means that there is a series of input signals which produce different outputs than those of the fault-free circuit.

14. Only logically testable faults are considered in this Chapter. A fault is logically testable if it is detectable using the history of zeros and ones at the output nodes of the circuit. Each input vector is applied for δ units of time and the circuit stabilizes. Then the next input is applied.

It is assumed in this Chapter that the circuit under test remains combinational in the presence of the fault(s). This implies that there are no feedback paths created by the fault leading to the existence of spurious stored-state devices. It is further assumed that the circuit under test is constructed from unidirectional gates. In a unidirectional gate the output(s) of the gate depend only on the input(s) and there is no dependency of input values on the outputs.

7.4 Redundancy

Figure 7.3 shows a redundant circuit. The output of this circuit is zero in its steady state when delays of the wires and the gate are exhausted. The circuit can be removed and the output node is connected to '0' permanently. As we will show, the redundancy will not help to reduce the size of the complete test set. There are

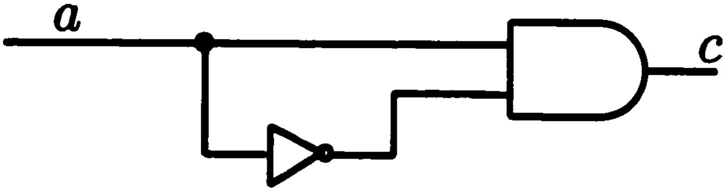


Figure 7.3: *Redundant circuit.*

a few Boolean rules which are useful in eliminating the redundancy.

$$\begin{cases} x+x=x \\ x.x=x \end{cases}, \quad \begin{cases} x+\overline{x}=1 \\ x.\overline{x}=0 \end{cases}, \quad \begin{cases} x+\overline{x}.y=x+y \\ \overline{x}+x.y=\overline{x}+y \end{cases}, \quad \begin{cases} x.(\overline{x}+y)=x.y \\ \overline{x}.(x+y)=\overline{x}.y \end{cases}$$

$$\text{where} \qquad \begin{cases} x=g(a_1,a_2,\cdots,a_M) \\ y=h(a_1,a_2,\cdots,a_M) \end{cases}$$

This does not mean the circuit should be minimum in terms of wires or gates if the circuit is to be irredundant, see Figure 7.4.

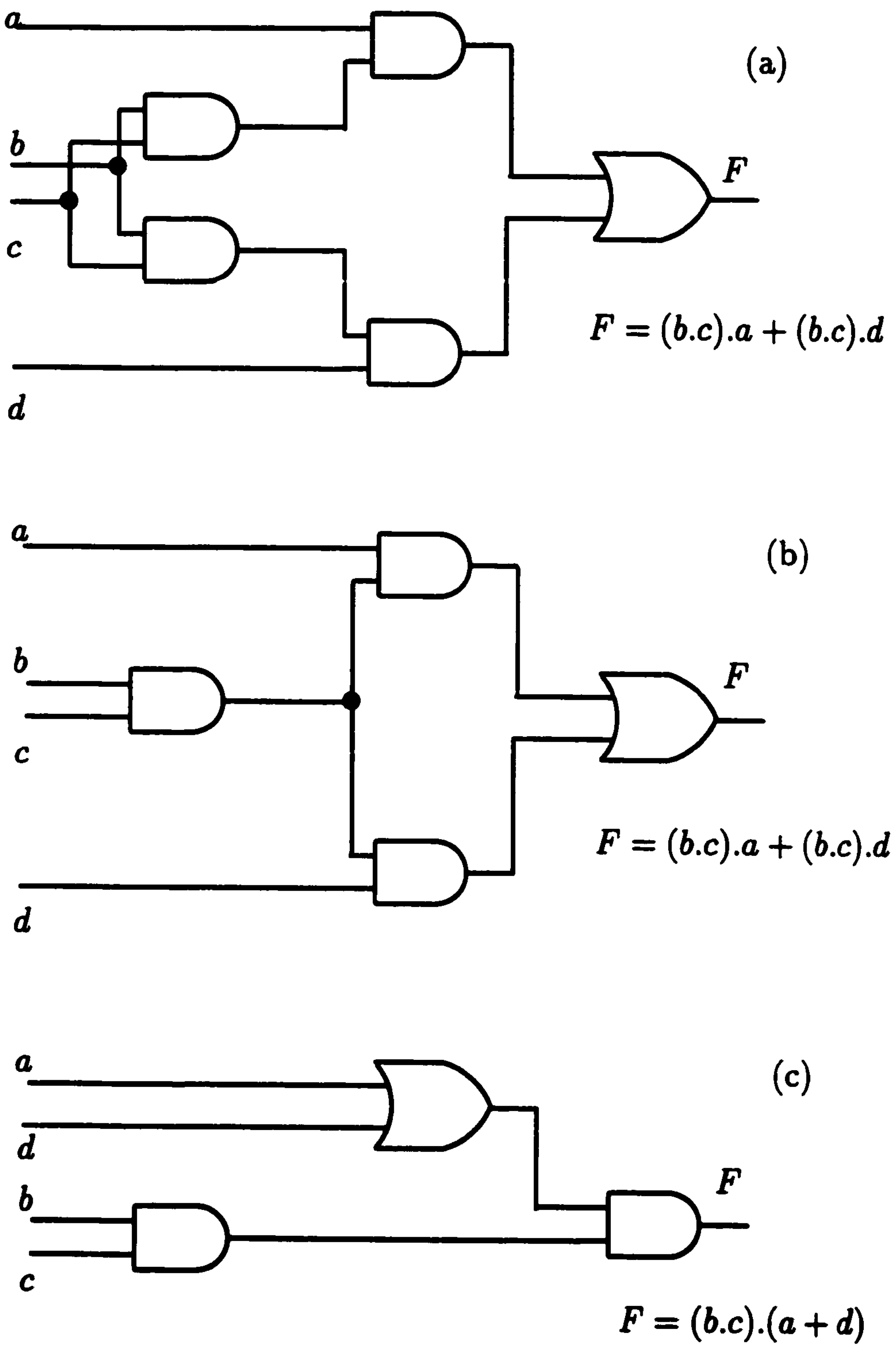


Figure 7.4: *An irredundant circuit is not necessarily minimized in terms of the the number of gates or wires.*

7.5 Network Model

7.5.1 Minimal Feedback Set

We define a minimal feedback set in this Section using a circuit graph. First, we identify some nodes on the circuit with which the feedback state variables are associated with. When the set is detected, a network model is established using the set of state variables.

The circuit graph is a 3-tuple, $G_I = \langle \mathcal{X}, \mathcal{W}, \mathcal{E} \rangle$, where

- $\mathcal{X}$ is a set of *input vertices*, labeled a_1, a_2, ..., a_n,
- $\mathcal{W}$ is a set of *wire vertices*, labeled z_1, z_2, ..., z_p, associated with the output wires of each fork[4] element only,
- $\mathcal{E} \subseteq (\mathcal{X} \times \mathcal{W}) \cup (\mathcal{W} \times \mathcal{W})$ is a set of *edges*.

As an example, consider the gate circuit for the majority C-element shown in Figure 7.5. This example is used throughout this Chapter to illustrate the proposed approach. In the circuit graph we have $\mathcal{X}= \{a, b\}$, and $\mathcal{W}= \{z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8\}$. All other wires and gates are assumed without delay.

Each external input is represented by an input vertex. There is an edge connecting the input vertex a_i or a wire vertex z_i to a wire vertex z_j, if the corresponding node for a_i or z_i has a connection to wire vertex z_j through wires or gates which have no delay unit. Our circuit model is constructed by bypassing any gate or wire vertex in the complete circuit graph except wire vertices which are assigned to the

[4] A fork has one input wire, called a *fork's input*, a connection node, called *fork's node*, and several outgoing wires, each called a *fork's output*.

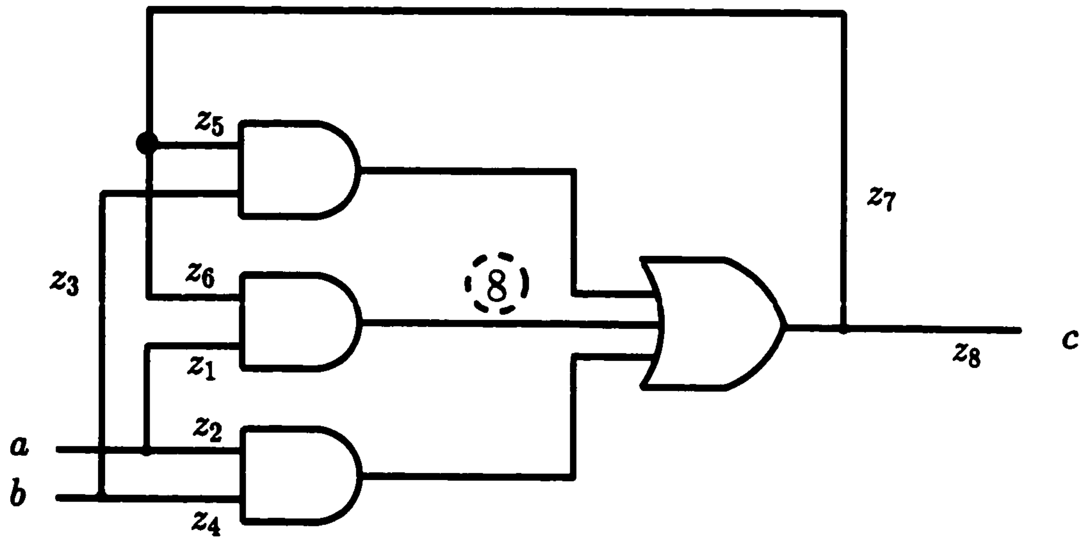


Figure 7.5: *The majority C-element: the gate-circuit with associated state variables.*

outputs of a fork. Figure 7.6 shows the circuit graph obtained from the circuit of Figure 7.5.

In order to pick up feedback variables, cycles must be detected. A cycle is a sequence of vertices and edges, $(z_i, \varepsilon_i, z_{i+1}, \varepsilon_{i+1}, \cdots, z_j)$, such that the initial vertex, z_i, is the same as the final vertex, z_j, and each edge, ε_k, is connecting z_k to z_{k+1} in this sequence.

A set of vertices of the circuit graph is called a feedback vertex set, if every cycle of G_I contains *only* one vertex from the set. In our example, there are two feedback vertex sets, $\{z_7\}$ and $\{z_5, z_6\}$. Although removing variables in set $\{z_7, z_5\}$ from the circuit graph G_I eliminates all cycles in the circuit graph, the set is not considered to be a feedback variable set in this dissertation.

A feedback variable set is considered to be minimal if it contains the least number of variables. In our example, only $\{z_7\}$ is considered to be the minimal feedback variable set. Although our model for testability works correctly using any feedback variable set, we adopt a minimal feedback variable set to reduce computational costs.

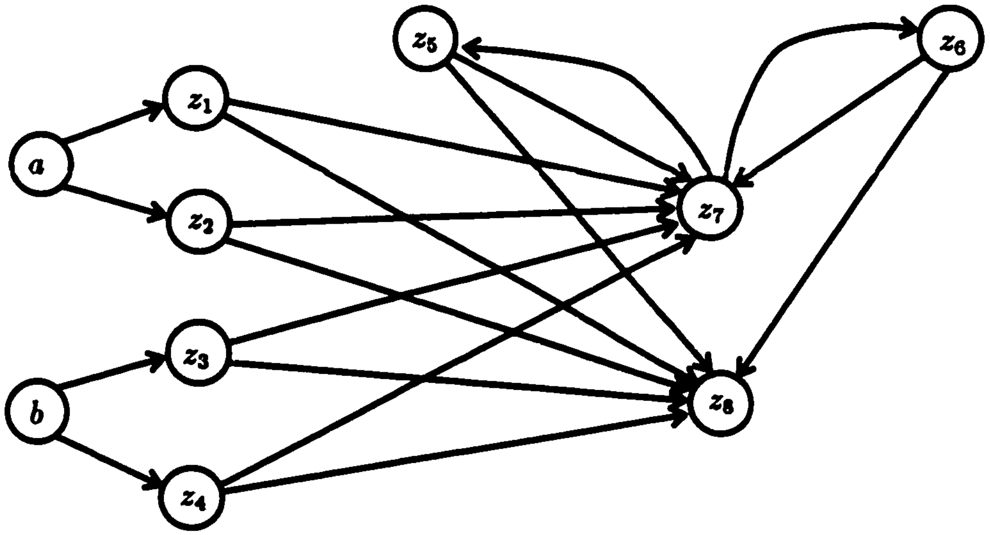
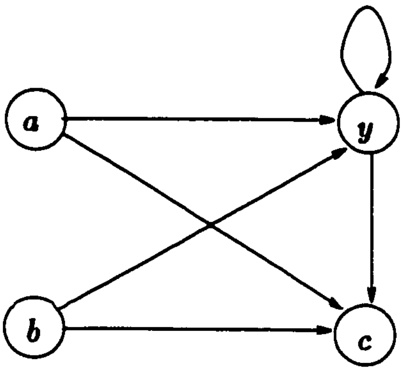


Figure 7.6: *Circuit graph used to obtain feedback variable set. Edges are not named to simplify the figure.*

Let $\mathcal{Y}$ be the union of the minimal feedback set and the outputs of any state-holding element which is considered by its function regardless of how it is realized. The input vertices are kept in our network model, and any other variable in our circuit graph is disregarded. Let C be the set of primary outputs, and $\Gamma = \mathcal{Y} \cup C$. Delays are associated with any variable from the set Γ, and consequently, the network model is obtained. Any variable from the set Γ is called a Γ-variable or state variable in this Chapter. Figure 7.7 shows the outcome of the procedure just explained for the majority C-element. z_7 is replaced by y for consistency reason. It is worth noting that the primary output variable is not necessarily a fork's output, as it may be generated through some logic gates after a fork node.

A feedback vertex function, f_y, or an output vertex function, f_c, maps a set



$$\begin{cases} Y = ab + ay + by \\ C = ab + ay + by \end{cases}$$

Figure 7.7: *Network model using* $\{y, c\}$.

of binary $(n + l)$-tuples, $(a_1, a_2, \cdots, a_n, y_1, y_2, \cdots, y_l)$ where $a_i \in \mathcal{X}$ and $y_j \in \mathcal{Y}$, to $\{0, 1\}$. Note that a feedback vertex function does not depend on output variables since primary outputs are not connected to any gate of the circuit. However, the primary output variable may depend on itself if it is the output of a state-holding element. The value computed by the vertex function is the excitation of the vertex function and may be different from the present value of that vertex variable. The excitation functions for our example are as follows:

$$\begin{cases} Y = ab + ay + by \\ C = ab + ay + by \end{cases} .$$

The network model which contains input vertices, Γ-vertices, and edges connecting vertices, along with its domain and excitation functions is sufficient information to describe the behavior of the gate circuit in which the UIN delay elements are associated with Γ-variables only.

Necessary steps to generate the behavioral model for testability (BMT) graph are explained for our asynchronous gate-circuit example in the next Section.

7.6 The Behavioral Model for Testability (BMT)

BMT is a general single winner (GSW) model with nodes and edges defined in this Section. It is general since the exact delay values are not known. It is single winner model because only one unstable state variable is changed in case a race is possible. The BMT graph contains nodes and edges described as follows.

- Each node is an $(l+m)$-tuple $\gamma = (y_1, y_2, \cdots, y_l, c_1, c_2, \cdots, c_m) \in \{0,1\}^{l+m}$, where $y_i \in \mathcal{Y}$ and $c_j \in \mathcal{C}$. While γ_i is a state variable, γ^k represents an $(l+m)$-tuple, which is called *total state*. We may separate feedback variables from primary output variables by a dot (.) when increased readability is required. The l-tuple $(y_1, y_2, \cdots, y_l)$ is called the *internal* state of the circuit, and the m-tuple $(c_1, c_2, \cdots, c_m)$ is called the *external* state of the circuit.

- Each edge is labeled by an n-tuple $X = (a_1, a_2, \cdots, a_n) \in \{0,1\}^n$, being the values of the input excitations. X^k represents an n-tuple. The input values are kept constant for as long as at most one state variable being changed. The rules for placing edges are explained in the next paragraphs.

For a given node, γ^k, and under the input excitations, X^j, the set of unstable state variables is defined as $\mathcal{U}(X^j, \gamma^k) = \{\gamma_i | \gamma_i \neq S_i(X^j, \gamma^k)\}$, where $S_i(X^j, \gamma^k)$ is the excitation value for state variable γ_i with $X = X^j$ and $\gamma = \gamma^k$ using excitation functions.

The BMT graph is generated using the following rules.

- A node, γ^k, is connected to itself through an edge labeled by X^j if $\mathcal{U}(X^j, \gamma^k) = \emptyset$. Such a node is called stable under X^j.

- Consider node γ^k, input excitations X^j, and the set of unstable state variables $\mathcal{U}(X^j, \gamma^t) \neq \emptyset$. Let γ^t be obtained from γ^k by complementing only one state variable which belongs to $\mathcal{U}(X^j, \gamma^k)$. Other state variables are kept constant in this process. Nodes γ^k and γ^t are connected by a directed edge labeled X^j from node γ^k to node γ^t.

No other pairs of nodes are connected. If $\mathcal{U}(X^j, \gamma^k)$ has m separate state variables, then a race may occur and node γ^k may go to m different nodes by directed edges which are all labeled X^j. One way to build the BMT graph is to consider one node at a time, and complete its connections to itself or other nodes under all possible input excitations. To find the next node in case some instability exists, flip only one unstable variable to its opposite logic value at a time.

7.6.1 An Example

Here we present the detailed steps for the majority C-element. Initially, we begin with $\gamma^0 = (y, c) = (0, 0) = 00$.

- step 1: Let the input excitations be $X^0 = (a, b) = (0, 0) = 00$. Using excitation functions, we have $\mathcal{U}(X^0, \gamma^0) = \emptyset$. Therefore, there is an edge from node 00 to itself labeled 00, see Figure 7.8(a).

- step 2: Consider $X^1 = 01$ for γ^0. Since $\mathcal{U}(X^1, \gamma^0) = \emptyset$, there is an edge from 00 to itself labeled 01, see Figure 7.8(b).

- step 3: Consider $X^2 = 10$ for γ^0. Since $\mathcal{U}(X^2, \gamma^0) = \emptyset$, there is an edge from 00 to itself labeled 10, see Figure 7.8(c).

- step 4: Consider $X^3 = 11$ for γ^0. Since $\mathcal{U}(X^3, \gamma^0) = \{y, c\}$, a race can happen. We obtain the next nodes by complementing only one state variable at a time. Therefore, node 00 is connected to node $\gamma^1 = 01$ and node $\gamma^2 = 10$ using edges labeled 11, see Figure 7.8(d).

- step 5: Consider $X^0 = 00$ for $\gamma^1 = 01$. Since $\mathcal{U}(X^0, \gamma^1) = \{c\}$, there is only one unstable variable. Hence, node 01 is connected to node 00 by an edge labeled 00, see Figure 7.8(e).

- step 6: Consider $X^1 = 01$ for $\gamma^1 = 01$. Since $\mathcal{U}(X^1, \gamma^1) = \{c\}$, there is only one unstable variable. Hence, node 01 is connected to node 00 by an edge labeled 01, see Figure 7.8(f).

- step 7: Consider $X^2 = 10$ for $\gamma^1 = 01$. Since $\mathcal{U}(X^2, \gamma^1) = \{c\}$, there is only one unstable variable. Node 01, therefore, is connected to node 00 by an edge labeled 10, see Figure 7.9(g).

- step 8: Consider $X^3 = 11$ for $\gamma^1 = 01$. Since $\mathcal{U}(X^3, \gamma^1) = \{y\}$, no race is possible. Therefore, node 01 is connected to node $\gamma^3 = 11$ using a directed edge labeled 11, see Figure 7.9(h). Node 01 is now completed.

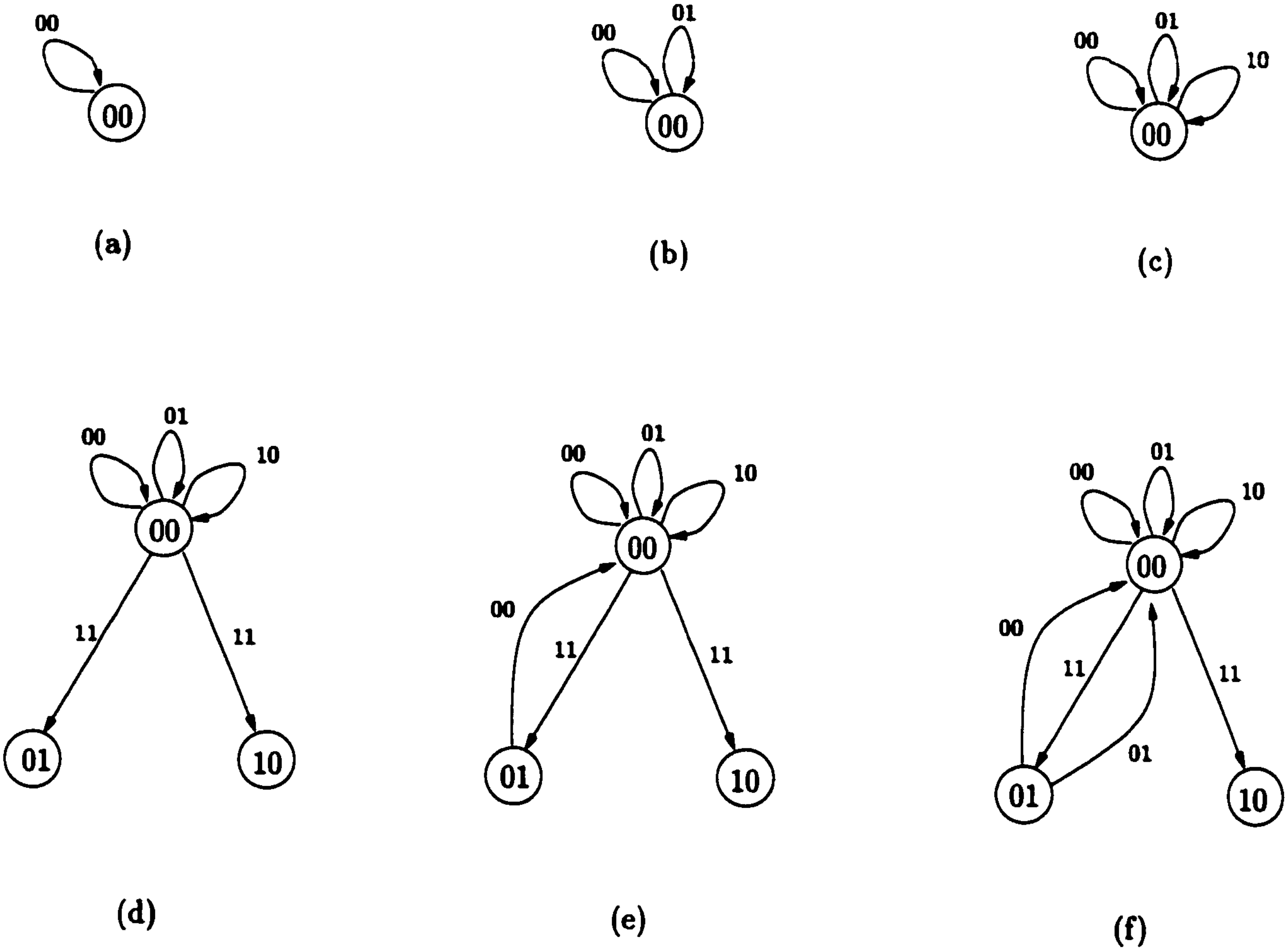


Figure 7.8: *The BMT graph: steps (a) through (f).*

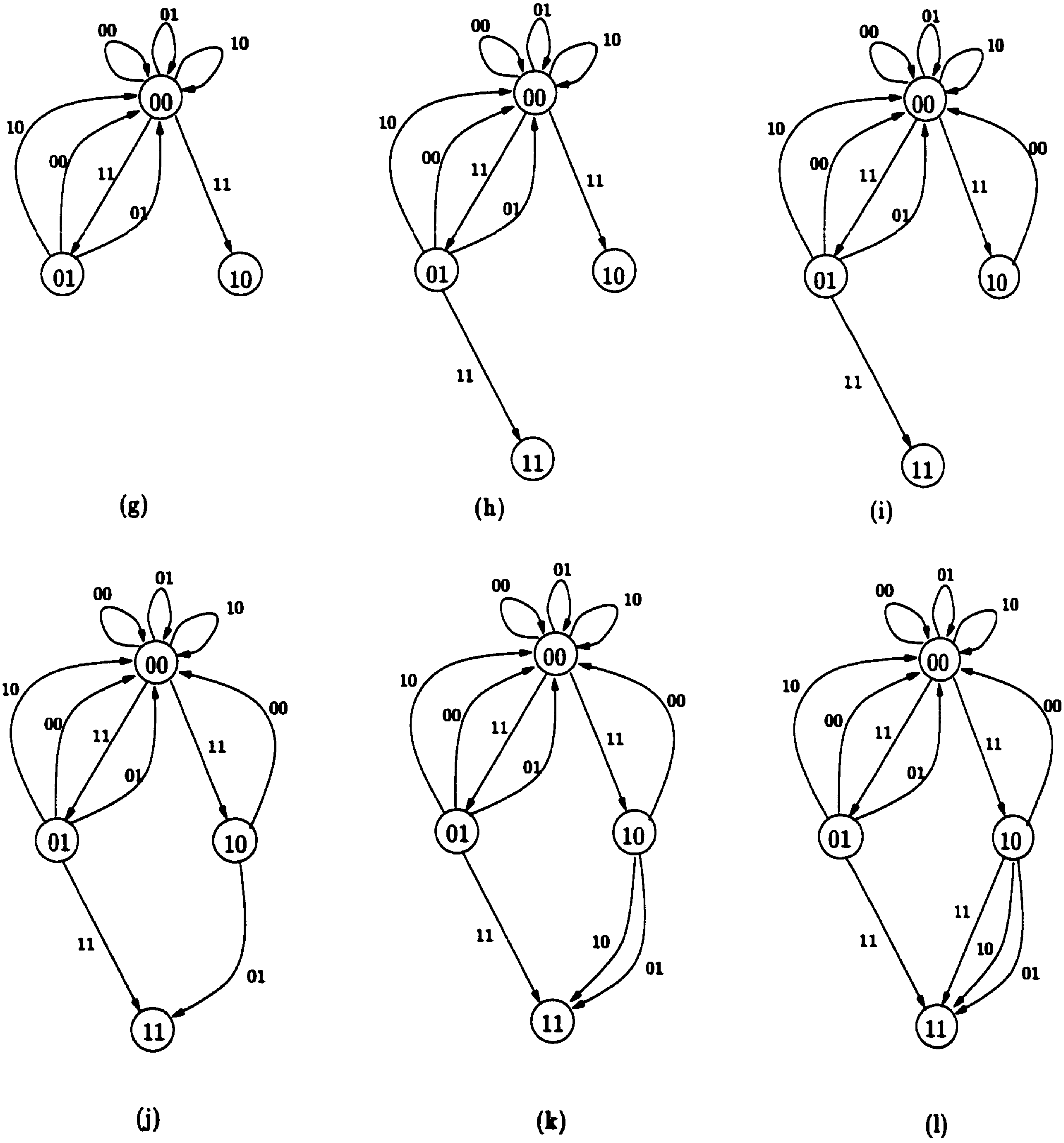
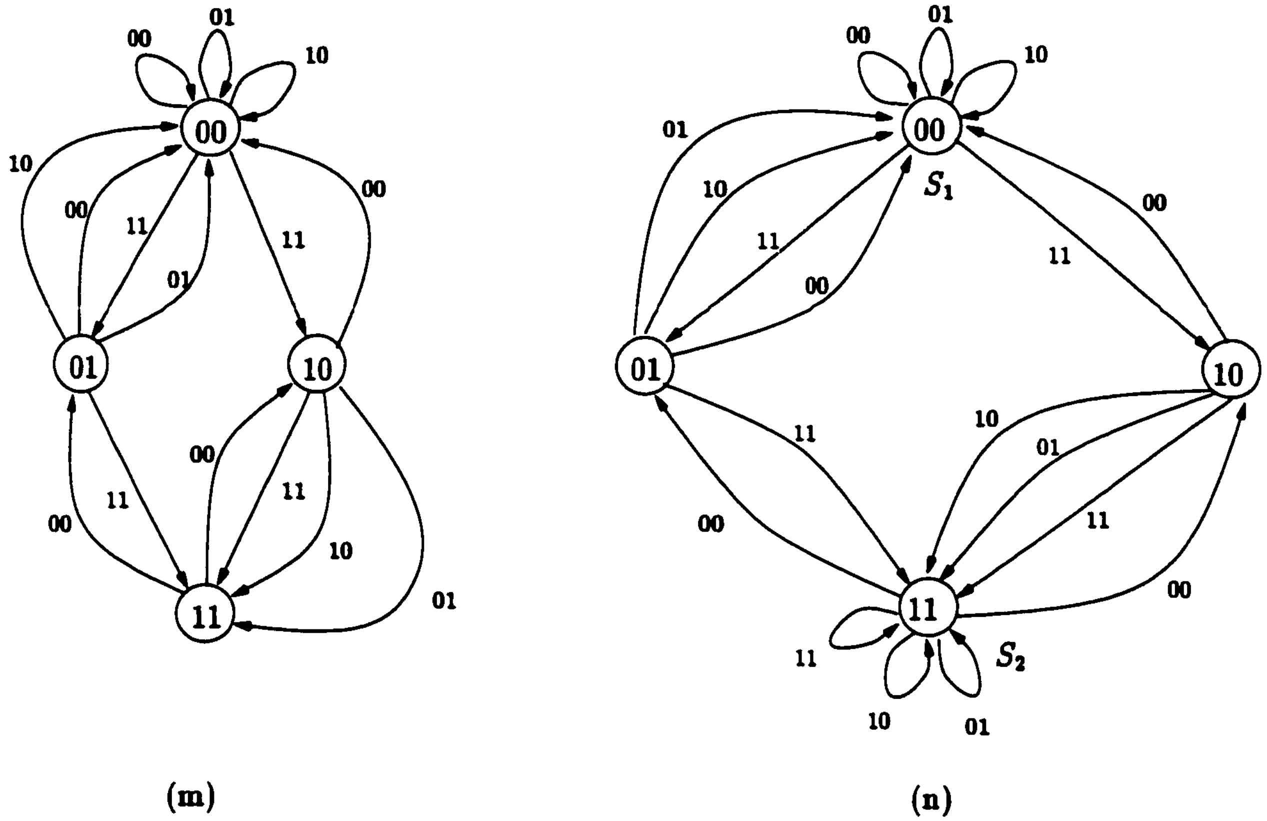


Figure 7.9: *The BMT graph: steps (g) through (l).*



(m)

(n)

Figure 7.10: *The BMT graph: steps (m) and (n).*

We simplify the next steps using brief notations.

$$\text{For } \gamma^2 = 10 \Rightarrow \begin{cases} \text{and } X^0 = 00, & \mathcal{U}(X^0, \gamma^2) = \{y\}, & \text{see Figure 7.9(i)}, \\ \text{and } X^1 = 01, & \mathcal{U}(X^1, \gamma^2) = \{c\}, & \text{see Figure 7.9(j)}, \\ \text{and } X^2 = 10, & \mathcal{U}(X^2, \gamma^2) = \{c\}, & \text{see Figure 7.9(k)}, \\ \text{and } X^3 = 11, & \mathcal{U}(X^3, \gamma^2) = \{c\}, & \text{see Figure 7.9(l)}. \end{cases}$$

$$\text{For } \gamma^3 = 11 \Rightarrow \begin{cases} \text{and } X^0 = 00, & \mathcal{U}(X^0, \gamma^3) = \{y, c\}, & \text{see Figure 7.10(m)}, \\ \text{and } X^1 = 01, & \mathcal{U}(X^1, \gamma^3) = \emptyset, & \text{see Figure 7.10(n)}, \\ \text{and } X^2 = 10, & \mathcal{U}(X^2, \gamma^3) = \emptyset, & \text{see Figure 7.10(n)}, \\ \text{and } X^3 = 11, & \mathcal{U}(X^3, \gamma^3) = \emptyset, & \text{see Figure 7.10(n)}. \end{cases}$$

7.7 The BMT Correct Behaviors

The next step involves mapping the correct circuit behavior, which is provided by Karnaugh map or state table, to the BMT graph. The mapping results in transitions from one stable state to another.

Consider two stable states S_n and $S_m \neq S_n$. Let $\mathcal{M}_m$ denote the set of input excitations, v_i, under which S_m is stable. Consider a path between these two stable states represented by:

$$\mathcal{P}_{nm} \equiv S_n : v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_i \rightarrow \cdots \rightarrow v_k \rightarrow S_m,$$

and $\mathcal{N}_{nm}$ denote input excitations on the path: i.e.,

$$\mathcal{N}_{nm} = \{\cup\{v_i, v_{i+1}\} \mid (v_i \rightarrow v_{i+1}) \in \mathcal{P}_{nm}\}.$$

$\mathcal{P}'_{nm}$ is an *image* of $\mathcal{P}_{nm}$ if $S'_n = S_n$, $S'_m = S_m$, and $\mathcal{N}'_{nm} \subset \mathcal{N}_{nm}$: i.e.,

$$\mathcal{P}'_{nm} \equiv S_n : v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_i \rightarrow S_m.$$

$\mathcal{P}_{nm}$ is called the *cover* of $\mathcal{P}'_{nm}$.

7.7.1 Selection Rules

The correct paths are selected using the following rules.

1. *$\mathcal{N}_{nm} \subset \mathcal{M}_m$: i.e., state S_m should be stable under any input excitation from $\mathcal{N}_{nm}$.*

   Justification: If the circuit enters the stable state correctly and remains there for all such acceptable paths, then the circuit is either fault-free or has undetectable faults.

2. *There should not exist any static or dynamic hazards on $\mathcal{P}_{nm}$.*

   Justification: The circuit is designed hazard-free. If a hazard happens, then the circuit is either faulty or it has delay fault. Either way, there exists a path by which the circuit goes into the wrong stable state.

3. *There should not exist an image of $\mathcal{P}_{nm}$ which ends in a different stable state $S_k(\neq S_m)$.*

   Justification: In practice, each input excitation, v_i, is applied at least as long as the minimum acceptable delay. If the circuit goes to the wrong stable state while applying the covering path, then the fault is detected.

7.8 Redundancy

Some of the test sequences obtained from the BMT graph, although they represent correct circuit behaviour, are sometimes redundant. The redundancy is explained as follows.

Redundant Edge:

The circuit is combinational between each two states in the BMT graph. There are numerous techniques to drive the complete test set for such a combinational circuit. A test vector which is not in the complete test set is a redundant input vector. One technique to obtain redundant input excitations for test analysis has been reported in [148] for combinational circuits. In [148] we detect redundant edges using an objective function.

Redundant Path:

$\mathcal{P}'_{nm}$ is a *Redundant Path* if its cover, $\mathcal{P}_{nm}$, exists.

The majority C-element is considered as an example. The correct circuit behavior is given by the following rules, which also represent the test sequences.

$$\begin{cases} Rule(1) & S_1 : 11 \rightarrow 01 \Rightarrow S_2 \\ Rule(2) & S_1 : 11 \rightarrow 10 \Rightarrow S_2 \\ Rule(3) & S_2 : 00 \rightarrow 10 \Rightarrow S_1 \\ Rule(4) & S_2 : 00 \rightarrow 01 \Rightarrow S_1 \\ Rule(5) & S_1 = 00, \quad S_2 = 11 \end{cases}$$

This is schematically shown in Figure 7.11. An input excitation which stems from a stable state is called *essential*[5] since it facilitates leaving that state. A single

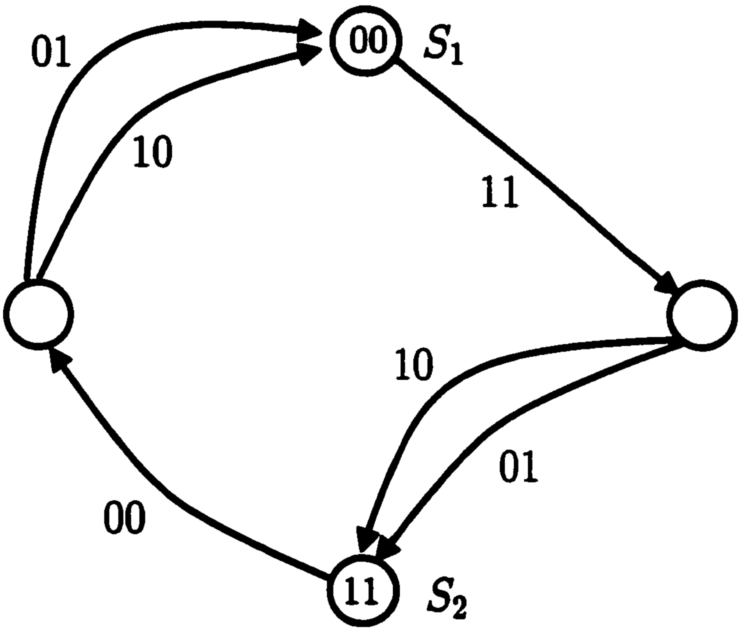


Figure 7.11: *The BMT graph after removing incorrect and redundant transitions.*

[5] It is essential because without applying it, the circuit will not start changing stable state. This terminology is used after the BMT graph is reduced. To reduce the BMT graph, we remove

number, θ =(outdegree − indegree), is assigned to each input excitation, v_i. *Indegree* shows the number of incoming edges connected to v_i and *outdegree* shows the number of outgoing edges. The indegree of essential input excitations is zero, and the outdegree of of any v_i ending in a stable state is also zero. For example, $\theta_{11} = 2$ and $\theta_{01} = -1$.

The BMT graph of the faulty circuit is different than that of the fault-free. This is due to the following fact. The BMT graph captures arbitrary delay units assigned to wires and gates in a fault-free circuit. These delay units, although they are not known, are bounded. There are transitions that happen at the primary input(s) and propagate to the primary output. Stuck-at fault, however, is an infinite delay unit. There exists at least one critical path which behaves differently at the presence of the fault. The following scenarios have been observed at the presence of a stuck-at fault.

1. An unstable state becomes stable.
2. A stable state becomes unstable.
3. A transition between two nodes is disabled.
4. An invalid transition between two nodes is enabled.

Consider a faulty majority C-element at the presence of 8-stuck-at zero. The only sequence which detects this fault is 11 → 10. The BMT graph for the faulty majority C-element is shown in Figure 7.12. Dashed-arrows show the difference

redundant excitations. One redundant excitation can happen when an input forces the circuit state to change in many different directions. Such behavior is taken care of before the linking process begins. It is only here that we define "essential" input.

between the BMT graphs for the faulty and the fault-free C-elements. Rule (2) in the specification is, therefore, violated.

Rule (5) is necessary to distinguish the circuit from its inverse. If we choose rules (1) to (4) and $S_1 = 11, S_2 = 00$, then the circuit would operate as $\overline{C}$.

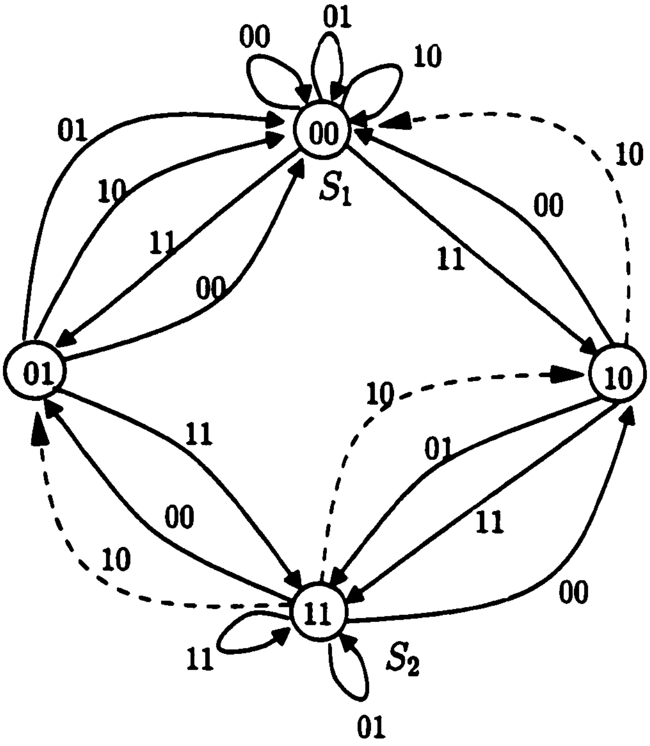


Figure 7.12: *The BMT graph for the majority C-element at the presence of 8-sa0.*

If the BMT Graph does not change at the presence of a fault, the fault is not logically testable.

Another property of the BMT graph is that any state variable should change from 0 to 1 and from 1 to 0 at least once. If this never happens, then the state variable or its corresponding circuit line is redundant.

7.8.1 State-Holding Component

There are three rules which must be followed when deriving the BMT graph for the output of the state-holding element.

1. The output of the state-holding component depends on itself and its excitation function. A component C-element has the excitation function given by $C = ab + ac + ac$.

2. The component has a delay described by UIN delay model.

3. No output delay is associated with the primary output of the component. The component, however, appears as a gate-vertex in circuit graph when feedback variable set is found.

Similar to any state variable, the output of the state-holding component must change from 0 to 1 and from 1 to 0.

7.9 Linking Test Sequences Together

The next step in generating complete test sequence is to link the pieces of test sequences obtained in Section 7.8 using the following rules.

1. *Select an input excitation with zero indegree.*

2. *Select input excitation v_j after v_i if $\theta_{v_j} \leq \theta_{v_i}$. This is continued until an input excitation with zero outdegree is selected.*

3. *Remove an input excitation which has zero outdegree and is selected at least once. Re-calculate θ for each edge and go to step 1.*

4. *Every input excitation derived in Section 7.8 must be taken at least once. Go to step 1, otherwise.*

(v_i, v_j) are connected by a directed edge from v_i to v_j. The algorithm explained above provides the necessary steps to generate a sequence of input excitations, $(v_i, v_j, \cdots, v_k)$, which detects faults in the circuit. The proof is to follow. Minimizing the length of this sequence is an $\mathcal{NP}$-complete problem [66, 88, 93]. The following step is helpful in reducing the length of test sequence.

- Input excitations which are between two nodes of the BMT graph have equal *indegrees* and *outdegrees*. These nodes are connected together in any order. This is a valid statement because the circuit is combinational between the two nodes and, as we have seen in Chapter 6, the order of the test inputs is not important.

A complete test sequence must always start with an input excitation which has a zero indegree. Such input excitation would force the circuit to initialize. If the circuit is not directly initialized as predicted by the correct circuit behavior, the circuit is faulty. If the circuit is not initializable, the initialization technique explained in Chapter 7 is used to facilitate the initialization process.

Each of the the following test sequences provide a complete test sequence for the majority C-element. We have verified exhaustively that each test sequence detects single and multiple faults and that there is not smaller test sequence.

$$\begin{cases} 11 \rightarrow 10 \rightarrow 01 \rightarrow 00 \rightarrow 10 \rightarrow 01, \\ 11 \rightarrow 10 \rightarrow 01 \rightarrow 00 \rightarrow 01 \rightarrow 10, \\ 11 \rightarrow 01 \rightarrow 10 \rightarrow 00 \rightarrow 10 \rightarrow 01, \\ 11 \rightarrow 01 \rightarrow 10 \rightarrow 00 \rightarrow 01 \rightarrow 10, \\ 00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 01 \rightarrow 10, \\ 00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 10 \rightarrow 01, \\ 00 \rightarrow 10 \rightarrow 01 \rightarrow 11 \rightarrow 10 \rightarrow 01, \\ 00 \rightarrow 10 \rightarrow 01 \rightarrow 11 \rightarrow 01 \rightarrow 10. \end{cases}$$

One can verify for each test sequence that the majority C-element would pace both stable states S_1 and S_2 according to BMT graph. One may select the test sequence as

$$11 \rightarrow 10 \rightarrow 00 \rightarrow 10 \rightarrow 11 \rightarrow 01 \rightarrow 00 \rightarrow 01,$$

if the minimum length complete test sequence is not chosen. It is easy to verify that the mentioned sequence of test vectors is hazard-free. Other possible hazard-free complete test sequences are as follows.

$$\begin{cases} 11 \rightarrow 10 \rightarrow 00 \rightarrow 01 \rightarrow 11 \rightarrow 01 \rightarrow 00 \rightarrow 10, \\ 11 \rightarrow 01 \rightarrow 00 \rightarrow 10 \rightarrow 11 \rightarrow 10 \rightarrow 00 \rightarrow 01, \\ 11 \rightarrow 01 \rightarrow 00 \rightarrow 01 \rightarrow 11 \rightarrow 10 \rightarrow 00 \rightarrow 10, \\ 00 \rightarrow 10 \rightarrow 11 \rightarrow 01 \rightarrow 00 \rightarrow 01 \rightarrow 11 \rightarrow 10, \\ 00 \rightarrow 10 \rightarrow 11 \rightarrow 10 \rightarrow 00 \rightarrow 01 \rightarrow 11 \rightarrow 01, \\ 00 \rightarrow 01 \rightarrow 11 \rightarrow 01 \rightarrow 00 \rightarrow 10 \rightarrow 11 \rightarrow 10, \\ 00 \rightarrow 01 \rightarrow 11 \rightarrow 10 \rightarrow 00 \rightarrow 10 \rightarrow 11 \rightarrow 01. \end{cases}$$

One can obtain the hazard-free test sequences by touring the reduced BMT graph shown in Figure 7.11. Any tour should observe one input change at a time. It may be necessary to travel through the graph and take input excitations more than once to fulfill this condition.

7.10 Computational Cost

The BMT technique is much cheaper than exhaustive testing techniques in terms of the computation costs. In order to detect a fault in an asynchronous sequential circuit, a sequence of input excitations is necessary. Consider a sequential circuit with n inputs. The number of possible sequences of length m for such circuit is $(2^n - 1)^{m-1}2^n$. For our C-element where $n = 2$, we may need to check 2916 test sequences of length 7. There are realizations of C-element in which some faults are detected by a test sequence of length 14. To find one complete test sequence for such a circuit, we may need to check 6.4×10^6 test sequences of length 14. This approach is not practical since the number of possible sequences grows exponentially as m grows. Using the BMT graph, we study the effects of the sequence of input excitations on the fault-free circuit and consider only those sequences which excite or propagate a transition. This graph can tell as whether a test sequence needs to be of length 2, 3, or m. It can also predict which order of input excitations is most important.

One available test technique for asynchronous sequential circuits is to find the faulty machines for all possible single and multiple faults and use **observer**. This is done in two steps. The first step is to generate the faulty machines. A circuit with k lines can potentially contain $(3^k - 2k - 1)$ distinct [6] multiple stuck-at faults.

[6] Each line can be either stuck-at zero, stuck-at one, or fault-free, so there will be 3^k different cases. $2k$ of them, however, correspond to single stuck-at faults and one case corresponds to fault-free circuit; therefore the remaining will be $3^k - 2k - 1$.

We have to generate faulty machines by hand at the present time since there are no techniques to generate them automatically. It is virtually impossible to find all faulty machines for a complex circuit.

The next step is to produce an input file which contains possible faulty machines and run **observer** on the input file. **Observer** then generates a graph of all possible sequences which can detect some faults. For the majority C-element, for example, there are 45 different states and 95 edges. However, our BMT graph requires only 4 states and 18 edges for the same circuit. As the number of inputs increases, the number of states generated by the **observer** increases too. **Observer** generates 512 states for Khoche's C-element [96], which has two extra test inputs. We do not know how many states are necessary for Bartky or Mayevsky's C-element [98, 100] using **observer** since it was impossible to consider multiple faults for them[7]. The algorithm used by **observer** is expensive in terms of computation time.

The cost for the BMT graph is relatively small. There are at most 2^h total states for a circuit which has h number of state variables. This part of the cost is independent of the number of inputs. There are only 4 states for the majority C-element, Khoche's C-element [96], or Mayevsky's C-element [98, 100]. The computation, however, can be relatively costly. There are 2^n possible transitions at each total state for a circuit with n inputs. The number of transitions, however, is less than those generated by the **observer**.

7.11 Rationale

The question addressed in this Section is whether or not the test sequence obtained in Sections 7.7, 7.8, and 7.9 detects all single and multiple faults. The answer is

[7] The single fault results have been reported in Appendix C.

expressed in the form of the following theorems.

Theorem 7.1 *An initializable sequential circuit is fault-free if it has the following properties.*

1. *The circuit is initialized to any valid stable state.*
2. *The state of the circuit changes from stable state S_n to S_m when the valid input excitations are applied.*

Proof: First, if the circuit is not initialized to a stable state, it is either unstable or it is not initializable. Both possibilities are invalid for the fault-free circuit as it is assumed to be initializable to a stable state. Second, the primary objective of a digital designer is to provide a circuit which produces correct output sequence when a known input sequence is applied. QED.

Theorem 7.2 *The BMT graph of the faulty circuit with a detectable fault is different from that of the fault-free for at least one transition.*

Proof: The circuit is combinational between each two states. Each input excitation which changes at least one state variable provides a transition path from the inputs to the outputs of the circuit between the two states. Therefore, each edge of the BMT graph corresponds to a red edge in the combinational circuit.

Consider two combinational circuits which are identical but one circuit has a single or a multiple fault, see Figure 7.1. In order to detect the fault, both circuits are excited with the same inputs and the outputs are compared. Let the output of the fault-free circuit be α and that of faulty circuit be β. The fault is logically testable if:

1. There exists an input excitation, v_i, for which $\alpha \neq \beta$. Input v_i is, therefore, a test vector.

2. Consider a different input excitation, v_j, for which the output response of the fault-free and the faulty circuits are the same, i.e., $\alpha' = \beta'$ but $\alpha' = \alpha$ and $\beta' \neq \beta$. An invalid output transition starting from $C = \beta'$ and ending in $C = \beta$ under the input sequence $v_i \rightarrow v_j$ is enabled. Therefore, $v_i \rightarrow v_j$ is the test sequence which would detect the fault.

3. Similar statement is correct if there exists v_j for which $\alpha' = \beta'$ but $\alpha' \neq \alpha$ and $\beta' = \beta$. In this case, a valid output transition is disabled.

Therefore, there exists at least one edge originating from $C = \beta$ and ending in $C = \bar{\beta}$ which is either disabled or its direction is reversed at the presence of the fault. If $\alpha = \beta$ for all possible inputs, then the fault is not logically testable. QED.

Theorem 7.3 *The algorithm described in Sections 7.7, 7.8, and 7.9 provides a test sequence which detects all single and multiple faults for a sequential circuit.*

Proof. Consider two sequential circuits which are identical but one circuit has a single or a multiple fault. To test a sequential circuit, it has to be initialized first. If the output of the faulty circuit fails to be initialized to the same logical value as the output of the fault-free circuit, then the fault is already detected. If these two circuits are initialized to the same total state and the circuit state changes to the same total states when the inputs are applied, the fault is not detectable. Note that a fault needs to be excited and propagated in order to be detected. If both circuits are initialized to the same output values, but not necessarily the same total states, then the following cases may arise.

- If both circuits show the same output transitions under the test sequences devised in section 7.7, then the fault is not detectable.
- If there is one test sequence which forces the faulty circuit to show a different output transition than that of the fault-free circuit, then the fault is detected by the set of applied input excitations.

Theorem 7.1 describes a fault-free circuit precisely. If the circuit fails to follow fault-free characteristics, then it is faulty. Since we have made no reference as to what multiplicity of faults are under investigation, the test sequence will detect single and multiple faults as intended. QED.

This proof shows that if the fault is detectable, then there exists at least one initial total state and a set of input excitations which change the state of the circuit to a different total state than the one described by the BMT graph for the fault-free circuit.

Transitions in the BMT graph do not necessarily correspond to the correct circuit behavior expected by the designer. If the design is correctly defined using fundamental (input/output) mode of operation, then parts of the BMT graph which concur with that mode of operation is considered to be part of the correct circuit behavior. Some parts of the BMT graph, although they show the behavior of a fault-free gate-circuit, will correspond to faulty behavior. This may happen due to the relative values of delays.

Two different gate-circuits may have the same correct circuit behavior even though their BMT graphs are different. It is also possible that two gate-circuits, such as the majority C-element and Wuu's C-element [192], have the same BMT graph and the same correct circuit behavior. Such circuits have the same faulty behaviors and the same complete test sequence.

7.12 Important Results

There are a number of novel results stated in this Section which are stated in the form of the following general theorems.

Theorem 7.4 *The complete test set of an irredundant sequential logic circuit depends only on the inputs, the forks inputs and outputs (feedback variables are included), the outputs of the state-holding elements, and the BMT graph. It does not depend on the interconnections of gates, or the gate type used.*

Proof: The proof is straightforward since the algorithm explained in Sections 7.7, 7.8, and 7.9 does not depend on the circuit structure. QED.

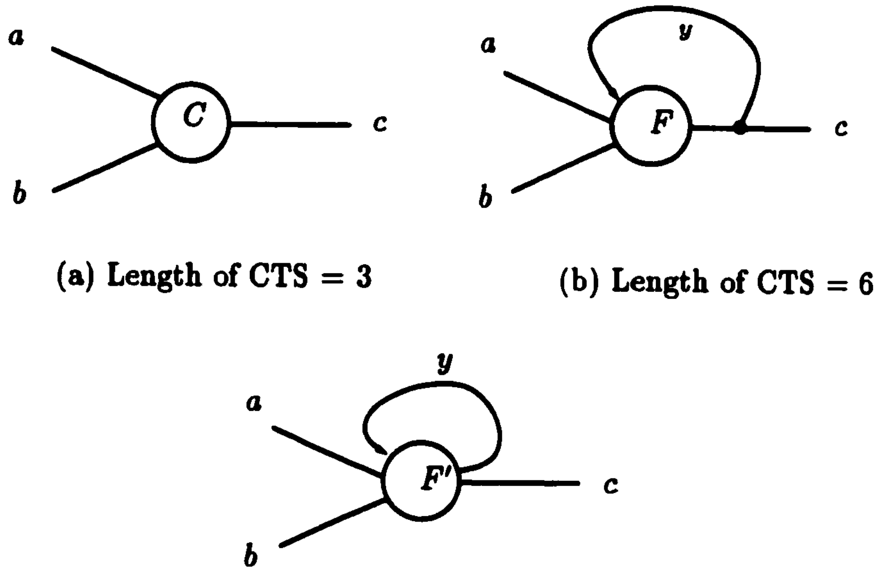
Figure 7.13(a) shows the C-element as a component. The designer does not have any information regarding how this component is realized. Whether this component has a feedback variable or not is not known. It is unclear if this component is static or dynamic. The only fact known is the correct circuit behavior using the BMT graph. The complete test sequence is of length three.

So many C-elements can be realized using standard gates. If the C-element is realized using one and only one feedback variable, then it needs a minimum length sequence of six tests to detect single and multiple faults.

Consider Figure 7.13(b). The C-element is realized using two inputs, one output, one feedback variable, and a component which is called F. Note that this component is different than the component C-element shown in Figure 7.13(a). While component F has three inputs a, b, and y, the component C-element has only two inputs a and b. The excitation functions for component F is different than that of the component C-element. While the output of the former does not depend on itself, the output of the latter does. Component F is a combinational component, whereas component C-element is a state-holding element or a sequential circuit.

Figure 7.13(c) shows a more general realization of the C-element. Component F' is different than component F since it has two outputs. Nevertheless, component F and F' have the same properties.

Both Figures 7.13 (b) and (c) have a minimum test sequence of length six regardless of how they are realized. We have examined test sequences at the presence of all possible single and multiple stuck-at faults. Our simulations show that the test sequence is complete and minimal.



(a) Length of CTS = 3

(b) Length of CTS = 6

(c) Length of CTS = 6

Figure 7.13: *The component C-element realized using (a) no feedback, (b) component F and feedback variable y connected to the primary output, (c) component F′ and an internal node of component F.*

Theorem 7.4 states that whether we use NAND-NAND, NAND-NOR, AND-AND, OR-OR, NOR-NAND, or AND-OR-Inverter technique to realize the logic circuit, the complete test set would not be any different. It also states that a logic circuit has the same complete test set as the circuit with inverted logic. Both majority C-element and Wuu's C-element [192] have the same number of inputs, feedback variables, and BMT graphs. The complete test sequences for both circuits are, therefore, the same.

The majority C-element and Mayevsky C-element [98,100] have the same number of inputs and feedback variables. The BMT graph of these circuits are different in only one transition. The complete test sequences, however, are the same in size and order of input excitations.

As the number of feedback variables increases, the length of the complete test sequence will also increase in size. We have studied Bartkey C-element [100] which has an additional feedback variable compared to that of the majority C-element. The complete test sequence of the former is longer by one to that of the latter.

Theorem 7.5 *Adding extra test signals does not shrink the size of test sequence.*

Proof: An additional test signal is applied through an additional input. The complete test sequence depends on the number of inputs as stated in the algorithm. Therefore, the size of the the complete test sequence will have to remain the same or to increase. QED.

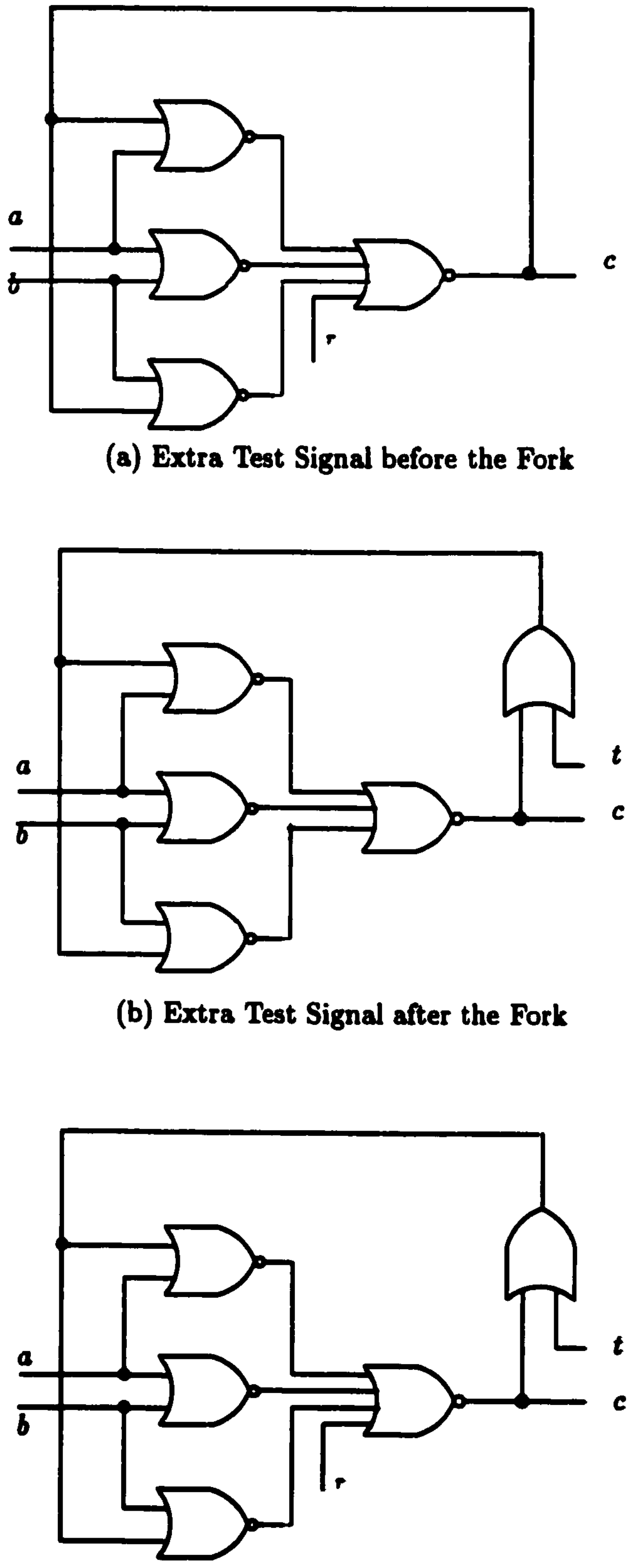
This theorem states that adding test signal will not provide better controllability as many researchers have believed. The reason is that the additional test input needs to be tested like other inputs because it can be faulty with the same probability. Our study of Khoche C-element [96] shows that the length of the complete test sequence is greater than six. This is justified since the number of forks and feedback variables have not changed. The number of inputs, however, has increased by one for Figure 7.13 (a) and (b), or two for Figure 7.13 (c).

Theorems in Section 7.12 provide us with the necessary means to design circuits which have complete test sequence of smaller length. A circuit which realizes the same logic function is considered to be better testable if it has:

- Fewer inputs
- Fewer Fork inputs
- Fewer Fork outputs (and Fewer Feedback variables)

7.13 Benchmark Circuits

There are only a few reports for test generation of synchronous sequential circuits due to the complexity of their test generation problem. No previous results have been reported for single *and* multiple fault testing on ISCAS'89 benchmark circuits [27].



(a) Extra Test Signal before the Fork

(b) Extra Test Signal after the Fork

(c) Extra Test Signals before and after the Fork

Figure 7.14: *Different realizations of Khoche C-element.*

Test generation of asynchronous circuits has not been studied in depth. Many researchers assumed that asynchronous circuits are self-checking. This assumption does not hold if the faults inside the basic component (such as C-element) are considered. There are no benchmark circuits for asynchronous sequential circuits with relatively large number of gates and interconnection lines or primary inputs. There is no previous attempt on circuits we studied. As a result, comparison is not possible.

Hazewindus has studied testing asynchronous circuits using D-algorithm. However, only theoretical results were provided. Similar to Hazewindus work [82], we have explained how our algorithm works in this Chapter, and provided some theoretical results. A comparison between our technique and Hazewindus technique was not possible since there was no empirical results reported in [82] nor any circuit in detail was studied.

ISCAS'89 benchmark circuits are sequential, synchronous, and use only D-type flip-flops. One possible solution to generate asynchronous sequential circuits is to substitute some or all of the flip-flops with their appropriate gate-level designs. This solution is not applicable to our test generation technique for the following reason. It has been known [22,23] that test sequences are invalidated due to hazards and races. Recent research [13] has shown that there are hazards in asynchronous circuits produced by replacing flip-flops in ISCAS'89 benchmark circuits. On the other hand, our test generation technique is asynchronous in nature because only delays of wires and gates should be considered. As we have mentioned in Section 7.3, an assumption for our technique is that the circuit under test should be hazard and race free. Therefore, we cannot report results on ISCAS'89 benchmark circuits.

Another interesting set of benchmark circuits is ISCAS'85 combinational circuit. A wire is an interconnections between two nodes, namely, *source* node and *sink* node. It is possible to insert a D Flip-Flop in a wire, replace D Flip-Flop by its gate-level design, similar to what we discussed in Appendix C, and generate asynchronous circuits. This is not helpful either since hazards have been reported [64] for each ISCAS'85 combinational circuits.

For the reasons mentioned above, we have applied our algorithm on some small circuits which have been provided in Appendix C. Generating a suitable benchmark for asynchronous sequential circuits remains open.

7.14 Summary

We have presented a General-Single-Winner (GSW) model where internal transitions in an irredundant sequential circuit are examined in details. Given a circuit with its components, wires, and topology, the following algorithm finds complete test sequence(s).

1. Assign variables to forks outputs only, discover the cycles, and find the minimal feedback set. Derive the network model, and the excitation functions for state variables, and generate the BMT graph.

2. Check whether all state variables show transitions 0 → 1 and 1 → 0 in the BMT graph for at least once. If there exists a state variable which does not, then such state variable is redundant.

3. Map the correct circuit behavior provided by Karnaugh tables or state tables onto the BMT graph, and find the set of input sequences.

4. Remove redundant paths.

5. Link pieces of test sequences to generate the complete test sequence.

The outcome is a test sequence which detects single and multiple faults in a sequential circuit without fault simulation or fault enumeration. This is the first test generation algorithm for asynchronous sequential circuits without making any reference to the faulty circuit. The outcome is either minimal or nearly-minimal for the following reason. Our algorithm is a combination of branch-and-bound technique and a heuristic method. It resembles a branch-and-bound technique since it searches the entire solution space. It resembles a heuristic technique since it uses an objective function and considers upper-bound and lower-bound limits for the size of the test set. We believe that the quality of the test sequence in terms of length depends on what technique is used to obtain the test sequence from the *reduced* BMT graph. Although we have found the optimal solutions for small circuits we considered, there are certainly examples for which our algorithm may fail to produce optimal solution. This is an inherent nature of heuristic algorithm.

Chapter 8

Summary and Future Work

8.1 Summary

The research reported in this dissertation is a comprehensive study of testing clockless digital circuits. The first part of this study is devoted to the transistor-level realizations of BiCMOS/CMOS logic families. Since the totem-pole family is used for most of the implementations of the digital circuits, its testing has been widely studied in this dissertation and several DFT techniques are developed for this family. In particular, we investigated various techniques which can result in on-line fault detection for this logic family. In the second part of the dissertation, test generation algorithms for gate-level realizations of asynchronous digital circuits are developed.

The first contribution of this dissertation is a DFT technique, presented in Chapter 4, for detecting short defects in this family. We have proposed an I_{DDQ}-testable design, which is also capable of detecting most other faults caused by short defects.

The second contribution of this dissertation is a novel circuit for detecting delay faults, proposed in Chapter 5. Defects manifest themselves first as delay faults. Since BiCMOS/CMOS circuits are normally designed where high-speed performance is of great concern, it is important to detect these faults as soon as they occur. Also, if some defects are not detected early, they may result in functional faults. Early detection of these defects by the proposed scheme is an important contribution since it guarantees the correct operation of the circuit.

The third contribution of this dissertation is the development of a novel on-line built-in self-test (BIST) technique for detecting stuck-open faults, discussed in Chapter 5.

We have shown that 100% stuck-at fault coverage is achieved if I_{DDQ} faults, delay faults, and stuck-open faults are detected.

All of the three DFT techniques proposed in Chapters 4 and 5 are capable of being employed for concurrent testing. The importance of on-line delay fault detection is that a digital circuit, which has passed the initial tests, is prone to some defects which may occur during the normal operation of the circuit. These schemes eliminate the requirement of test pattern generation.

Two of the three DFT techniques require a circuit for generating the control signals. In a synchronous sequential design, the sampling clock required for the delay and stuck-open tests can be generated from the master clock. In an asynchronous circuits, however, this control signal is generated using *request* and *acknowledge* signals.

In order to derive the fault characterizations reported in this study, and to evaluate the feasibility of the proposed design techniques, HSPICE simulations were performed. Device model parameters were taken from 0.8μ BiCMOS Design Kit

V0.3 for Cadence Analog Artist, developed by the Canadian Microelectronics Corporation. The device parameters used in our simulations are shown in Appendices A and B.

In order to use a realistic approach for modeling physical defects, defects are modeled as *shorts* and/or *opens* in the circuit level. Shorts are modeled as a small resistor between the two nodes. Open-circuits are modeled as a large resistor inserted between the affected node and the node to which it would normally have been connected. The open-circuit in the gate of an MOS transistor is treated as a floating point, and for simulation, the gate terminal is tied to the corresponding bulk terminal. The values of the resistors, modeling shorts and opens, were varied in a wide range to accommodate the effects of short and open defects of variable strengths. This approach results in more accurate modeling of physical defects, compared to the simple *stuck-on* and *stuck-open* models normally employed in the literature.

A new approach to testing sequential circuits has been proposed in the second half of this dissertation. Instead of studying the faulty circuits or machines, we have derived the complete test sequence using the specification of the fault-free circuit. This is a sound method since the fault-free specifications of digital circuits are provided by designer at the synthesis level. This approach is computationally superior to other proposed techniques since there is only one fault-free machine compared to numerous possible faulty machines. Our approach is more comprehensive than any convential techniques as it addresses both single and multiple faults without actually simulating or enumerating faults.

The fourth contribution of this dissertation is an initialization technique for asynchronous sequential circuits. This technique is always applicable, and has very little overhead.

The fifth contribution of this dissertation is testing asynchronous sequential circuits. Convential techniques have failed to be extended to these types of circuits due to the absence of the clock. We have proposed a Behavioral Model for Testability (BMT) graph which captures internal and external transitions. The correct circuit behavior is derived from the BMT graph, and redundant transitions are removed. Finally, pieces of test sequences are linked to form a complete test sequence. Techniques to devise *minimum* length complete test sequence are also discussed.

8.2 Future Directions

The following are recommendations for future work to further extend the testability of BiCMOS logic circuits.

1. The proposed techniques were applied to typical CMOS/BiCMOS logic gates. The results reported in this dissertation should be further expanded by applying them to commercial products, which employ several building blocks in a single chip. Such implementations will show the ultimate importance of the proposed circuits. Our experiments show that the fault coverage is relatively high for large circuits such as multiplier or decoder. We speculate that applying our DFT testing technique will evaluate the great increase in the testability of digital circuits. This will justify the minor delay/area/power overheads contributed by the proposed schemes.

2. The circuit proposed in Chapter 4 provides a means for detecting short defects. It would be interesting to extend this technique and provide completion signals in asynchronous circuits. One such important application of the current sensor proposed is Current Sensing Completion Detection (CSCD) [45,

73]. In such application, the sensor circuit should respond to any momentary transition at its virtual ground.

3. The length of the complete test sequence contributes to how fast the testing process takes place. A circuit which is tested by smaller test sequence than another circuit which performs the same task is considered to be better testable. Is it possible to use the BMT graph for designing circuits with improved testability?

4. We assumed thoughrout this dissertation that transistor- or gate-level implementations of digital circuits are irredundant. In [148] we have stated that redundant circuits are equally or less testable. The results reported in this dissertation should be further expanded by applying them to redundant circuits.

5. The theorems expressed in Chapter 7 point out that an asynchronous sequential circuit is easily tested if it is initializable to each of its stable states. On the other hand, the initialization technique in Chapter 6 provides simple design technique to initialize a sequential circuit. These Chapters together provide bases for designing asynchronous sequential circuit. Such design methodology should be further investigated.

6. No synchronization is assumed for inputs throughout this dissertation. Synchronous circuits are easier to test since there is a global clock which controls propagation of transitions. A clock is an input which changes periodically. This constraint will make testing more difficult since data is not allowed to march forward before the next clock comes in. The techniques proposed and the theorems stated should be studied for synchronous circuits as special case.

This may be desirable since most circuits designed at the present time are synchronous. More specifically, scan design techniques for synchronous circuits and addition of other control lines or observation points should be studied in light of our theorems. Furthermore, it would be interesting to know where these additional signals should be supplied and when the logic values should change to reduce the test length for synchronous circuits.

7. We have provided a test generation algorithms for gate-level circuits in the second half of this dissertation. It is desirable to extend these techniques to transistor-level circuits and to provide the complete test set or sequence for them. More specifically, how do we obtain test sets for delay faults and I_{ddq} faults using the BMT graph?

8. There is no computer-aided design tool for design and testing of asynchronous circuits at the present time. Our algorithms and the BMT graph provide bases for writing commercial software which benefit both industry and the academic community.

Given the encouraging results obtained from our simulations and algorithms, we are hopeful that future work will realize the full potential of these novel techniques for testing digital circuits.

Appendix A

HSPICE Device Model Parameters

The following NTE HSPICE device model parameters have been used in the circuit simulations. These parameters have been chosen from:
Canadian Microelectronics Corporation
BiCMOS Design Kit V0.3 for Cadence Analog Artist
January 16, 1995
0.8-micron BiCMOS HSPICE library

The device model parameters for the NPN bipolar transistor.

IS= 3.10E-18	BF= 1.03E+02	NF= 1.00E+00
AF= 1.18E+02	IKF= 5.36E-03	ISE= 4.57E-18
NE= 1.50E+0		
BR= 1.00E+00	NR= 1.00E+00	VAR= 5.50E+00
IKR= 0.00E+00	ISC= 0.00E+00	NC= 2.00E+00
RB= 5.76E+02	IRB= 0.00E+00	RBM= 5.76E+02
RE= 1.20E+01	RC= 5.94E+0	
CJE= 1.50E-14	VJE= 8.00E-01	MJE= 2.67E-01
TF= 1.10E-11	XTF= 7.36E+02	VTF= 1.31E+00
ITF= 9.38E-02	PTF= 2.00E+0	
CJC= 1.58E-14	VJC= 7.10E-01	MJC= 3.97E-01
XCJC= 5.10E-01	TR= 4.00E-0	
CJS= 3.44E-14	VJS= 3.70E-01	MJS= 1.34E-01
XTB= 1.62E+00	EG= 1.11E+00	XTI= 5.82E+00
KF= 4.27E-09	AF= 2.07E+00	FC= 8.00E-01

The device model parameters for the NMOSFET.

level=3	acm=2	
vto=0.7888	uo=502.9	tox=17.52e-9
nsub=3.231e16	nfs=819.9e9	delta=0.7279
theta=63.38e-3	wd=0.00	pb=0.9236
ld=59.55e-9	ldif=940.5e-9	xj=274.9e-9
vmax=150.3e3	eta=45.26e-3	kappa=6.71e-3
rs=595.6	rsh=0.00	hdif=1.2e-6
rd=595.6		
tlev=1	tcv=21.35e-6	bex=-2.183
tlevc=1	trs=-1.121e-3	trd=-1.121e-3
js=5.0e-4	jsw=5.5e-9	
cj=250.0e-6	mj=0.50	
cjsw=205.0e-12	mjsw=0.30	
cgso=273.9e-12	cgbo=570.9e-12	
cgdo=273.9e-12		
nlev=2	kf=600e-27	af=0.80

The device model parameters for the PMOSFET.

level=3	acm=2	
vto=-0.8838	uo=165.1	tox=17.52e-9
nsub=3.367e16	nfs=763.7e9	delta=0.3359
theta=135.0e-3	wd=0.00	pb=0.9210
ld=0.0	ldif=999.9e-9	xj=230.4e-9
vmax=189.7e3	eta=121.0e-3	kappa=1.449
rs=1189.0	rsh=0.00	hdif=1.2e-6
rd=1189.		
tlev=1	tcv=335.1e-6	bex=-1.292
tlevc=1	trs=3.109e-3	trd=3.109e-3
js=5.0e-4	jsw=5.5e-9	
cj=450.0e-6	mj=0.50	
cjsw=212.0e-12	mjsw=0.30	
cgso=214.8e-12	cgbo=570.9e-12	
cgdo=214.8e-12		
nlev=2	kf=500e-27	af=1.01

Appendix B

A Typical HSPICE Simulation

This appendix contains a typical HSPICE file, used for simulation and evaluation of the performance of the I_{DDQ} fault testing.

The file "JKFF.sp" is the input HSPICE file. It uses the subcircuit "cnot.sp", which is enclosed in this appendix, as well. The subcircuit is a CMOS inverter.

B.1 HSPICE files

JKFF.sp

```
*********************************************
*            input signals                  *
*********************************************


vdd dd gnd 5v
.global dd
.global cgnd


.tran .5n 70n UIC


.IC v(qout)=0 v(qnot)=5 v(11)=0 v(9)=5
.include 'cnand.sp'
.include 'cfaulty.sp'


vin1 J   0 pwl 0n 0, 9.9n 0, 10n 5, 33.9n 5, 34n 0, 45.9n 0, 46n 5,
+              57.9n 5, 58n 0, 70n 0
vin2 K   0 pwl 0n 0, 21.9n 0, 22n 5, 33.9n 5, 34n 0, 45.9n 0, 46n 5,
+              70n 5
vin3 clk 0 pwl 0n 0, 0.9n 0, 1n 5, 5.9n 5, 6n 0, 12.9n 0, 13n 5,
+              17.9n 5, 18n 0, 24.9n 0, 25n 5, 29.9n 5, 30n 0, 36.9n 0,

+              37n 5, 41.9n 5, 42n 0, 48.9n 0, 49n 5, 53.9n 5, 54n 0,
+              60.9n 0, 61n 5, 65.9n 5, 66n 0, 70n 0
```

```
******************************************************************************
.param lm=.8u
.param  wn1=18u
+               wn=18u
+               wp=9u
+       ap='wp*lm'
+       an='wn*lm'
+       an1='wn1*lm'
+       ppd='3*(lm+wp)'
+       pnd='3*(lm+wn)'
+       pnd1='3*(lm+wn1)'

******************** MOSFET transistors ***************
Xnand1 J clk 1 cnand
**********************************************************
mi1 not1 1 dd   dd mpch.0p8     w=wp l=lm ad=ap as=ap pd=ppd ps=ppd
mi2 not1 1 cgnd gnd mnch.0p8    w=wn l=lm ad=an as=an pd=pnd ps=pnd
**********************************************************
Xnand2 K clk 3 cnand
**********************************************************
mi3 not2 3 dd   dd mpch.0p8     w=wp l=lm ad=ap as=ap pd=ppd ps=ppd
mi4 not2 3 cgnd gnd mnch.0p8    w=wn l=lm ad=an as=an pd=pnd ps=pnd
**********************************************************
Xnand3 not1  qnot  5 cnand
Xnand4 not2  qout  7 cnand
**********************************************************
```

```
Xnand5 5  9  11 cnand
Xnand6 7  11  9 cnand
******************************************************
mi5 clkn clk dd   dd mpch.0p8     w=wp l=lm ad=ap as=ap pd=ppd ps=ppd
mi6 clkn clk cgnd gnd mnch.0p8    w=wn l=lm ad=an as=an pd=pnd ps=pnd
******************************************************
Xnand7  11  clkn  13 cnand
Xnand8  9   clkn  15 cnand
******************************************************

m1 qout  13 dd  dd MPCH.0P8     w=wp l=lm ad=ap as=ap pd=ppd ps=ppd
.param val=10meg val2=5
Rshort 4  qnot  val
m2 qout  qnot dd  dd MPCH.0P8     w=wp l=lm ad=ap as=ap pd=ppd ps=ppd
m3 qout  13  4   gnd MNCH.0P8   w=wn1 l=lm ad=an1 as=an1 pd=pnd1
ps=pnd1
m4 4   qnot cgnd gnd MNCH.0P8   w=wn1 l=lm ad=an1 as=an1 pd=pnd1
ps=pnd1

Xnand10 15  qout  qnot cnand
***************  Added circuit for Iddq DFT method ***********
vtin tin1 gnd val2
.param val=10meg val2=5
Rtin tin1 tin 15K
ctin tin1 tin .1p
Qmt1 cgnd tin gnd gnd NN51111X AREA=1 AREAB=2 AREAC=3
```

```
vtest cgnd cgnd2 0
mt2 tout cgnd2 gnd gnd MNCH.OP8  w=1u l=1u ad=an1 as=an1 pd=pnd1
ps=pnd1
mt3 tout cgnd2  dd  dd MPCH.OP8  w=1u l=1m ad=ap as=ap pd=ppd ps=ppd
mt4 cgnd2 tout gnd gnd MNCH.OP8  w=4u l=2u ad=an1 as=an1 pd=pnd1
ps=pnd1
mt5 cgnd2 tout  dd  dd MPCH.OP8  w=1u l=1m ad=ap as=ap pd=ppd ps=ppd
***************************************************************
.include 'hspice.model'
.option nomod acct post=2
.alter
.param val=10meg val2=0
.alter
.param val=10 val2=5
.alter
.param val=10 val2=0
.END
```

cnand.sp

```
* PUPD CMOS NAND gate (DFT)


.SUBCKT cnand  in1 in2 out
*********************************************
.param lm=.8u
.param  wn1=18u
+         wn=36u
+         wp=9u
+        ap='wp*lm'
+        an='wn*lm'
+        an1='wn1*lm'
+ ppd='3*(lm+wp)'
+ pnd='3*(lm+wn)'
+ pnd1='3*(lm+wn1)'
******************* MOSFET transistors **************
m1 out   in1 dd   dd MPCH.OP8     w=wp l=lm ad=ap as=ap pd=ppd ps=ppd
m2 out   in2 dd   dd MPCH.OP8     w=wp l=lm ad=ap as=ap pd=ppd ps=ppd
m3 out   in1  4    gnd MNCH.OP8   w=wn1 l=lm ad=an1 as=an1 pd=pnd1
ps=pnd1
m4 4     in2 cgnd gnd MNCH.OP8   w=wn1 l=lm ad=an1 as=an1 pd=pnd1
ps=pnd1
```

cfaulty.sp

```
* PUPD CMOS NAND gate (DFT)


.SUBCKT cfaulty  in1 in2 out
*********************************************
.param lm=.8u
.param  wn1=18u
+        wn=36u
+        wp=9u
+       ap='wp*lm'
+       an='wn*lm'
+       an1='wn1*lm'
+ ppd='3*(lm+wp)'
+ pnd='3*(lm+wn)'
+ pnd1='3*(lm+wn1)'
******************* MOSFET transistors **************
m1 out  in1 dd  dd MPCH.OP8     w=wp l=lm ad=ap as=ap pd=ppd ps=ppd
Rshort 4  dd  val
m2 out  in2 dd  dd MPCH.OP8     w=wp l=lm ad=ap as=ap pd=ppd ps=ppd
m3 out  in1  4   gnd MNCH.OP8   w=wn1 l=lm ad=an1 as=an1 pd=pnd1
ps=pnd1
m4 4    in2 cgnd gnd MNCH.OP8   w=wn1 l=lm ad=an1 as=an1 pd=pnd1
ps=pnd1

.ENDS cfaulty
```

cnot.sp

```
*  CMOS INVERTER


* input :  a1
* output : inv1


.SUBCKT cnot  A1  INV1
.SUBCKT inv in out
*************************************************
.param lm=.8u
.param  wn=2u
+ wp=3u
+       ap='wp*lm'
+       an='wn*lm'
+       pp='3*(lm+wp)'
+       pn='3*(lm+wn)'
*************************************************
m1 out in  dd  dd mpch   w=wp l=lm ad=ap as=ap pd=pp ps=pp
m2 out in gnd gnd mnch   w=wn l=lm ad=an as=an pd=pn ps=pn

.ENDS inv

X1 A1 INV1 inv

.ENDS cnot
```

Appendix C

Examples for the BMT graph

C.1 Majority C-element

Majority C-element has been discussed in Chapter 8 in details. Figure C.1 shows the fault locations and Table C.1 shows the test sequences to detect single stuck-at faults. The test sequences have been obtained using conventional techniques such as D-algorithm. Each test sequence detects the single stuck-at fault mentioned in the Table. However, such test sequence is not unique: There are other possible test sequences to detect the same single stuck-at fault. The complete test sequence (CTS) for majority C-element is of length 6 as mentioned in Chapter 8. Such CTS detects single and multiple stuck-at faults.

C.2 Wuu's C-element

Figure C.2(a) shows Wuu's C-element [192]. $\{z_1, z_2, z_3, z_4, z_5, z_6\}$ are variables assigned to the forks' outputs in order to find the feedback variable. z_5 is chosen as

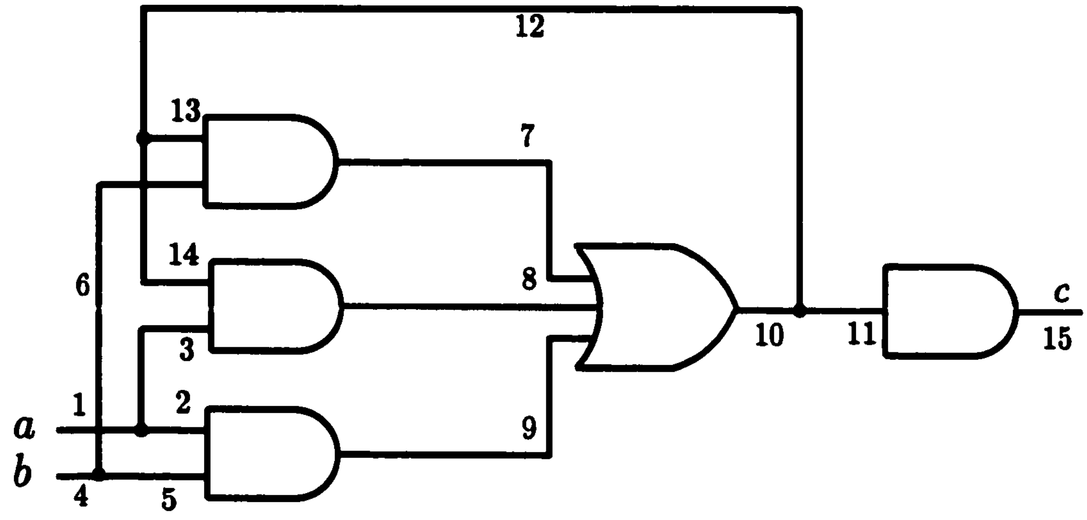


Figure C.1: *The stuck-at fault locations in majority C-element*

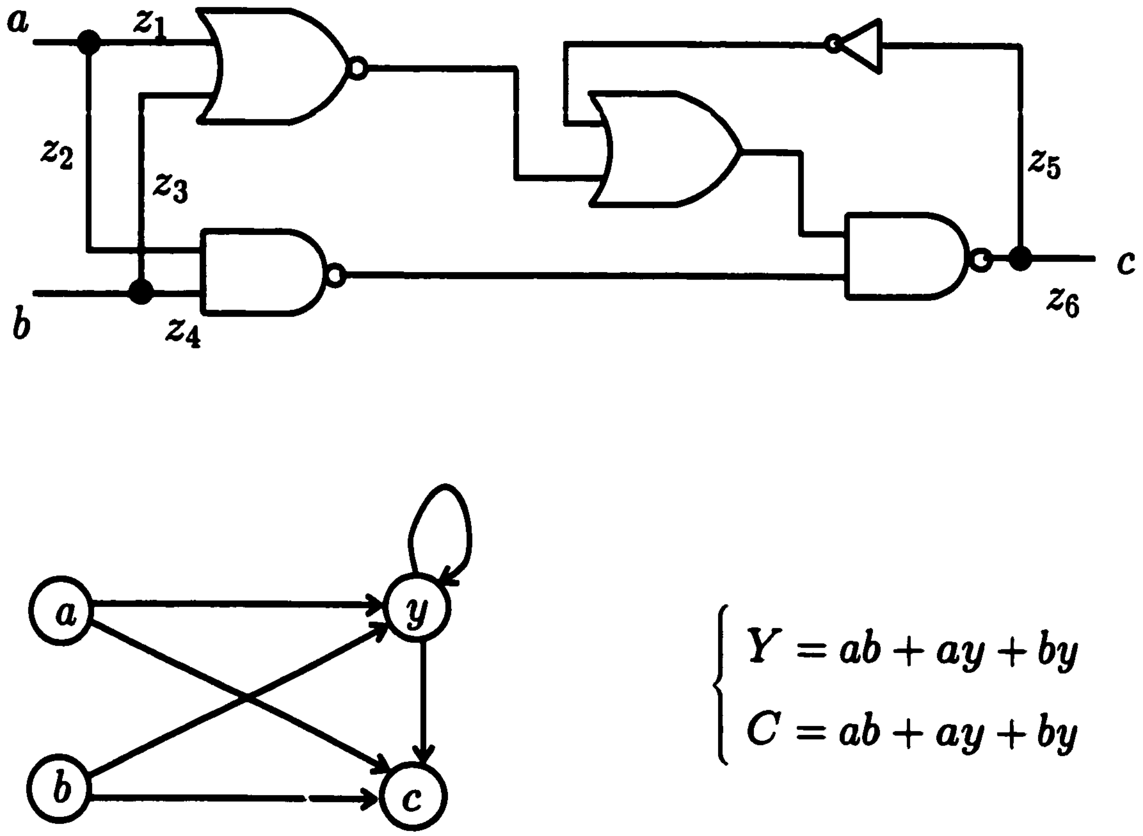
Equivalent single faults	Test(ab)
10-sa0, 11-sa0, 15-sa0	11
4-sa0, 12-sa0, 6-sa0, 7-sa0, 13-sa0	11, 01
1-sa0, 3-sa0, 8-sa0, 14-sa0	11, 10
4-sa1, 1-sa1, 3-sa1, 6-sa1	11, 00
7-sa1, 8-sa1, 9-sa1, 10-sa1, 11-sa1, 15-sa1	00
12-sa1, 2-sa1, 13-sa1	00, 01
5-sa1, 14-sa1	00, 10
2-sa0, 5-sa0, 9-sa0	00, 11

Table C.1: *Single stuck-at faults in the majority C-element and their test sequences.*

the feedback variable from Figure C.2(b), and is replaced by y in Figure C.2(c), which shows the network model for Wuu's C-element. The excitation functions for

$\{y, c\}$ are as follows.

$$\begin{cases} Y = ab + ay + by \\ C = ab + ay + by \end{cases}$$



$$\begin{cases} Y = ab + ay + by \\ C = ab + ay + by \end{cases}$$

Figure C.2: *Wuu's C-element: (a) Logic circuit, (b) Circuit graph, (c) Network model.*

The network model and the excitation functions for Wuu's C-element are the same as those of the majority C-element. This results the same BMT graph, concludes the same essential pieces of the complete test sequence, and shows the same faulty behaviors.

Figure C.3 shows the fault locations in Wuu's C-element, and Table C.2 shows the test sequences to detect single stuck-at faults.

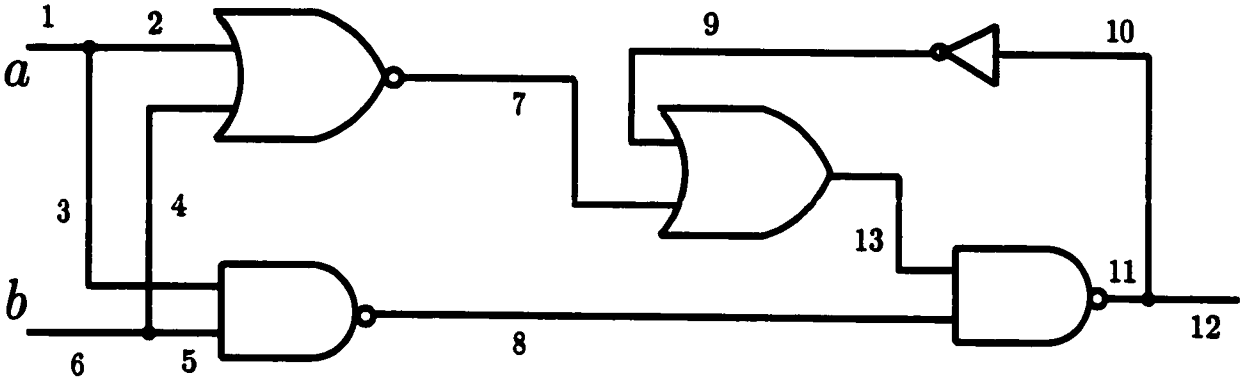


Figure C.3: *Fault locations in Wuu's C-element.*

Equivalent single faults	Test
11-sa0, 12-sa0, 14-sa0	11
6-sa0, 7-sa1, 9-sa1, 10-sa0, 13-sa1, 4-sa0	11, 01
1-sa0, 2-sa0	11, 10
6-sa1, 1-sa1, 2-sa1, 4-sa1, 7-sa0	11, 00
8-sa0, 11-sa1, 12-sa1, 13-sa0, 14-sa1	00
9-sa0, 10-sa1, 3-sa1	00, 01
5-sa1	00, 10
3-sa0, 5-sa0, 8-sa1	00, 11

Table C.2: *Single stuck-at faults in the Wuu's C-element and their test sequences.*

C.3 Mayevsky's C-element

Figure C.4(a) shows gate realization of Mayevsky's C-element [98,124]. $\{z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, z_{10}\}$ are assigned to the forks' outputs in order to find the minimal feedback vertex set. One can verify from Figure C.4(b) that any of $\{z_7\}$, $\{z_5, z_6\}$, or $\{z_6, z_9\}$ is a feedback variable set. Figure C.4(c) shows the network model with z_7 replaced by y. Note that the primary output variable, c, is different from z_8 or z_{10}. The excitation functions are as follows.

$$\begin{cases} Y = ab + ay + by \\ C = (a+b)y \end{cases}$$

The excitation function for C is different from that of the majority C-element. The BMT graph, therefore, shows a different behavior when $ab = 11$ is applied at $S_1 = 00$ or 01. This is shown by a dashed arrow in Figure C.4 (d).

The complete test sequence for Mayevsky's C-element is the same as that of the majority C-element since the number of inputs, feedback variables, and the logic functions are the same. As it was stated in Chapter 8, this test sequence detects all single and multiple stuck-at faults. It is tedious to examine all possible multiple faults and necessary test sequences. It is, however, easy to use the BMT graph and claim that the given test sequence is enough to detect all multiple faults as well.

The BMT graph for the Mayevsky's C-element at the presence of 2-sa0 is the same as shown in Figure 7.12. The excitation functions for the faulty circuit are:

$$\begin{cases} Y = ab + by \\ C = by \end{cases} .$$

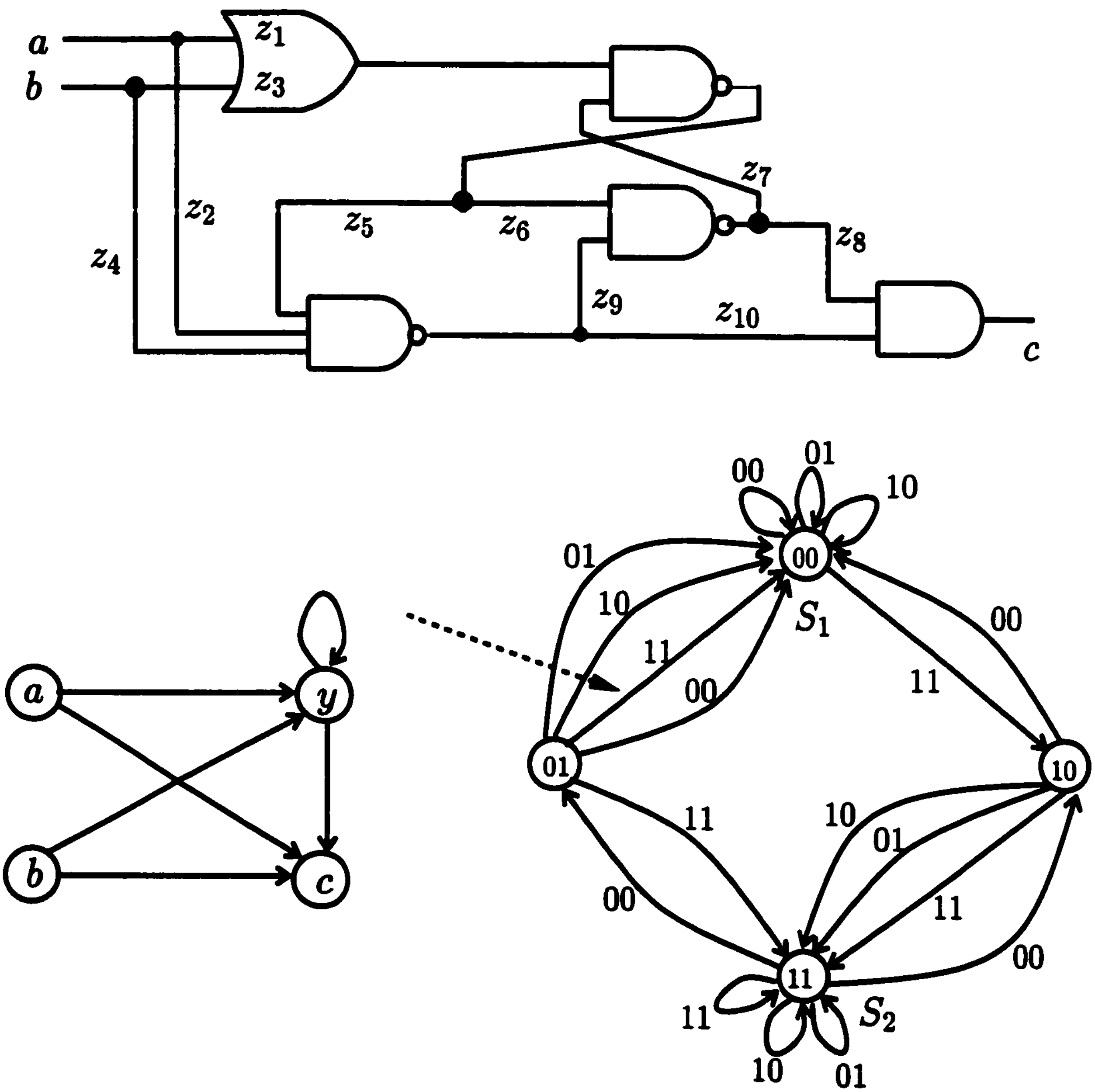


Figure C.4: *Mayevsky's C-element: (a) Gate circuit, (b) Circuit graph, (c) Network model.*

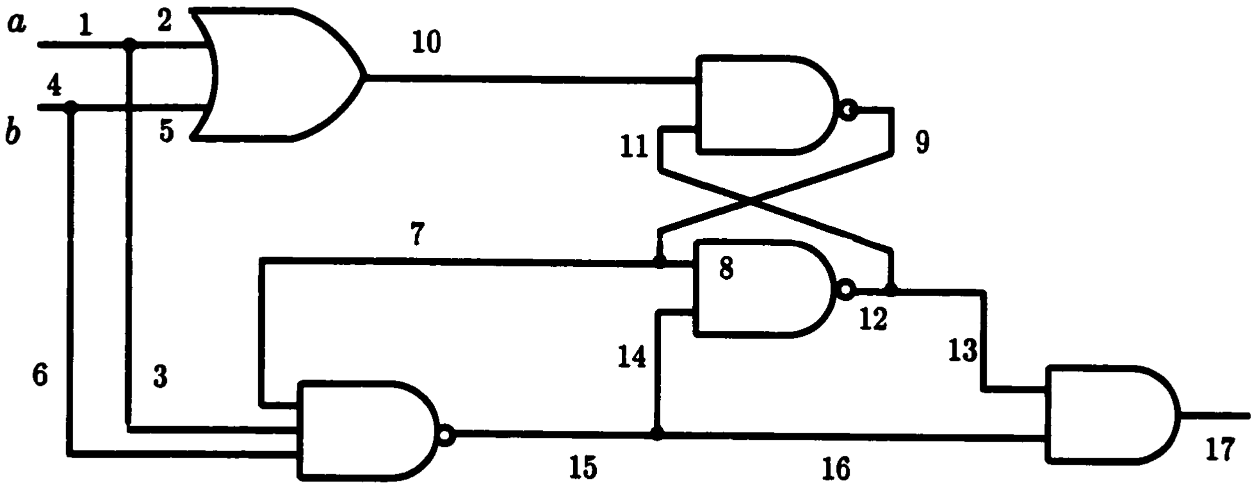


Figure C.5: *Fault locations in Mayevsky's C-element.*

Equivalent single faults	Test
9-sa1, 10-sa0, 11-sa0, 12-sa0, 13-sa0, 14-sa0, 15-sa0, 16-sa0, 17-sa0, 7-sa1	11
4-sa0, 5-sa0	11, 01
1-sa0, 2-sa0	11, 10
3-sa0, 6-sa0, 7-sa0, 14-sa1, 15-sa1	00, 11
8-sa0, 9-sa0, 12-sa1, 13-sa1, 17-sa1	00
4-sa1, 1-sa1, 2-sa1, 5-sa1, 10-sa1	11, 00
11-sa1, 3-sa1	00, 01
6-sa1	00, 10

Table C.3: *Single stuck-at faults in the Mayevsky's C-element and their test sequences.*

C.4 Bartkey's C-element

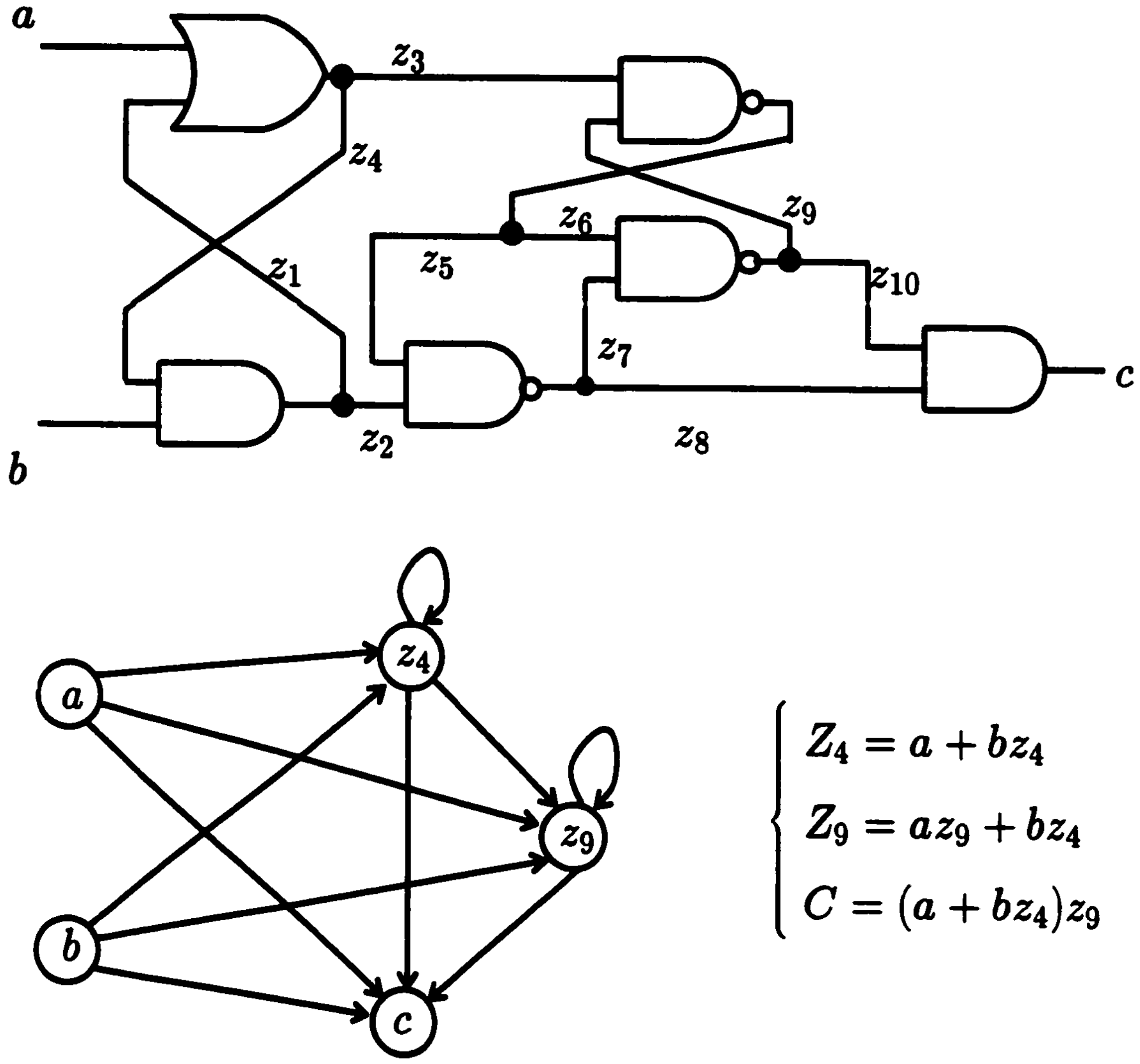
Figures C.6 (a) shows Bartkey's C-element with variables associated to the forks' outputs. Any set of $\{(z_1),(z_4)\} \times \{z_9,(z_6,z_5),(z_6,z_7)\}$ is a feedback variable set. Figure C.6 (b) shows the network model using $\{z_4,z_9\}$ as the minimal feedback set, and c as the primary output variable. Figure C.7 is presented to simplify the process of finding circuit equations. Boolean steps are as follows:

$$\begin{cases} z_8 &= \overline{bz_4.\overline{(a+bz_4)z_9}} \\ &= \overline{bz_4} + (a+bz_4)z_9 \\ &= \bar{b} + \overline{z_4} + (a+bz_4)z_9 \\ &= \bar{b} + \overline{z_4} + az_9 + bz_4z_9 \\ &= \bar{b} + \overline{z_4} + z_4z_9 + az_9 \\ &= \bar{b} + \overline{z_4} + z_9. \end{cases}$$

$$\begin{cases} Z_9 &= \overline{\overline{(a+bz_4)z_9}.(\bar{b}+\overline{z_4}+z_9)} \\ &= (a+bz_4)z_9 + bz_4\overline{z_9} \\ &= az_9 + bz_4 \end{cases}$$

$$\begin{cases} C &= (\bar{b}+\overline{z_4}+z_9)(az_9+bz_4) \\ &= \bar{b}az_9 + az_9\overline{z_4} + az_9 + bz_4z_9 \\ &= (a+bz_4)z_9 \end{cases}$$

The excitation functions are as follows.



$$\begin{cases} Z_4 = a + bz_4 \\ Z_9 = az_9 + bz_4 \\ C = (a + bz_4)z_9 \end{cases}$$

Figure C.6: *Bartkey's C-element (a) The gate circuit, (b) The circuit graph.*

$$\begin{cases} Z_4 = a + bz_4 \\ Z_9 = az_9 + bz_4 \\ C = (a + bz_4)z_9 \end{cases}$$

The BMT graph for the fault-free Bartkey's C-element is shown in Figure C.8.

The BMT graph for Bartkey's C-element is quite different from that of the Majority C-element. While the latter is described using only one feedback variable, the former has two feedback variables. Removing only redundant paths, the correct circuit behavior derived from the BMT graph is represented by the following rules.

$$\left\{\begin{array}{llll}
(1) & S_1 : 10 \Rightarrow S_3 & (5) & S_3 : 11 \rightarrow 10 \Rightarrow S_2 \\
(2) & S_3 : 00 \Rightarrow S_1 & (6) & S_1 : 11 \rightarrow 10 \Rightarrow S_2 \\
(3) & S_3 : 01 \Rightarrow S_2 & (7) & S_3 : 11 \rightarrow 01 \Rightarrow S_2 \\
(4) & S_2 : 00 \rightarrow 01 \Rightarrow S_1 & (8) & S_1 : 11 \rightarrow 01 \rightarrow 10 \Rightarrow S_2
\end{array}\right.$$

$$(9) \quad S_1 = 000, \quad S_2 = 111 \quad S_3 = 100$$

One complete test sequence is as follows.

$$00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 01 \rightarrow 10 \rightarrow 00$$

Figure C.9 shows the fault locations, and Table C.4 shows test sequences to detect the single stuck-at faults.

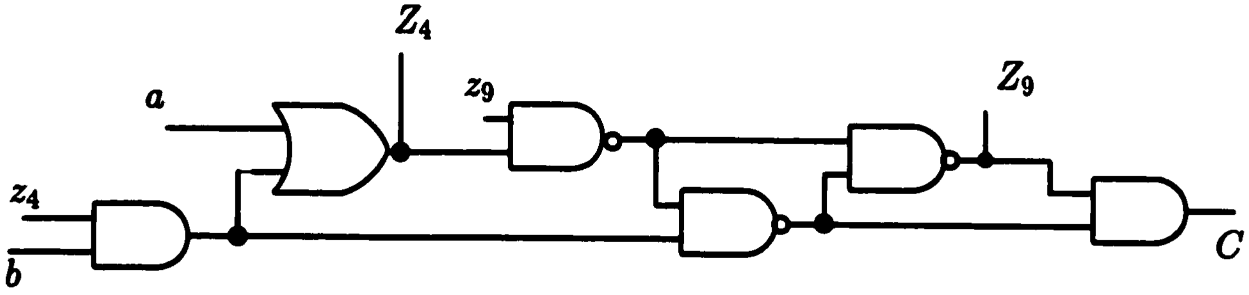


Figure C.7: *Bartkey's C-element with z_4, z_9 as feedback variables, and c as the primary output variable.*

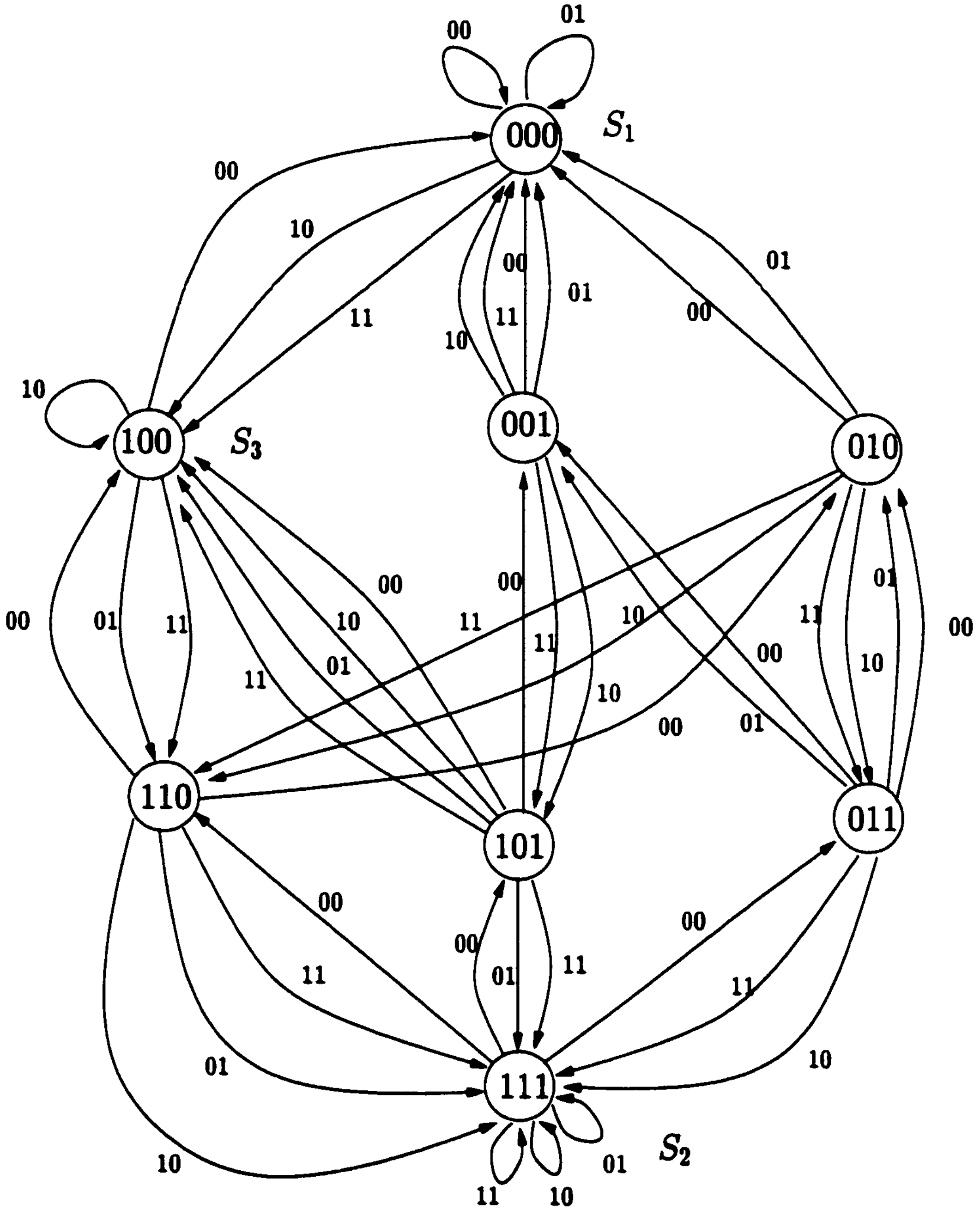


Figure C.8: *The BMT graph for Bartkey's C-element.*

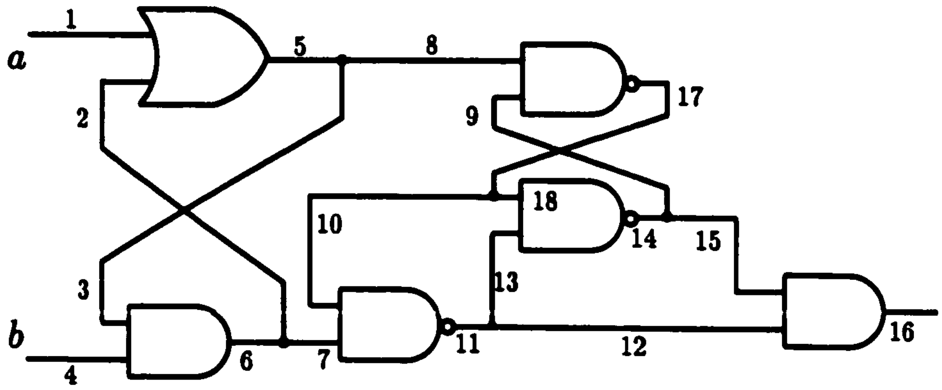


Figure C.9: *Fault locations in Bartkey's C-element.*

Equivalent single faults	Test
5-sa0, 8-sa0, 9-sa0, 18-sa1, 17-sa1, 14-sa0, 11-sa0, 12-sa0, 15-sa0, 16-sa0, 10-sa1	11
4-sa0, 3-sa0, 6-sa0, 2-sa0	11, 01
1-sa0, 2-sa1, 5-sa1	11, 10
7-sa0, 10-sa0, 13-sa1, 11-sa1	00, 11
6-sa1, 18-sa0, 17-sa0, 13-sa0, 14-sa1, 15-sa1, 16-sa1	00
4-sa1, 1-sa1, 8-sa1	11, 00
3-sa1	00, 01
7-sa1, 9-sa1	00, 10

Table C.4: *Single stuck-at faults in the Bartkey's C-element and their test sequences.*

C.5 NOR-Latch

Figure C.10 shows the gate circuit of a NOR-latch, its network model, and its BMT graph. The excitation functions are:

$$\begin{cases} Y = \bar{a}(b+y) \\ O = \bar{a}(b+y) \end{cases}.$$

The reduced BMT graph after removing redundant edges is shown in Figure C.10(d). Rules to describe the NOR-latch are given as follows.

$$\begin{cases} (1) \quad S_1 : 01 \rightarrow 00 \Rightarrow S_2 \\ (2) \quad S_2 : 11 \rightarrow 00 \Rightarrow S_1 \\ (3) \quad S_1 = 00, S_2 = 11 \end{cases}$$

The complete test sequence is:

$$11 \rightarrow 00 \rightarrow 01 \rightarrow 00$$

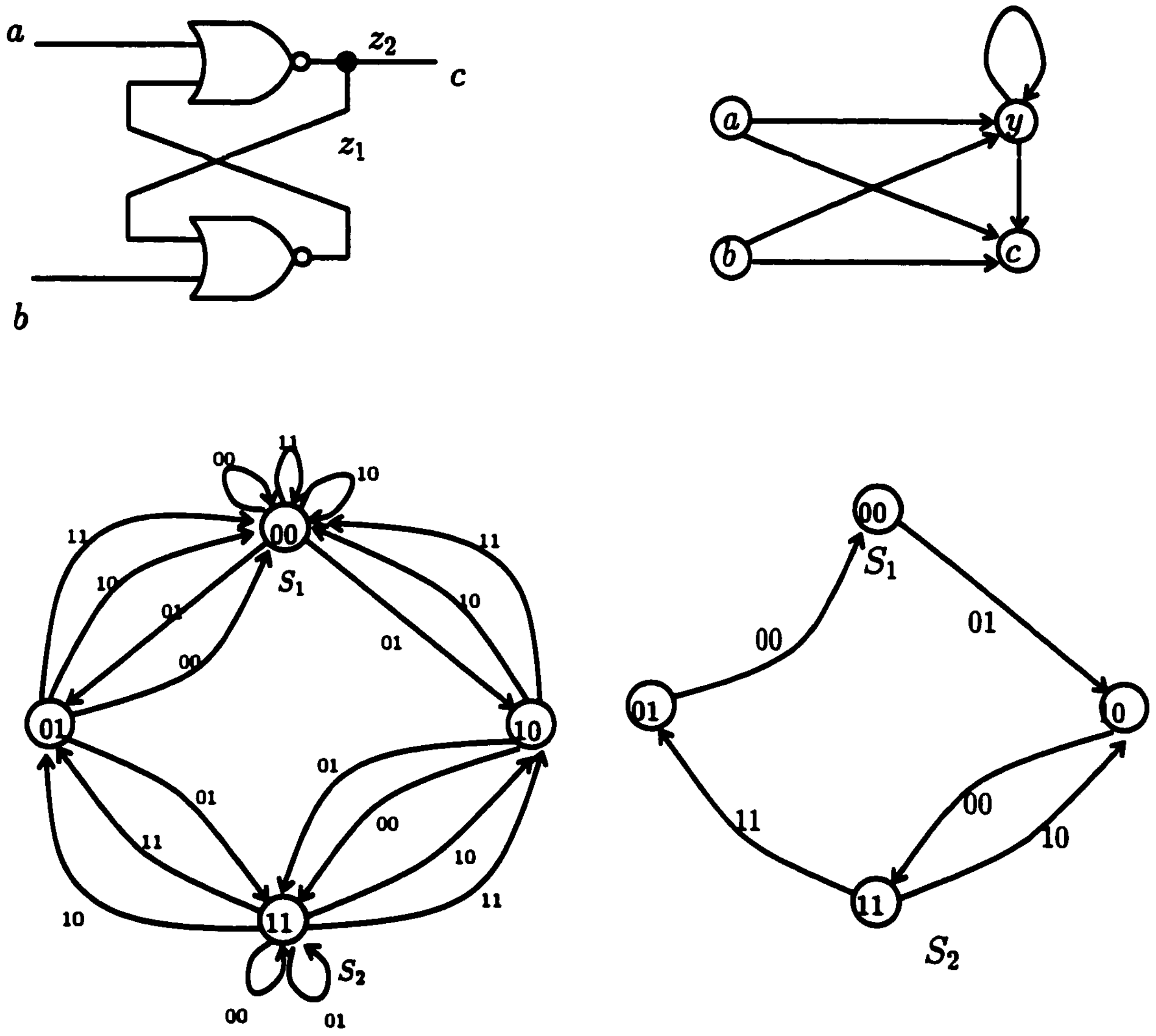


Figure C.10: *(a) NOR-latch logic circuit, (b) The network model, (c) The BMT graph, (d) The reduced BMT graph.*

C.6 CEL-NOR-latch

Figure C.11 shows a latch composed of NOR gate and C-element, its circuit graph, its network model, and its BMT graph. The excitation functions are given by:

$$\begin{cases} G_c = az_1 + z_1 g_c + ag_c \\ Z_1 = \overline{a + g_c} \\ Z_2 = \overline{a + g_c} \end{cases} .$$

The correct circuit behavior is as follows.

$$\begin{cases} (1) \quad S_1 : 1 \Rightarrow S_2 & (2) \quad S_1 : 1 \Rightarrow S_3 \\ (3) \quad S_2 : 0 \Rightarrow S_1 & (4) \quad S_3 : 0 \Rightarrow S_1 \\ (5) \quad S_1 = 011, S_2 = 000, S_3 = 100 & \end{cases}$$

The circuit is initializable to state 011 only. The complete test sequence is 0 → 1 → 0. Our analyses confirm that this test sequence detects all single and multiple stuck-at faults which are detectable.

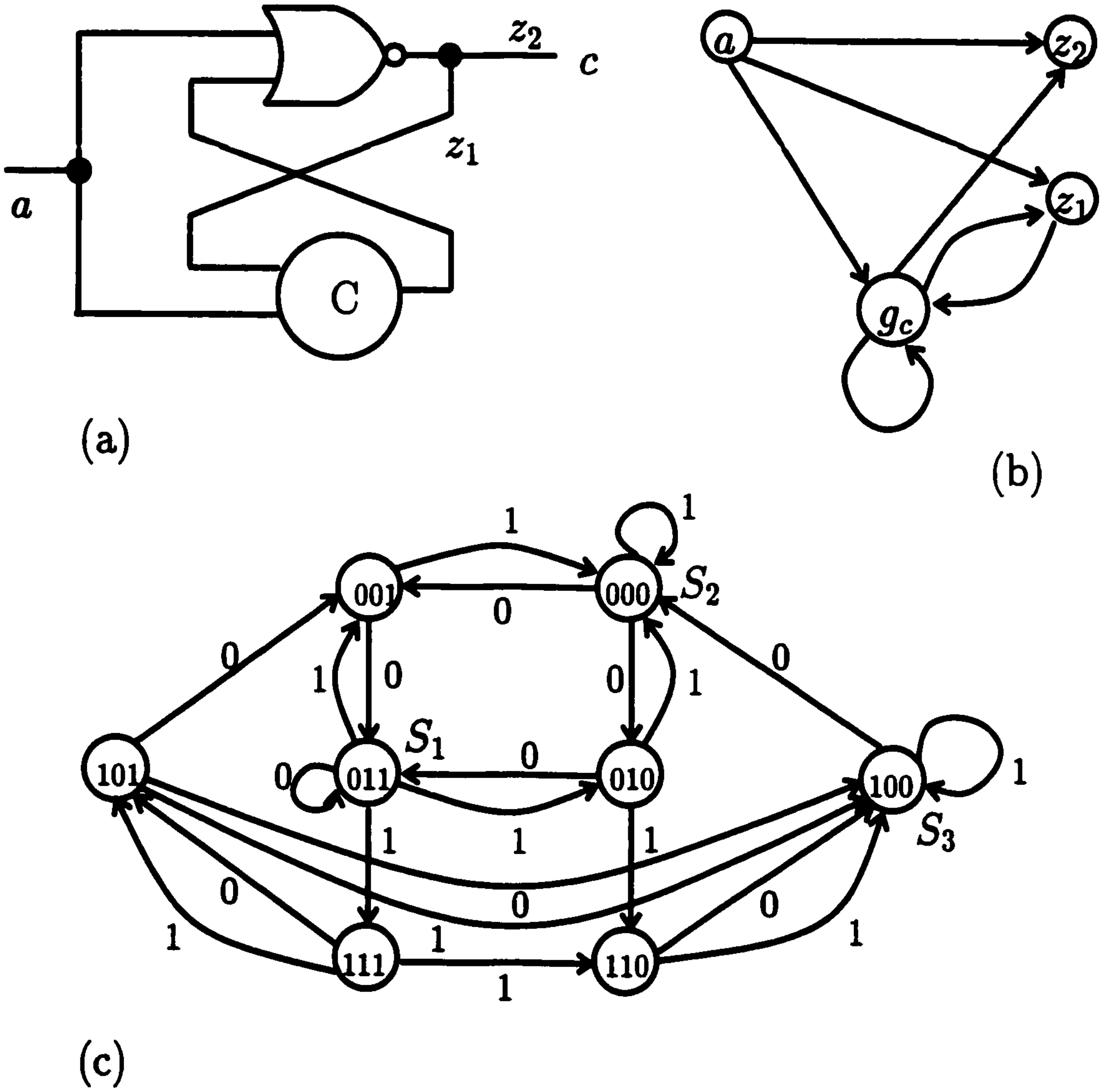


Figure C.11: *(a) The logic circuit, (b) The network model, (c) BMT graph.*

C.7 D Flip-Flop

Figure C.12 shows the gate circuit for a D flip-flop, its circuit graph, and the combinational circuit obtained by breaking $\{y_1 = z_1, y_2 = z_5, y_3 = z_4\}$. CLK is named C for convenience.

The excitation functions are as follows.

$$\begin{cases} Z = y_1 y_3 + \overline{C}, & Y_2 = (y_1 y_3 + \overline{C}) y_2 + \overline{D} \\ Y_1 = y_1 y_3 + \overline{C}, & Y_3 = (y_1 y_3 + \overline{C}) y_2 + \overline{D} \end{cases}$$

The BMT graph contained 16 states. It had too many edges. It was tedious to draw the BMT graph using conventional tools in the short time. I will present the PhD advisory committee with a hand copy of this graph. The correct circuit behavior is given by the following rules.

$$\begin{cases} (1) \quad S_1 : 01 \Rightarrow S_2 & (2) \quad S_1 : 10 \Rightarrow S_3 \\ (3) \quad S_1 : 00 \rightarrow 10 \Rightarrow S_4 & (4) \quad S_1 : 00 \rightarrow 01 \rightarrow 10 \Rightarrow S_4 \\ (5) \quad S_2 : 11 \Rightarrow S_1 & (6) \quad S_2 : 10 \Rightarrow S_3 \\ (7) \quad S_2 : 00 \rightarrow 01 \Rightarrow S_4 & (8) \quad S_2 : 00 \rightarrow 10 \Rightarrow S_4 \\ (9) \quad S_3 : 11 \Rightarrow S_1 & (10) \quad S_3 : 01 \Rightarrow S_4 \\ (11) \quad S_3 : 00 \rightarrow 01 \Rightarrow S_4 & (12) \quad S_3 : 00 \rightarrow 10 \Rightarrow S_4 \\ (13) \quad S_1 = 0000, S_2 = 1001, & S_3 = 0110, S_4 = 1111 \end{cases}$$

The circuit is not initializable if its state is $S_4 = 1111$. However, our technique provides the test sequence, and leaves initialization to the designer. One can use our initialization technique described in Chapter 7 and set the state to desired logical values.

One complete test sequence is as follows.

$$S_2 : 11 \rightarrow 10 \rightarrow 11 \rightarrow 00 \rightarrow 01 \rightarrow 10$$

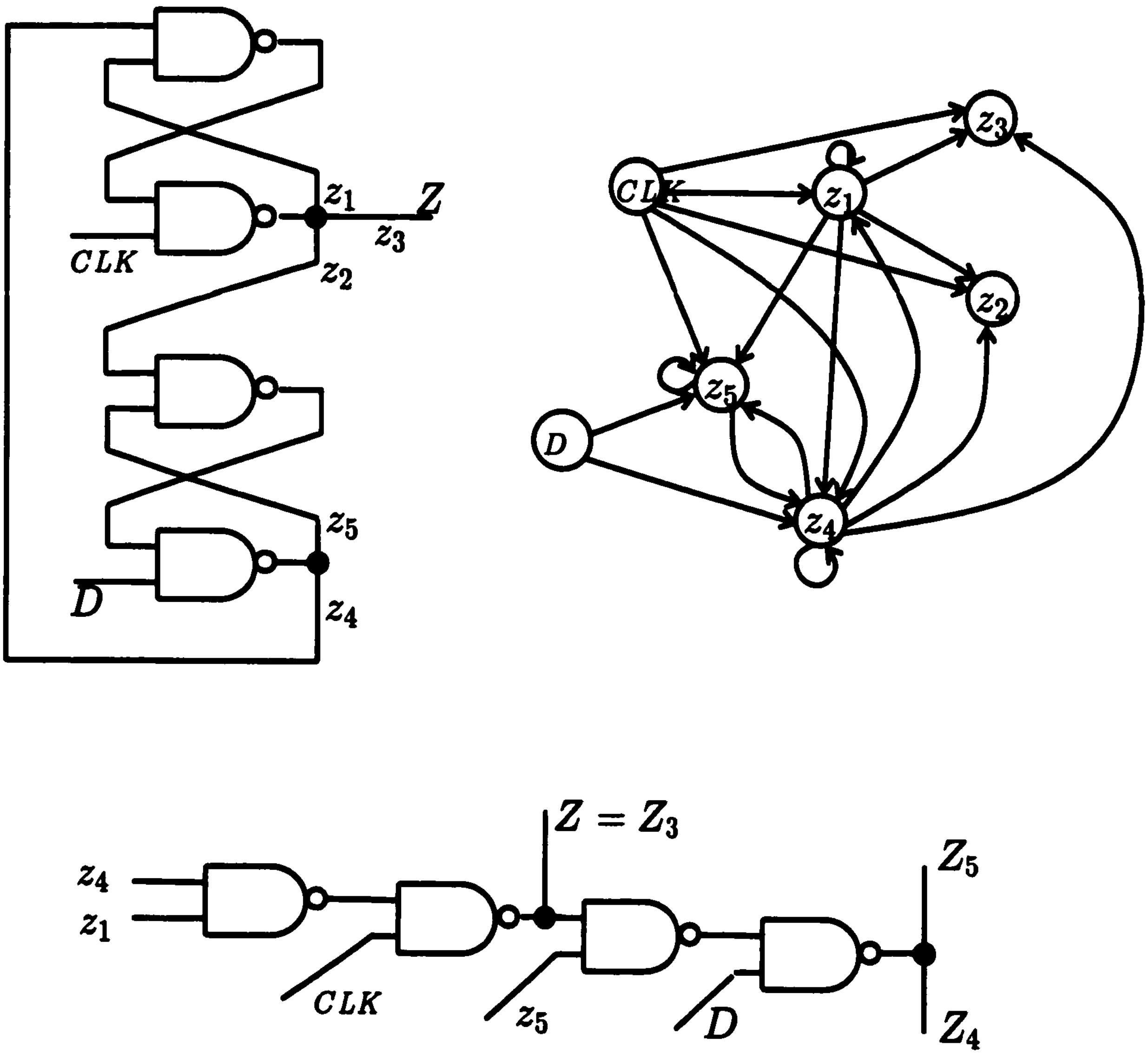


Figure C.12: *Partial D Flip-flop, (a) The gate circuit, (b) The circuit graph, (c) The gate circuit with broken feedback lines.*

Exhaustive test analysis shows that the mentioned test sequence detects single and multiple stuck-at faults.

Bibliography

[1] J. A. Abraham and V. K. Agrawal, *Test Generation for Digital Systems*. Prentice Hall, 1986. chapter 1 in Fault-Tolerant Computing, I.

[2] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*. Computer Science Press, 1990.

[3] M. Abramovici, P. R. Menon, and D. T. Miller, "SMART and FAST: Test Generation for VLSI Scan-Design Circuits," *IEEE Design & Test of Computers*, vol. 3, no. 4, pp. 43–54, August 1986.

[4] T. Agerwala, "Putting Petri Nets to Work," *Computer Magazine*, pp. 85-94, Dec. 1979.

[5] V. D. Agrawal and M. R. Mercer, "Testability Measures - What Do They Tell Us?," *Digest of Papers: International Test Conference*, pp. 391–396, Nov. 1982.

[6] V. D. Agrawal, C. R. Kime, and K. K. Saluja, "A Tutorial on Built-in Self-Test, Part 1: Principles," *IEEE Design and Test of Computers*, vol. 10, pp. 73–82, Mar. 1993.

[7] V. D. Agrawal, C. R. Kime, and K. K. Saluja, "A Tutorial on Built-in Self-Test, Part 2: Applications," *IEEE Design and Test of Computers*, vol. 10, pp. 69–77, June 1993.

[8] V. K. Agrawal and A. S. F. Fung, "Multiple Fault Testing of Large Circuits by Single Fault Test Sets," *IEEE Transactions on Computers*, vol. C-30, pp. 855–865, Nov. 1981.

[9] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[10] S. B. Akers. "Partitioning for testability," *Journal of Design Automation & Fault-Tolerant Computing*, vol. 1, Feb. 1977.

[11] A. R. Alvarez, *BiCMOS Technology and Applications*. Norwell, MA: Kluwer Academic Publishers, second ed., 1993.

[12] P. Ashar, S. Devadas, and A. R. Newton, "Irredundant Interacting Sequential Machines Via Optimal Logic Synthesis," *IEEE Transactions on CAD/ICAS*, vol. CAD-10, pp. 311-325, March 1991.

[13] S. Banerjee, S. T. Chakradhar, and R. K. Roy, "Synchronous Test Generation Model for Asynchronous Circuits," *9th International Conference on VLSI Design*, pp. 178–185, January 1996.

[14] K. Bartlett, R. K. Brayton, G. D. Hatchel, R. M. Jacoby, C. R. Morrison, R. L. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "Multi-Level Logic Minimization Using Implicit Don't Cares," *IEEE Transactions on CAD/ICAS*, vol. CAD-7, pp. 723-740, June 1988.

[15] P. A. Beerel and T. H. Y. Meng. "Semi-Modularity and Self-Diagnostic Asynchronous Control Circuits," In *Advanced Research in VLSI, Proceedings of the 1991 University of California/Santa Cruz Conference*, The MIT Press, pp. 118–132, 1991.

[16] P. A. Beerel and T. H. Y. Meng. "Testability of asynchronous self-timed control circuits with delay assumptions," In *Proceedings 28th ACM/IEEE Design Automation Conference*, IEEE Press, pp. 446-451, 1991.

[17] P. A. Beerel and T. H. Y. Meng. "Semi-modularity and Testability of Speed-Independent Circuits," *Integration, The VLSI Journal*, 13, pp. 301–322, 1992.

[18] K. V. Berkel, R. Burgess, J. Kessels, M. Roncken, and F. Schalij, "Characterization and Evaluation of a Compiled Asynchronous IC," *IFIP Asynchronous Design Methodologies*, pp. 209–221, 1993.

[19] K. V. Berkel, "Beware the Isochronic Fork," *INTEGRATION, the VLSI journal*, vol. 13, pp. 103–128, 1992.

[20] B. B. Bhattacharya and B. Gupta, "Logical Modeling of Physical Failures and Their Inherent Syndrome Testability in MOS LSI/VLSI Networks," *Proc. IEEE International Test Conference*, pp. 847–855, Oct. 1984.

[21] D. Brand, "Redundancy and Don't Cares in Logic Synthesis," *IEEE Transactions on Computers*, vol. C-32, pp. 947-952, Oct. 1983.

[22] M. A. Breuer and R. Harrison, "Procedures for Eliminating Static and Dynamic Hazards in Test Generation," *IEEE Transactions on Computers*, pp. 1069–1077, October 1974.

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

[24] M. A. Breuer, "New Concepts in Automated Testing of Digital Circuits," *Proc. EEC Symp. on CAD of Digital Electronic Circuits and Systems*, pp. 69–92, North-Holland Publishing Co., Nov. 1978.

[25] M. A. Breuer and A. D. Friedman, *Diagnosis & Reliable Design of Digital Systems.* Computer Science Press, 1989.

[26] F. Brglez and H. Fujiwara, "Neural Netlist of Ten Combinational Benchmark Circuits and a Target Translator in FORTRAN," *Proc. Int. Symp. on Circuits and Systems*, pp. 663-698, June 1985.

[27] F. Brglez, D. Brayan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *Proc. Int. Symp. on Circuits and Systems* pp. 1929-1934, May 1989.

[28] E. Brunvand and R. F. Sproull, "Translating Concurrent Programs into Delay-Insensitive Circuits," *International Conference on Computer-Aided Design, ICCAD-1989*, IEEE Computer Society Press, 1989.

[29] J. A. Brzozowski and J. C. Ebergen. *Recent Developments in the Design of Asynchronous Circuits.* In J. Csirik, J. Demetrovics, and F. Gécseg, editors, *Proceedings of Fundamentals of Computation Theory, Lecture Notes in Computer Science*, 380, Springer-Verlag, pp. 78–94, 1989.

[30] J. A. Brzozowski and J. C. Ebergen. "On the Delay-Sensitivity of Gate Networks," *IEEE Transactions on Computers*, C-41(11), pp. 1349-1360, November 1992.

[31] J. A. Brzozowski and H. Jürgensen. "A Model for sequential Machine Testing and Diagnosis," *Journal of Electronic Testing: Theory and Applications*, 3(3), pp. 219–234, 1992.

[32] S. M. Burns, A. J. Martin, "Syntax-Directed Translation of Concurrent Programs into Self-Timed Circuits," *Fifth MIT Conference on Advanced Research in VLSI*, 1988.

[33] K. M. Butler and M. R. Mercer, *Assessing Fault Model and Test Quality.* Kluwer Academic Publishers, 1992.

[34] G. R. Carson and G. Borriello. "A Testable CMOS Asynchronous Counter," *IEEE Journal of Solid-State Circuits*, 25(4), August 1990.

[35] C. W. Cha, W. E. Donath, and F. Ozguner, "9-V Algorithm for Test Pattern Generation of Combinational Digital Circuits," *IEEE Transactions on Computers*, vol. C-27, no. 3, pp. 193–200, March 1978.

[36] S. T. Chakradhar, S. Banerjee, R. K. Roy, and D. K. Pradhan, "Synthesis of initializable asynchronous circuits," *International Conference on Computer Design*, 1994.

[37] S. J. Chandra and J. H. Patel, "Experimental Evaluation of Testability Measures for Test Generation," *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 1, pp. 93–98, January 1989.

[38] M. S. Cheema and P. K. Lala, "Totally Self-Checking CMOS Circuit Design for Breaks and Stuck-on Faults," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 8, pp. 1203–1206, August 1992.

[39] T.-A. Chu, *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications.* Ph.D. Thesis, Dept. of EECS, MIT, May 1987.

[40] W. A. Clark, "Macromodular Computer Systems," *Proceedings of the Spring Joint Computer Conference*, AFIPS, Apr. 1967.

[41] W. A. Clark, C. E. Molnar "Macromodular Computer Systems," *Computers in Biomedical Research*, vol. IV, Academic Press, 1974.

[42] I. David, R. Ginosar, and M. Yoeli, "Self-Timed is Self-Diagnostic," *Journal of Electronic Testing: Theory and Applications*, vol. 6, pp. 219-228, 1995.

[43] P. Day and J. V. Woods, "Investigation into Micropipeline Latch Design Styles," *IEEE Transactions on VLSI Systems*, vol. 3, no. 2, pp. 264-272, June 1995.

[44] I. David, R. Ginosar, and M. Yoeli. *Self-Timed is Self-Diagnostic.* Technical Report UT 84112, Department of Computer Science, University of Utah, Salt Lake City, UT, 1990.

[45] M. E. Dean, D. L. Dill, and M. Horowitz, "Self-Timed Logic Using Current-Sensing Completion Detection (CSCD)," *Journal of VLSI Signal Processing*, vol. 7, pp. 7–16, 1994.

[46] S. Devadas and K. Keutzer, "Synthesis of Robust Delay-Fault-Testable Circuits: Practice," *IEEE Transactions on Computer-Aided Design*, vol. 11, pp. 277–300, Mar. 1992.

[47] S. Devadas, H.-K. T. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli, "Optimal Logic Synthesis and Testability: Two Faces of the Same Coin," *IEEE International Test Conference*, pp. 4–12, 1988.

[48] S. Devadas and K. Keutzer, "Synthesis of Robust Delay-Fault-Testable Circuits: Theory," *IEEE Transactions on Computer-Aided Design*, vol. 11, pp. 87–101, Jan. 1992.

[49] D. L. Dill, "Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits," *ACM Distinguished Dissertation, The MIT Press*, 1988.

[50] D. L. Dill, "Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits," *Fifth MIT Conference on Advanced Research in VLSI*, 1988.

[51] D. L. Dill, E. M. Clarke, "Automatic Verification of Asynchronous Circuits Using Temporal Logic," *Chapel Hill Conference on VLSI*, 1985.

[52] J. C. Ebergen, *Translating Programs into Delay-Insensitive Circuits*. Ph.D. Thesis, Eindhoven University of Technology, 1987

[53] J. C. Ebergen, M. G. Peeters, *Design and Analysis of Delay-Insensitive Module-N Counters*, in *Formal Methods in System Design*. Kluwer Academic Publishers, 1993.

[54] E. B. Eichelberger and T. W. Williams. "A Logic Design Structure for LSI Testability," *Journal of Design Automation & Fault-Tolerant Computing*, vol. 2, pp. 165-178, May 1978.

[55] E. B. Eichelberger *et al.*, *Structured Logic Testing*. Prentice Hall, 1991.

[56] R. D. Eldred, " Test Routines Based on Symbolic Logical Statements," *J. Assoc. Comput. Mach.*, vol. 6, no. 1, pp. 33–36, 1959.

[57] S. H. Embabi, A. Bellaouar, and M. I. Elmasry, *Digital BiCMOS Integrated Circuit Design*. Kluwer Academic Publishers, Norwell, MA, 1993.

[58] M. Favalli, P. Olivo, M. Damiani, and B. Ricco, "Novel Design for testability Schemes for CMOS IC's," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 1239–1245, October 1990.

[59] M. Favalli *et al.*, "Analysis of Steady State Detection of Resistive Bridging Faults in BiCMOS Digital ICs," *Proc. IEEE International Test Conference*, pp. 466–475, 1992.

[60] M. Favalli and C. Metra, "Sensing Circuit for On-line detection of delay faults," *IEEE Transactions on VLSI Systems*, vol. 4, no. 1, pp. 130–133, 1996.

[61] F. J. Ferguson and J. P. Shen, " Extraction and Simulation of Realistic CMOS Faults using Inductive Fault Analysis," *Proc. IEEE International Test Conference*, pp. 475–484, Sept. 1988.

[62] F. J. Ferguson, M. Taylor, and T. Larrabee, "Testing for Parametric Faults in CMOS Circuits," *Proc. IEEE International Test Conference*, pp. 436–443, 1990.

[63] R. J. Feugate Jr. and S. M. McIntyre, *Introduction to VLSI Testing*. Prentice Hall, 1988.

[64] P. Franco and E. J. McCluskey, " Delay Testing of Digital Circuits by Output Waveform Analysis," *Proc. IEEE International Test Conference*, pp. 798–807, 1991.

[65] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithm," *IEEE Transactions on Computers*, vol. C-32, no. 12, pp. 1137–1144, Dec. 1983.

[66] H. Fujiwara, "Computational Complexity of Controllability/Observability Problems for Combinational Circuits," *Proc. of the 18th Fault-Tolerant Computing Symposium*, pp. 64–69, 1988.

[67] H. Fujiwara, *Logic Testing and Design for Testability*. MIT Press, 1990.

[68] S. Funatsu, N. Wakatsuki, and T. Arima, " Test Generation Systems in Japan," *Proceedings on Design Automation Symposium*, pp. 114-122, June 1975.

[69] S. B. Furber, P. Day, J. D. Garside, N. C. Paver, and J. V. Woods, " A Micropipelined ARM," *Proceedings of the IFIP TC10/WG 10.5 International Conference on Very Large Scale Integration (VLSI'93)*, Grenoble, France, September 1993.

[70] S. B. Furber, P. Day, J. D. Garside, N. C. Paver, and J. V. Woods, " AMULET1: A Micropipelined ARM," *Proceedings on ComCon'94*, IEEE Computer Society Press, ComCon'94, San Francisco, March 1994.

[71] J. Galiay, Y. Crouzet, and M. Vergniault, "Physical Versus Logical Fault Model in MOS LSI Circuits: Impact on Their Testability," *IEEE Transactions on Computers*, vol. C-29, pp. 537–541, June 1980.

[72] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[73] E. Grass, S. Jones, "Asynchronous Circuits Based on Multiple Localized Current Sensing Completion Detection," *Second Working Conference on Asynchronous Design Methodologies*, London, England, May 30, 1995.

[74] P. Goel, "Test Generation Costs Analysis and Projections," *Proc. 17th Design Automation Conference*, pp. 77–84, Minneapolis, June 1980.

[75] P. Goel and B. C. Rosales, "PODEM-X: An Automatic Test Generation System for VLSI Logic Structures," *Proc. 18th Design Automation Conf.*, pp. 260–268, June 1981.

[76] L. H. Goldstein, "Controllability/Observability Analysis of Digital Circuits," *IEEE Transactions on Circuits and Systems*, vol. CAS-26, no. 9, pp. 685–693, Sept. 1979.

[77] B. Gupta, Y. K. Malaiya, *et al.*, "CMOS Combinational Circuit Design for Stuck-Open/Short Fault Testability," *Proc. Int. Symp. Elec. Dev. Circuits and Syst.*, pp. 789–791, Dec. 1987.

[78] D. S. Ha and S. M. Reddy, "On Testable Self-Timed logic Circuits," *Proceedings of International Conference on Computer Design (ICCD)* pp. 296–301, 1984.

[79] S. Hauck. "Asynchronous Design Methodologies: An Overview." Technical Report TR 93-05-07, Department of Computer Science and Engineering, University of Washington, Seatle, Washington, USA, 1993.

[80] J. P. Hayes. "On Modifying Logic Networks to Improve Their Diagnosability," *IEEE Transactions on Computers* , vol. C-23, pp.56-62, Jan. 1974.

[81] J. P. Hayes and A. D. Friedman. "Test Point Placement to Simplify Fault Detection," *International Symposium on Fault-Tolerant Computing*, June 1973, pp.73-78.

[82] P. J. Hazewindus. *Testing Delay-Insensitive Circuits*. PhD. thesis, California Institute of Technology, Pasadena, CA, 1992.

[83] C. A. Hoare, *Communicating Sequential Processes*. Prentice-Hall, 1985.

[84] D. A. Hodges and H. G. Jackson, *Analysis and Design of Digital Integrated Circuits*. McGraw-Hill Inc., second ed., 1988.

[85] D. A. Huffman, "The Design and Use of Hazard-Free Switching Networks," *Journal of the ACM*, vol. 41, no. 4, pp. 47-62, 1957.

[86] J. L. A. Hughes and E. J. McCluskey, "Multiple Stuck-at Fault Coverage of Single Stuck-at Fault Test Sets," *Proc. IEEE International Test Conference*, pp. 368–374, Sept. 1986.

[87] H. Hulgaard, S. M. Burns, and G Borriello. *Testing Asynchronous Circuits: A Survey.* Technical Report FR–35, Department of Computer Science, University of Washington, Seattle, WA, 1994.

[88] O. H. Ibarra and S. K. Sahni, "Polynomially Complete Fault Detection Problems," *IEEE Transactions on Computers*, vol. C-24, pp. 242–249, March 1975.

[89] A. Ivanov and V. K. Agarwal, "Dynamic Testability Measures for ATGP," *IEEE Transactions on Computer-Aided Design*, vol. 7, no. 5, pp. 598–608, May 1988.

[90] V. S. Iyengar and G. Vijayan, "Optimized Test Application Timing for AC Test," *IEEE Transactions on Computer-Aided Design*, vol. 11, no. 11, pp. 1439–1449, November 1992.

[91] M. Jamoussi, B. Kaminska, and D. Mukhedkar, "Testability of Iterative Arrays Using a Variable Testability Measure," *IEEE Transactions on Circuits nad Systems*, vol. 41, pp. 82–86, Jan. 1994.

[92] A. P. Jayasumana, Y. K. Malaiya, and R. Rajsuman, "Design of CMOS Circuits for Stuck-open Fault Testability," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 58–61, Jan. 1991.

[93] Y. Karkouri and E. M. Aboulhamid, "Complexite' du test des circuits logigues," *Technique et Science Informatiques*, vol. 9, pp. 273–287, April 1990.

[94] M. Katoozi and M. Soma. "A Testable CMOS Synchronous Counter," *IEEE Journal of Solid-State Circuits*, 5(23):1241-1248, October 1988.

[95] A. Khoche and E. Brunvand, "Testing Micropipelines," *Proceedings International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Salt Lake City, UT, IEEE Computer Society Press, pp. 239–246, 1994.

[96] A. Khoche and E. Brunvand, *A Partial Scan Methodology for Testing Self-Timed Circuits*. Technical Report UUCS-95-001, Department of Computer Science, University of Utah, Salt Lake City, UT, 1995. (See also these proceedings.)

[97] A. Khoche and E. Brunvand, "Testing Self-Timed Circuits using Partial Scan," *Second Working Conference on Asynchronous Design Methodologies*, London, England, May 1995, to appear.

[98] M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky. *Concurrent Hardware: The Theory and Practice of Self-Timed Design*. John Wiley & Sons, 1994.

[99] S. Kundu, S. M. Reddy, and N. K. Jha, "Design of Robustly Testable Combinational Logic Circuits," *IEEE Transactions on Computer-Aided Design*, vol. 10, pp. 1036–1048, Aug. 1991.

[100] M. Kuwako and T. Nanya. Timing-Reliability Evaluation of Asynchronous Circuits Based on Different Delay Models. In *Proceedings International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Salt Lake City, UT, IEEE Computer Society Press, pp. 22–31, 1994.

[101] Parag K. Lala. *Practical digital logic design and testing*. Englewood Cliffs, N.J. : Prentice Hall, 1996.

[102] Parag K. Lala. *Digital Circuit Testing and Testability*. Sandiago, CA; London: Academic Press, 1997.

[103] L. Lavagno, K. Keutzer, and A. L. Sangiovanni-Vincentelli, "Synthesis of Hazard-Free Asynchronous Circuits with Bounded Wire Delays," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 1, pp. 61–86, January 1995.

[104] M. Lanzoni, M. Favalli, M. Ambanelli, P. Olivo, and B. Ricco, "An experimental

study of testing techniques for bridging faults in CMOS IC's," *IEEE Journal of Solid-State Circuits*, vol. 28, no. 6, pp. 686–690, June 1993.

[105] C. H. Lau, "SELF: A Self-Timed Systems Design Technique," *Electronic Letters*, vol. 23, no. 6, pp. 269-270, March 1987.

[106] M. E. Levitt, K. Roy, and J. A. Abraham, " BiCMOS Fault Models: Is Stuck-at Adequate?," *Proc. IEEE International Conference on Computer Design*, pp. 294 – 297, 1990.

[107] S. L. Lu, "Improved design of CMOS multiple-input Muller C-element," *Electronic Letters*, vol. 29, no. 19, pp. 1680-1682, September 1993.

[108] S. L. Lu, "Implementation of Micropipelines in Enable/Disable CMOS Differential Logic," *IEEE Transactions on VLSI Systems*, vol. 3, no. 2, pp. 338-341, June 1995.

[109] D. L. Liu and E. J. McCluskey, "Designing CMOS Circuits for Switch Level Testability," *IEEE Design and Test*, vol. 4, pp. 42–49, Aug. 1987.

[110] S. C. Ma and E. J. McCluskey, " Non-Conventional Faults in BiCMOS Digital Circuits," *Proc. IEEE International Test Conference*, pp. 882 – 891, Oct. 1992.

[111] S. C. Ma and E. J. McCluskey, " Open Faults in BiCMOS Gates," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, pp. 567–575, May 1995.

[112] Y. K. Malaiya and R. Narayanswamy, "Testing for Timing Faults in Synchronous Sequential Integrated Circuits," *Proc. IEEE International Test Conference*, pp. 560–571, Oct. 1983.

[113] C. Maly, P. K. Nang, and P. Nigh, "Testing Oriented Analysis of CMOS ICs with Opens," *Proc. of International Conference on Computer-Aided Design*, pp. 344–347, 1988.

[114] W. Maly and S. Naik, "Defect and Design Error Diagnosability Measure," *Proceedings of the second European Test Conference, Munich*, pp. 83–90, Apr. 1991.

[115] W. Maly and M. Patyra, "Design of ICs Applying Built-In Current Testing," *Journal of Electronic Testing: Theory and Applications*, no.. 3, pp. 397–406, Kluwer Academic Publishers, Boston, 1992.

[116] M. M. Mano, *Digital Design*. Prentice-Hall, Inc., 1984.

[117] A. Marshall, B. Coates, and P. Siegel, "The Design of An Asynchronous Communications Chip," *IEEE Design & Test of Computers*, June 1994.

[118] A. J. Martin, "Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits," *Technical Report, Caltech CS-TR-89-23*, 1989.

[119] A. J. Martin, "The Limitations to Delay-Insensitivity in Asynchronous Circuits," *Sixth MIT Conference on Advanced Research in VLSI*, 1990.

[120] A. J. Martin and P. J. Hazewindus. "Testing delay-insensitive circuits," In *Advanced Research in VLSI, Proceedings of the 1991 University of California/Santa Cruz Conference*, The MIT Press, pp. 118–132, 1991.

[121] A. J. Martin, S. M. Burns, T. K. Lee, D. Borkovic, and P. J. Hazewindus, "Design of an Asynchronous Microprocessor," *Advanced Research in VLSI 1989: Proceedings of the Decennial Caltech Conference on VLSI*, MIT Press, pp. 351–373, 1989.

[122] A. J. Martin, *A Delay-Insensitive Fair Arbiter*. Technical Report, Caltech 5193:TR-85, 1985.

[123] A. J. Martin, "Compiling Communication Processes into Delay-Insensitive VLSI Circuits," *Distributed Computing*, vol. 4, no. 1, pp. 226-234, 1986.

[124] O. Mayevsky, V. Varshavsky, V. Marahovsky, and Yu. Mamrukov. USSR Patent Certificate N1081801, The Inventions Bulletin N 13, 1993.

[125] E. J. McCluskey. Fundamental Mode and Pulse Mode Sequential Circuits. In C. M. Popplewell, editor, *Proceedings of the IFIP Congress 62*, North-Holland Publishing Company, pp. 725–730, 1963.

[126] M. C. McFarland, A. C. Parker, and R. Camposano, "Tutorial on High-Level Synthesis," *25th ACM/IEEE Design Automation Conference*, pp. 330 – 336, 1988.

[127] T. H.-Y. Meng, *Asynchronous Design for Digital Signal Processing Architecture* . *Ph.D. Thesis*, Dept. of EECS, University of California at Berkeley, 1988.

[128] T. H.-Y. Meng, R. W. Brodersen, D. G. Messerschmitt, "Automatic Synthesis of Asynchronous Circuits from High-Level Specifications," *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 11, pp. 1185-1205, Nov. 1989.

[129] S. M. Menon, A. P. Jayasumana, and Y. K. Malaiya, "Test Generation for BiCMOS Circuits," *IEEE International Symposium on Circuits and Systems*, pp. 1987–1990, 1993.

[130] S. M. Menon, A. P. Jayasumana, and Y. K. Malaiya, "Testable Design for BiCMOS Stuck-open Fault Detection," *VLSI Test Symposium*, pp. 296–302, Apr. 1993.

[131] S. M. Menon, Y. K. Malaiya, and A. P. Jayasumana, "Behavior of Faulty Single BJT BiCMOS Logic Gates," *Proc. IEEE VLSI Test Symposium*, pp. 315–320, Apr. 1992.

[132] S. M. Menon, Y. K. Malaiya, and A. P. Jayasumana, "Bridging Faults in BiCMOS Circuits," *Proc. of the 5th NASA Symposium on VLSI Design*, pp. 7.1.1–7.1.10, Nov. 1993.

[133] R. E. Miller, "Sequential Circuits and Machines," in Switching Theory, Volume II, *John Wiley and Sons*, 1965.

[134] D. Misunas, "Petri Nets and Speed-Independent Design," *Communications of the ACM*, vol. 16, no. 8, pp. 474-481, Aug. 1973.

[135] C. E. Molnar, T-P. Fang, and F. U. Rosenberg, " Synthesis of Delay-Insensitive Modules," *Chapel Hill Conference on VLSI*, pp. 67 – 86, 1985.

[136] D. E. Muller, W. S. Bartky, "A Theory of Asynchronous Circuits," in The Proceedings of an International Symposium on the Theory of Switching, Part I, pp. 204-243, Harvard University Press, 1959.

[137] C. Myers and T. H. Y. Meng, "Synthesis of Timed Asynchronous Circuits," *ICCD*, 1992.

[138] P. Nigh, W. Needham, K. Butler, P. Maxwell, and R. Aitken, "An Experimental Study Comparing the Relative Effectiveness of Functional, Scan, Iddq, and Delay-Fault Testing," *Proceedings of the IEEE VLSI Test Symposium*, pp. 459–464, 1997.

[139] S. M. Nowick and D. L. Dill, "Automatic Synthesis of Locally-Clocked Asynchronous State Machines," *ICCAD*, 1991.

[140] S. M. Nowick and D. L. Dill, "Synthesis of Asynchronous State Machines Using a Local Clock," *ICCD*, 1991.

[141] S. M. Nowick, M. E. Dean, D. L. Dill, and M. Horowitz, "The Design of a high-Performance Cache Controller: A Case Study in Asynchronous Synthesis," *Hawaii International Conference on Systems Science*, 1993.

[142] S. M. Nowick, K. Y. Yun, and D. L. Dill, "Practical Asynchronous Controller Design," *ICCD*, October, 1992.

[143] S. M. Nowick and D. L. Dill, "Exact Two-Level Minimization of Hazard-Free Logic with Multiple-Input Changes," *ICCAD*, November, 1992.

[144] M. Y. Osman and M. I. Elmasry, "Highly Testable Design of BiCMOS Logic Circuits," *IEEE Journal of Solid-State Circuits*, vol. 29, pp. 671–678, June 1994.

[145] J. L. Peterson, "Petri Nets," *Computing Surveys*, vol. 9, no. 3, pp. 223-252, Sep. 1977.

[146] T. J. Powell, J. R. Pair, and B. G. Carbajal, "Correlating Defects to Functional and Iddq Tests," *Proc. IEEE International Test Conference*, pp. 501–510, 1996

[147] D. K. Pradhan, editor. *Fault-Tolerant Computing: Theory and Techniques*, vol. 1. Prentice Hall, 1986.

[148] K. Raahemifar. *Testing Asynchronous Logic Circuits from Transistor Networks to Gate-Level Designs*. Technical report, Electrical and Computer Engineering Department, University of Windsor, Windsor, Ontario, Canada, 1999.

[149] K. Raahemifar, S. Hessabi, and M. I. Elmasry, "A Design-for-Testability Technique for Shorts and Bridging Faults in BiCMOS Logic Families," in *1995 Canadian Conference on Electrical and Computer Engineering*, vol. 1, pp. 221–224, Montreal, Canada, Sep. 5–8, 95.

[150] R. Rajsuman, A. P. Jayasumana, and Y. K. Malaiya, "CMOS Open-Fault Detection in the Presence of Glitches and Timing Skews," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 1055–1061, Aug. 1989.

[151] R. Ramachandran and S. L. Lu, "Efficient Arithmetic Using Self-Timing," *IEEE Transactions on VLSI Systems*, vol. 4, no. 4, pp. 445–454, Dec. 1996.

[152] D. Rana, S. P. Levitan, D. A. Carlson, and C. E. Hutchinson, "A Testable Asynchronous Systolic Array Implementation of an IIR Filter," *Proceedings of the IEEE 1986 Custom Integrated Circuits Conference*, pp. 90–93, May 1986.

[153] I. M. Ratiu, A. Sangiovanni-Vincentelli, and D. O. Pederson, "VICTOR: A Fast VLSI Testability Analysis Program," *Digest of Papers: International Test Conference*, pp. 397–401, Nov. 1982.

[154] S. M. Reddy and M. K. Reddy, "Testable Realizations for FET Stuck-open Faults in CMOS Combinational Logic Circuits," *IEEE Transactions on Computers*, vol. C-35, pp. 742–754, Aug. 1986.

[155] A. W. Righter, J. M. Soden, and R. W. Beegle, "High Resolution Iddq Characterization and Testing – Practical Issues," *Proc. IEEE International Test Conference*, pp. 259–268, 1996.

[156] M. Roncken and R. Saejis, "Linear Test Times for Delay-Insensitive Circuits: a Compilation Strategy," *IFIP Asynchronous Design Methodologies*, pp. 13–27, 1993.

[157] M. Roncken, E. Aarts, and W. Verhaegh, "Optimal Scan Pipelined Testing: An Asynchronous Foundation," *International Test Conference*, pp. 215–224, 1996.

[158] M. Roncken, "Partial Scan Test for Asynchronous Circuits Illustrated on a DCC Error Corrector," *Proceedings International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Salt Lake City, UT, IEEE Computer Society Press, pp 247–256, November 1994.

[159] F. U. Rosenberger, C. Molnar, T. J. Chaney, T.-P. Fang, "Q-Modules: Internally Clocked Delay-Insensitive Modules," *IEEE Transactions on Computers*, vol. 37, no. 9, pp. 1005-1018, Sep. 1988.

[160] K. Roy, M. E. Levitt, and J. A. Abraham, " Test Considerations for BiCMOS Logic Families," *Proc. IEEE Custom Integrated Circuit Conference*, pp. 17.2.1 – 17.2.4, 1991.

[161] J. P. Roth, W. G. Bouricious, and P. R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Transactions on Electronic Computers*, EC–16, no. 5, pp. 567–579, 1967.

[162] R. A. Rutman, "Fault Detection Test Generation for Sequential Logic by Heuristic Tree Search," *IEEE Computer Group Repository*, Paper no. R-72-187, 1972.

[163] A. E. Salama and M. I. Elmasry, " Fault Characterization, Testing Considerations, and Design for Testability of BiCMOS Logic Circuits," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 944–947, June 1992.

[164] J. Savir and W. H. McAnney, "Random Pattern Testability of Delay Faults," *IEEE Transactions on Computers*, vol. 37, pp. 291–300, Mar. 1988.

[165] C. L. Seitz, *System Timing.* Chapter 7, *Introduction to VLSI Systems*, C. A. Mead and L. A. Conway, Eds. Addison-Wesley, 1980.

[166] J. A. Segura *et al.*, "Quiescent Current Analysis and Experimentation of Defective CMOS Circuits," *Journal of Electronic Testing: Theory and Applications*, vol. 3, pp. 337–348, 1992.

[167] J. P. Shen and F. J. Ferguson, "The Design of Easily Testable VLSI Array Multipliers," *IEEE Transactions on Computers*, vol. C-33, pp. 554–560, June 1984.

[168] J. P. Shen, W. Maly, and F. J. Ferguson, "Inductive Fault Analysis of MOS Integrated Circuits," *IEEE Design and Test of Computers*, vol. 2, pp. 13–26, Dec. 1985.

[169] G. L. Smith, "Model for Delay faults Based upon Paths," *Proc. IEEE International Test Conference*, pp. 342–349, Nov. 1985.

[170] J. Snepscheut, "Trace Theory and VLSI Design," LNCS 200, *Springer-Verlang*, 1985.

[171] R. F. Sproull, I. E. Sutherland, and C. E. Molnar, "Counterflow Pipeline Processor Architecture," to appear in *IEEE Design and Test of Computers*, 1995.

[172] B. E. Stewart, D. Al-Khalili, and C. Rozon, " Defect Modeling and Testability Analysis of BiCMOS Circuits," *Canadian J. Elect. & Comp. Eng.*, vol. 16, no. 4, pp. 148 – 153, 1991.

[173] A. K. Susskind, "A technique for making asynchronous sequential circuits readily testable," *IEEE International Test Conference*, pp. 842-846, 1984.

[174] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720-738, June 1989.

[175] I. E. Sutherland and R. F. Sproull, "Logical Effort: Designing for Speed on the Back of an Envelope," *Advanced Research in VLSI*, pp. 1–16, UC Santa Cruz, 1991.

[176] Y. K. Tan, Y. C. Lim, "Self-Timed System Design Technique," *Electronic Letters*, vol. 26, no. 5, pp. 284-285, March 1990.

[177] T. Nanya, Y. Ueno, H. Kagotani, M. Kuwako, and A. Takamura, "TITAC: Design of a Quasi-Delay-Insensitive Microprocessor," *IEEE Design and Test of Computers*, vol. 11, no. 2, pp. 50-73, 1994.

[178] C. Timoc *et al.*, "Logical Models of Physical Failures," *Proc. IEEE International Test Conference*, pp. 546–553, Oct. 1983.

[179] J. T. Udding, *Classification and Composition of Delay-Insensitive Circuits*. Ph.D. Thesis, Eindhoven University of Technology, 1984.

[180] S. H. Unger, *Asynchronous Sequential Switching Circuits*. John Wiley & Sons Inc., 1969.

[181] S. H. Unger, "Asynchronous Sequential Switching Circuits with Unrestricted Input Changes," *IEEE Transactions on Computers*, vol. C-20, No. 12, December 1971.

[182] V. I. Varshavsky, M. A. Kishinevsky, V. B. Marakhovsky, V. A. Peschansky, and L. Y. Rosenblum. *Self-Timed Control of Concurrent Processes*. Kluwer Academic Publishers, Russian edition, 1986. English edition, 1990.

[183] S. Paggy, G. Venkatesh, and S. Sherlekar, "Issues in Fault Modeling and Testing of Micropipelines," *First Asian Test Symposium*, Hiroshima, Japan, November 1992.

[184] J. A. Wehbeh and D. G. Saab, "On the Initialization of Sequential Circuits," *Proc. IEEE International Test Conference*, pp. 233–239, 1994.

[185] N. H. E. Weste and K. Eshraghian, *Principles of CMOS Design: A Systems Perspective, second edition.* Addison-Wesley Publishing Company, 1994.

[186] C. L. Wey, M. D. Shieh, and P. D. Fisher. "ASLCScan: A Scan Design Technique for Asynchronous Sequential Logic Circuits," *IEEE International Test Conference*, pp. 159–162, 1993.

[187] R. V. D. Wiel, "High-Level Test Evaluation of Asynchronous Circuits," *Second Working Conference on Asynchronous Design Methodologies*, London, England, May 1995, to appear.

[188] S. R. Whitaker and G. K. Maki, "Pass-Transistor Asynchronous Sequential Circuits," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 1, February 1989.

[189] T. W. Williams, and K. P. Parker. "Design for Testability-A Survey," *IEEE Transactions on Computers*, vol. C-31, No. 1, pp. 2–15, January 1982.

[190] T. W. Williams and K. P. Parker. "Testing Logic Networks and Design for Testability," *Computer*, pp.9-21, Oct. 1979.

[191] M. J. Y. Williams and J. B. Angell. "Enhancing Testability of Large Scale Integrated Circuits via Test Points and Additional Logic," *IEEE Transactions on Computers*, vol. C-22, pp.46-60, Jan. 1973.

[192] T. Y. Wuu and S. B. K. Vrudhula. A design of a fast and area efficient multi-input Muller C-element. *IEEE Transaction on VLSI systems*, 1(2), pp. 215–219, June 1993.

[193] A. Yakaviev, "Designing Self-Timed Systems," *VLSI Systems Design*, pp. 70-86, Sep. 1985.

[194] K. Y. Yun, D. L. Dill, and S. M. Nowick, "Practical Generalization of Asynchronous State Machines," *EDAC*, February, 1993.