Study on the Reliability Enhancement of Edge Computing Devices

A Thesis

Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in Engineering

By

Xihong Zhou

Supervisor: Hiroshi Takahashi (Full Professor)

Graduate School of Science and Engineering

Ehime University

Abstract

The recent rapid evolution of Internet of Things (IoT) and Artificial Intelligence (AI) technologies has accelerated the emergence of a smart society where every object can interface with the internet. This connectivity facilitates real-time data collection and analysis. One of the pivotal challenges within this context is maintaining the integrity and reliability of edge devices—key components of IoT systems operating within a 5G environment. The accumulation of inaccurate data from compromised edge devices may incur erroneous decision-making, thus compromising the overall system reliability and deteriorating trust in the smart society.

This study addresses the reliability issues of two distinct types of edge devices: electronic control units (ECUs) deployed within advanced driving-assistant systems (ADAS), and an innovative memory-based programmable logic device (MPLD) specifically engineered for executing AI functionality in IoT systems.

The initial objective of this study is to enhance the reliability of ECU devices, instrumental in automotive control systems. Adherence to functional safety standards, specifically ISO 26262, is a mandatory requirement for these devices. Under the functional safety standard ISO 26262, automotive systems necessitate in-field testing, such as the power-on self-test (POST). The POST examines safety-critical components during the system's startup, prior to executing any functional operations. This test is vital for the early detection of potential internal faults to prevent system failures. Nevertheless, for testing automotive ECUs, the POST requires minimal test application time to achieve essential test quality (e.g., >90% latent fault metric) to meet the functional safety criteria of ISO 26262. This research proposes a performance-enhanced POST, specifically the Multi-Cycle Power-on Self-Test, which applies multiple test clocks to execute numerous function operations after a root test pattern is set into the Circuit Under Test (CUT). To address fault-masking and fault detection degradation under multi-cycle testing, this study presents a test point insertion technique to reduce test application time while maintaining superior fault detection for multi-cycle POST. Moreover, a method is devised to identify a user-specified number of test points capable of achieving the greatest scan-in pattern reduction to attain a target test coverage.

The secondary objective concentrates on enhancing the reliability of MPLD edge devices. These are particularly designed for low power consumption and low latency in edge computing applications. This study explores two fundamental aspects of MPLD

reliability: interconnect defect testing during the manufacturing phase and aging monitoring techniques to ensure field reliability. During the manufacturing process, the detection of interconnect faults in the memory-based programmable elements (MLUTs: multiple look-up tables) array is of critical importance to yield improvement and the assurance of high reliability. The study proposes a comprehensive test method capable of detecting and identifying stuck-at and bridge faults in the interconnects between MLUTs. Moreover, to guarantee long-term field reliability, an aging monitoring technique is proposed that employs a ring oscillator circuit to periodically measure the delay of MLUTs. This method facilitates the detection of aging-induced delays, potentially leading to performance degradation and system failures, thereby ensuring the in-field reliability of MPLD devices.

Extensive experiments and simulations on benchmark circuits demonstrate the effectiveness of the test point insertion technique, achieving a significant reduction in test application time while maintaining high fault detection quality for the automotive ECUs. The interconnect defect test method for MPLDs successfully identifies and locates single interconnect faults, contributing to enhanced manufacturing processes and field reliability. The aging monitoring technique accurately gauges the delay of MLUTs, yielding invaluable insights into the operational aging state of MPLD devices.

This study constitutes a significant contribution to the field of reliability enhancement for IoT and AI edge devices, with a specific emphasis on automotive ECUs and MPLDs. The proposed techniques grapple with the challenges of ensuring functional safety and long-term reliability in these devices, which are vital for the development of a smart society. Future research directions include the exploration of additional fault detection methods, test generation techniques, and design for testability approaches for MPLDs. Furthermore, the study of quantitative analysis and on-chip test methods will be pursued to deepen the understanding and management of aging phenomena in these devices.

Table of Contents

List of Figures

List of Tables

Part I: Introduction and Preliminary

Chapter 1

1. Introduction

1.1 Background

In recent years, rapid advancements in IoT (internet of things) and AI (artificial intelligence) technologies have made the realization of an ultra-smart society more plausible. In such a society, every object connects to the internet, enabling real-time data collection and analysis. Particularly, in IoT systems within a 5G environment, a vast number of edge devices (integrated circuits) connect to the cloud, facilitating data collection and analysis.

However, if we don't ensure the integrity of edge devices, inaccurate data could be collected in the cloud, leading to erroneous decision-making based on data analysis. This might result in decreased system reliability, undermining the safety and confidence of the ultra-smart society.

Simultaneously, with the rapid progress of AI technology, AI edge devices with intelligent capabilities are evolving at the data generation source, i.e., the edge endpoints. This reduces the dependence on cloud systems and enables real-time data analysis and processing on edge devices. However, physical defects in AI edge devices may decrease the accuracy of intelligent processing.

To secure the ultra-smart society, high-reliability technology for IoT and AI edge devices is indispensable. The primary factor that impairs the reliability of edge devices is "failure." While various approaches such as high-quality manufacturing tests before shipment, redundancy, and duplication techniques have been proposed, establishing field testing techniques during edge device operation remains a challenge. Furthermore, there is a need for testing techniques to guarantee the reliability of specially designed edge computing devices that have been newly developed.

1.2 Objective

Edge devices are broadly classified into two categories: non-reconfigurable devices, such as ECUs (electronic control units), and reconfigurable devices, such as FPGAs (field-programmable gate arrays). In recent years, with the progress of self-driving cars, the functional safety of ECUs in automotive systems has become a fundamental

requirement according to the ISO 26262 standard. On the other hand, the requirements of edge devices, such as low power consumption and small latency, hinder the application of traditional reconfigurable devices in edge processing. To address this, a new type of reconfigurable device named MPLD (memory-based programmable logic device) specially designed for edge processing is being developed. Therefore, this study focuses on two objectives:

(1) Reliability enhancement for automotive ECU edge devices.

(2) Reliability enhancement for MPLD edge devices.

1.2.1 Reliability Enhancement for Automotive ECU Edge Devices

First, this study aims to develop fault detection enhancement technologies to meet functional safety standards for automotive ECU edge devices.

Automotive ECU edge devices play a critical role in automotive control systems, and improving their reliability is essential. According to the functional safety standard ISO 26262, automotive systems must undergo field-testing, such as power-on self-test (POST). Unlike production testing, POST needs to reduce test application time and meet the test quality (e.g., >90% latent failure indicator), indispensable for ISO 26262. By developing high-speed and high-quality fault detection methods in field testing, we can ensure accurate fault detection and thus functional safety of automotive ECU edge devices.

Specifically, to enhance the reliability of automotive ECUs, our goal is to develop a fast built-in self-test (BIST) method that can satisfy test quality and detect faults in automotive ECUs edge devices in real-time. Based on this goal, this study proposes a *test point insertion technique for multi-cycle power-up self-test* to reduce test application time with indispensable test quality.

1.2.2 Reliability Enhancement for MPLD Edge Devices

Next, this study will shift our focus to the reconfigurable edge device MPLD and develop fault detection and fault state warning techniques for its reliability.

The MPLD is built exclusively with an array of MLUTs (multiple look-up tables) without any additional programmable interconnect resources. An MLUT is the essential reconfigurable element constructed by SRAMs (static random-access memories). In contrast to traditional reconfigurable device FPGAs, the MPLD can achieve a high density of programmable devices with low power consumption and minimal delay.

During the production phase of the MPLD, a variety of defects may exist in the SRAM memory of the MLUT. Conventional memory testing methods are available for these memory defects. However, a significant number of defects could also be present on the interconnects between MLUTs; these defects could cause considerable yield loss and reliability degradation.

In addition, when the MPLD operates in the field, various aging phenomena such as HCI (hot carrier injection) and BTI (bias temperature instability) could cause aging-induced delays in the MLUT array of the MPLD. The rate of aging progress in the MLUT array may vary. Frequently-used MLUTs may exhibit faster aging speeds, meaning the aging-induced delay would be larger. These variations in aging-induced delay could affect the performance of configured logic circuits and even cause a system failure, threatening the in-field reliability of the device.

Therefore, to guarantee the long-term reliability of the MPLD, this study proposes test techniques tailored to its specific needs. These include a *test method to detect and identify interconnect defects* in the MLUT array during the production phase and a *delay monitoring technique* to detect aging-induced failures in the field.

1.3 Structure of this Dissertation

This dissertation is organized as follows:

Part I Introduction and Preliminary *Chapter 1* introduces this study. *Chapter 2* introduces some important concepts in integrated circuits and test techniques that are relevant to this study.
Part II Test Point Insertion for Multi-Cycle Power-On Self-Test *Chapter 3* introduces multi-cycle test scheme. *Chapter 4* introduces fault detection model in multi-cycle BIST. *Chapter 5* introduces proposed methods of test point insertion and selection for multi-cycle BIST.
Part III Test to Memory-based Programmable Logic Device *Chapter 6* gives an introduction to the MPLD. *Chapter 7* introduces reliability issue in MPLD. *Chapter 8* proposes test method to identify interconnect defect in MPLD. *Chapter 9* proposes aging monitoring technique for MPLD.
Part IV Application of MPLD *Chapter 10* introduces a solution to implement neural networks into MPLD
***Chapter 11* outlines a summary of this study.**

Chapter 2

2. Preliminary

This chapter introduces some important concepts related to this study, including integrated circuit (IC) concepts, reliability of IC, principles of IC testing, fault modeling, test generation, fault simulation, and design for test (DFT) techniques.

2.1 Integrated Circuit

An *integrated circuit* (IC), also known as a *microchip* or simply a *chip*, is a miniature electronic device that contains thousands, millions, or even billions of electronic components, such as transistors, resistors, capacitors, and diodes, fabricated onto a single semiconductor material, typically silicon. These components are interconnected by conductive pathways etched into the chip's surface, forming a complex network of electronic circuits. The integration of numerous components onto a single chip allows for the creation of compact, lightweight, and highly efficient electronic systems.

Integrated circuits can be classified into various types, including *digital*, *analog*, and *mixed-signal* (which consist of both digital and analog signaling on the same IC) integrated circuits. Each type is tailored for specific applications. However, this dissertation will not cover analog and mixed-signal integrated circuits.

2.1.1 Digital Logic Circuit

A *digital circuit* is also known as a *logic circuit* because it carries out *logical operations* on digital signals. Logic (or digital) circuits are constructed by interconnecting elements called *gates* (or *logic gates*) whose inputs and outputs represent only the values in terms of 0 and 1 [1].

Some of the common logic gates are AND, OR, NOT, NAND, NOR (an inverter), and XOR (Exclusive-OR); with symbols as shown in Figure 2.1. The output of each gate can be represented by a *logic function* of the inputs, i.e., a *Boolean function*. The *Boolean (logic) operations* $\wedge$ ($\cdot$), $\vee$ ($+$), $\neg$ ($\bar{}$), and $\oplus$ correspond to AND, OR, NOT, and XOR, respectively. A logic function can often be also specified by a *truth table*. These gates are fundamental building blocks in digital logic circuits and are used to perform various logical operations in computer systems and electronic devices. A summary of the logical operations performed by these gates is as following [1].

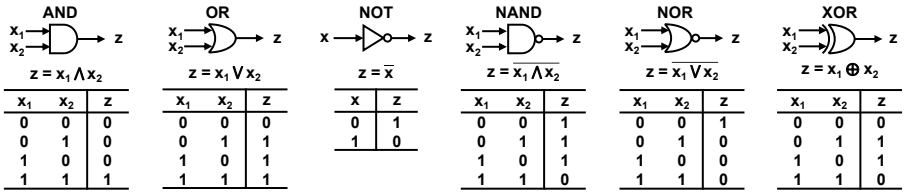


Figure 2.1 Symbols, logic functions, and truth tables of some common logic gates.

Logical Operations

AND gate: The output is 1 only when both inputs are 1. E.g., the output z of an AND gate with inputs x_1 and x_2 is 1 if and only if both of its inputs are 1 simultaneously, and the logic function can be expressed as

$$z = x_1 \wedge x_2 \text{ (or } z = x_1 \cdot x_2 \text{ or } z = x_1x_2)$$

OR gate: The output is 1 if at least one input is 1. E.g., the output z of an OR gate with inputs x_1 and x_2 is 1 if and only if any of its inputs are 1, and the logic function can be expressed as

$$z = x_1 \vee x_2 \text{ (or } z = x_1 + x_2)$$

NOT gate: It simply negates the input value. E.g., the output z of an NOT gate with inputs x is 1 if and only if its input is 0, and the logic function can be expressed as

$$z = \bar{x} \text{ (or } z = \neg x)$$

NAND gate: It is the negation of an AND gate, meaning the output is 1 when at least one input is 0. E.g., the output z of a NAND gate with inputs x_1 and x_2 is 1 if and only if any of its inputs are 0, and the logic function can be expressed as

$$z = \overline{x_1 \wedge x_2} = \overline{x_1} \vee \overline{x_2}$$

NOR gate: It is the negation of an OR gate, so the output is 1 only when both inputs are 0. E.g., the output z of a NOR gate with inputs x_1 and x_2 is 1 if and only if both of its inputs are 0, and the logic function can be expressed as

$$z = \overline{x_1 \vee x_2} = \overline{x_1} \wedge \overline{x_2}$$

XOR gate: The output is 1 when the inputs have different values. E.g., the output z of an XOR gate with inputs x_1 and x_2 is 1 if and only if its inputs are not simultaneously equal, and the logic function can be expressed as

$$z = \overline{x_1}x_2 \vee x_1\overline{x_2} = x_1 \oplus x_2$$

2.1.2 Combinational and Sequential Logic Circuit

Logic circuits can be categorized as *combinational (logic) circuits* or *sequential (logic) circuits*, depending on whether the logic circuits contain a feedback loop which is a directed path from the output of some gate to an input of that gate [1].

A combinational (logic) circuit consists of an interconnected set of gates with no feedback loops. A block diagram for combinational circuits is shown in Figure 2.2(a), where the inputs and outputs are used to interact with the circuit and are also known as the *primary inputs* (PIs) and *primary outputs* (POs), respectively. The output values of a combinational circuit at a given time depend only on the present applied input values. Hence, each output can be specified by a logic function of its input variables.

A sequential (logic) circuit consists of two sections: a combinational circuit part and the feedback loops containing memory circuits. The output values of a sequential circuit at a given time depend on the present applied input values and previous applied input

values. The history information of previous applied inputs is summarized as the *state* of the circuit and stored in memory. Figure 2.2(b) shows a block diagram for sequential circuits, where the outputs of the memory as the feedback inputs feeding the *present state* of the sequential circuit; and the inputs of the memory as the feedback outputs summarizing the *next state* for the sequential circuit. A sequential circuit can be modeled mathematically by a *finite-state machine* (FSM) or *sequential machine*, and each primary output and state can be specified in the FSM according to the primary input variables.

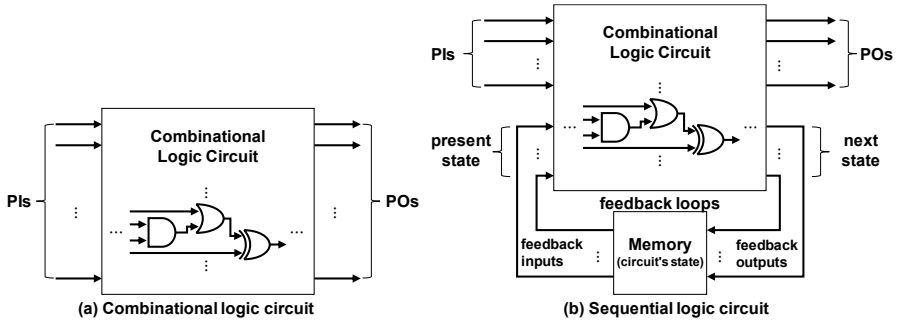


Figure 2.2 Combinational and sequential logic circuits.

Sequential circuits can be further categorized as either *synchronous circuits* or *asynchronous circuits*, depending on whether or not the memory portion of the circuit is controlled (or clocked) at discrete instants of time (*time-frame*) by a synchronizing pulse signal called a *clock pulse* or simply a *clock*.

A synchronous circuit is applied to a clock to the memory portion, and all feedback loops are controlled synchronously by the clock. Figure 2.3(a) shows a block diagram for synchronous circuits. The memory element in a feedback loop is a *flip-flop* (FF). Only at a clock pulse can the FFs be stored with new information, i.e. at this time, the present state can be updated, simultaneously with the next state being stored in the FFs.

An asynchronous circuit operates asynchronously, and its memory portion does not need to be clocked by a synchronizing pulse signal. Figure 2.3(b) shows a block diagram for asynchronous circuits. The memory element in each feedback loop is either a *latch* or a *time-delay element*.

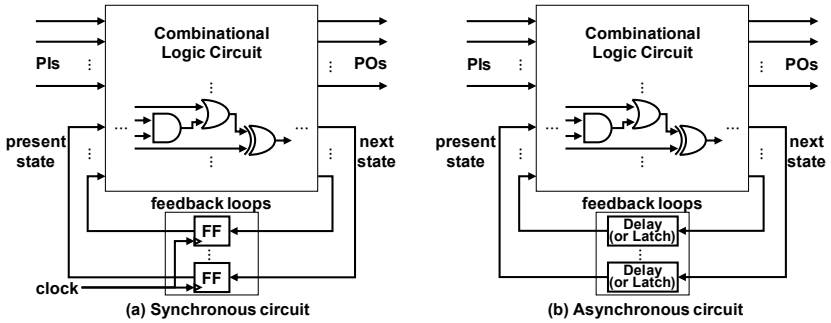


Figure 2.3 Synchronous and asynchronous sequential circuits.

2.1.3 Latch and Flip-flop

Latches are the fundamental element that stores binary information in logic circuits. A latch stores one bit of a binary value as long as power is applied and holds its value until it is updated by new input signals. The latch is built from logic gates to derive a *bistable circuit* to keep the stable value by itself.

Figure 2.4 shows a basic *bistable circuit* that is built by two NOT gates in a feedback loop, it is also known as two *cross-coupled* inverters. This circuit keeps two stable states Q and $\overline{Q}$, which means it can store a bit value. The value of the stable states can be updated for storing a new value by additional gates to control the two cross-coupled inverters.

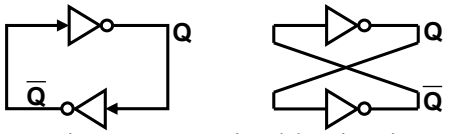


Figure 2.4 Bistable circuit.

As shown in Figure 2.5(a), two additional OR gates control the two cross-coupled inverters (it can also be considered as two cross-coupled NOR gates), and it can set (S) or reset (R) the value of the Q and $\overline{Q}$ by input signals of the OR gates. This is known as an *SR-latch*. The SR-latch can also be designed by adding AND gates to the two cross-coupled inverters or with two cross-coupled NAND gates, as shown in Figure 2.5(b). The SR latch can be added gates at inputs to provide an additional control input (C) that determines when the state of the latch can be changed. This is known as a *gated SR-latch* (with control input), as shown in Figure 2.5(c).

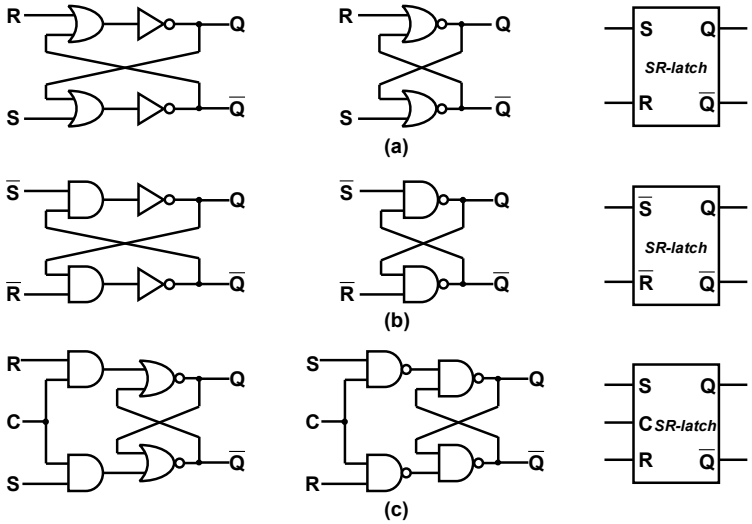


Figure 2.5 SR-latch.

The SR-latch is rarely used in practice because it makes the circuit difficult to manage due to an indeterminate condition that may occur when all inputs are equal

simultaneously. But it holds significance as it serves as the basis for implementing other latches and flip-flops. Figure 2.6 shows a *D-latch* implemented by using an SR-latch. The D-latch is more commonly used than the SR-latch because it eliminates the indeterminate state of the SR-latch by making the *S* and *R* of the SR-latch never equal simultaneously. Although latches are valuable for storing binary information and designing asynchronous sequential circuits, they are not suitable for synchronous sequential circuits due to the lack of time control leading to an immediate output response, unlike flip-flops.

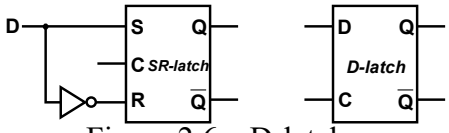


Figure 2.6 D-latch.

Flip-flops are used as memory elements in synchronous sequential circuits. Various types of flip-flops, such as *D flip-flop (D-FF)*, *JK flip-flop (JK-FF)*, *T flip-flop (T-FF)*, are realized by configuring SR-latches or D-latches. The D-FF is a frequently used flip-flop in synchronous circuits. Figure 2.7 shows a D-FF realized with two D-latches connected in a master-slave configuration. Figure 2.7(a) shows a *negative edge triggered* D-FF, where the circuit samples the *D* input during the high level of the clock (*CLK*) and changes the *Q* output only at the negative edge of the *CLK*. In contrast, Figure 2.7(b) shows a *positive edge triggered* D-FF, in which samples during the low level and changes only at the positive edge.

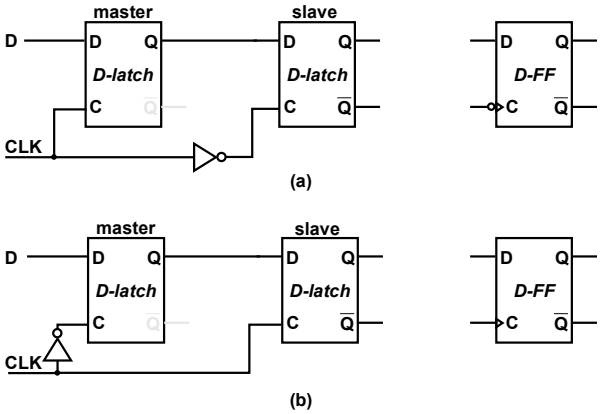


Figure 2.7 D flip-flop.

2.1.4 Register

A latch or flip-flop memory element can store only one bit of binary value. By organizing multiple of these memory elements, multi-bit storage can be implemented. A basic multi-bit storage element is known as a *register*. Figure 2.8 shows a basic structure of a typical register consisting of a set of *n* D-FFs. It is capable of storing *n* bits of binary numbers, where each D-FF shares a common clock. Since the D-FFs have only one data

input, whatever input we apply on the input side, at the same time as a clock transition will be stored in the D-FF. Therefore, it is convenient to use the D-FF in the registers.

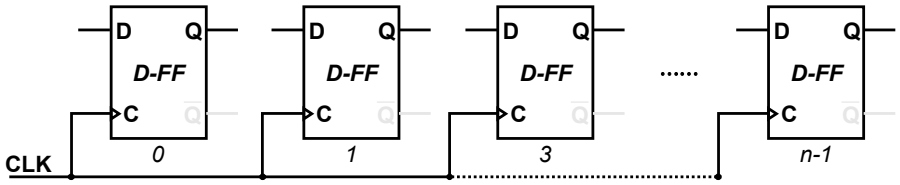


Figure 2.8 Register.

There is another type of register where it is possible to shift binary data between adjacent flip-flops of the register. This type of register is known as the *shift register*. In a shift register, the output of one flip-flop is connected to the input of the next flip-flop. There are two ways to input data into a shift register: serial input and parallel input. The serial input means that only one new bit of data is loaded into the register at one clock pulse. A shift register like this has only one input. In contrast, the parallel input means that all bits of data are loaded into the register at one clock pulse. Such a shift register has multiple inputs. Similarly, there are two ways to output data from the shift register: serial output and parallel output. Figure 2.9 shows the structure of an *n*-bit serial-input serial-output shift register consisting of *n* D-FFs.

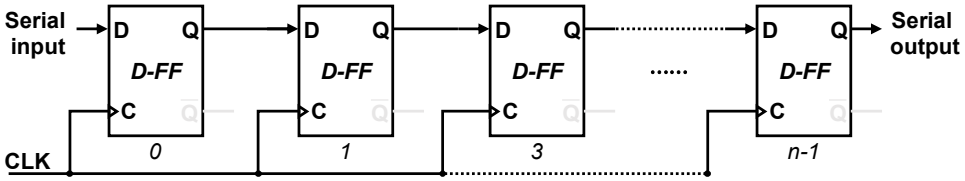


Figure 2.9 Shift register.

2.1.5 SRAM

Compared to registers that store multiple bits of binary data, if a large quantity of binary data needs to be stored, one of the extremely optional storage devices is random-access memory (RAM). A relatively fast RAM is *static random-access memory* (SRAM). SRAM is composed of a large number of basic binary storage cells. A binary storage cell is built from the basic storage element such as a bistable circuit (two cross-coupled inverters) or a latch. A set of binary bits of data stored in a group of storage cells is known as a *word*. A set of eight bits of data is referred to as a *byte*. The capacity of an SRAM is usually expressed as the total number of bytes (or bits) it can store.

In an SRAM memory, in addition to the storage cell (SC), other circuits are needed to control the reading and writing of SRAM, such as the decoder circuit used to select the memory word specified by the address. Figure 2.10 shows an example of the schematic

diagram for an SRAM that can store 2^k*word*×*m-bit*. Which shows $2^k \times m$ binary storage cells and the decoder for selecting individual words.

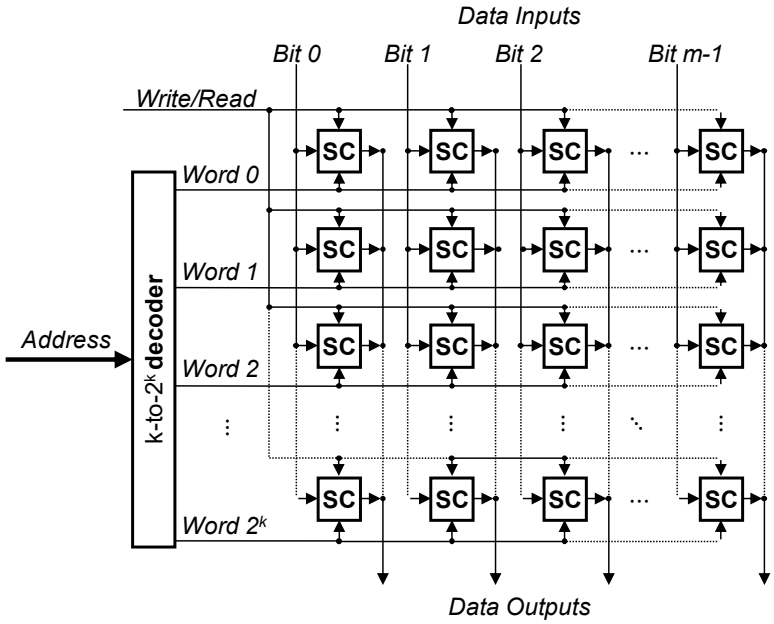


Figure 2.10 Schematic diagram for an SRAM.

2.1.6 Transistor

Logic gates are realized by transistors, and today most integrated circuits are implemented by *metal oxide semiconductor field effect transistors* (MOSFET, or simply MOS) because larger integrations can be obtained with them [1]. The most basic MOSFET-based logic families are *p-channel MOSFET* (PMOS) and *n-channel MOSFET* (NMOS). Another dominant MOS-based logic family is the *complementary MOSFET* (CMOS), which consists of a pair of complementary NMOS and PMOS transistors.

Figure 2.11(a) shows the circuit symbols for NMOS and PMOS transistors [2]. For NMOS transistors, when the gate-to-source voltage V_{gs} is less (more) than the threshold voltage V_{th}, the drain will be in a cut-off (turn-on) state to the source. For PMOS, it is in a cut-off state when V_{sg} is less than V_{th}, and in a turn-on state when V_{sg} is more than V_{th}. Figure 2.11(b) shows a COMS invertor.

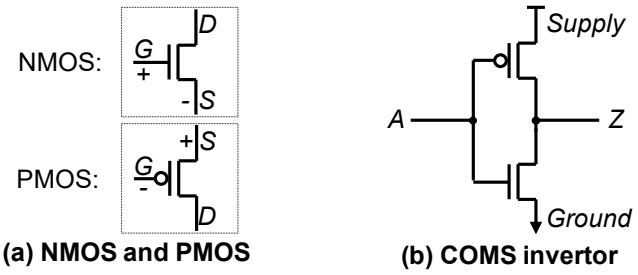


Figure 2.11 Transistors.

2.2 Integrated Circuit Reliability

As IC technology continues to advance greatly, the integration levels of ICs have increased dramatically. The increase in the number of integrated transistors has led to the emergence of *large-scale integration* (LSI), *very large-scale integration* (VLSI), and even *ultra-large-scale integration* (ULSI). The high circuit density of ICs improves their performance and reduces their cost. On the other hand, high integration density requires extremely fine manufacturing processes, where even minute variations in these processes can easily lead to *defects* (A defect in an IC is a flaw or physical imperfection that may result in a fault manifestation [3][4].), thus may resulting in a *failure* IC. Furthermore, high-integration ICs, due to their tiny transistors and delicate interconnections, are prone to damage from various factors during field use, such as aging or wear. These challenges highlight the increasing importance of *reliability* in high-integration ICs.

The reliability of an IC varies with the *failure rate* over its life cycle [5]. One common approach to analyzing the reliability of ICs, particularly LSI devices, is by using the "bathtub curve" model. It is a widely adopted model used in reliability engineering to describe the failure pattern of electronic components, including ICs. This model characterizes the failure rate of ICs over time. As shown in Figure 2.12, the bathtub curve consists of three phases: the *early failure phase*, the *random failure phase*, and the *wear-out failure phase* [5].

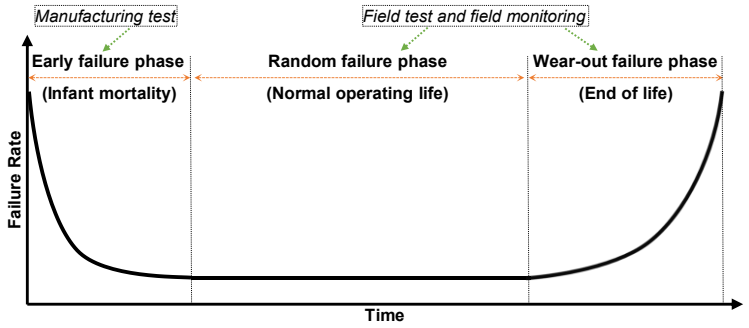


Figure 2.12 Bathtub curve for IC reliability.

Early Failure Phase: In this phase, the failure rate of ICs is relatively high during the initial period of operation. This phase is often associated with manufacturing defects or issues arising from the "infant mortality" phenomenon, where components fail early due to latent defects introduced during the manufacturing process.

Random Failure Phase: After the early failure phase, the ICs enter a phase where the failure rate remains relatively low and constant over an extended period. This phase

represents the *normal operating life* of ICs, where failures occur randomly due to various factors such as external stresses, electrical overstress, or component wear-out.

Wear-out Failure Phase: Over time, as ICs age and accumulate usage, they enter the wear-out failure phase. In this phase, the failure rate starts to increase, indicating the degradation of components and a higher likelihood of failures. Wear-out failures are typically associated with aging effects, such as electromigration, oxide breakdown, or material fatigue.

By understanding the bathtub curve model and its application to IC reliability, researchers and engineers can assess and improve the reliability of ICs. For this purpose, one of the important roles involves the *IC test* technique. By employing effective IC test techniques to detect and minimize the defects causing failures, the normal operating life of ICs can be extended. As shown in Figure 2.12, before an IC is shipped to the market, in early failure phase conducting high quality *manufacturing test* can eliminate the defective IC or that with a high potential for failure. By employing strategies such as *field test* and *field monitoring* to report and predict the random failures and wear-out failures, the overall reliability of shipped ICs can be enhanced.

2.3 Integrated Circuit Test

2.3.1 Purpose of Test

An IC test is a procedural examination aimed at detecting and/or localizing faults resulting from defects (or design errors) within ICs [1]. It can be carried out at various stages in the lifecycle of an IC chip to ensure reliability, including during the design (involving design verification), manufacturing (involving manufacturing test), and field operation (involving field test or field monitoring) stages [4]. This Dissertation mainly focuses on the test during the manufacturing stage and field operation stage.

Depending on the specific purpose of the testing, the tests may be categorized as *fault detection* and *fault location* (also known as *fault diagnosis*) [1]. The purpose of fault detection is to determine whether an IC is defective (*faulty*) or free of faults (*fault-free*), while the fault diagnosis goes further by pinpointing the location and type of the fault, along with other pertinent information necessary for resolving the diagnostic issue. The fault detection is prioritized as the initial step during fault diagnosis.

In the manufacturing test, fault detection is a mandatory step, because if any fault is present, the entire chip must be discarded and cannot be shipped to the market. At this stage, fault diagnosis is usually not necessary; it can of course be carried out selectively

for the purpose of improving the manufacturing process by identifying the location, type, and cause of faults present in the defective chips.

In the field test (or field monitoring), if it is established that a fault exists by fault detection, typically, fault diagnosis may be subsequently employed to identify and isolate the specific faulty node or component for necessary repairs.

2.3.2 Test Principle

The basic scheme of IC testing is shown in Figure 2.13 [3][4][6]. A set or a sequence of input patterns is applied to the inputs of the *circuit under test* (CUT or DUT: *device under test*) that produce output responses at the outputs of the CUT, and then the output responses are compared with the expected (correct) responses to determine whether the CUT is fault-free (good) or faulty. It is considered fault-free and passes the test if the CUT produces the correct output responses (matched with the expected ones), otherwise, is faulty and fails the test.

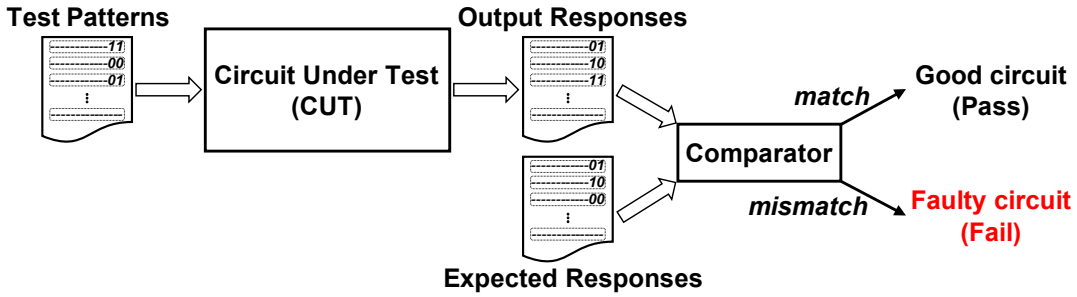


Figure 2.13 Basic scheme of IC testing.

Where an input pattern utilized for testing purposes is commonly referred to as a *test pattern*, also known as *test stimuli* or *test vector*. Typically, a test for a CUT encompasses multiple test patterns, which are collectively referred to as a *test set* or *test sequence*. If the test patterns must be applied in a specific order, the term "test sequence" is used to denote a series of test patterns. Test patterns, along with the corresponding output responses, are occasionally referred to as *test data* [1].

In the design verification, the test patterns and the expected responses are decided by the designer according to the requirements specified in the design specifications.

In the manufacturing and field tests, the expected responses can be obtained from circuit simulation of the fault-free (design error-free) circuit that has been verified by the design verification. The input patterns during the design verification process also can be used as test patterns for the manufacturing test and field test. But, typically, to find efficient test patterns that detect all faults considered for that circuit, the test patterns are decided by a process known as *test generation* for the specific *fault model*.

2.4 Fault Models

A chip may be produced various types of defects. Since the complexity and diversity of the defects, it is difficult to generate test patterns for the real defects. To generate test patterns more easily, it is necessary to build mathematical models that can accurately describe the behavior of the real defects and that must be computationally efficient in simulation environments. A mathematical model like this is known as the *fault model*.

There are many fault models reflecting various defects. The most popular and common fault models, the *stuck-at fault* model, the *bridging fault* model, and the *delay fault* model, will be introduced in the following subsections.

2.4.1 Stuck-at Fault Model

A stuck-at fault describes a faulty behavior of the defect causing the value of a signal on lines (including PIs, POs, and interconnects) in a logic circuit to be stuck at a constant, either a logic 1 or a logic 0, referred to as *stuck-at-1 (sa1)* or *stuck-at-0 (sa0)*, respectively. A defect such as this could be a short circuit between the signal wire and the power supply or ground, or it could be something else. Figure 2.14 shows an example of a stuck fault. Figure 2.14(a) shows a stuck-at-1 fault on line *c*, which is fixed to a value of 1 by a defect that could be a short to the power supply, and Figure 2.14(b) shows a stuck-at-0 fault on line *d*, which is fixed to a value of 0 by a defect that could be a short to ground.

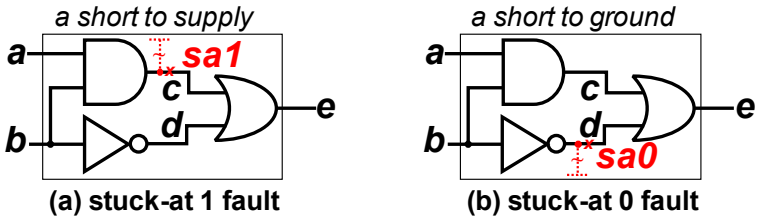


Figure 2.14 Example of stuck-at fault model.

If only one fault exists in a logic circuit, it is referred to as a *single fault*. If two or more faults are present at the same time, then the set of faults is referred to as a *multiple fault*. For a circuit with n signal lines and a given fault model with k different types of faults (for the stuck-at model $k=2$: *sa1* and *sa0*), there may be at most $k \times n$ single faults; and fault collapsing techniques can help reduce these numbers [06]. For multiple faults, the number of possible faults increases sharply up to $(k+1)^n - k \times n - 1$. Testing for multiple faults is difficult due to too many faults to be assumed; however, testing for single fault models can be utilized to test multi-fault models; therefore, single fault models are typically used for test generation [4].

2.4.2 Bridging Fault Model

A bridging fault reflects the behavior of a defect causing that the value of a signal line is dominated by the value of another signal line. A typical type of such defect is a short circuit in a certain situation between two signal lines, as shown in Figure 2.15(a). Depending on the short circuit situation, the values of the bridged signal lines are dominated in different ways, which leads to several different types of bridging faults. Generally, the values of shorted signal lines are dominated by either logic value 0 or 1 [1]. If it is 0-dominant, it is referred to as the *wired-AND* bridging fault model (AND-bridge), as shown in Figure 2.15(b); If it is 1-dominant, it is referred to as the *wired-OR* bridging fault (OR-bridge) model, as shown in Figure 2.15(c) [6]. These two types of bridging faults are the most frequently used in practice.

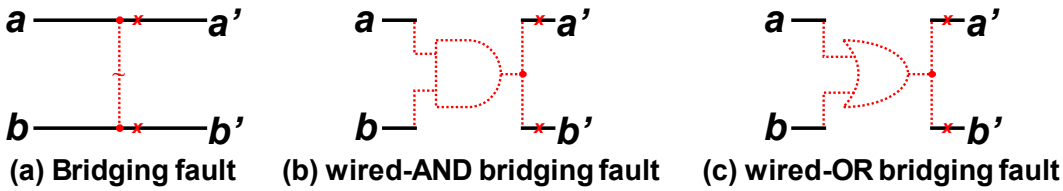


Figure 2.15 Bridging fault: wired-AND/wired-OR bridging fault models.

Other bridging fault models are the *dominant bridging* fault model, and the *dominant-AND/dominant-OR* bridging fault model. The dominant bridging fault model was developed to more accurately reflect the behavior of certain short circuits in CMOS circuits, in which case one line is assumed to act as the driver, dominating the logic values on the two short lines [4][6]. In certain cases, the dominant bridge fault model fails to accurately reflect the behavior of a resistive short. To address this limitation, the dominant-AND/dominant-OR model has been proposed to take into account the observed behavior of resistive shorts in specific CMOS circuits, where one driver exerts dominance over the logic value of the shorted lines, but only under certain logic conditions [4].

2.4.3 Delay Fault Model

A delay fault refers to a type of fault or error that occurs in a circuit when certain signal delays exceed the specified time limits. In logic circuits, signals are expected to propagate through various logic gates and interconnects within a specific time frame. If the propagation delay of a signal exceeds the predetermined threshold, it can lead to functional errors or failures in the circuit. Delay faults can arise due to various factors, such as design errors (e.g., aggressive place and route), process variations (e.g., gate threshold variations), manufacturing defects (e.g., resistive bridges/opens), environments (e.g., severe temperature fluctuations), or field aging phenomenon (e.g., hot-carrier

injection, bias temperature instability). These faults can manifest in different ways, including hold time violations, setup time violations, clock skew, or interconnect delays.

Depending on the ways to model delay faults, there are several typical delay fault models considered, which are the *transition fault* model, *gate-delay fault* model, *line-delay fault* model, *segment-delay fault* model, and *path-delay fault* model [7]. Transition, gate, and line delay models are utilized to characterize delay defects that concentrate at individual gates. Conversely, path and segment delay models are employed to address delay defects that are dispersed across multiple gates [7]. These models are specifically designed to capture and represent the various types of delay defects in the timing behavior of integrated circuits.

In these models, it is assumed that in a fault-free circuit, each gate, along with its interconnects on the input and output pins, possesses a predefined nominal rise (fall) delay from each input to the output pin. When delay defects increase the nominal rise (fall) delay, it results in a *slow-to-rise* (*slow-to-fall*) fault [7], as depicted in Figure 2.16. This fault implies that the transition from 0 to 1 (or 1 to 0) will not reach any output within the specified time limits and result in faulty circuit behavior.

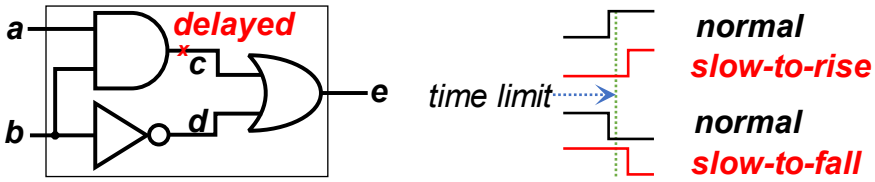


Figure 2.16 Delay fauls: slow-to-rise (slow-to-fall) faults.

2.5 Test Generation

To generate effective test patterns to identify the various potential fault models in the circuit, a process referred to as test generation is to be done [3]. Figure 2.17 shows a procedure for test generation. Once a circuit has successfully passed design verification, the circuit is considered to be a design error-free (fault-free). In test generation, first, a *logic simulation* is performed on the fault-free circuit to generate candidate test patterns and their expected responses. Then, *fault simulation* is carried out by injecting fault models into the fault-free circuit to filter out invalid test patterns and remove already detected faults. Finally, the quality of selected valid test patterns is evaluated using *fault coverage*. If the evaluation results are adequate, the process ends. Otherwise, the process continues by adding new test patterns through logic simulation until the desired fault coverage is achieved. The details about the logic simulation, fault simulation, and fault coverage in the test generation procedure are as follows.

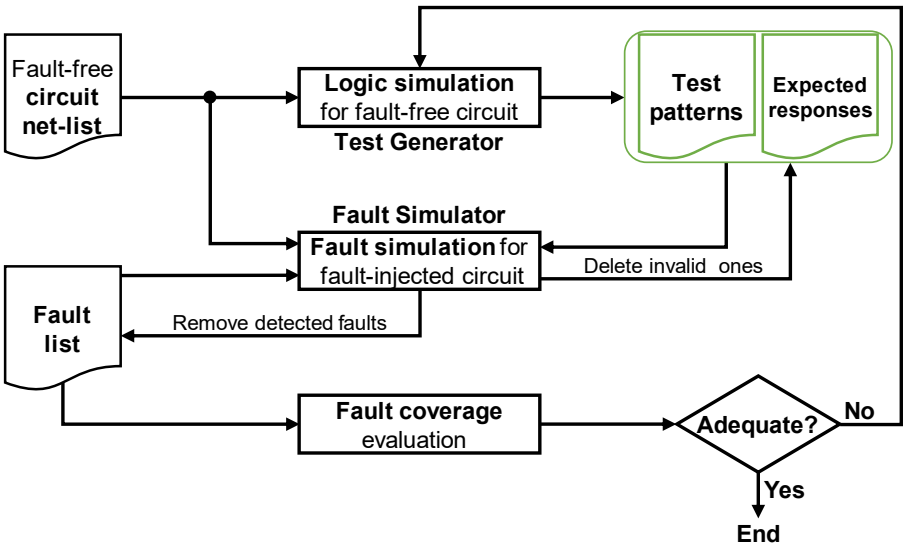


Figure 2.17 Test generation procedure.

2.5.1 Logic Simulation

Logic simulation is a process that uses test vectors to simulate the behavior of a logic circuit. It is typically performed during design verification to ensure that the circuit is an error-free design. After the design verification, it is also employed for fault-free circuits in order to generate candidate test patterns in the test generation process. In this case, it is typically performed in a *test generator*. In the test generator, the fault-free circuit is loaded into the *logic simulator* while input patterns, that may be from a *random pattern generator*, are fed to the fault-free circuit to generate the output responses. These input patterns and the corresponding output responses are used as candidate test patterns and expected responses, which will be filtered in the fault simulation process to detect specific fault models.

2.5.2 Fault Simulation

Fault simulation is an important step in the test generation process as it evaluates the ability of the test patterns to detect specific fault models. In this phase, various faults in a fault list, such as stuck-at faults, bridging faults, or delay faults, are injected into the fault-free circuit. The circuit is then simulated with these injected faults. In the simulation, for each applied test vector, the output response is compared with the expected response. If the comparison result is a mismatch, the test vector is considered to be able to detect the injected fault, and the detected fault will be removed from the fault list; otherwise, it means that the test vector cannot detect the injected fault, and the invalid test vector will be deleted from the test patterns. After all the test vectors in the test patterns have been applied, a quality evaluation for these test patterns will be performed by calculating the fault coverage.

2.5.3 Fault Coverage

Fault coverage is used to evaluate the effectiveness of the generated test patterns in detecting faults for high test quality. As defined in the formula below, it evaluates the quality of the test patterns by quantifying the ratio of the number of detected faults to the total number of faults in the fault list.

$$Fault\ coverage = \frac{Number\ of\ detected\ faults}{Total\ number\ of\ faults}$$

Higher fault coverage indicates that the set of test patterns is more effective and able to detect more faults specified in the fault list. During the test generation process, if the result of the fault coverage evaluation for the applied test patterns to the fault simulation is unsatisfactory, new test vectors should be added through the test generator until the fault coverage after the fault simulation reaches a satisfactory value.

However, sometimes, it is very hard to obtain high fault coverage for some circuits, especially for very highly integrated chips, because some faults in the circuit are difficult to detect or even undetectable using conventional tests. For this reason, a technique called *design-for-testability* has been proposed for achieving high test quality.

2.6 Design for Testability

As circuit integration increases, testing becomes increasingly difficult. Design for testability (DFT) is a crucial aspect of modern circuit design that focuses on making circuits easier to test. By incorporating specific techniques and hardware into the design, DFT enables efficient control and observation of the internal state of the circuit from external access points. This ensures that products are thoroughly and accurately tested, guaranteeing their reliability and performance. This section will discuss three popular DFT techniques: *scan design*, *logic built-in self-test*, and *test point insertion*.

2.6.1 Scan Design

Scan design is a widely used DFT technique that enhances testability by introducing *scan chains* into a circuit to obtain the controllability and observability of the internal state in the circuit. Typically, a scan chain is a shift register formed by connecting certain selected flip-flops in a circuit in a linear fashion, allowing for the insertion and extraction of test data during the testing process. These flip-flops that are selected for the scan design are called *scan flip-flops* (SFFs) or *scan cells*. The number of these SFFs in a scan chain is the length of this scan chain. By incorporating scan design, the internal state of the

circuit can be efficiently controlled, enabling the application of various test patterns and the observation of circuit responses. The scan chain facilitates the shift of test data in and out of the circuit, simplifying the testing procedure and improving fault coverage.

Figure 2.18 shows a schematic for a scan design [4]. A test control signal (TC) added to all flip-flops in the scan chain controls the three operation modes of the scan chain: normal mode, shift mode, and capture mode. In normal mode, all flip-flops operate in the normal functional configuration of the circuit. In shift mode, any desired test data can be set to all the flip-flops of the scan chain by shifting from the *scan-in*. These test data will be applied to the circuit in the normal mode. In capture mode, the test responses stored in the flip-flops can be observed from the *scan-out* by shifting.

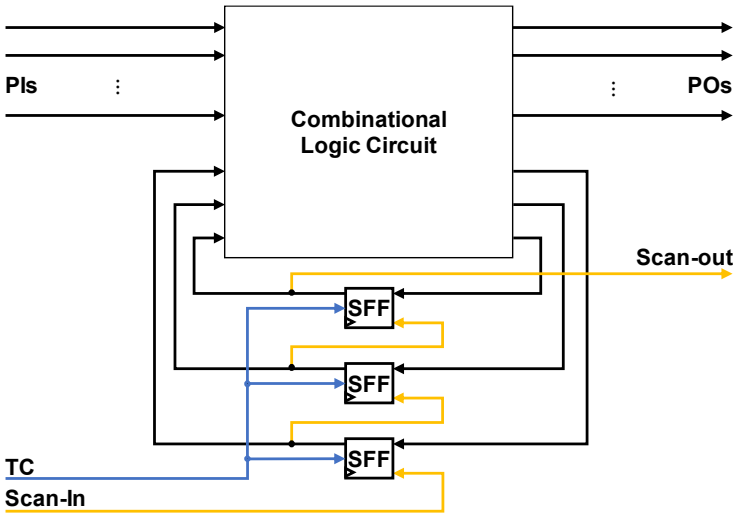


Figure 2.18 Schematic for a scan design.

There is usually more than one implementation way to convert the selected flip-flop in a circuit into a scan cell SFF. The most widely used scan cell is the Muxed-D scan cell [4], as shown in Figure 2.19. The Muxed-D scan cell is implemented by using a D flip-flop and a multiplexer. A *scan enable* input (SE) on the multiplexer is used to select the *data input* (DI) and *scan input* (SI).

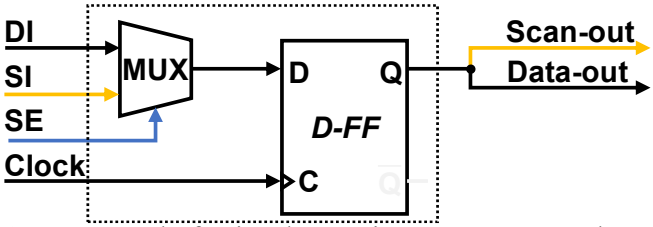


Figure 2.19 Example for implementing a SFF: Muxed-D scan cell.

2.6.2 Logic Built-In Self-Test

LBIST (*Logic BIST*: logic built-in self-test) is another powerful DFT technique that incorporates self-test circuitry directly into the design. This self-test circuitry generates

and applies test patterns to the circuit, autonomously detecting and identifying faults or errors. LBIST eliminates the need for external test equipment, making the testing process more autonomous and efficient. By embedding self-test circuitry, LBIST enables comprehensive testing of the circuit's internal logic and facilitates fault diagnosis and localization.

Figure 2.20 illustrates the basic architecture of LBIST [4][6]. Three key components are designed within the circuit for implementing self-testing. One of these components is the *test pattern generator* (TPG), which automatically generates test patterns to be applied to the inputs of the circuit under test (CUT). These test patterns stimulate the CUT and help detect potential faults. The *output response analyzer* (ORA) is responsible for compacting the output responses of the CUT into a *signature* through signature analysis and comparing it with the expected signature. Additionally, the logic *BIST controller* (or *test controller*) generates specific test control signals to coordinate the BIST operation among the TPG, CUT, and ORA. Once the BIST operation is completed, the ORA provides a pass/fail indication, indicating whether the circuit has passed or failed the test.

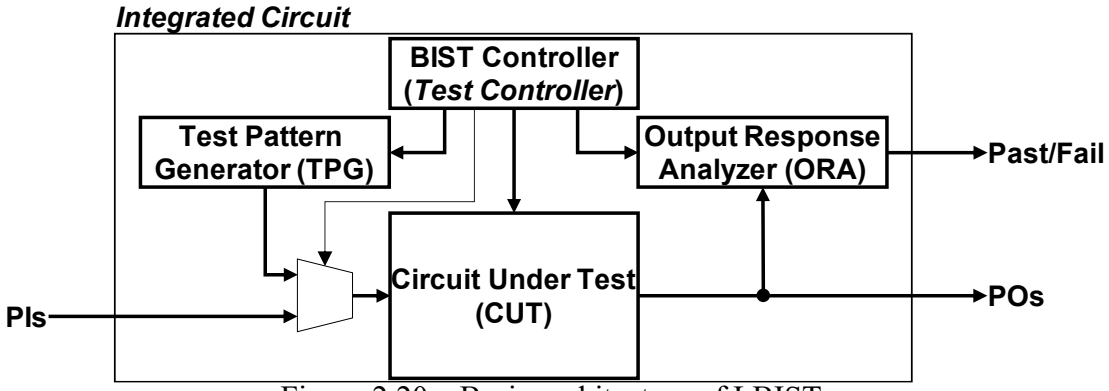


Figure 2.20 Basic architecture of LBIST.

In LBIST applications, TPGs are commonly implemented using *linear feedback shift registers* (LFSRs). Figure 2.21 illustrates the structure of an *n*-stage *modular LFSR* typically used for generating test patterns or test sequences. It consists of *n* D flip-flops and a selected number of XOR gates. It can efficiently generate sequences with good randomness (pseudo-random sequences) at a relatively small area cost.

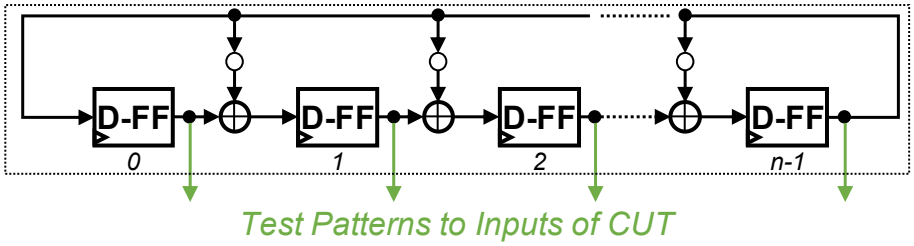


Figure 2.21 *n*-stage modular LFSR.

In ORAs, signature analysis schemes are often employed to compress the output responses. These schemes typically use *multiple-input signature registers* (MISRs). Figure 2.22 shows the structure of an *n*-stage MISR. This MISR compresses multiple output sequences by simultaneously feeding them into extra XOR gates added to the modular LFSR.

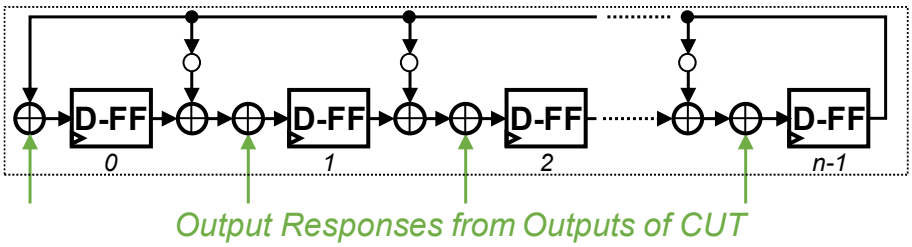


Figure 2.22 *n*-stage MISR.

2.6.3 Test Point Insertion

Test Point Insertion (TPI) is a DFT technique aimed at enhancing observability and controllability within the circuit. It is typically used to improve the detection probability of RP-resistant (random-patterns resistant) faults so they can be detected during pseudo-random testing, to increase the circuit's fault coverage to a desired level [4]. TPI involves strategically inserting additional circuit nodes, called *test points*, including *control points* and *observation points*, throughout the design. These test points provide access to internal signals, allowing for the monitoring and control of specific areas within the circuit during testing. By carefully selecting and placing test points, engineers can target critical areas or potential fault sites, improving fault detection and enabling more effective debugging and characterization of the circuit.

Figure 2.23 shows two typical types of test points [4]: a test point with a multiplexer and a test point with AND-OR gates. Where the control point (CP) can be connected to a primary input, an existing scan cell output, or a dedicated scan cell output; the observation point (OP) can be connected to a primary output through an additional multiplexer, an existing scan cell input, or a dedicated scan cell input; and a test control signal (TC) controls the test mode and normal operation mode of the test point.

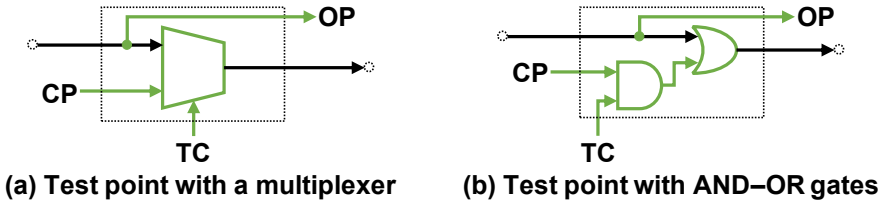


Figure 2.23 Two typical types of test points.

Part II: Test Point Insertion for Multi-Cycle Power-On Self-Test

With the rapid evolution of technologies in developing automotive systems toward fully autonomous vehicles, various complex integrated circuits (ICs) are embedded in a car. The functional safety of the automotive system becomes a fundamental requirement as indexed by the ISO26262 standard [8].

Power-on Self-Test (POST) is the most common test solution to ensure the safety of a system. It tests the safety-critical components during the system's startup before running any functional operations. It is thus helpful to detect any potential faults inside the components early to avoid a system failure. For testing automotive ICs, The POST needs to meet several constraints as follows.

- Indispensable test quality: The ISO26262 standard imposes at least 90% latent fault metric to meet the most stringent a*utomotive safety integrity level* (ASIL D) for avoiding a random hardware failure (e.g., stuck-at faults) during the lifetime of ICs [8];
- Limited *test application time* (TAT): test must be complete during the power-on reset at the engine startup (e.g., 10~50ms);
- Low power: the consideration of power consumption during the test is helpful to avoid false tests under the delay fault model [9];
- Low silicon overhead: suppress the increase of the Design for Test (DFT) hardware due to the ever-increasing complexity of ICs.

The simple way to run the POST is to utilize the logic built-in self-test (LBIST) which is the general test infrastructure for the manufacturing test. An LBIST is typically a scan-based DFT scheme running test-per-scan testing, where each test (capture operation) is executed after a serial scan-shift of pseudo-random patterns/responses. It usually requires a large test volume to attain a reasonable test coverage due to the lower quality of the pseudo-random patterns generated by the on-chip test pattern generator (TPG). Consequently, the TAT increases under a usually slow scan-shift clock for scan-shifting the test patterns.

In the past, many sophisticated solutions have been introduced to reduce the test volume of the standard LBIST, such as scan structure optimization [10][11], the weighted random pattern [12][13], random vector perturbing [14], Bit-flipping [15], and reseeding [16][17][18]. Test point insertion (TPI) technology improves the testability of CUT by inserting test logic into the CUT to deal with the detection of random pattern resistant (RPR) faults [19][20][21]. Other solutions focus on improving the test scheme of LBIST to enable the test-per-clock testing [22][23], such as the shadow flip-flops insertion [24], and the Tri-Modal Scan test scheme [25] with reconfigurable scan cell design. Recently

the deterministic test compression technology is also applied to the automotive ICs to run a POST or an in-system test [26][42]. The deterministic compression test requires a modified BIST structure to allow applying the external test patterns generated by ATPG or continuous reseeding for high fault coverage. In this work, we focused on improving the test quality of an on-chip TPG-based LBIST by introducing a multi-cycle test described later to make the standard LBIST comply with the ISO26262 standard.

Multi-cycle test applying more than one clock to run many times function operation after the scan-shift operation is a smart way to complement the quality of the test patterns (scan-in patterns) for test compaction [27][28][29], low-power testing [30][31][32], and logic diagnosis [33]. In a multi-cycle test, the response of CUT at each capture cycle is applied to the CUT in parallel as a capture pattern at the subsequent capture cycle. This feature is helpful to reduce the volume of scan-in patterns for attaining a target test coverage when the capture patterns can detect any additional faults that are missed by the root scan-in pattern. The multi-cycle test has the behavior to take the CUT closer to its functional operation conditions that can generate functional vectors with lower power consumption, which are very helpful to at-speed testing for delay fault detection [31][32]. It is also easy to implement the multi-cycle test w/o the extra overhead in terms of software (e.g., modified ATPG for deterministic pattern generation) and hardware (e.g., reseeding logic & memory). Therefore, the multi-cycle test is expected to be a promising test scheme to a trade-off among the test coverage, TAT, silicon area, and low power for POST.

A multi-cycle test may not always be effective for test reduction when the functional sequences generated by the CUT are not helpful to fault detection. Appropriate DFTs that can complement the value of the functional sequences are necessary to enhance the ability of the multi-cycle test to test reduction. Many DFT approaches to improve delay fault detection were presented in the past. In [34], the authors proposed a new scan cell named Transition-Launch Flip-Flop to complement the test vectors by modifying the value of partial FFs after the launch cycle in a two-cycle broadside test. In [35] and [36], the author expanded the approach of [34] to multi-cycle tests and proposed the DFT approaches to enhance the ability of the capture states to delay fault detection by holding [35] or reversing [36] the value of all FFs at the appropriate capture cycles. These DFT approaches considered the condition/requirement of the hard-to-detect delay faults in the multi-cycle test.

In our previous works [37] [40], we have discussed the fault masking problem and the *fault detection degradation problem* (FDD) that would obstruct the effect of multi-

cycle tests to detect stuck-at faults. It is necessary to solve the problems to reduce the test application time of POST under the indispensable test quality specified in the ISO26262 standard. The main difference between the existing DFT approaches and our works is to solve the Fault Masking and FDD problems for the stuck-at fault detection under the multi-cycle LBIST.

The fault-masking problem denotes that the fault effects excited at the intermediate capture cycles might be masked before the effects are propagated to the final capture cycle for observation. To address this issue, we proposed a novel scan cell named the *fault-detection-strengthened FF* (FDS-FF) that directly observes and keeps the value of a faulty effect as it arrives at the FF during the capture operations [37][38][39].

The FDD denotes that the capability of capture patterns to detect additional stuck-at faults degrades with the increase of the number of capture cycles [40]. In [41], we have proposed a *control point insertion* (CPI) method to overcome the FDD by inserting control logic into scan FFs that modifies the value captured into the FFs during intermediate cycles, named the FF-CPI. While the basic idea is similar to the DFT proposed in [35][36], our approach targeted controlling partial FFs but not the whole scan chain. We also proposed an approximate evaluation approach to identify CPs by analyzing the circuit structure without fault simulation. In general, the fault-simulation-based evaluation in [35][36] needs more processing time than our method. Moreover, in this study, we expand the FF-CPI approach of [41] to control the internal state of the combinational logic for stuck-at fault detection, which is different from the existing DFT approaches that will be described in *Chapter 5*.

This part consolidates the FDS-FF insertion approach denoted by OP (observation point) insertion and the FF-CPI approach into a complete DFT technique referred to as the test point insertion (TPI). Unlike the conventional TPIs which detect the random pattern resistant faults, our TPI focus on addressing the Fault Masking problem and FDD problem under a multi-cycle LBIST scheme to reduce the volume of scan-in patterns to meet the indispensable test quality specified by ISO26262.

The main contributions of this part are as follows.

(1) *We clarify the mechanism of Fault Masking and FDD by analyzing the stuck-at fault detection model in the multi-cycle BIST scheme.*
(2) *We expand the FF-CPI approach to control the internal state of combinational logic by a newly proposed control logic circuit named Self-flipping CP to improve the testability for stuck-at fault detection under multi-cycle tests.*

(3) *We propose a new metric to evaluate the effect of candidate signal lines for CP insertion under the multi-cycle BIST scheme.*

(4) *We introduce an improved probabilistic cost function to estimate the effect of CP and OP insertion.*

(5) *We introduce a consistent procedure to identify a user-specified number of CPs and OPs to achieve the most scan-in pattern reduction for attaining a target test coverage in the multi-cycle BIST scheme.*

(6) *We evaluate the effectiveness of the proposed TPI for shortening the test application time based on the experimental results of ISCAS'89 and ITC'99 benchmark circuits under the single stuck-at fault model.*

The remainder of this part is organized as follows. Chapter 3 introduces the basic concept of test-per-scan BIST, the multi-cycle test, and its issues. Chapter 4 describes the fault detection model under multi-cycle BIST. Chapter 5 presents the TPI approach for multi-cycle BIST, shows the experimental results on benchmark circuits, and concludes the part.

Chapter 3

3. Multi-Cycle Test Scheme

In this chapter, we first review the characteristic of scan BIST and multi-cycle BIST and discuss the problems of multi-cycle tests.

3.1 Scan BIST

In a traditional test-per-scan BIST, pseudo-random vectors generated by the on-chip TPG are serially loaded into the scan chains driven by scan-shift clocks, known as scan operation. When all the scan registers are filled up, a complete scan-in pattern is latched to the inputs of the circuit. The circuit is then switched to the functional operation that generates the corresponding functional response at the outputs of the circuit. The FFs will be updated with the functional responses of the circuit when the trigger edge of the functional clock arrives, known as the capture operation. The captured functional response will be unloaded for fault detection as loading the next scan-in pattern. It is easy to observe that test is conducted only once by applying a complete scan-in pattern. Almost all test application time is consumed in the serial scan shift operation for test data delivery.

3.2 Multi-cycle BIST

The multi-cycle test applies more than one functional clock to run many capture operations for every single scan-in pattern. In Figure 3.1, we show the operations during the multi-cycle test in the time-frame expansion of CUT. Let's define a multi-cycle test by $<s_i, v_i, c_{ij}, o_i>$, where s_i denotes a scan-in pattern; v_i denotes a primary input vector; c_{ij} denotes the responses of CUT captured into the scan chains represented by the capture patterns at the jth functional clock; and o_i denotes a scan-out pattern which is the response of the combinational circuit when c_{ij} is applied at the last capture. After a scan-in pattern s_i is loaded into the scan chain in serial, the corresponding response c_{i1} is generated at the outputs of combinational logic (FFs drawn in dashed line) and captured into the FFs in parallel by the functional clock T1. Then, c_{i1} is used as test stimuli and latched to the circuit to generate a new response c_{i2}, and c_{i2} is applied and generates the corresponding response c_{i3} in parallel until the final capture clock is applied. The response captured at the final capture o_i is unloaded for observation. It should be noted that the state of primary

inputs v_i will be kept constant and the primary outputs in the intermediate capture cycle are considered unobservable during the capture operation.

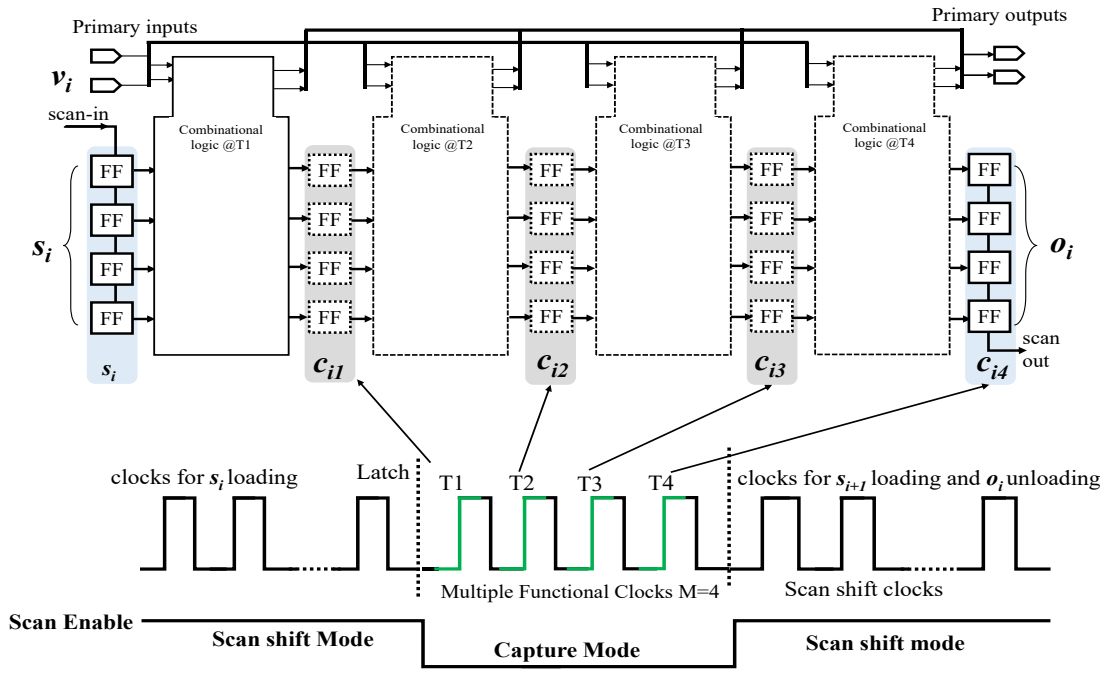


Figure 3.1 Test operations in multi-cycle BIST.

From Figure 3.1, it can be observed that conducting a multi-cycle test for each scan-in pattern $<s_i, v_i>$ could provide more chances to detect additional faults through the functional capture patterns c_{ij}. Therefore, it has promising potential to reduce the number of scan-in patterns for attaining a target test coverage that contributes to shortening the test application time due to fewer scan-shift operations. In addition, since the time of capture operation is negligible compared to the serial scan-shift operation, the reduction of the total test application time for POST is expectable.

3.3 The Problems of Multi-cycle BIST

In our earlier works, we have raised two issues that would obstruct the effect of multi-cycle test to reduce the scan-in patterns for shortening the TAT of POST, called the Fault Masking [37][38][39] and Fault Detection Degradation of Capture Pattern [40][41], respectively. The following gives a brief overview of these problems for this study.

3.3.1 Fault Masking

Fault Masking denotes that the fault effects excited at the intermediate capture cycles by capture patterns might disappear before these effects are propagated to the final capture cycle for observation. Suppose that a fault f is excited at the first capture cycle by

the scan-in pattern. To detect *f*, its faulty value has to be propagated through all *M-1* capture cycles until the final capture cycle is applied. When the CUT has a deep combinational logic or the capture operation runs in a large cycle number, the time-expanded propagation path of the faulty value would become too long to be activated for detection, and the faulty value might be masked at certain logic due to the un-controllable logic state during the capture operation. Severe fault-masking would decrease the test quality of the scan-in patterns and capture patterns, and finally, obstruct the effect of the multi-cycle test for reducing the scan-in patterns.

3.3.2 Fault Detection Degradation Problem (FDD)

FDD means the capability of capture patterns to detect more additional stuck-at faults degrades as increasing the number of capture cycles. This is based on the observation that multi-cycle tests can take the CUT closer to its functional operation conditions with small internal transitions when increasing the capture cycles [31]. The functional operation would consequently cause the states of the large number of FFs to become constant when a number of capture cycles are applied. Since the value of FFs is reused as test stimuli at the subsequent capture cycles, the large number of FFs with constant values would cause the loss of randomness property of the capture patterns that obstructs the detection of additional faults.

Chapter 4

4. Fault Detection Model in Multi-Cycle BIST

In this chapter, we give a detailed analysis of the stuck-at fault detection model in a multi-cycle BIST scheme to elucidate the mechanism of Fault Masking and FDD as follows.

For a stuck-at fault F_i, its faulty effect will always exist at each capture cycle during the capture operation, and we express it by f_{ij} in the time-expanded circuit as shown in Figure 4.1. The faulty effect of F_i at each capture cycle might be excited by the inputs of CUT. We use Pe_{ij} to denote the probability to excite fault F_i at the jth capture cycle. To detect F_i, the excited faulty value of F_i at the jth capture cycle (f_{ij}) must be propagated to the scan FFs for observing after the final capture, and we denote the propagation probability as Pp_{ij}.

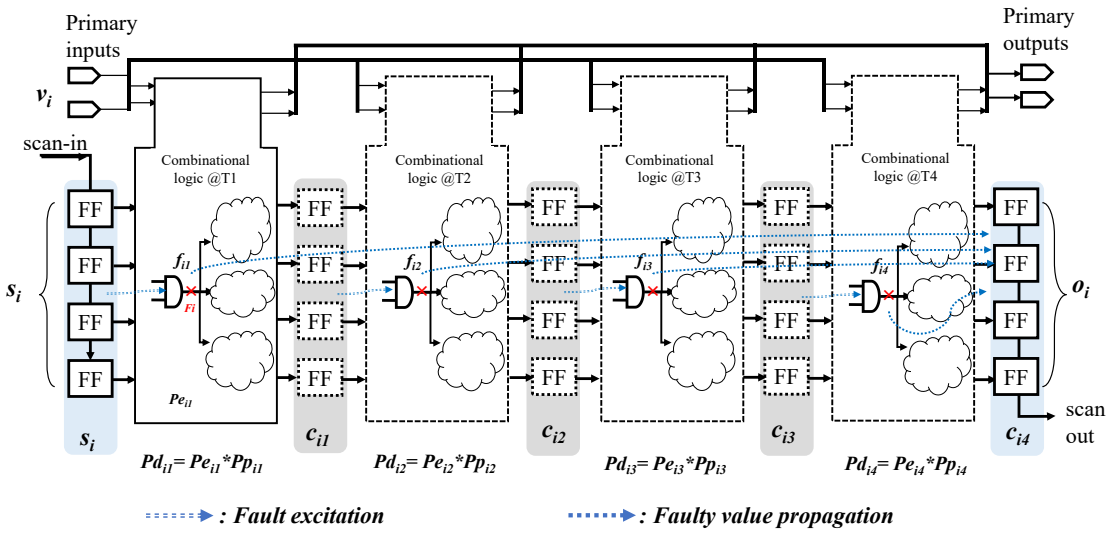


Figure 4.1 Single stuck-at fault detection in time-expanded circuit.

In LBIST, Pe_{ij} and Pp_{ij} of a stuck-at fault $F_{i/s}$ can be estimated by computing the $s^{\neg}$ controllability ($C_{ij/s^{\neg}}$) and the observability (O_{ij}) of signal line i through the probabilistic random pattern testability measure such as COP (*controllability observability procedure*). Hence, the detection probability of $F_{i/s}$ at the jth capture cycle denoted by $Pd_{ij/s}$ can be expressed by $Pd_{ij/s}=C_{ij/s^{\neg}} \times O_{ij}$. For a multi-cycle test with M capture cycles, the fault $F_{i/s}$ would have M times opportunity to be excited by the capture patterns c_{i1}~c_{iM}, and $F_{i/s}$ will be detected out of once the fault is excited and propagated to the outputs. Hence, the detection probability of $F_{i/s}$ denoted by $Pd_{i/s}$ in a multi-cycle test is the complementary

probability of the case that $F_{i/s}$ cannot be excited and propagated for detection at all capture cycles, which can be expressed by:

$$Pd_{i/s} = 1 - \prod_{j=1}^{M}(1 - C_{ij/s^{\neg}} \times O_{ij}) \qquad (4.1)$$

To calculate the controllability and the observability of signal lines at each capture cycle, we transform the CUT to *M* cycles time-frame expansion combinational circuit. We initialize the 0/1 controllability ($C_{i1/0}$ and $C_{i1/1}$) of PI (primary input) and PPI (*pseudo-primary input*: FF) at the first capture cycle to 0.5/0.5, then, calculate the value of $C_{ij/0}$ and $C_{ij/1}$ for each gate at each time-frame. The observability of signal line at each time-frame is calculated starting from the PO (primary output) and PPO (*pseudo-primary output*) at the last capture cycle with initial value of 1.0, tracing back to the PI and PPI until the first capture cycle.

Compared to the traditional scan test with a single capture, the multi-cycle test shows the potential to improve the probability of fault detection for every single scan-in pattern followed by multiple capture patterns. However, the fault detection in the multi-cycle test depends on the controllability and observability of signal lines in the time-expanded circuit, which is generally deteriorating, as the number of capture cycles increases.

For demonstration, we conducted preliminary experiments on ISCAS89 and ITC99 benchmark circuits to evaluate the average 1-controllability and the observability of signal lines at each capture cycle. Figure 4.2 shows the results. In Figure 4.2(a), it can be observed that ITC99 circuits show higher 1-controllability, which implies the internal states of these circuits are easy to be 1, whereas ISCAS89 circuits likely trend to be 0. Figure 4.2(b) shows the standard deviation of 1-controllability of signal lines at each capture cycle corresponding to the results of Figure 4.2(a), to demonstrate the impact of increasing capture cycles on the controllability. The results show that the standard deviation of 1-controllability becomes higher as the capture cycles increase, which implies the controllability of more signal lines is biasing toward either 0 or 1; in other words, the value of more signal lines in a large capture cycle would be most likely fixed at 0 or 1 during the tests. For a signal line with stuck-at fault, higher 0-controllability (0-bias) is helpful to excite the *s-1* fault, whereas exciting the *s-0* fault becomes difficult. Moreover, the biased controllability of signal lines in a time frame would also affect the path sensitization for propagating the excited faulty values to the FFs in the current time frame. We insist on it as the root cause of FDD observed in our previous works.

Regarding the observability shown in Figure 4.2(c) and Figure 4.2(d), it can be observed that in a 10-times expand circuit the value of more signal lines in earlier capture cycles is more difficult to be observed from the outputs (scan FFs) after the final capture. The deterioration of observability of signal lines at early capture cycles causes the faults excited at an early capture cycle difficult to be propagated to the final capture cycle for detection, which is the root cause of the fault masking.

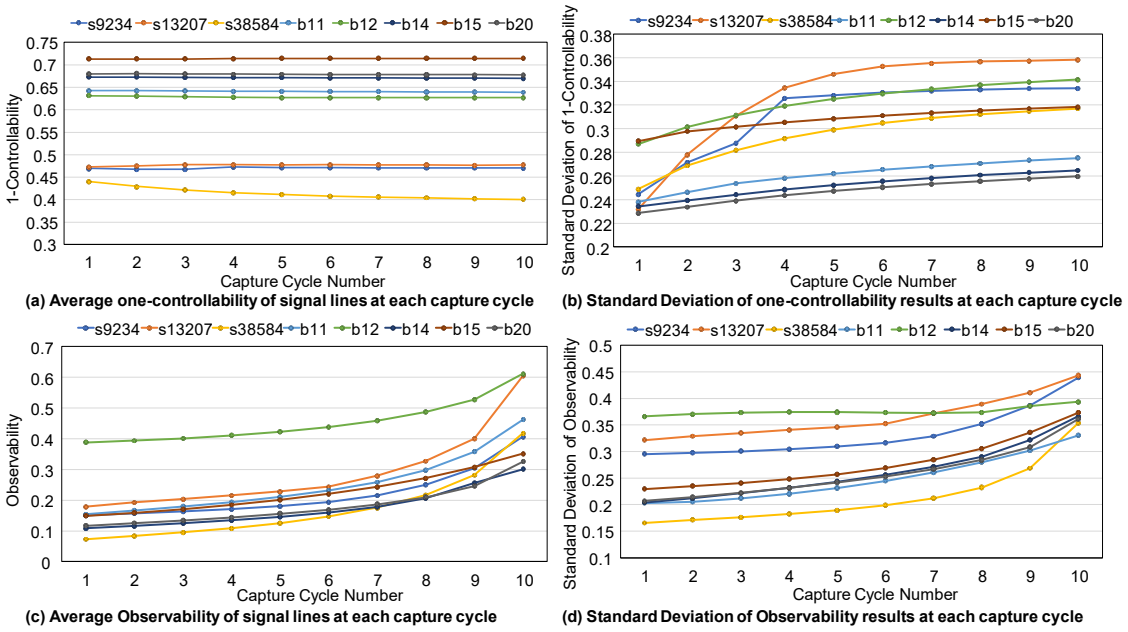


Figure 4.2 Testability vs. Capture Cycles

Based on the analysis presented above, we have determined that the primary factor that affects the effectiveness of the multi-cycle test in reducing the number of scan-in patterns and shortening the TAT of POST is the incompatibility between controllability and observability in the time-expanded circuit under multiple capture cycles. Specifically, we have found that the controllability bias of the signal line at earlier capture cycles is smaller than that of later cycles, resulting in lower observability. This difference in controllability bias and observability between earlier and later cycles can lead to reduced effectiveness of the multi-cycle test.

We insist that reconciling the incompatibility of testability under the multi-cycle test is necessary to improve the performance of multi-cycle BIST for scan-in pattern reduction.

Chapter 5

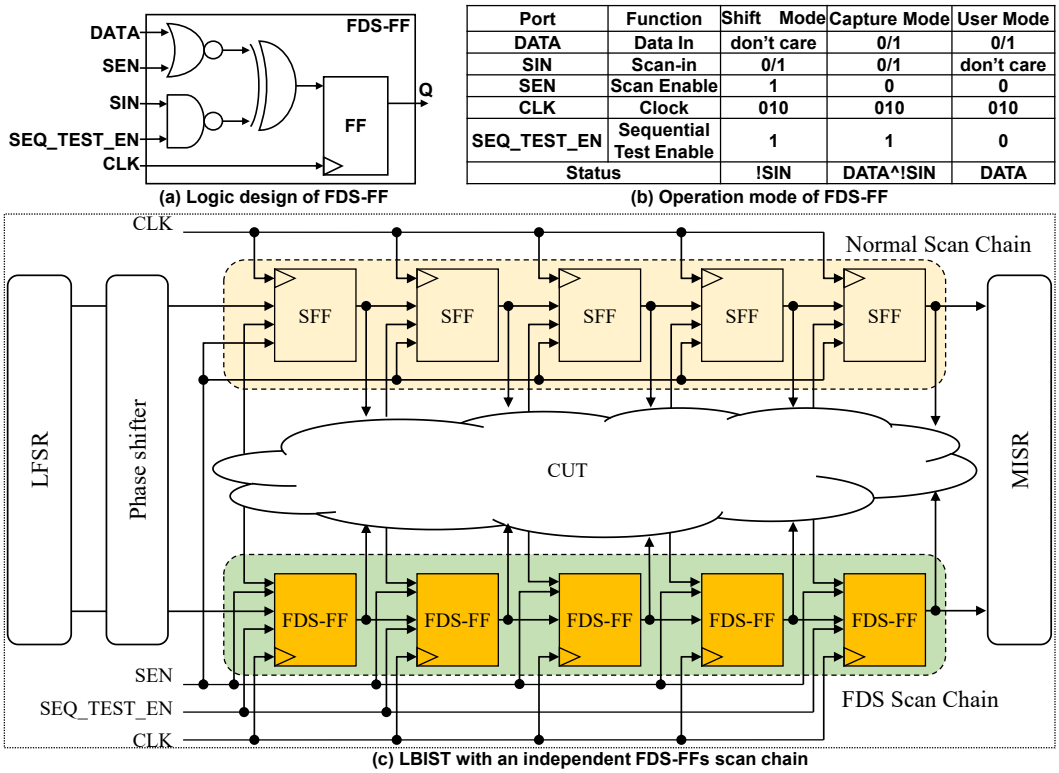
5. Test Point Insertion and Selection for Multi-Cycle BIST

5.1 Test Points for Multi-Cycle BIST

In this chapter, we introduce the observation point and control point proposed in our previous works [37][38][39][40][41] to address the Fault Masking and the FDD of the multi-cycle test, respectively.

5.1.1 Observation Point: FDS-FF

To improve the observability of scan FFs at the intermediate capture cycles in the time-expanded circuit, we proposed a new scan-cell design named *fault-detection-strengthened FF* (FDS-FF) that can directly observe and keep the value of FFs captured at each cycle.



Port	Function	Shift Mode	Capture Mode	User Mode
DATA	Data In	don't care	0/1	0/1
SIN	Scan-in	0/1	0/1	don't care
SEN	Scan Enable	1	0	0
CLK	Clock	010	010	010
SEQ_TEST_EN	Sequential Test Enable	1	1	0
Status		!SIN	DATA^!SIN	DATA

Figure 5.1 The DFT architecture of FDS-FF insertion for LBIST.

Figure 5.1 shows the structure of FDS-FF and the DFT architecture for LBIST with

FDS-FF insertion. The output of CUT denoted as DATA is controlled by *scan enable signal* (SEN) through a NOR gate, and a sequential test controls the scan-in test enable signal (SEQ_TEST_EN) through a NAND gate. The FDS-FF can work in three modes: *Shift*, *Capture*, and *User* mode as shown in the table, where *Shift* and *Capture* modes refer to as the Test mode, and *User* Mode refer to as the functional operation. The CUT will work at the user mode when SEQ_TEST_EN and SEN are set to 0. In Test Mode, SEQ_TEST_EN is set to 1, and the SEN signal controls the shift and the capture modes. If SEN=1, the test pattern is loaded into FF. If SEN=0, the corresponding test responses are captured. It should be noted that the test responses of CUT captured at each cycle are XORed with the SIN, which is the data stored in the neighbor FF in the scan chain. In this way, the test responses of each cycle can be compacted by the XOR gate, and the compacted test responses will be applied to the next capture cycle as a new test pattern. Since FDS-FFs observe and keep the capture response of CUT during the capture operation, in order to avoid functional timing issues, all FDS-FFs are extracted to constructed into a daisy chain and isolated from the other normal scan chains.

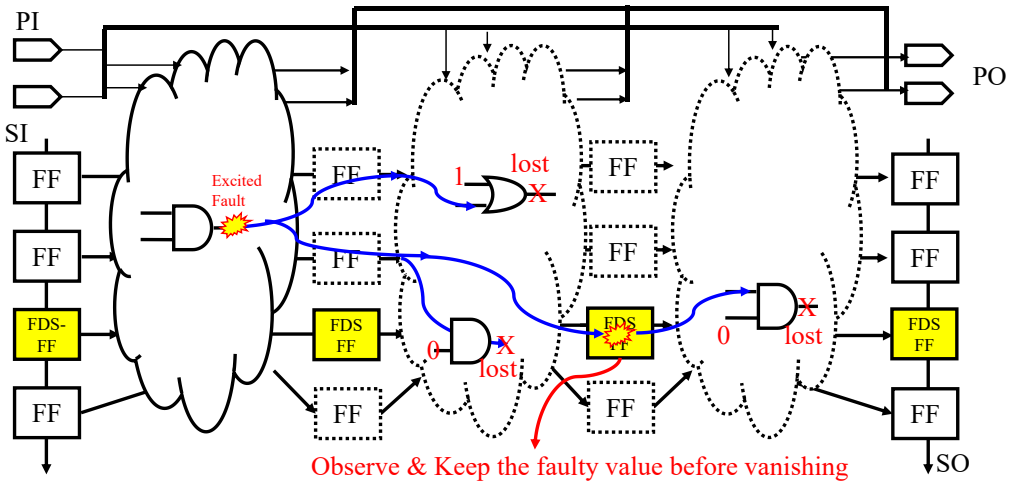


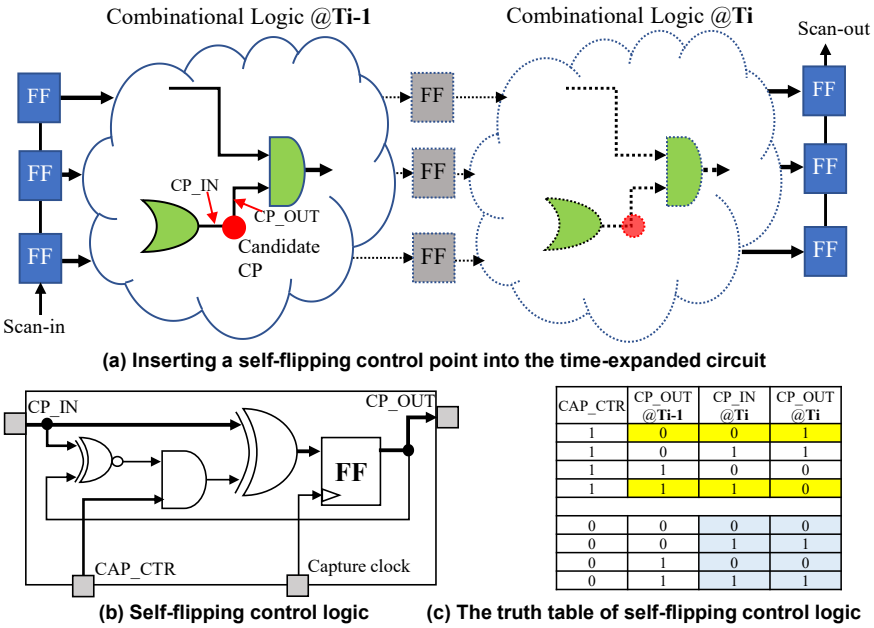
Figure 5.2 Replace a scan-FF with FDS-FF to address fault masking.

Figure 5.2 shows the effect of FDS-FF to address the fault masking problem. In a time-expanded circuit, some faulty values are successfully propagated to the FF at the intermediate capture cycles; however, they would be masked before the final capture. Replacing a scan FF with the FDS-FF is equivalent to inserting an observation point into the time-expanded circuit to observe and keep these faulty values before they are masked. However, it is impractical to replace all scan-FFs with FDS-FFs; the fault effects that never pass through the selected FDS-FFs may disappear if they cannot be propagated to the final capture cycle. Fortunately, replacing a small count of scan cells with FDS-FFs could achieve significant fault detection improvement [38], which is beneficial for low hardware overhead.

5.1.2 Control Point: Self-Flipping CP

Inserting control points into the circuit to force the target signal line to 0 (0-control) or 1 (1-control) is a popular way to improve the testability of the circuit. However, it would be challenging to adapt the conventional CPI to the time-expanded circuit under multi-cycle BIST because 1) CP with a fixed control value during a complete capture operation is less helpful to relax the controllability biasing; 2) generating the control values for each capture cycle requires complex sequential ATPG; 3) applying the dynamic control value to CP during the capture operation requires intricately designed control logic.

In [40][41], we have proposed an FF-CPI approach to improve the controllability of the time-expanded circuit by modifying the captured values of partial scan FFs at each capture cycle. The FF-CP compares the state of FF at the current capture cycle with its state at the previous capture cycle and changes the current state to its inverse value if no state transition occurs on the FFs during the capture cycles. In this study, we expand the FF-CPI approach to control the combinational logic and propose the control logic that can flip the value of the signal line of CP during the capture operation. We call it the Self-Flipping control in this study described as follows.



CAP_CTR	CP_OUT @Ti-1	CP_IN @Ti	CP_OUT @Ti
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1

Figure 5.3 Self-Flipping CP insertion for multi-cycle LBIST.

Figure 5.3 shows the design of the Self-Flipping control logic. In the capture mode, the present state (CP_OUT@T_{i-1}: the state of CP after the previous capture cycle) and the new state (CP_IN@T_i: the input value of the candidate CP of the current capture cycle) of the CP are checked whether there is a transition occurs in the current capture cycle or not. If not, the Self-Flipping control logic will generate the inverse value of the input state to

the CP output (CP_OUT@T_i). An external control signal “CAP_CTR” is used to enable the self-Flipping when set to 1. Otherwise, the input value of CP passes through the CP logic to the output. It thus can keep the value of a target signal line at the adjacent time-frame always different to relax the bias of 0/1-controllability caused by successive capture cycles.

It is worth noting that a traditional inversion CP using an XOR gate would be ineffective in reducing the 0/1-controllability bias because the 0/1-controllability of XOR-CP output depends on the input signal line, which is biasing as increasing the capture cycles. While inserting the Self-Flipping CP will cause hardware increase, it operates automatically during the capture cycles w/o any external control, which does not cause the extra cost in updating the ATPG to generate the deterministic control vectors.

To implement the proposed multi-cycle LBIST scheme for POST, the unknown values (Xs) generated as switching the operation mode from test and function need to be dealt with carefully. This issue can be addressed by separating the control logic of POST from the test target of POST (CUTs) through wrapper logic. During the test operation, the wrapped control logic of POST will be kept in function mode. When the test is completed, the CUTs will be switched to the functional mode by a system reset through the control logic of POST. The detailed solutions to handle the implementation issues for in-system-testing have been published in [42][43].

5.2 TP Selection for Multi-Cycle BIST

This section introduces the procedure to determine the locations of CPs and OPs to address the Fault Masking and the FDD problem induced by the controllability biasing and observation deterioration under a multi-cycle LBIST scheme.

While the selection procedure inherits some underlying techniques proposed in our previous works, such as the structure-based evaluation metric and the probabilistic testability analysis of circuits for FDS-FFs and FF-CP insertion, in this study, we consolidate them into a consistent process for TP selection under multi-cycle BIST through the following efforts:

- We propose a new metric to evaluate the effect of candidate signal lines for CP insertion under a multi-cycle BIST scheme.
- We introduce an improved probabilistic cost function for estimating the effect of candidate TPs.

- We introduce an OP Pruning approach into the TP procedure to improve the efficiency of TP selection under the multi-cycle BIST scheme.

It is worth noting that the proposed TP selection procedure conducts the probabilistic evaluation to identify the candidate CPs and Ops. The proposed TP selection procedure is a time-saving process because the procedure does not use the conventional fault simulation. As a result, we obtain the list of the TPs, then we evaluate the fault coverage achieved under the circuit with TPs by conducting the multi-cycle test fault simulation at one time.

5.2.1 A New Evaluation Metrics for CP Selection

As discussed in *Chapter 4*, increasing the number of capture cycles would cause a significant 0/1-controllability bias on signal lines at later capture cycles, which implies the value of more signal lines in a large capture cycle would most likely fix at 0 or 1 in most capture cycle. For a signal line *x*, if setting its value to 0(1) would cause fewer gates with fixed output value in its arrival logic region to FFs than that of setting to 1(0), 0(1)-controllability bias of *x* due to the multiple capture cycles would be helpful to fault excitation and propagation we call it positive bias, 1(0)-controllability bias would obstruct the fault detection called the negative bias. For the signal line shows a negative bias in controllability, it is suggested to insert a self-flipping CP to relax its controllability bias in the multi-cycle test. Following this, we propose the method to calculate the degree of the 0/1 controllability bias when inserting a CP into the signal line.

- *x: a signal line in the combinational circuit*
- $p_{x/0}$*: the probability of line x's value being logic 0*
- $p_{x/1}$*: the probability of line x's value being logic 1, where,* $p_{x/1}+p_{x/0}=1.0$
- $fg_{x/0}$*: the number of gates in the arrival logic region from line x to POs/PPOs whose output value will be fixed, as setting the value of x to 0*
- $fg_{x/1}$*: the number of gates in the arrival logic region from line x to POs/PPOs whose output value will be fixed by setting the value of x to 1*
- *BD(x): the degree of controllability bias at line x that would impact the fault detection, where BD(x)>0 denotes a positive bias, BD(x)<0 denotes a negative bias.*

$$BD(x)=(p_{x/0}-p_{x/1})\times(fg_{x/1}-fg_{x/0}) \quad (5.1)$$

- *CD(x): the degree of contribution to relax the controllability bias as forcing the 0/1-controllability of line x to 0.5/0.5. CD(x)>0 denotes a positive contribution, CD(x)<0 denotes a negative contribution that would be achieved by CP insertion.*

$$CD(x)=(p_{x/0}-0.5)\times fg_{x/0}+(p_{x/1}-0.5)\times fg_{x/1}=(0.5-p_{x/0})\times(fg_{x/1}-fg_{x/0}) \quad (5.2)$$

We use the s27 circuit as an example for illustration, see Figure 5.4. For signal line *i*, two paths connect with the PPO (FF2) through path 1: $i \rightarrow G5 \rightarrow G7 \rightarrow G8 \rightarrow w$, and

path 2: $i \rightarrow G6 \rightarrow G7 \rightarrow G8 \rightarrow w$. When the value of i is 1, the output of G5, G6, and G7 will be fixed at 1, 1, 0, respectively, thus $fg_{i/1}$=3. When i is 0, the output of G5 and G6 depends on the other input signal lines n and c, which implies a 0 value at i cannot directly cause any fixed gates on the two paths to FFs, thus $fg_{i/0}$=0. The probability of signal line i's values $p_{i/1}$ and $p_{i/0}$ can be calculated using the COP measurement, which is 0.25 and 0.75. The degree of controllability bias is hereby $BD(i)$ = 0.5×3 = 1.5, which represents that the controllability bias at signal line i is positive to fault detection.

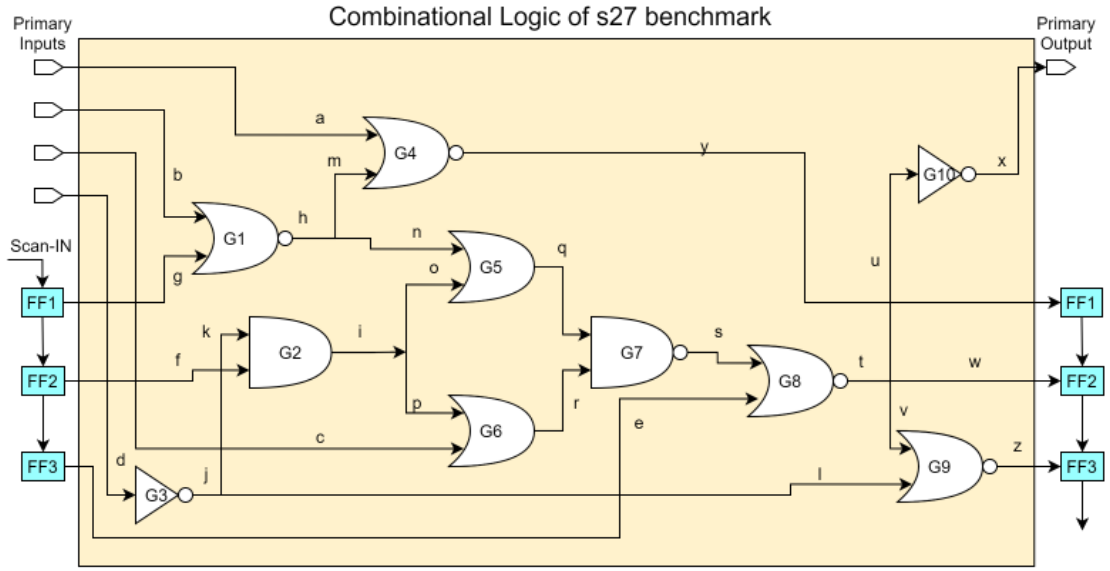


Figure 5.4 The combinational logic frame of s27 circuit.

For the signal line with positive controllability bias, inserting a CP would cause more fixed gates on the fault propagation paths to FFs with a negative contribution to fault detection, e.g., $CD(i)$=-0.25×3=-0.75. Table 5.1 gives the evaluation value of some signal lines shown in Figure 5.4. It can be observed that signal lines q and s show the negative controllability bias in BIST, and inserting a CP to s would achieve the most contribution to fault detection.

Table 5.1 Evaluation metrics of signal lines in s27

line #	$fg_{x/0}$	$fg_{x/1}$	$p_{x/0}$	$p_{x/1}$	*BD*	*CD*
h	0	2	0.75	0.25	1	-0.5
i	0	3	0.75	0.25	1.5	-0.75
n	0	1	0.75	0.25	0.5	-0.25
q	3	0	0.56	0.44	-0.36	0.18
s	0	2	0.27	0.73	-0.92	0.46

As shown in Figure 4.2, the controllability bias on signal line changes at different capture cycles in the multi-cycle BIST. It becomes larger as increase the number of capture cycles. Thus, the degree of controllability bias of signal line x: $BD(x)$ and the

contribution of CP insertion to relax the impact of controllability bias on fault detection: *CD(x)* can be easily extended for the multi-cycle test as follows.

$$BD(x) = \frac{fg_{x/1} - fg_{x/0}}{M} \sum_{j=1}^{M} (p_{xj/0} - p_{xj/1}) \quad (5.3)$$

$$CD(x) = \frac{fg_{x/1} - fg_{x/0}}{M} \sum_{j=1}^{M} (0.5 - p_{xj/0}) \quad (5.4)$$

where $p_{xj/1}$ and $p_{xj/0}$ denote the probability of line x's value being logic 1/0 at the *j*th capture cycle. We use CD as the evaluation metrics for searching the candidate signal lines for CP insertion under multi-cycle BIST, which is described in the next section.

5.2.2 TP Selection Procedure for Multi-cycle BIST

The procedure consists of two phases, Phase 1: CP insertion under a time-expanded circuit with full FF-observation, and Phase 2: OP is pruning to remove the impotent observation points (FDS-FF).

In Phase 1, the CP selection will be performed at the time-expanded circuit with full observation where the FFs at intermediate capture cycles are supposed to be observable. This is because the purpose of self-flipping CP insertion is to relax the controllability bias caused by functional operation at each time frame, but not to create long propagation paths that can cross multiple time frames to the final capture cycle for observation which is an arduous task. The algorithm for CP insertion is shown in *Algorithm 1*.

To evaluate the quality of CPs and OPs, cost function *U* as follows is widely used in various TPI techniques.

$$U = \frac{1}{|F|} \sum_{\forall x/s \in F} \frac{1}{Pd_{x/s}} \quad (5.5)$$

In this work, we expand the cost function considering the fault detection model under the multi-cycle BIST scheme, where the detection probability of the faults at a signal line denoted by $Pd_{x/s}$ is calculated by

$$Pd_{x/s} = 1 - \prod_{j=1}^{M} (1 - (1 - C_{xj/s}) \times O_{xj}) \quad (5.6)$$

$C_{xj/s}$ and O_{xj} denote the *s-controllability* and observability of signal line *x* at the *j*th time-frame, respectively, computed by COP measure as discussed in *Chapter 4*. The difference in *U* before (U^{org}) and after (U^{tp}) inserting a TP can be calculated by the following equation to identify the most effective TP from a candidate TP list.

$$\Delta U = U^{org} - U^{tp} = \frac{1}{|F|} \sum_{\forall i/s \in F} \left(\frac{1}{Pd_{i/s}^{org}} - \frac{1}{Pd_{i/s}^{tp}} \right) \quad (5.7)$$

Algorithm 1: CP insertion

Inputs:
net: original CUT netlist
M: the number of capture cycles
N_{cp}: Maximum number of CPs
Outputs:
cplist[N_{cp}]: Selected CP list
net_{cp}: CUT with CP insertion
Optional parameter:
$CR_{threshold}$: the threshold of cost reduction (≥0)
N_{cand}: number of candidate CP at each iteration of CP decision
Process:
1: *cplist* ← ∅
2: *cand* ← ∅ */*Candidate CP list for determining a CP*/*
3: *read_circuit (net);*
4: *fix_gate_cal(net); /*computing $fg_{x/0}$, $fg_{x/1}$*/*
5: *time_expansion (net, M);*
6: *full_observation_point_insertion(net);*
7: **while** |*cplist*|< N_{cp} **do**
8: *cop_controllability(net_{cp});*
9: *cop_observability(net_{cp});*
10: *CD_calculate (net_{cp});*
11: *cand* ← ∅*;*
12: **for** *j=1 to # of available candidate CP* **do**
13: *cand[j]* ←*unchecked signal line with the largest CD;*
14: **end for**
15: **if** *cand*=∅ **then**
16: **return** *cplist, net_{cp};* **stop the process**
17: **else**
18: U^{org}=*cost_computation(net_{cp},M);*
19: **for** *k=1 to N_{cand}* **do**
20: net_{cp}=*insert cand[k] cp to net_{cp};*
21: *update_controllablity_observability($net_{cand[k]}$, M);*
22: $\Delta U_k = U^{org}$-*cost_computation(net_{cp}, M);*
23: *Remove cand[k] from net_{cp};*
24: **end for**
25: **If** *maximum(ΔU_k)>= $CR_{threshold}$* **then**
26: *cplist[i]*←*cand[k];*
27: **end if**
28: **end if**
29: *insert cplist[i] to net → update net_{cp};*
30: **end while**
31: **return** *cplist, net_{cp};*
End process

In Phase 2, we will remove the impotent observation points from the time-expanded circuit to reduce the hardware overhead caused by FDS-FFs insertion, named OP pruning, as shown by *Algorithm 2*. In OP pruning, the input is the time-expanded circuit with CP insertion achieved at Phase 1 where all scan FFs are replaced with FDS-FFs for observation during multi-cycle captures. We target on reducing the amount of FDS-FFs to a user-specified number N_{op} by restoring the FDS-FFs that do not affect the fault detection to scan FFs. As shown from line 14 to line 24 of algorithm 2, we compute the cost U of each candidate FDS-FF when temporarily changing it to a scan FF which is

equivalent to a wire in the intermediate time-frame circuit, and remove the one which has the least cost increase from the OP list.

The OP pruning is considered effective based on the following observations. 1) The large number of signal lines usually can be observed by multiple FFs; 2) The FFs which have larger observable logic regions could observe more fault effects. 3) The number of FFs in a design is much smaller than that of signal lines, and exploring the inactive FFs would be more time-saving than inserting OPs into the CUT.

Algorithm 2: OP Pruning

Inputs:
net_{cplist}: CUT with CP insertion
M: number of capture cycles
N_{op}:Target number of OPs (FDS-FFs)
Outputs:
oplist[N_{op}]: FF list for FDS-FFs insertion
net_{tp} : CUT with TPs (CP and OP)
Optional parameter:
N_{cand}: # of candidate OPs at each iteration for OP pruning
Process:

```
1:  oplist ← ∅
2:  cand ← ∅   /* Candidate target OP list for pruning */
3:  read_circuit (net_cp);
4:  oplist← all FFs
5:  structure_analysis(net_cp);
6:  FF_ranking(oplist) /*Ranking the FFs by the approximate evaluation metrics proposed in [39]*/
7:  while |oplist|>N_op do
8:      for j=1 to # of available candidate OP do
9:          cand[j] ←select an OP in the oplist in descending order which is unchecked;
10:     end for
11:     if cand=∅ then
12:         return oplist, net_tp; stop the process
13:     else
14:         U^org=cost_computation(net_tp,M);
15:         for k=1 to N_cand do
16:             remove cand[k] OP from net_tp;
17:             update_observability(net_tp);
18:             ΔU_k=U^org-cost_computation(net_tp, M);
19:             restore cand[k] OP to net_tp;
20:         end for
21:         If ΔU_k =minimum then
22:             remove cand[k] OP from net_tp;
23:             remove cand[k] from oplist;
24:         end if
25:     end if
26: end while
27: return oplist, net_tp;
End process
```

5.3 Experimental Results

Experiments are conducted on ISCAS89 and ITC99 benchmark circuits to evaluate

the effect of TPI under multi-cycle BIST. A 16-bits internal type LFSR (characteristic polynomial: $X^{16}+X^{15}+X^{13}+X^4+1$) with Phase Shifter generates pseudo-random patterns. A parallel scan structure is introduced into the CUT that consists of multiple scan chains up to 100 FFs in length (when the total number of FFs > 1600, the maximum length of the chain is up to 200). A multi-cycle BIST logic/fault simulator that can simulate at most 50 cycles capture per pattern is implemented in-house for **stuck-at faults** testing. For automotive ICs, the ISO26262 functional safety standard imposes at least 90% latent fault metric (permanent fault) to meet the safety goal ASIL D. Therefore, in this study, we set a target fault coverage 90% and evaluate the effect of the proposed multi-cycle TPI that would make the classical on-chip pseudo-random TPG-based LBIST comply with the ISO26262 standard. Table 5.2 gives the details of CUTs.

Table 5.2 Detailed information of benchmark circuits

Circuit	# gate	# FF	# of stuck-at fault	N_{cp} (<1% of gates)	N_{cp} (<5% of FFs)	#OPs (FDS-FFs) (<20% of FFs)
s9234	5597	228	6927	55	11	45
s13207	7951	669	9815	79	33	133
s15850	9772	597	11725	104	29	120
s38417	22179	1636	31180	1141	85	327
s38584	19253	1452	36303	97	72	290
b11	437	31	1322	2	1	6
b12	904	121	2797	9	6	24
b14	4444	245	12811	44	12	49
b15	8338	449	23528	8	8	89
b17	22645	1415	65464	201	70	283
b20	8875	490	25338	88	24	98

5.3.1 Evaluation of the Efficiency of the Multi-Cycle Test

We first performed fault simulations on the regular scan testing with single capture (SCAN) and multi-cycle testing with 2, 4, 6, 8, and 10 capture cycles, respectively, to evaluate the effect of multi-cycle testing for fault detection, using 100k test patterns (scan-in) generated by LFSR. **Figure** 5.5 shows the fault coverage of each circuit at different capture cycle test when 100k patterns are applied. It can be seen that for most circuits, multi-cycle test achieved an increase in fault coverage at 2 and 4 capture cycle. As continuous increasing the capture number to 10 cycles, the increment of fault coverage is slowing down or getting degraded. For s9234, multi-cycle test shows significant decrease of fault coverage. It can be explained by the incompatibility of testability shown in Figure 4.2, where 10-cycle test caused a little controllability bias, however, significant deterioration of the observability in the expanded circuit.

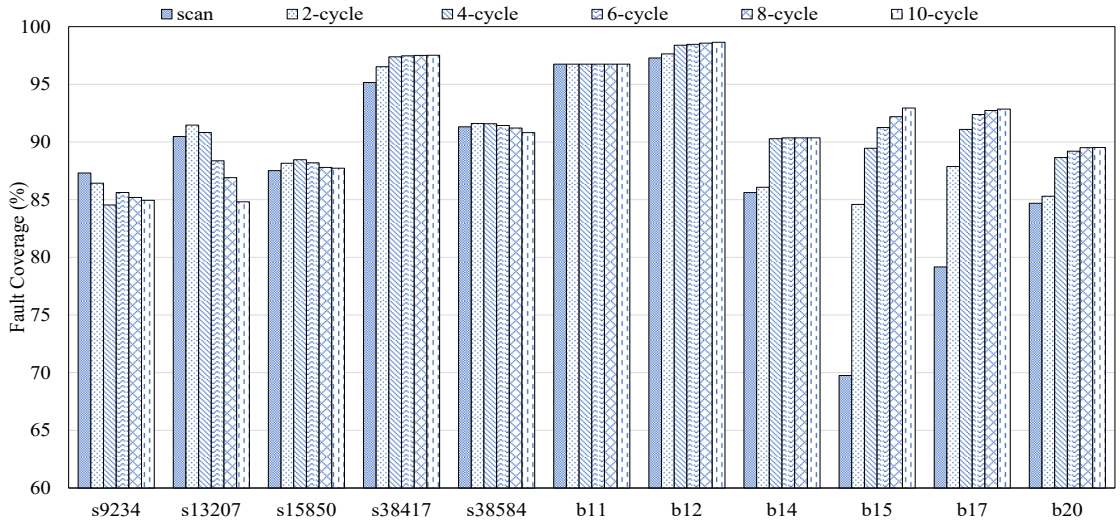


Figure 5.5 Fault coverage of benchmark circuit with 100k patterns.

Figure 5.6 shows the curve of the average fault coverage of all CUTs by increasing the number of patterns. The horizontal and vertical axis shows the pattern number and the corresponding fault coverage, respectively. The average fault coverage of all CUTs confirms that multi-cycle test has a statistical improvement in fault detection for most benchmark circuits compared with scan testing (SCAN). Applying 4 and 6 capture cycles achieved the most fault coverage improvement, and the increment of fault coverage becomes less as the number of capture cycles increases to 8 and 10 cycles.

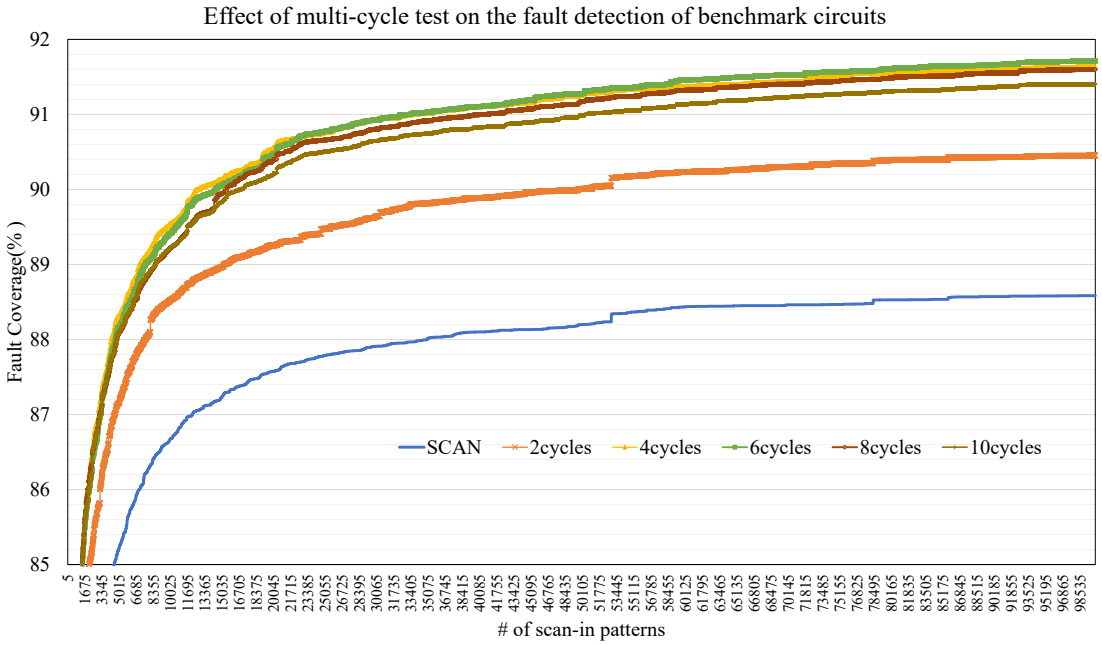


Figure 5.6 Scan testing vs multi-cycle testing.

The above observations fit the basic feature of multi-cycle testing discussed in *Chapter 4*. Multiple capture operations would provide more detection opportunities for

the fault that the scan-in pattern cannot detect. Therefore, it is possible to improve the fault detection of the scan-in pattern. However, the deterioration of testability of signal lines in the time-expanded circuit would interfere with future fault detection as increasing the capture cycles. To reinforce the effect of multi-cycle BIST on scan-in pattern reduction, CP insertion and FDS-FFs insertion are introduced, and their effects are described as follows.

5.3.2 Evaluation of the Efficiency of the CPI and the OPI

We conducted the CP selecting and OP pruning algorithm proposed in Section 5.2 on the benchmark circuits to identify a specified amount of CPs and FDS-FFs, where the maximum number of CPs N_{cp} was set to 1% of the gate number of CUT, and 5% of the FFs, respectively. The expected amount of FDS-FFs is set to 20% of the total number of FFs in the CUT. The number of CPs and FDS-FFs are shown in the fifth and the sixth column of Table 5.2, respectively. To demonstrate the difference between OPI and CPI, we performed fault simulation under 10 cycles by individually inserting the identified CPs and OPs into the CUTs, denoted by CP-ONLY and OP-ONLY, respectively. CPI&OPI denotes inserting both the CPs and OPs into the CUTs. Figure 5.7 shows the curve of average fault coverage of the benchmark circuits, increasing scan-in patterns to 100K under different DFT strategies. The figure only presents the curves up to 50K patterns for demonstration. Full observation replaces all FF with FDS-FFs has been conducted on the CUTs w/(w/o) CP insertion denoted by FullOB, CP&FullOB, respectively. While it is not for practical use due to hardware overhead concerns, the results represent the upper bound of the fault coverage possibly achieved by OPI, which is used to evaluate how far the OP pruning has reached in identifying the OPs for FDS-FFs insertion.

Compared to the regular scan test (SCAN), the multi-cycle test w/o TPI denoted by *"10-Cycle"* in the figure achieved a significant fault coverage improvement where a target fault coverage (90%) is attained by applying 14,260 scan-in patterns under a 10-cycle test, which cannot be achieved by scan testing even with more than 100K scan-in patterns. Replacing 20% of FFs with FDS-FFs (OPI-ONLY) further improved the fault coverage and reduced the necessary scan-in patterns to 5,175 for the target 90% fault coverage as well as the effect achieved by full observation (replacing all scan FFs with FDS-FFs). The results demonstrate the effect of FDS-FFs insertion that directly observes the values of FFs at each capture cycle to relax the fault masking problem in the time-expanded circuit.

Note the fault coverage curve of CPI-ONLY in Figure 5.7, where we inserted 1% of

the gate number of CPs into the CUTs, and it shows almost the same fault coverage improvement (≈93%) and much more scan-in pattern reduction (2,195 for 90% fault coverage) than that of OPI-ONLY. The results indicate 1) inserting Self-Flipping CP into the CUT that relaxes the controllability bias in a time-expanded circuit is helpful in improving the fault detection of capture patterns 2) the effect of CPI is limited due to the fault masking problem in the time-expanded circuit. When inserting both the identified CPs and OPs into the CUTs denoted by CPI&OPI, we achieved a sharp increase in fault coverage compared to inserting CPs and OPs individually. The final fault coverage of 100K scan-in patterns increases to 95.01%. The number of scan-in patterns for achieving 90% fault coverage is drastically reduced to 585 (24.4X reduction compared to the multi-cycle test). Note the fault coverage curve of 10-cycle (multi-cycle test) and the CP&FullOB, which represents the upper bound of the fault coverage possibly achieved by CPI&OPI, inserting both the CPs and OPs (CPI&OPI) identified by our proposed method achieved remarkable fault coverage increase and pattern reduction that is very close to the upper bound.

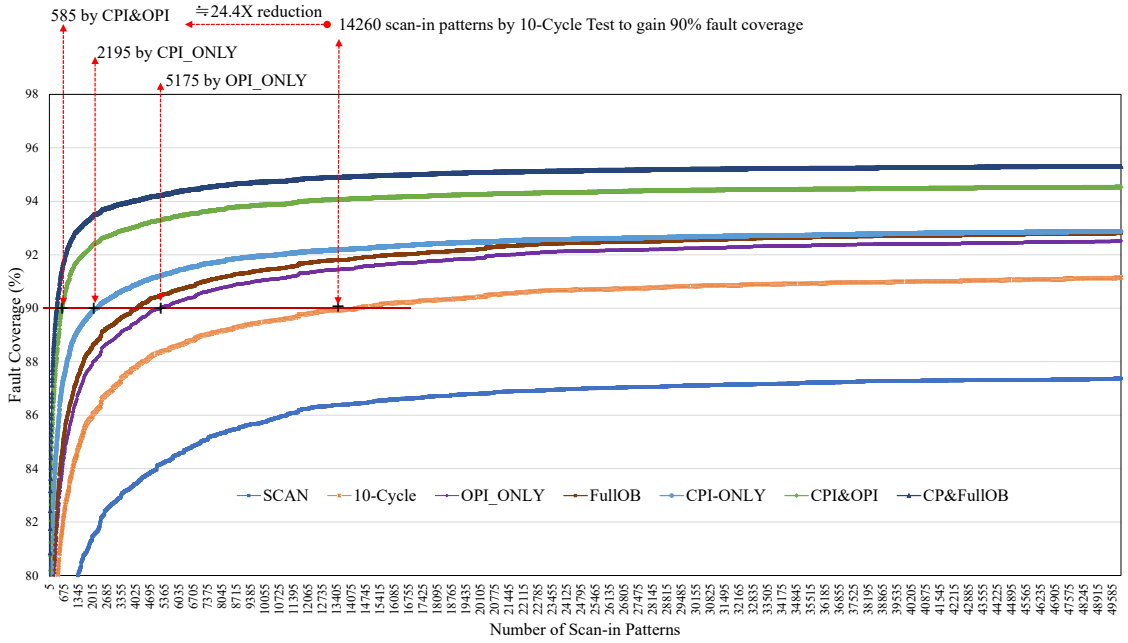


Figure 5.7 Fault coverage vs. Pattern number (scan testing, multi-cycle testing, OPI and CPI under multi-cycle testing).

Table 5.3 and Table 5.4 show the detailed results of the final fault coverage achieved by applying 100K patterns and the number of scan-in patterns for attaining 90% stuck-at fault coverage, respectively. The experimental results when inserting fewer CPs (<5% of FFs) into the benchmark circuits are also presented in the tables. The results show that the 10-cycle test would cause the fault coverage loss in most ISCAS89 circuits (s9234,

s13207, s38584); however, it achieves a significant increase in ITC99 benchmark circuits. Where ISCAS89 circuits show a much more testability bias, increasing the capture cycles are vulnerable to fault masking and FDD problem. While inserting OP or CP individually both improved the fault coverage and reduce the patterns for attaining 90% fault coverage for all circuits, it suggests that combining the CPs and OPs can achieve the most pattern reduction under the multi-cycle BIST scheme. Reducing the number of CPs causes a corresponding degradation in the fault coverage and the scan-in pattern reduction, however, the degradation is small, e.g., when reduce the number of CPs from 201 to 70 for b17 circuit, the fault coverage with 100k patterns decreased from 97.81% to 96.29%, scan-in patterns for 90% fault coverage increased from 130 to 185. The reduction of scan-in patterns compared to the multi-cycle test is remarkable for shortening the TAT of POST with less hardware overhead.

Table 5.3 The final fault coverage reached by 100K scan-in patterns

Circuit	Design for Testability Approaches											
	SCAN	10-Cycle	OPI_ONLY	FullOB	# of CP<1% of gates				# of CP<5% of FFs			
					# of CPs	CPI-ONLY	CPI&OPI	CP&FullOB	# of CPs	CPI-ONLY	CPI&OPI	CP&FullOB
s9234	87.31	84.94	89.94	90.00	55	82.69	89.68	91.80	11	83.3	87.96	88.02
s13207	90.47	84.81	92.20	92.96	79	86.16	92.75	93.89	33	85.6	90.1	91.01
s15850	87.51	87.73	88.48	90.18	104	85.09	87.41	91.52	29	86.47	87.71	90.77
s38417	95.16	97.52	97.96	98.03	141	98.19	98.66	98.72	85	98.00	98.55	98.62
s38584	91.31	90.81	91.59	92.07	97	91.16	91.70	92.28	72	90.27	90.93	91.53
b11	96.75	96.75	96.75	96.75	2	98.03	98.03	98.03	1	96.82	96.82	96.82
b12	97.28	98.64	98.68	98.68	9	99.18	99.21	99.21	6	97.6	97.6	97.64
b14	85.61	90.36	90.38	90.40	44	93.71	93.96	94.07	12	93.81	94.04	94.08
b15	69.75	92.94	92.95	92.95	8	98.35	98.36	98.36	8	98.35	98.36	98.36
b17	79.17	92.85	92.85	92.86	201	97.76	97.81	97.83	70	96.27	96.29	96.32
b20	84.69	89.52	89.66	89.69	88	93.10	93.61	93.94	24	92.68	93.17	93.22

Table 5.4 The number of scan-in patterns to achieve 90% fault coverage

Circuit	Design for Testability Approaches											
	SCAN	10-Cycle	OPI_ONLY	FullOB	# of CP<1% of total gates				# of CP<5% of FFs			
					# of CPs	CPI-ONLY	CPI&OPI	CP&FullOB	# of CPs	CPI-ONLY	CPI&OPI	CP&FullOB
s9234	>100K	>100K	>100K	>100K	55	>100K	>100K	9180	11	>100K	>100K	>100K
s13207	20560	>100K	11565	7375	79	>100K	6050	4175	33	>100K	59835	8885
s15850	>100K	>100K	>100K	68905	104	>100K	>100K	2380	29	>100K	>100K	4710
s38417	5780	1710	590	460	141	250	80	55	85	310	85	60
s38584	8180	10645	3700	1960	97	1555	575	305	72	12795	960	330
b11	475	120	120	120	2	45	40	35	1	115	115	105
b12	1280	175	170	170	9	100	45	45	6	260	210	195
b14	>100K	58280	58280	53425	44	1285	870	770	12	885	675	605
b15	>100K	4180	4180	4115	8	285	230	170	8	285	230	170
b17	>100K	4305	4300	4300	201	180	130	100	70	260	185	140
b20	>100K	>100K	>100K	>100K	88	4330	2045	935	24	7480	3740	3685

5.4 Conclusions

The multi-cycle BIST has room for improvement to reduce the volume of scan-in patterns. Therefore, this study investigated the stuck-at fault detection model in the time-expanded circuit.

We revealed that the incompatibility between the controllability and observability of signal line as increasing the capture cycles would induce the fault masking and fault detection degradation problem. Those problems obstruct the effect of multi-cycle tests to test pattern reduction. We introduced the TPI technique to a multi-cycle LBIST scheme focused on reducing the volume of scan-in patterns for a target fault coverage to address this issue. The TPI approach replaces partial scan cells with FDS-FF referred to as OPI to enhance the observability and inserts Self-Flipping control logic into the combinational logic referred to as CPI to relax the controllability bias of signal lines of CUT at the intermediate capture cycles.

To identify the TPs that could achieve the most scan-in pattern reduction, we proposed a metric called the CD (the degree of contribution to relax the controllability) to evaluate the effect of candidate CPI signal lines and introduced an improved probabilistic cost function for estimating the effect of CP and OP insertion under multi-cycle BIST scheme. A TPI procedure including CP insertion and OP pruning is also proposed to identify the effective TPs to achieve the most scan-in pattern reduction. The experimental results on ISCAS89 and ITC99 benchmarks show 24.4X pattern reduction on average that confirming the effectiveness of the proposed TPI for shortening the test application time of POST.

In the future work, we will implement the proposed TP selection algorithm to support the industrial design, to evaluate the effectiveness of the multi-cycle LBIST scheme on the commercial automotive ECUs.

Part III: Test to Memory-based Programmable Logic Device

Reconfigurable devices (e.g., FPGAs: field-programmable gate arrays) allow users to customize the functions in-field that provides a flexible (custom logic and routing) and scalable (add new functions) platform for system development, with faster development cycle time (better time-to-market), low design cost (e.g., IP reuse), high-performance (high-speed hardware), and long-term maintenance (update function). Benefiting from such abilities, FPGAs have gone successfully for many applications such as the IoT (internet of things) [44], SDV (self-driving vehicle) [45], and AI (artificial intelligence) [46].

However, FPGAs suffer from area, power, and delay due to programmable interconnect resources [47][48]. Large amounts of interconnect resources also require multi-layer wiring architecture and advanced manufacturing technology that causes significant production costs. Large area, power, delay, and production cost issues prevent the FPGA from more.

Recently, a new type of reconfigurable device called MPLD (memory-based programmable logic device) [49] is under development for edge computing devices in IoT and AI applications. In contrast to FPGAs, which require large amounts of programmable interconnect resources to achieve programmability, MPLD is constructed only with an array of MLUTs (multiple look-up tables) without any extra programmable interconnect resources. An MLUT is the essential reconfigurable element constructed using general SRAMs and connects with its neighbors via Address-inputs/Data-outputs called AD interconnects. Users can configure wires and logic into MLUTs by writing the corresponding truth tables into the SRAMs. This feature enables high-density reconfigurable devices with low production costs, low power consumption, and minimal delay.

To guarantee the long-term reliability of MPLD, extensive production testing with high quality is first required to identify as many manufacturing defects as possible in the MLUT array. When the device is operating in the field, various hard-to-predict factors, such as aging phenomena [50][51], and environmental factors including operating temperature, power supply, noise, etc., can cause delay degradation in the MLUT array of MPLD and threaten its long-term reliability [52].

In this part, we focus on the issues that would affect the long-term reliability of MPLD. We propose a test method to address these reliability concerns to detect and identify interconnect defects in the MLUT array during the production phase. We also propose a delay monitoring technique to detect aging-caused failures in the field.

The proposed test method creates route maps in MPLD for fault propagation by configuring pre-designed test cubes into the SRAM array, it then excites faults by applying an external walking-zero/one vector to the external input ports of MPLD and identifies any faults through fault effects propagated to the external output ports. The delay monitoring method configures a novel ring oscillator (RO) logic design into MPLD to measure aging-induced delays. We designed an MPLD with a 6×6 MLUT array to evaluate the proposed methods by performing logic simulations. The simulation results with fault injection confirmed the effectiveness of the proposed methods.

The main contributions of this part are as follows.

1. *We explore the fault models of interconnect defects within the MLUT array of MPLD.*
2. *We propose approaches to test stuck-at faults and bridge faults caused by interconnect defects in the MLUT array during MPLD production. This contributes to high reliability and yield improvement.*
3. *We propose a test method to accurately identify the location of faults. The proposed test method improves the manufacturing process and enables the avoidance of faulty MLUT blocks, thereby ensuring high reliability when the MPLD is put into practical use.*
4. *We investigate the reliability issues induced by aging when the MPLD operates in the field and propose a monitoring technique to measure the aging-induced delay variations by configuring a novel ring oscillator logic design into MPLD.*

The remainder of this part is organized as follows: Chapter 6 introduces the architecture of MPLD and its basic working principle. Chapter 7 discusses the reliability concerns in the lifecycle of MPLD. Chapter 8 proposes the production test solution for interconnect defects. Chapter 9 presents the delay monitoring method.

Chapter 6

6. Memory-based Programmable Logic Device (MPLD)

This chapter gives an introduction to the architecture of the MPLD and its working principle, in order to serve as a basis for the work that follows in *Chapters 7*, *8*, *9*, and *10*.

The rest of this chapter is organized as the following: *Section 6.1* introduces the architecture of the MPLD by describing detailed its structure, main component elements, and operation functions of these elements. *Section 6.2* describes the working principle of the MPLD through two examples of configuring logic circuits to a single MLUT and multiple MLUTs, respectively. Finally, the chapter concludes in *Section 6.3*.

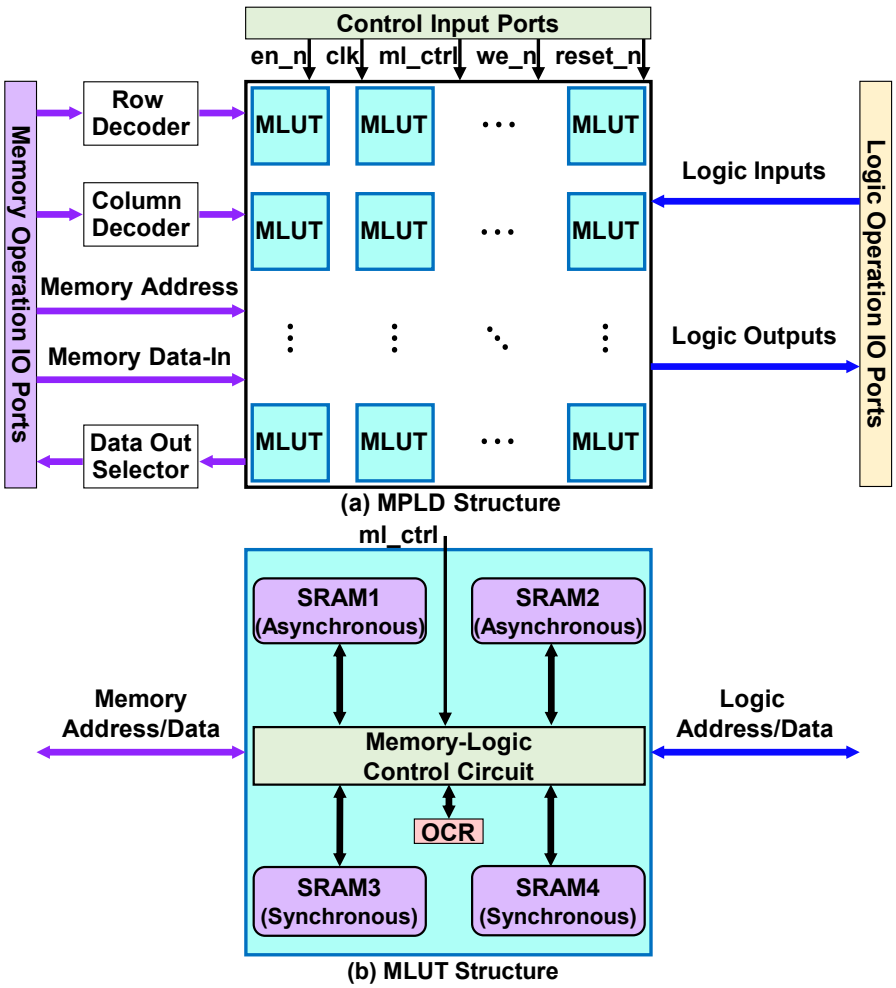


Figure 6.1 MPLD Architecture.

6.1 MPLD Architecture

6.1.1 MPLD Structure: MLUTs (Multiple Look-up Tables) Array

Figure 6.1(a) shows the structure of the MPLD. it is constructed by an array of reconfigurable cells named the MLUTs (*multiple look-up tables*). The MLUTs array can work in either two operation modes: *memory* or *logic* operation mode, by manipulating a *memory-logic control* signal *ml_ctrl* at an external input port of the MPLD. In memory operation mode, a *row decoder, column decoder*, *memory address bus*, *memory data-in bus*, and *data out selector* are used to configure (read out) data into (from) the MLUT in the array, via external *memory operation IO (input/output)* ports. In logic operation mode, configured data can function logically in the array, driven by the external *logic operation input* ports, and the function results are out to the external *logic operation out* ports.

Figure 6.1(b) shows the structure of the MLUT. Which is constructed by two *asynchronous SRAMs* (SRAM1, SRAM2), two *synchronous SRAMs* (SRAM3, SRAM4), a *memory-logic control circuit,* and an *output control register* (OCR). The asynchronous SRAMs are used to create combinational logic functions, and the synchronous SRAMs are used to create sequential logic functions. The memory-logic control circuit, driven by the *ml_ctrl* signal, controls the memory and logic operation of the MLUT. The OCR is used to control the logic data output of the MLUT.

6.1.2 MPLD Memory Operation Mode

The MPLD works in the memory operation mode when by setting the value of the *ml_ctrl* signal to 0; this operation mod includes two operation options: the configuring operation and the reading operation. In the configuring operation, the data contents in the SRAMs and OCR of each MLUT can be configured. In the reading operation, the data contents in the SRAMs can be read out.

Figure 6.2 shows the schematic of the MPLD working in the memory operation mod. A *we_n* signal handles the two operation options; setting the value of the *we_n* signal to 0 allows the configuring operation, and conversely, setting it to 1 allows the reading operation.

For the MPLD with a size of $X \times Y$ (X columns and Y rows) MLUTs array, there are log_2X-bit *column selection input ports*, *mlut_x*, and log_2Y-bit *row selection input ports*, *mlut_y*, are used to select an MLUT to be configured or read in the array, via respectively the column decoder and row decoder, and are used to select the data output of an MLUT from the array via the data out selector. The column decoder and row decoder respectively generate *X-bit column enable signals* and *Y-bit row enable signals*, *mlut_c[X-1:0]* and *mlut_r[Y-1:0]*, to enable an MLUT in the array; where if the values both of *mlut_c[i]* and

mlut_r[j] are 1, the MLUT x_iy_j (where $i \in [0, X-1], j \in [0, Y-1]$) is valid for the configuring or reading operation. A *memory address input ports* provide the memory address bus, *mad*, directly connecting to each MLUT, to access the SRAMs or OCR in the MLUT, for specifying a target address for the configuring or reading operation. In the configuring operation, a *memory data input ports* provide the memory data-in bus, *mdata_i*, directly feeding to each MLUT for supplying the data contents configured to the target address of the SRAMs or OCR specified by *mad*. In the reading operation, a *memory data out ports, mdata_o*, via the data out selector, out the data contents at the target address of the SRAM specified by the *mad*, from the MLUT selected by *mlut_x* and *mlut_y*.

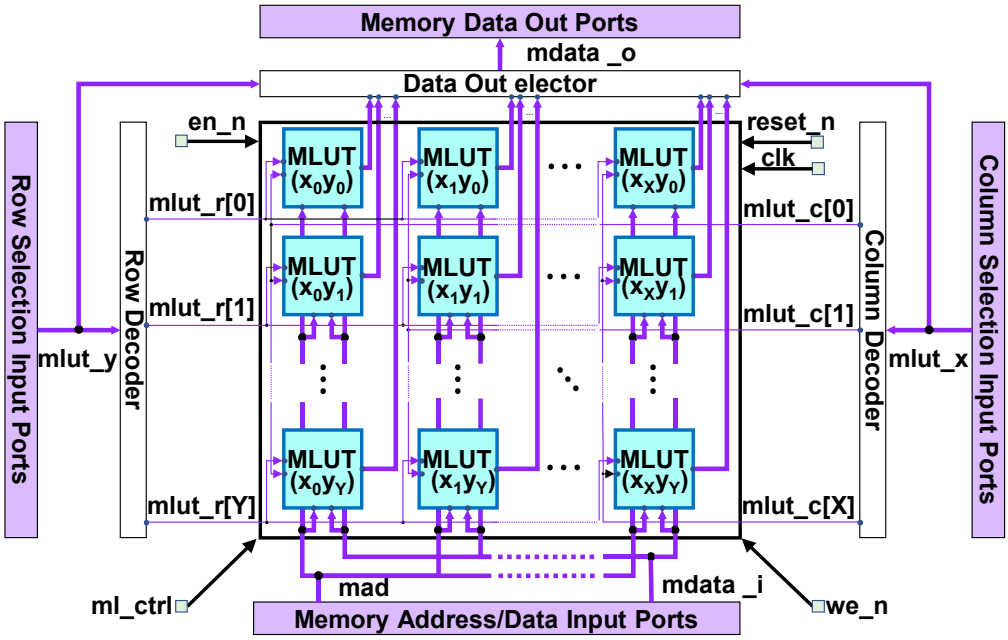


Figure 6.2 Schematic of MPLD working in memory operation mod.

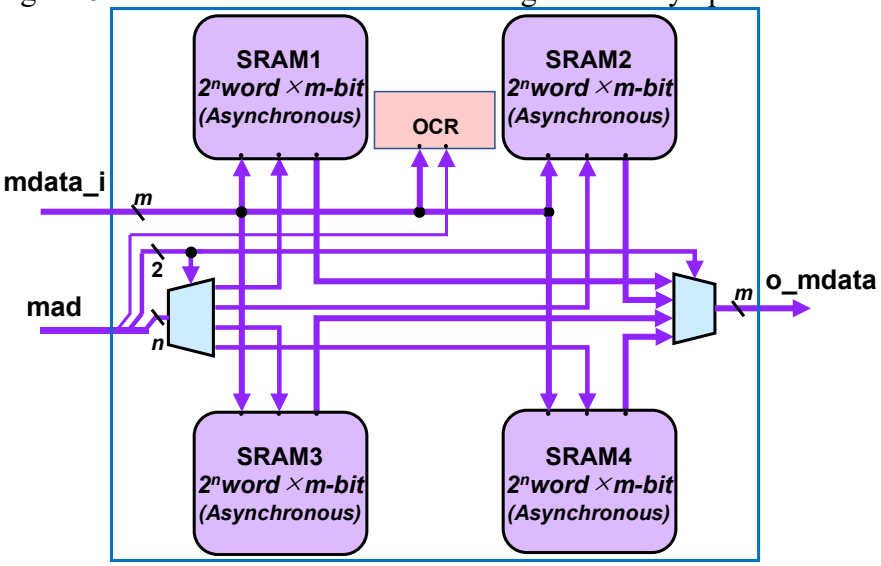


Figure 6.3 Schematic of MLUT working in memory operation mod.

Figure 6.3 shows the schematic of the MLUT working in the memory operation mod. Where for the SRAM with the size of $2^n word \times m\text{-}bit$, the *mad* are *n+3 bits*; both the *mdata_i* and the *mdata_o* are *m bits*. The highest bit of the *mad*, $mad_{[n+2]}$, is used to select the OCR; the two bits of the *mad*, $mad_{[n+1:n]}$, are used to specify an SRAM from four SRAMs; and the remainder n-bits of the *mad*, $mad_{[n-1:0]}$, are used to specify the address of the SRAM specified selected. The m-bit data contents in a specified target address can be configured and read via the m-bit memory data inputs *mdata_i* and the m-bit memory data outputs *o_mdata*, respectively.

6.1.3 MPLD Logic Operation Mode

The MPLD works in the logic operation mode when by setting the value of the *ml_ctrl* signal to 1; in this mode, the configured data can function logically in the MLUTs array. Figure 6.4 shows the schematic of the MPLD working in the logic operation mod. In the MLUTs array, each MLUT has *m-bit logic address inputs* $A_{m-1:0}$ and *m-bit logic data outputs* $D_{m-1:0}$, referred to as the *m-pair AD interconnects*, used for logic operation mode. The logic address inputs of the inner MLUTs are connected to the logic data outputs of its adjacent MLUTs. The logic address inputs and logic data outputs of the outermost MLUTs are connected to the logic operation IO (Input/Output) ports of the MPLD device.

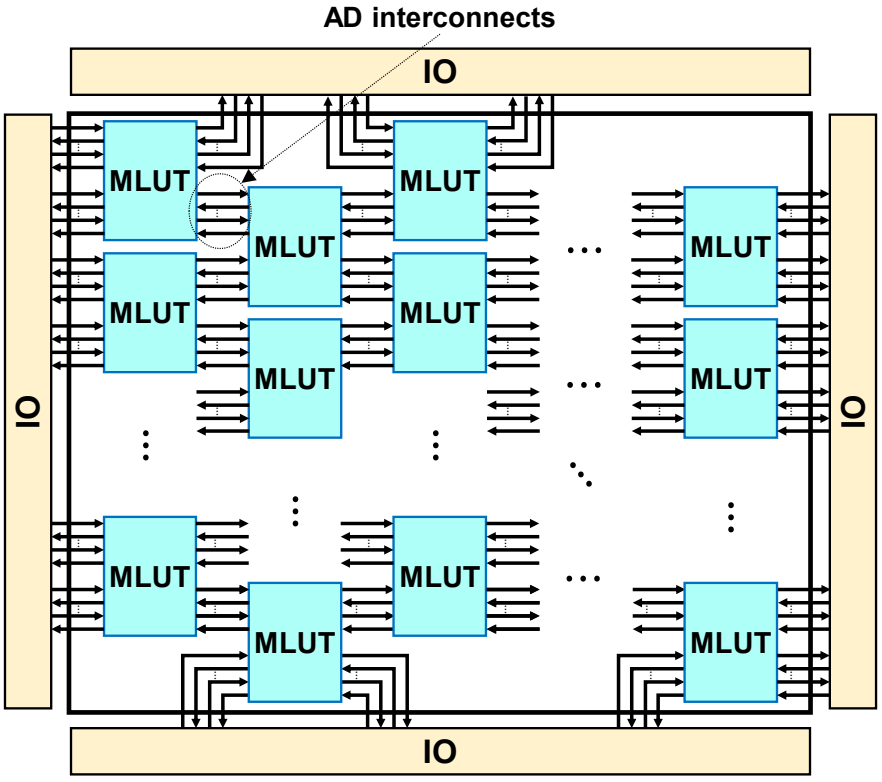


Figure 6.4 Schematic of MPLD working in memory operation mod.

Figure 6.5 shows the schematic in the logic operation mode of an MLUT. Each

SRAM has the $2^{m/2}word \times m$-bit size accessed by $m/2$-bit address inputs and m-bit data outputs. The low-order $m/2$-bit address inputs, $A_{m/2-1:0}$, are shared by SRAM1 and SRAM3, and the high-order $m/2$-bit, $A_{m-1:m/2}$, are shared by SRAM2 and SRAM4, respectively. An address transition detector (ATD) circuit is put into the address inputs of the asynchronous SRAM to detect the value changes coming from the data outputs of its adjacent MLUTs at high speed for combinational logic operation. A *logic output control circuit* controls the logic data outputs of MLUT by using an *m-bit* OCR, OR gates, and EOR gates.

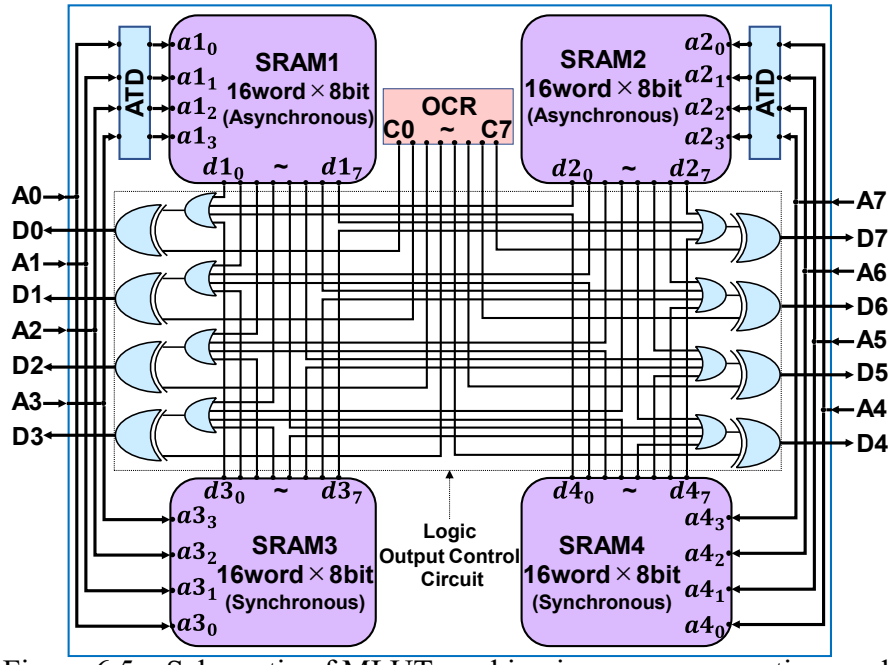


Figure 6.5 Schematic of MLUT working in memory operation mod.

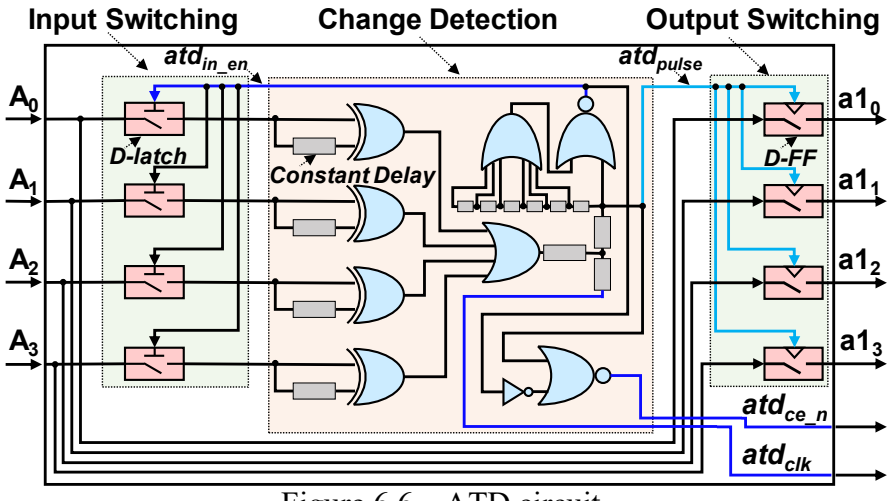


Figure 6.6 ATD circuit.

Figure 6.6 shows the ATD circuit structure. it comprises an *address input switching unit* (AISU, containing D-latch components), an *address change detection unit* (ACDU, containing constant delay components and signal change detection logic), and an *address output switching unit* (AOSU, containing D-FlipFlop components). The ACDU controls

the AISU via a signal atd_{in_en} and the AOSU via a signal atd_{pulse}. When the ACDU detects that no address change occurs, it generates an enable signal atd_{in_en} to enable the D-latches in the AISU to switch the address input of the MLUT (e.g., A_0 to A_3) and performs signal change detection for them. Further, if the ACDU detects an address change, a pulse signal is generated on atd_{pulse} to trigger D-FlipFlops in the AOSU to switch the address input (A_0~A_3) of the MLUT to the address input ($a1_0$~$a1_3$) of the SRAM. In addition, atd_{ce_n} and atd_{clk} signals will be output to drive the SRAM.

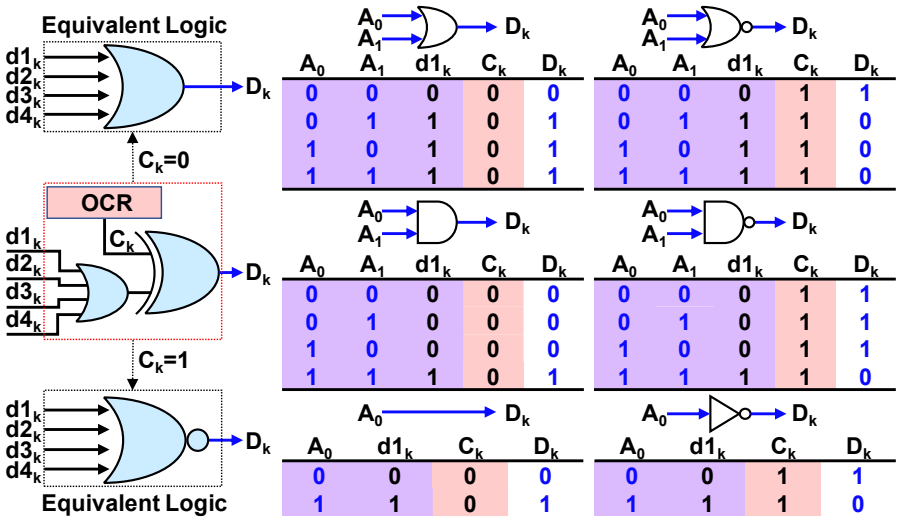


Figure 6.7 Functional operation of logic output control circuit.

Figure 6.7 shows the functional operation of the logic output control circuit. The C_k of the OCR controls the logic data output of the MLUT D_k, as an input to an EOR gate to enable this EOR logic. The other input of the EOR gate is the output of an OR gate. The inputs of the OR gate are the k-th data outputs of all SRAMs ($d1_k$, $d2_k$, $d3_k$, $d4_k$). In essence, this output control circuit performs the following logic function:

$$D_k = C_k \oplus (d1_k \vee d2_k \vee d3_k \vee d4_k) \qquad (6.1)$$

When $C_k = 0$, this output control circuit is equivalent to an OR logic for k-th data outputs of all SRAMs:

$$D_k = d1_k \vee d2_k \vee d3_k \vee d4 \qquad (6.2)$$

When $C_k = 1$, this output control circuit is equivalent to a NOR logic for k-th data outputs of all SRAMs:

$$D_k = \overline{d1_k \vee d2_k \vee d3_k \vee d4_k} \qquad (6.3)$$

By using the output control circuit, i.e., OCR, OR, and EOR, various types of output logic functions can be implemented in the MLUT for its logic inputs, such as common

logics like OR, NOR, AND, NAND, INVERTER, and wiring. The most important role of the OCR is to control the output logic function of the MLUTs in the logic mode without changing the contents of the truth table back to the memory mode.

In such architecture, each MLUT can work in either memory or logic operation mode. In memory operation mode, users can access (read/write) data as a regular memory block. When the MLUT is used as reconfigurable computing, first it is needed to put the MPLD in the memory operation mode to write the truth table of the logic function into the corresponding SRAMs, then, switch the device to the logic operation mode.

6.2 MPLD Work Principle

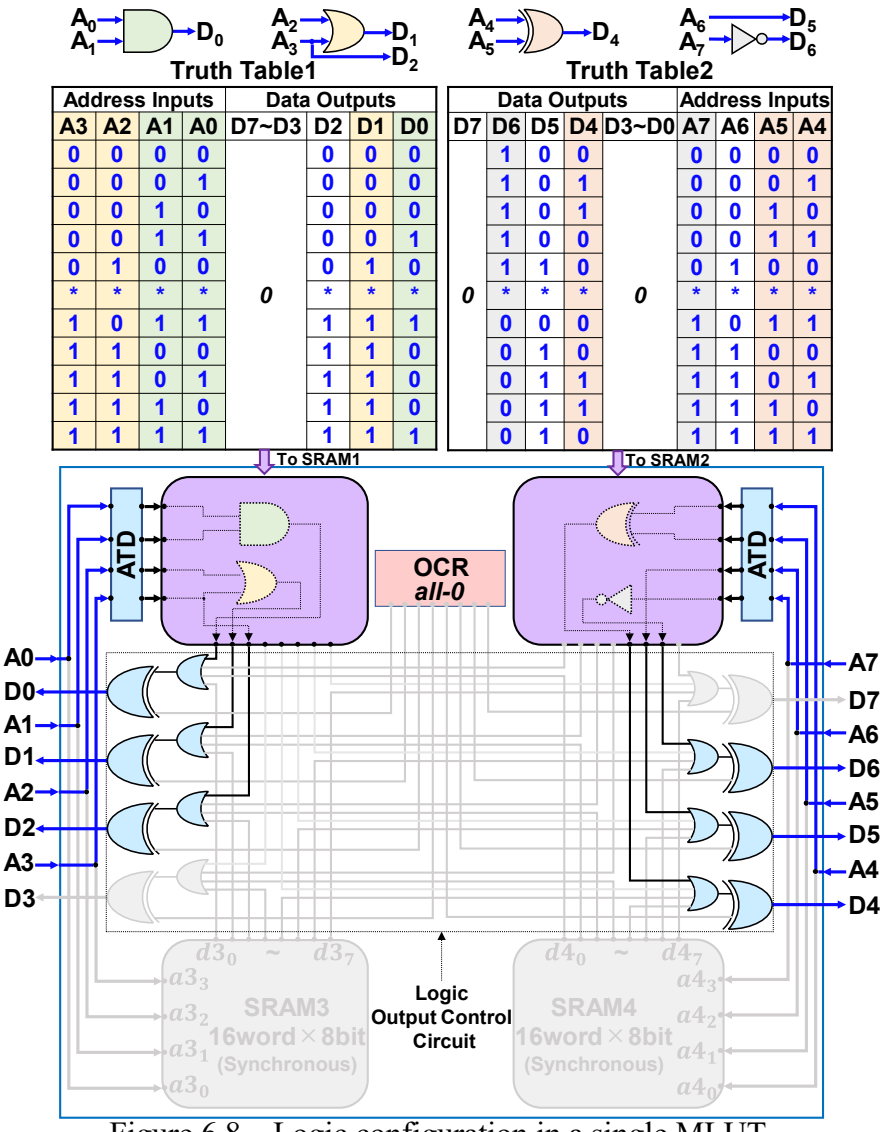


Figure 6.8 Logic configuration in a single MLUT.

Figure **6.8** shows an example to configure logic gates and wires in an MLUT. Here

we use two asynchronous SRAMs to configure an AND gate, an OR gate, and a wire into SRAM1, an XOR gate, a wire, and an INVERTER into SRAM2. We choose the address A_0 and A_1 of MLUT as the AND gate's inputs, A_2 and A_3 as the OR gate's inputs, A_4 and A_5 as the XOR gate's inputs, A_7 as the INVERTER's input, the data output D_0, D_1, D_2, D_4, D_5, and D_6 as the output of the AND gate, OR gate, wire $A_3 \rightarrow D_2$, XOR gate, wire $A_6 \rightarrow D_5$, and INVERTER, respectively. We represent the AND, OR logic, and the wire $A_3 \rightarrow A_2$ in the truth table1, the XOR logic, wire $A_6 \rightarrow D_5$, and the INVERTER in the truth table2. In memory operation mode, we write the truth table1 and truth table2 into the SRAM1 and SRAM2, respectively. Since the data outputs of SRAMs are connected to each other (by OR gate) and controlled by the OCR. We need to set the value of the remaining data outputs of SRAMs to all-zero and the OCR to all-zero. In logic operation mode, the MLUT will execute the configured logic and wires as a combinational logic block.

Figure 6.9 shows an example to configure a logic circuit in two MLUTs. The circuit has two inputs *a* and *b*, two internal signal lines *c* and *d*, and an output *e*. First, the logic partition is performed to divide the circuit into two sub-logics. Then, determining the address input and data output lines of the MLUTs according to each sub-logic (e.g.: $a \rightarrow A_0$, $b \rightarrow A_1$, $c \rightarrow D_5$, and $d \rightarrow D_4$), and computing the truth tables of the sub-logics. Finally, writing the truth tables in the SRAMs within the MLUTs.

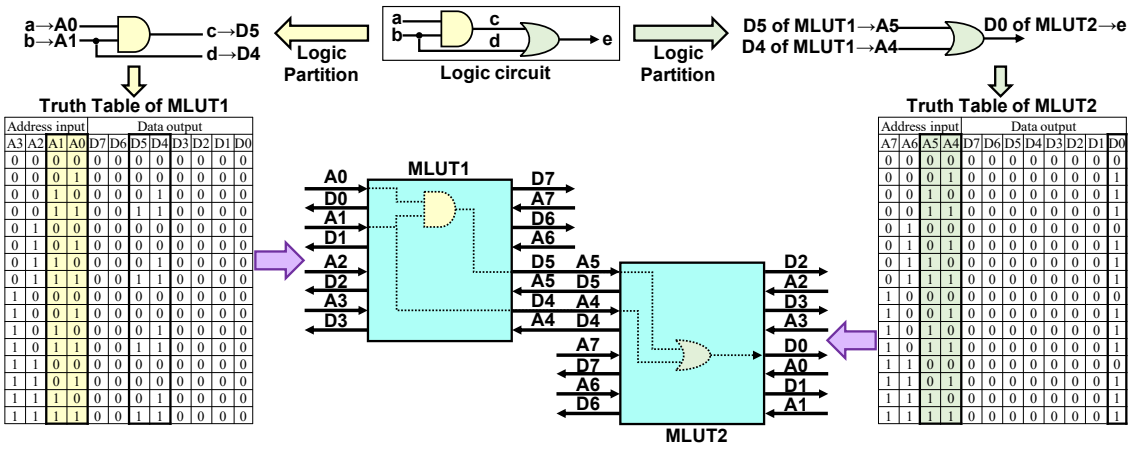


Figure 6.9 Configure a logic circuit in two MLUTs.

It is worth noting that wires can be configured in MLUTs as logic interconnects which can provide smaller delay and lower power consuming than FPGA.

Chapter 7

7. Reliability issue in MPLD

As a novel type of programmable logic device under development, and given its promising features of low production cost, reduced power consumption, minimal delay, fast data processing, and high flexibility, the MPLD is poised to revolutionize various applications, including IoT and AI edge devices. However, it is crucial to acknowledge that the MPLD's reliability can potentially be compromised by manufacturing defects during the production phase and aging in the field use phase. The implications of reliability degradation can impede the practical use of MPLD. Therefore, it becomes paramount to prioritize the resolution of reliability concerns to ensure the long-term dependability of the MPLD device. By addressing these challenges head-on, we can confidently unlock the full potential of the MPLD and enable its seamless integration into a wide range of cutting-edge technologies.

This chapter introduces the factors that affect the long-term reliability of MPLD in the production phase and the field use phase, respectively.

The rest of this chapter is organized as the following: *Section 7.1* introduces the reliability issues caused by manufacturing defects during the production phase of the MPLD, with the focus revolving on interconnection defects between MLUTs. *Section 7.2* describes reliability issues caused by aging when MPLD is operated in the field, focusing on ATD circuits in MLUTs that are sensitive to aging. Finally, the chapter concludes in *Section 7.3*.

7.1 Manufacturing-Defects-caused Reliability Issue

In the production phase of the MPLD, as depicted in Figure **7.1**, a multitude of defects would be present in the SRAM memory of the MLUT. While conventional memory testing methods can address these memory defects, however, there are also numerous defects that would arise between MLUTs, particularly in the form of interconnect defects that occur at the logic address input lines and logic data output lines of MLUTs. These interconnect defects encompass problems like shorts, bridges, and open circuits, all of which can lead to significant losses in yield and a decline in the reliability of the MPLD. Therefore, the focus of this study is placed specifically on tackling the interconnect defects that occur at AD interconnect between MLUTs.

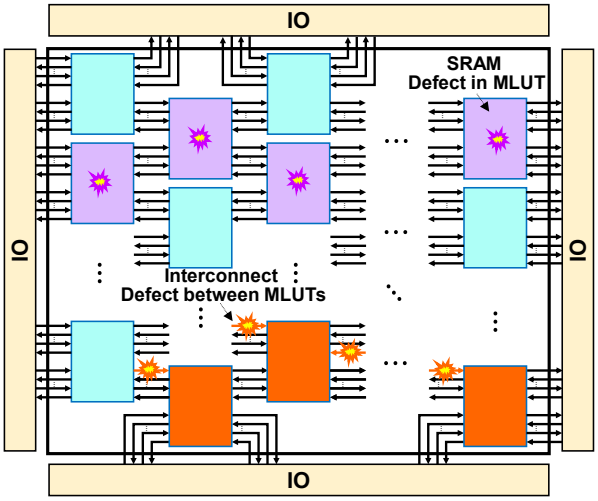


Figure 7.1 Manufacturing defects in MPLD.

Interconnect Defects

As described in *Section 6.1.2* of *Chapter 6*, the address inputs of a target MLUT come from the data outputs of its adjacent MLUTs. A defect at the AD interconnect would change the value of the address inputs of MLUTs, which cause access errors and results in logic faults in the configured circuit.

Figure 7.2 shows the example of an AD interconnect-defect-caused fault, which assumes that an OR-logic is configured in an MLUT ($D_5 = A_1 \vee A_0$) through the truth table. For a defect-free device, when the all-zero 00000000 are applied to the address inputs of the MLUT, the corresponding contents 00000000 stored in the SRAM will be readout and the OR gate will output 0 (D_5: 0). If there is a short interconnect defect between the supply and the address input A_0 of the MLUT, which will fix the value of A_0 at logic 1. In this case, the normal address of all-zero 00000000 will be changed to 00000001, which causes an access error where the content of 0010000 at the address 00000001 is read out, thus the output of the OR gate D_5 will output 1.

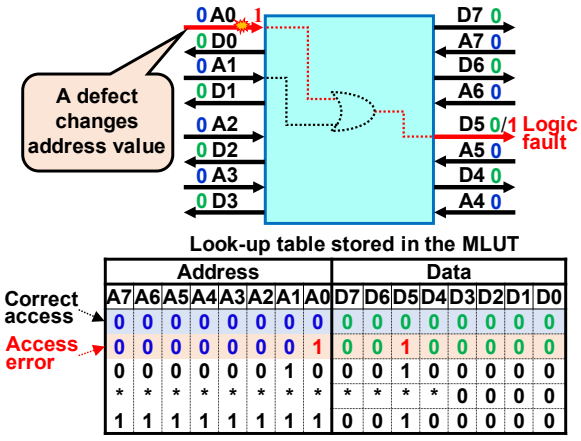


Figure 7.2 Interconnect defect causes logic fault in configured circuit.

7.2 Field-Aging-caused Reliability Issue

When the MPLD works in the field for a long term or under a severe environment, various aging phenomena such as HCI (hot carrier injection) and BTI (bias temperature instability) [50][51] would cause delay degradation that threatens the long-term reliability [52] of the MPLD. As described in *Chapter 6*, the MPLD is composed of a large number of MLUTs arranged in an array, and each MLUT is placed independently in the MPLD. During the operation of the MPLD, as shown in Figure 7.3, it is considered that the progress of the aging at each single MLUT is different. When configuring a logic circuit into the MPLD, the progress of aging at the often-used MLUTs would be faster which causes more extra delay. As the aging progresses, the variety of aging-induced delays at MLUTs would affect the performance of the configured logic circuit, and even worse, the delay at the MLUTs with faster aging progression could cause a sudden system failure.

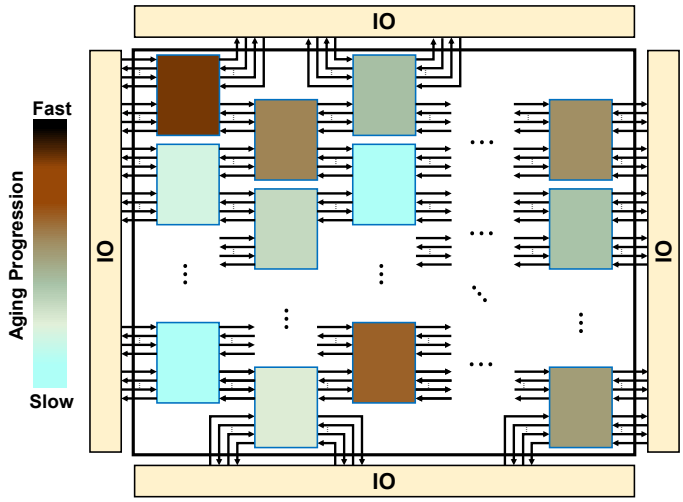


Figure 7.3 Aging progresses in MPLD.

In an MLUT, the two main components are vulnerable to the aging phenomena: the SRAM and the ATD circuit. The aging phenomena like BTI crucially impacts SRAM, which can degrade the static noise margin (SNM: a critical reliability metric for SRAM) and lead to read stability issues and potentially cause failure [54]. For the aging effects in the SRAMs of each MLUT, such as read/write delay, existing memory delay testing methods can be employed to detect it during the memory operation mode of the MPLD, which will not be described in more specific detail in this study. Therefore, this study focuses on the aging effects occurring in the ATD logic for each MLUT.

Aging Effect in Address Transition Detector (ATD) Circuit

As described in *Section 6.1.3* of *Chapter 6*, in each MLUT, the asynchronous

SRAMs use the ATD circuit to at high speed detect the input address change to execute asynchronous operations. The ATD circuit is extremely sensitive to the delay variation. The aging phenomena like HCI and BTI would increase the threshold voltage of the transistors in the ATD circuit, which could slow down the switching speed [53] and might cause false detection of the address change.

As demonstrated in Figure 7.4, the ATD circuit will generate an atd_{pulse} signal once detected any value changes in the address inputs ($A_{0:3}$) of MLUT and switches $A_{0:3}$ to the address inputs ($a1_{0:3}$) of the asynchronous SRAM1. Suppose that 01010 is applied to A_0 at time $t_0t_1t_2t_3t_4$, respectively. When a transition occurs at A_0, the ATD must detect the value change and transfer the transition to $a1_0$ in a very short delay. Aging-induced delay at the ATD logic would generate an anomalous atd_{pulse} signal to switch the A_0 to $a1_0$, which causes false detections for the A_0 at t_2, t_4 and result in $a1_0$ being 01111 at $t_0t_1t_2t_3t_4$.

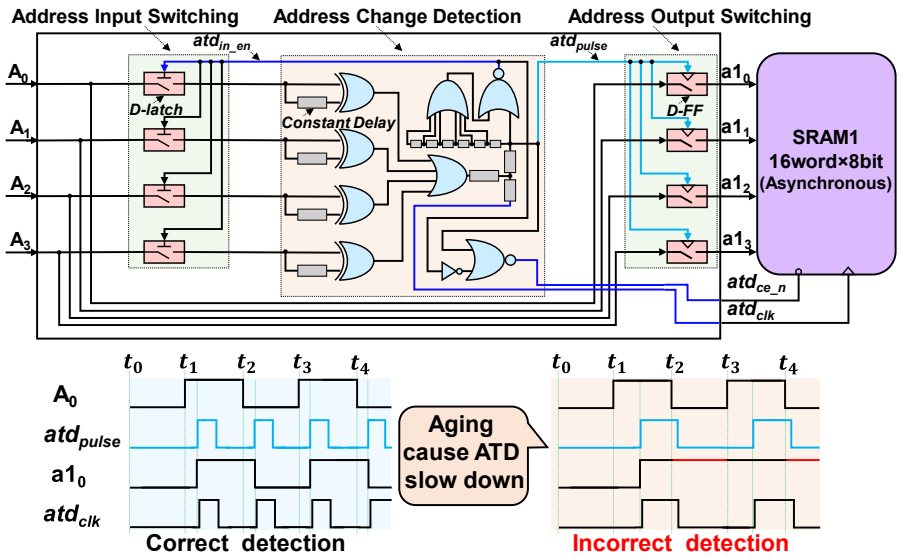


Figure 7.4 Aging caused ATD detection error.

7.3 Conclusions

In the production phase of the MPLD, there would be many kinds of defects exist in the SRAM memory of the MLUT. For these memory defects, conventional memory testing methods are available. On the other hand, there would also be lots of defects between MLUTs, especially the interconnect defect on the logic address input lines and logic data output lines, such as short, bridge, and open circuits, these defects would cause significant yield loss and reliability degradation.

In addition, when the MPLD works in the field, various aging phenomena such as the HCI, and BTI, would cause aging-induced delays in the MLUTs array of the MPLD.

The aging progress in the MLUTs array should be different. The often-used MLUTs would have a faster aging speed which means the aging-induce delay would be large. The variations of aging-induce delay would affect the performance of configured logic circuits and even cause a system failure, which would threaten the in-field reliability of the device.

Therefore, from this chapter, two important items can be put forward:

To guarantee the long-term reliability of the MPLD, the

1) During the production phase, it is necessary to perform high-quality tests for the interconnect defects on the logic address input lines and logic data output lines of the MLUTs.

2) In the field use phase, it is necessary to employ an aging monitoring approach to measure the aging-induced delay variations in the MLUTs.

To address the above two items, a test method for identifying interconnect defects is proposed in *Chapter 8*. A delay monitoring method to measure the aging-induced delay of the MLUTs is proposed in *Chapter 9*.

Chapter 8

8. Interconnect Defect Test for MPLD

In order to improve the yield and guarantee the reliability of the MPLD device, extensive production tests with high quality are required to detect as many manufacturing defects as possible that exist in the SRAMs and the AD interconnects between MLUTs. The former defects can be tested by conducting the existing test technologies of SRAM memory [55][56]. For the latter, we have analyzed the interconnect fault models including the stuck-at faults and bridge faults at the AD interconnects between the MLUTs, and proposed the test approaches for detecting the stuck-at fault and bridge fault, in [57][58].

Besides fault detection, fault diagnosis is also known to play an important role in improving the yield and reliability of products. In manufacturing, identifying the location of the interconnect faults in the MLUTs array is beneficial to improving the process. In addition, when the MLRD is put to actual use in the field, identifying the interconnect fault is helpful to avoid configuring the logic into a faulty MLUT block for high reliability.

The fault diagnosis for locating the interconnect fault in the FPGA device has been investigated deeply [59][60]. In [61] a universal fault diagnosis technique is presented for locating the interconnect fault in the CLBs array of an FPGA device. This method can identify all faulty points in the CLBs array through two steps: the horizontal diagnosis and the vertical diagnosis. For the MPLD device constructed by the MLUTs array, the basic idea presented in [61] would be also available to identify the AD interconnect faults between the MLUTs. However, implementing the horizontal and vertical diagnosis in MPLD must be considered carefully, because the interconnects between MLUTs are un-reconfigurable.

This chapter arm to present the test method to identify the AD interconnect faults between MLUTs of the MPLD device, for improving the manufacturing process and voiding a faulty MLUT block for high reliability when the MPLD is put to practical use.

The rest of this chapter is organized as follows: *Section 8.1* reveal the interconnect faults models including stuck-at and bridge faults between MLUTs in MPLD. *Section 8.2* present the test method to identify the interconnect faults and deals with the generation of data for testing. *Section 8.3* shows the results of the logic simulations for evaluating the proposed test method. The proposed methods are discussed in *Section 8.4*. Finally, *Section 8.5* make concludes this chapter.

8.1 Interconnect Fault Models in MPLD

This section scrutinizes the two primary types of faults resulting from interconnect defects between MLUTs, namely stuck faults and bridged faults. These interconnect faults lead to the alteration of the value of the logic address input of the MLUT, and have the potential to compromise the normal functioning of the logic circuitry configured in the MLUT. As a consequence, these interconnect faults may cause the generation of erroneous logic outputs at the value of the logic data output of the MLUT.

8.1.1 Stuck-at Interconnect Faults

A stuck-at fault that occurs at the AD interconnect between the MLUTs in the MPLD is referred to as a *stuck-at interconnect fault*. This type of fault may arise due to the presence of an interconnect defect such as a short between the ground or supply and the AD interconnect. As a result of this defect, the logic address input of the MLUT becomes fixed at either logic 0 or 1, which can significantly impact the normal functional operation of the MLUT. The presence of a stuck-at interconnect fault could lead to incorrect logic outputs being produced at the value of the logic data output of the MLUT. This issue, if left unaddressed, could ultimately cause significant damage or malfunction of the MPLD. Therefore, it is crucial to promptly identify and resolve any instances of stuck-at interconnect faults to ensure the continued reliable performance of the MPLD.

Figure 8.1 depicts the behavior of stuck-at interconnect faults, and in the event of a stuck-at occurring at $M_1D_5 \rightarrow M_2A_5$, the M_2A_5 value would be fixed to 1 or 0, depending on whether a *stuck-at-1* or *stuck-at-0* occurs.

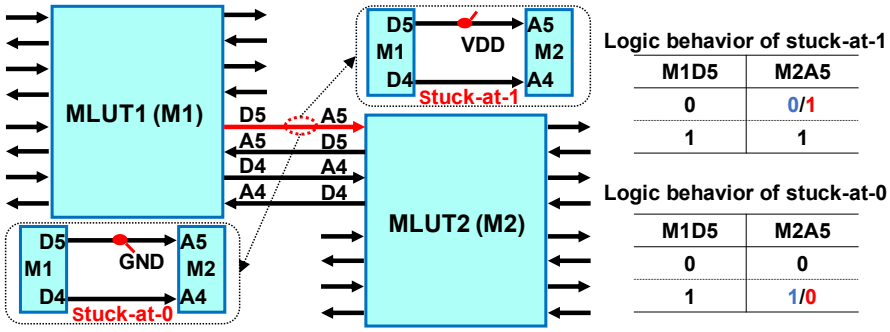


Figure 8.1 Stuck-at interconnect fault models.

8.1.2 Bridge Interconnect Faults

A bridge fault that occurs at the AD interconnect between the MLUTs in the MPLD is referred to as a *bridge interconnect fault*. This type of fault may arise due to the presence of an interconnect defect such as a short between AD interconnects. This defect may result

in either a *wired-OR* (*OR-bridge*) or *wired-AND* (*AND-bridge*) logic function depending on the utilized logic circuitry. The former is referred to as *OR-bridge interconnect fault* and the latter as *AND-bridge interconnect fault*. An OR-bridge interconnect failure causes shorted AD interconnects to be OR-ed together, and the output value of the OR-ed is assigned to each of the shorted AD interconnects. Similarly, an AND-bridge interconnect failure causes shorted AD interconnects to be AND-ed together, and the output value of the AND-ed is assigned to each shorted AD interconnect. Therefore, bridge interconnect faults caused by such an interconnect defect would also change the value of the logic address input of the MLUT, which also has a significant impact on the normal functional operation of the MLUT. Therefore, it is also crucial to promptly identify and resolve any instances of bridge interconnect faults to ensure the continued reliable performance of the MPLD.

Figure 8.2 demonstrates the behavior of bridge interconnect faults. In the event of a bridge occurring between $M_1D_5 \rightarrow M_2A_5$ and $M_1D_4 \rightarrow M_2A_4$, an AND-bridge interconnect fault would cause a faulty value of 0 at M_2A_5 (M_2A_4) when M_1 outputs logic 1 (0), 0 (1) at M_1D_5 and M_1D_4, respectively. Conversely, an OR-bridge interconnect fault would cause a faulty value of 1 at M_2A_5 (M_2A_4) when M_1 outputs logic 0 (1), 1 (0) at M_1D_5 and M_1D_4.

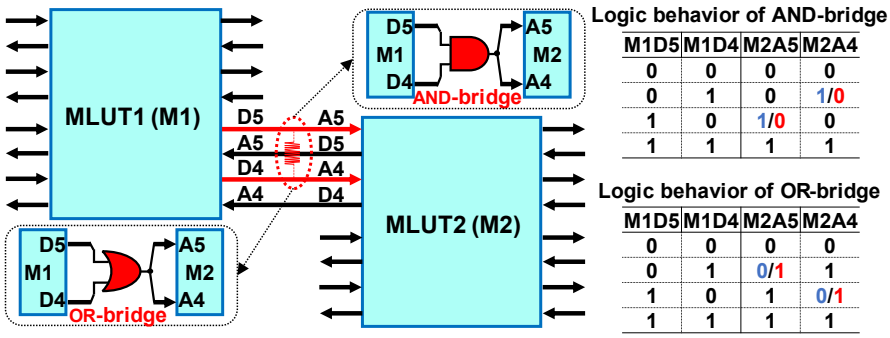


Figure 8.2 Bridge interconnect fault models.

8.2 Test Method for Interconnect Faults

In our research presented in [57][58], the test methods were proposed for detecting both stuck-at and bridge interconnect faults in the MLUT array. These methods were able to achieve high fault coverage while requiring fewer test configurations for fault detection. However, identifying the location of faults is crucial for improving the manufacturing process and ensuring the proper functioning of the MPLD when in practical use, such as to help the user to void the use of a faulty MLUT block. Therefore, in this study, we aim to extend the previous test methods proposed in [57][58] to include the identification of locations of interconnect defects.

For programmable devices, Prior research on fault localization for FPGAs has been conducted and is discussed in [59][60][61]. These studies proposed sophisticated methods for fault localization in FPGAs, with [61] presenting a universal fault diagnosis technique that can locate interconnect faults for the CLBs array of an FPGA. This method utilized a two-step horizontal and vertical diagnosis process to locate all faulty points for the CLBs array. Although the basic idea in [61] is applicable to the MPLD constructed using the MLUTs array, the interconnects between MLUTs are unconfigurable, unlike FPGAs. Therefore, careful consideration should be given to implementing the horizontal and vertical diagnosis in MPLD.

In this study, we present a novel test method for detecting and locating faults caused by AD interconnect defects, including stuck-at and bridge faults. Our approach is based on the fault detection idea presented in [57] and the fault localization idea in [61]. By building on the previous research, we aim to improve fault detection and localization in MPLDs and contribute to the advancement of manufacturing processes for these devices.

8.2.1 Test Strategy for Fault Detection and Location

Interconnect Fault Detection Idea. Since the logic address inputs of an MLUT come from the logic data outputs of its neighbor MLUTs, the *logic data flow* in MPLD expresses as the following:

external input → *[address* → $\boxed{data}$ → *address* → $\boxed{data}$ → *...]* → *external output*.

Where the $\boxed{data}$ indicate the data configured in an MLUT, and the *address* before a $\boxed{data}$ indicate the logic address inputs of the MLUT that stores the $\boxed{data}$. A fault at the AD interconnect can cause a change of the value at the logic address input that would access a different content of the data configured in the MLUTs and in turn may result in different output values than fault-free at the logic data outputs of the MLUT, and ultimately, may produce incorrect values to the external logical output ports of the MPLD. Therefore, the idea of the test method presented in [57] for detecting the interconnect faults of MPLD, is to configure the *internal test data* into the SRAMs of MLUTs, apply the *external test data* to external logic input ports, and observe the fault effects at the external logic output ports of MPLD by performing the logic operation. The two kinds of test data are defined in [57] as follows.

Definition 8.1 *Test cube.* the internal test data stored in the SRAMs of MLUT in MPLD, which can propagate fault excitations and fault effects for AD interconnect faults.

Definition 8.2 *External pattern.* the external test data applied to the external logic input ports of MPLD as fault excitations to excite AD interconnect faults.

The *test data flow* in MPLD expresses in the following:

external pattern → *[excitations*→ *test cube* → *fault* → *test cube* → *...]* → *effects*.

Figure 8.3 shows the concept of detecting interconnect faults under this idea. The test cubes are configured in the memory operation mode. In logic operation mode, the configured test cubes propagate the external patterns (fault excitations) applied to external logic input ports to excite the interconnect faults and propagate the fault effects to external logic output ports. At the external logic output ports, the fault effects can be observed, and the faults are detected.

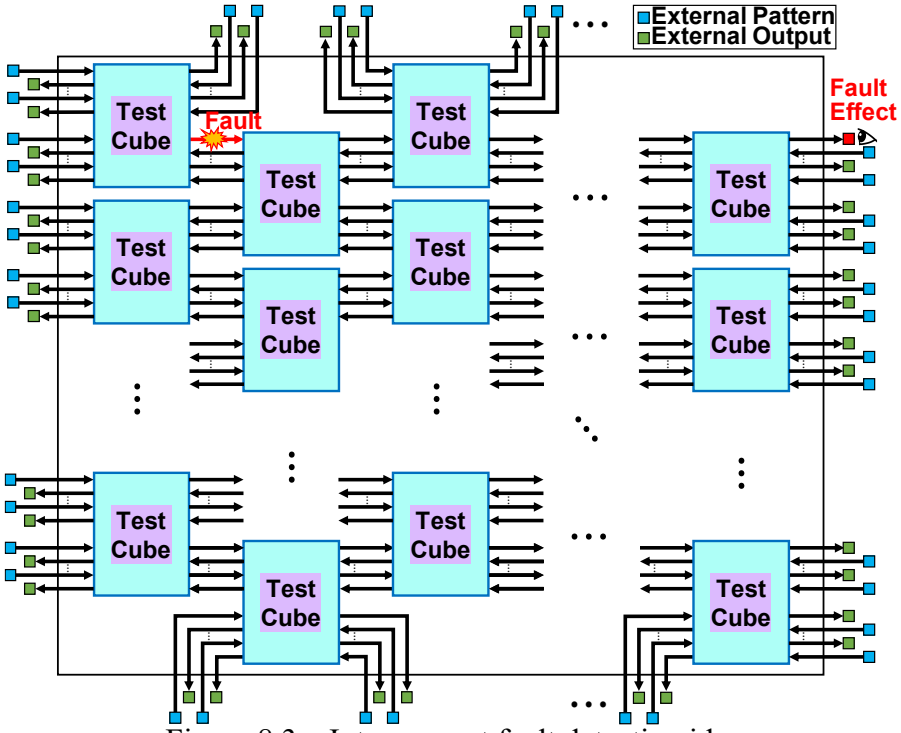


Figure 8.3 Interconnect fault detection idea.

The critical point of this test idea is to generate test cubes and external patterns. In [57], The test generations of the test cubes and external patterns for detecting stuck-at interconnect faults and bridge interconnect faults have been presented. In the presented test generations, the test generations for the stuck-at interconnect faults can detect all the stuck-at faults. However, at that time, the test generations for bridge interconnect faults did not consider detecting all bridge interconnect faults; in order to consider any bridge fault interconnects, also based on this test idea, in [58], the additional test generations were presented to cover the detecting of the bridge interconnect faults that were not detected in [57].

This study aims to present novel test generations for test cubes and external patterns, which can not only detect but also locate the interconnect faults, the idea of locating

interconnect faults are as follows.

Interconnect Fault Location Idea. For fault location of programmable devices, [61] presented a universal fault diagnosis technique that can locate interconnect faults on the CLBs array of an FPGA. The basic idea of this method is to utilize a two-step diagnosis process, step 1 for horizontal diagnosis and step 2 for vertical diagnosis, to locate the faulty CLBs in the array. Figure 8.4 shows this universal diagnosis procedure.

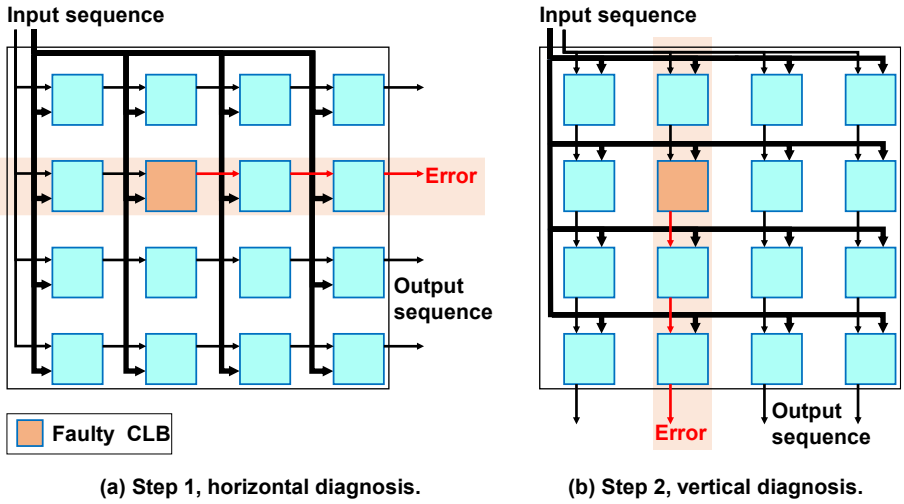


Figure 8.4 A universal diagnosis procedure for FPGA [61].

In this diagnosis procedure, the interconnect structures are specially configured. In the horizontal (vertical) diagnosis, for the CLBs in a row (column), the output line of each CLB except the last one is configured to connect with an appropriate input line of the CLB following it; the output line of the last CLB is configured to connect with an I/O block as the primary output for output sequence observing; the other input lines of each CLB are configured to connect with the remaining I/O blocks as the primary inputs for input sequence applying.

By applying the input sequence to the primary inputs at I/O blocks to perform respectively the horizontal and vertical diagnosis to the CLBs array, horizontally and vertically to propagate the fault, the faulty CLB can be located by observing the output sequences from the primary outputs at the I/O blocks.

This diagnosis method provides an applicable basic idea to locate the interconnect faults for the MPLD constructed using the MLUTs array, however, the interconnects between MLUTs are unconfigurable, unlike FPGAs. Therefore, careful consideration should be given to implementing the horizontal and vertical diagnosis in MPLD. The test strategy for locating the interconnect faults in MPLD is described as follows.

Test Strategy for Interconnect Fault Location The main idea for identifying an

interconnect fault between MLUTs is to create the proper paths on the MLUTs array which can propagate the fault effect across the MLUTs array to the expected logic output ports of the MPLD for fault localization. Since the interconnects between MLUTs are not configurable like FPGAs, it is impossible to directly configure the connections between the interconnect and any specified MLUT or any specified external port. Therefore, to create proper propagation paths, an optimal way is to take full advantage of these MLUTs, since they are reconfigurable, to realize proper propagation paths by configuring each MLUT with the *logic function* that can properly *route* the fault effects on the logic address inputs to the expected logic data outputs. The *routing logic* in each MLUT navigates the propagating of fault effects like a *map*, as defined in the following.

Definition 8.3 *Rout map.* a logic function configured in the MLUT that can properly propagate the fault effects on the logic address inputs to the expected logic data outputs.

A simple type of rout map is the *wiring logic*. For an MLUT with *m-pair* AD interconnects (*m-bit* logic address inputs and *m-bit* logic data outputs), there are ${}_mP_m = m!$ kinds of wiring patterns to connect the address inputs with the data outputs. Therefore, the number N_{rm} of the *wiring-type route map* for the MLUT with *m-pair* AD interconnects can be expressed as the following equation.

$$N_{rm}(m) = {}_mP_m = m! \qquad (8.1)$$

Figure 8.5 shows all 24 route maps to create the data paths between the address inputs and data outputs of an MLUT with 4-pair AD interconnects. Where the labeled *1*, *2*, and *3* are referred to as *horizontal*, *vertical*, and *diagonal* route maps, respectively. By implementing these route maps as test cubes configured into the SRAMs of the MLUTs, the test procedure can be realized to locate the AD interconnect faults in MPLD, like the horizontal and vertical diagnosis processes for FPGAs proposed in [61].

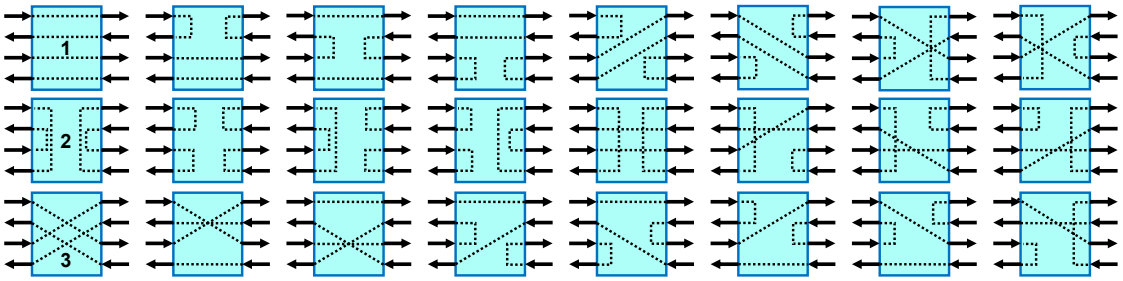


Figure 8.5 Route maps for an MLUT with 4-pair AD interconnects.

Figure 8.6 shows the testing mechanism under route maps to locate an interconnect fault. The (a) and (b) represent the testing under the horizontal route map and vertical route map, respectively. From these two testing processes, we can observe the fault effect from two different external output ports, and obtain two different fault propagation paths,

$FP^{(1)}$ and $FP^{(2)}$. Via the intersection of obtained fault propagation paths, the fault location can be identified.

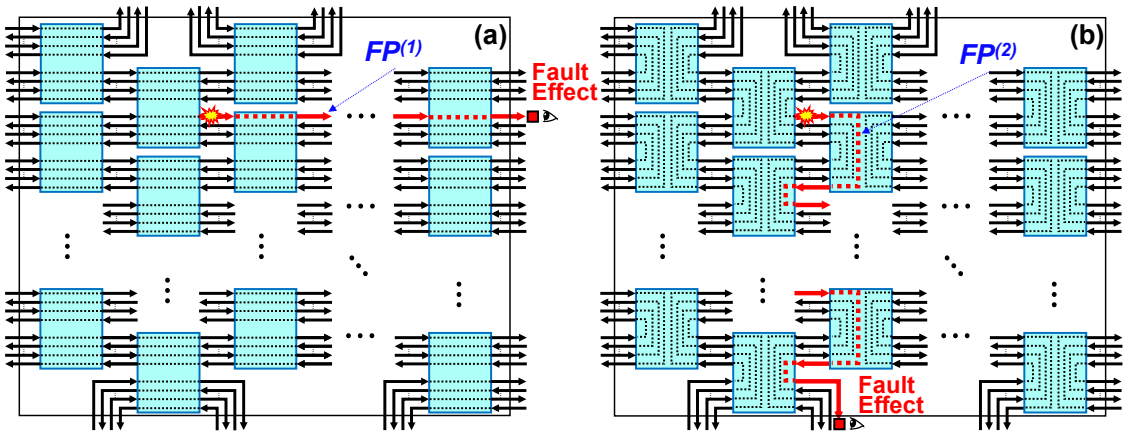


Figure 8.6 Testing mechanisms under route maps to locate an interconnect fault.

For the stuck-at faults in any AD interconnects of the MLUTs array, configuring either the horizontal route map or the vertical route map into the MLUTs would detect all faults and identify the location of the faults by configuring the two maps in order. For the bridge faults, an additional diagonal route map is considered necessary. The procedure of the proposed test method based on the route map is as follows.

Test Procedure

Definitions:

- N_{rm}: *number of route maps, it can be calculated by equation 8.1.*
- rm_i: *route map i;* $i \in [1, N_{rm}]$.
- $TC^{(i)}$: *test cubes implementing* rm_i.
- $N_{FE}^{(i)}$: *number of observed fault effects under* rm_i.
- $FP_k^{(i)}$: *fault propagation path k obtained under* rm_i; $k \in [1, N_{FE}^{(i)}]$.
- $FP^{(i)}$: *fault propagation path set under* rm_i.
- F_{loc}: *fault location*

Process:

(1) Test under rm_i *for* $i \in [1, N_{rm}]$:

(a) Configure $TC^{(i)}$ *into each MLUT to create* rm_i.

(b) Apply external patterns to the input ports of MPLD.

(c) Observe fault effects. If $N_{FE}^{(1)} == 0$, *end testing (fault-free).*

(d) Obtain the fault propagation path set:

$$FP^{(i)} = \bigcup_{k=1}^{N_{FE}^{(i)}} FP_k^{(i)} \qquad (8.2)$$

(2) Identify fault location:

$$F_{loc} = \bigcap_{i=1}^{N_{rm}} FP^{(i)} \qquad (8.3)$$

The next section presents the test generation of the test cubes ($TC^{(i)}$ for rm_i) and the test generation of the external patterns for exciting stuck-at and bridge faults.

8.2.2 Test Generation

Test generation of test cubes for implementing route maps Since the MLUT can be

allowed to arbitrarily configure the logic by designing and storing the corresponding truth tables in SRAMs of the MLUT, the test generation of the test cube for implementing the route map to an MLUT is to design the corresponding truth tables that can be stored in the SRAMs of the MLUT and are capable of representing the route map. For an MLUT with m pairs of AD interconnects ($A_{[m-1:0]}$, $D_{[m-1:0]}$) and constructed by four $2^{m/2}word \times m\text{-}bit$ size SRAMs, the test cubes, for implementing the horizontal, vertical, and diagonal route maps, are generated as follows.

Where $TC^{(1)}$, $TC^{(2)}$, and $TC^{(3)}$ indicate the test cubes that implement the horizontal, vertical, and diagonal route maps, respectively; and rm_1, rm_2, and rm_3 indicate the horizontal, vertical, and diagonal route maps, respectively. Each test cube consists of two truth tables, *truth table 1*, and *truth table 2*, to create the route of the address inputs $A_{m/2-1:0}$ and $A_{m-1:m/2}$ of the MLUT, respectively.

Test cube generation

[$TC^{(1)}$ for rm_1]

truth table 1: *For the SRAMs to share the low-order address inputs ($A_{[m/2-1:0]}$) of an MLUT, set the contents of the address lines $A_{[m/2-1:0]}$ to*
$D_{[m-1:m/2]} = A_{[0:m/2-1]}$,
$D_{[m/2-1:0]} = all\text{-}0$.

truth table 2: *For the SRAMs to share the high-order address inputs ($A_{[m-1:m/2]}$) of an MLUT, set the contents of the address lines $A_{[m-1:m/2]}$ to*
$D_{[m-1:m/2]} = all\text{-}0$,
$D_{[m/2-1:0]} = A_{[m/2:m-1]}$.

[$TC^{(2)}$ for rm_2]

truth table 1: *For the SRAMs to share the low-order address inputs ($A_{[m/2-1:0]}$) of an MLUT, set the contents of the address lines $A_{[m/2-1:0]}$ to*
$D_{[m-1:m/2]} = all\text{-}0$,
$D_{[m/2-1:0]} = A_{[0:m/2-1]}$.

truth table 2: *For the SRAMs to share the high-order address inputs ($A_{[m-1:m/2]}$) of an MLUT, set the contents of the address lines $A_{[m-1:m/2]}$ to*
$D_{[m-1:m/2]} = A_{[m/2:m-1]}$,
$D_{[m/2-1:0]} = all\text{-}0$.

[$TC^{(3)}$ for rm_3]

truth table 1: *For the SRAMs to share the low-order address inputs ($A_{[m/2-1:0]}$) of an MLUT, set the contents of the address lines $A_{[m/2-1:0]}$ to*
$D_{[m-1:m/2]} = A_{[m/4:m/2-1]} : A_{[0:m/4-1]}$,
$D_{[m/2-1:0]} = all\text{-}0$.

truth Table 2: *For the SRAMs to share the high-order address inputs ($A_{[m-1:m/2]}$) of an MLUT, set the contents of the address lines $A_{[m-1:m/2]}$ to*
$D_{[m-1:m/2]} = all\text{-}0$,
$D_{[m/2-1:0]} = A_{[3m/4:m-1]} : A_{[m/2:3m/4-1]}$.

For easier visibility, described above test generation of the test cubes for rm_1, rm_2, and rm_3 is also tabulated in Table 8.1.

Table 8.1 Test cubes to create route maps for the MLUT with m-pair AD interconnects

Route Maps	Test Cubes		
rm_1: horizontal route map	$TC^{(1)}$	truth table 1	$D_{m-1:m/2} = A_{0:m/2-1}$
			$D_{m/2-1:0} = all\text{-}0$
		truth table 2	$D_{m-1:m/2} = all\text{-}0$
			$D_{m/2-1:0} = A_{m/2:m-1}$
rm_2: vertical route map	$TC^{(2)}$	truth table 1	$D_{m-1:m/2} = all\text{-}0$
			$D_{m/2-1:0} = A_{0:m/2-1}$
		truth table 2	$D_{m-1:m/2} = A_{m/2:m-1}$
			$D_{m/2-1:0} = all\text{-}0$
rm_3: diagonal route map	$TC^{(3)}$	truth table 1	$D_{m-1:m/2} = A_{m/4:m/2-1} : A_{0:m/4-1}$
			$D_{m/2-1:0} = all\text{-}0$
		truth table 2	$D_{m-1:m/2} = all\text{-}0$
			$D_{m/2-1:0} = A_{3m/4:m-1} : A_{m/2:3m/4-1}$

Figure 8.7 shows an example of the $TC^{(1)}$ configured in asynchronous SRAMs of an MLUT with 8 pairs of AD interconnects. For each MLUT, we write truth table 1 and truth table 2 in the SRAM1 and SRAM2, respectively. In truth table 1, the *low 4-bit* data outputs $D_{[3:0]}$ for all addresses are *all-0*, and the *high 4-bit* data outputs $D_{[7:4]}$ for each address are $A_{[0:3]}$. In truth table 2, the high 4-bit data outputs $D_{[7:4]}$ for all addresses are *all-0*, and the low 4-bit data outputs $D_{[3:0]}$ for each address are $A_{[4:7]}$. Because the data outputs $D_{[7:0]}$ of SRAM1 and SRAM2 are connected by *OR function* as illustrated in Figure 6.5, the data outputs $D_{[7:0]}$ of each MLUT are the values of address inputs $A_{[0:7]}$. i.e. the address line A_k is connected to the data output line D_{7-k} for each MLUT, thus the construction of the propagation path of the fault from the horizontal direction is realized.

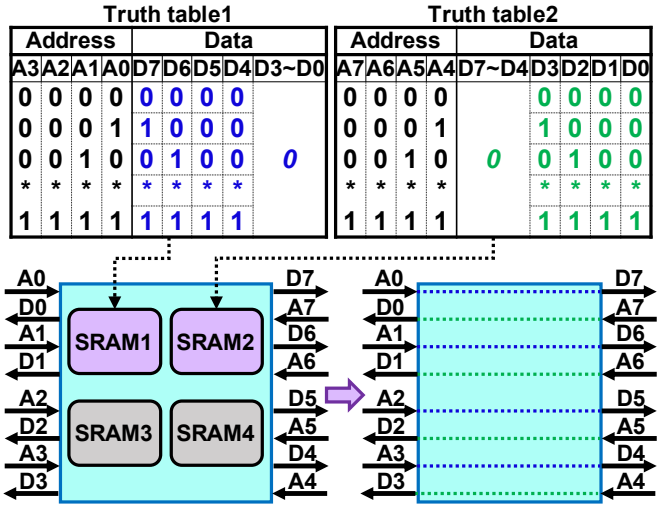


Figure 8.7 Example of test cube in the MLUT for horizontal route map.

Figure 8.8 shows an example of $TC^{(2)}$ configured in asynchronous SRAMs of an MLUT with 8 pairs of AD interconnects. Where, the data outputs $D_{[7:0]}$ for all addresses are $[0000A_0A_1A_2A_3]$ and $[A_4A_5A_6A_70000]$ for SRAM1 and SRAM2, respectively. i.e. for each MLUT, the address line A_k in the low 4-bit address ($k = 0, 1, 2, 3$) is connected to the

data output line $D_{7-(k+4)}$ and the address line A_k in the high 4-bit address ($k = 4, 5, 6, 7$) is connected to the data output line $D_{7-(k-4)}$, thus the propagation path of the fault from the vertical direction is created.

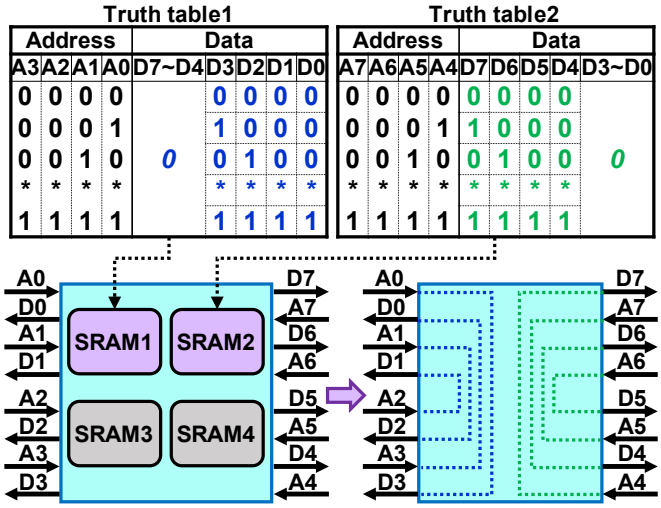


Figure 8.8 Example of test cube in the MLUT for vertical route map.

Figure 8.9 shows an example of $TC^{(3)}$ configured in asynchronous SRAMs of an MLUT with 8 pairs of AD interconnects. For each MLUT, the address line A_k for $k = 0, 1, 4, 5$ is connected to the data output line $D_{7-(k+2)}$, for $k = 2, 3, 6, 7$, to $D_{7-(k-2)}$, thus the propagation path of the fault from the diagonal direction is created.

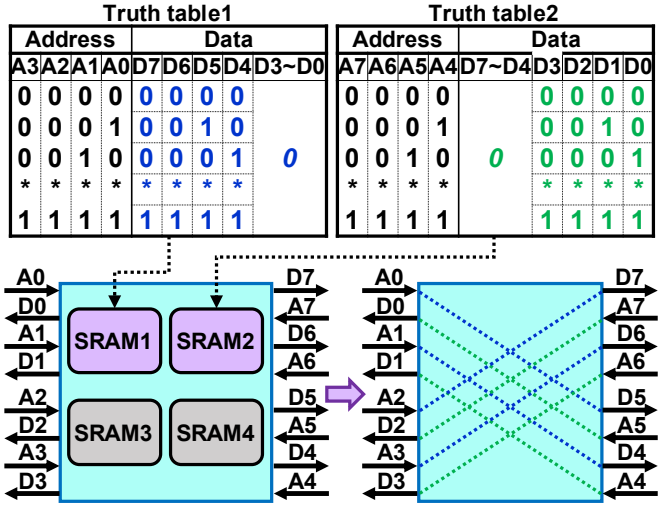


Figure 8.9 Example of test cube in the MLUT for diagonal route map.

Test generation of external patterns for exciting AD interconnect facts Since the conditions of fault excitation are different for different types of AD interconnect faults, it is necessary to consider different external patterns to excite different target faults at AD interconnect faults in MPLD. The conditions of fault excitation for different types of AD

interconnect faults are listed as the following.

* *Stuck-at-1 (stuck-at-0) interconnect fault excitation condition* is that the value 0 (1) must be assigned to the interconnect fixed at 1 (0).
* *AND-bridge (and OR-bridge) interconnect fault excitation condition* is that the two values 01 or 10 must be assigned to the two interconnects bridged.

Based on the above fault excitation conditions, as shown in Table 8.2, four external patterns are applied to the external input ports of MPLD to excite the AD interconnect faults and are defined as follows.

Table 8.2 External test patterns applied to external inputs of MPLD

Fault Types	External Test Patterns
stuck-at-1	*all-zero vector: [0…0]*
stuck-at-0	*all-one vector: [1…1]*
AND-bridge	*walking-zero vector: [1…1$\vec{0}$1…1]*
OR-bridge	*walking-one vector: [0…0$\vec{1}$0…0]*

Definition 8.4 *All-1/0 vector*. A sequence of binary values where all elements are 1/0.

Definition 8.5 *Walking-1/0 vector*. A sequence of binary values where a single 0/1 "walks" through a series of 1s/0s.

The *all-zero* (*all-one*) *vector* is used to excite the stuck-at-1 (stuck-at-0) interconnect fault and the *walking-zero vector* (*walking-one vector*) *vector* is used to excite the AND-bridge (OR-bridge) interconnect fault. Where the sequence length of each vector is equal to the number of logic input ports of MPLD, i.e., depends on the size of the MLUTs array.

Figure 8.10 shows the mechanisms of applying external patterns. The (a), (b), (c), (d) show detect a stuck-at-1, stuck-at-0, AND-bridge, and OR-bridge fault, respectively.

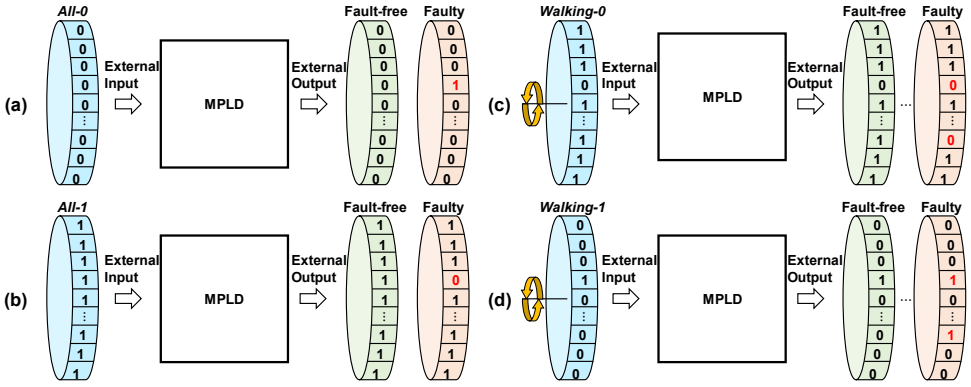


Figure 8.10 Applying mechanisms of external patterns.

Figure 8.11 and Figure 8.12 show the example of applying the *all-zero vector* and the *walking-zero vector* to excite a stuck-at-1 (*sa1*) and AND-bridge (*ANDbd*) interconnect fault in an MPLD with 2×2 MLUTs array, respectively. Figures (a) and (b) show the test under rm_1 and rm_2. The rm_1 and rm_2 are respectively created by $TC^{(1)}$ and

$TC^{(2)}$ stored in SRAMs to propagate the fault effect along with the horizontal and vertical routes. When applying the *all-zero*(*walking-zero*) *vector* to the external inputs, the *sa1*(*ANDbd*) fault is excited, and the fault effect(s) 1(0s) will be propagated to the external output ports for observation. As shown in the figures, the route that reaches the external output ports is individual; we can thus obtain the fault propagation path set on both the rm_1 and rm_2: $FP^{(1)}$ and $FP^{(2)}$, by observing the fault effects mapped on the external output ports. We can locate the fault by equation 8.2, and equation 8.3:

$$F_{loc} = FP^{(1)} \cap FP^{(2)} \text{ (for the } ANDbd, FP^{(1)} = \bigcup_{k=1}^{2} FP_k^{(1)}, FP^{(2)} = \bigcup_{k=1}^{2} FP_k^{(2)}).$$

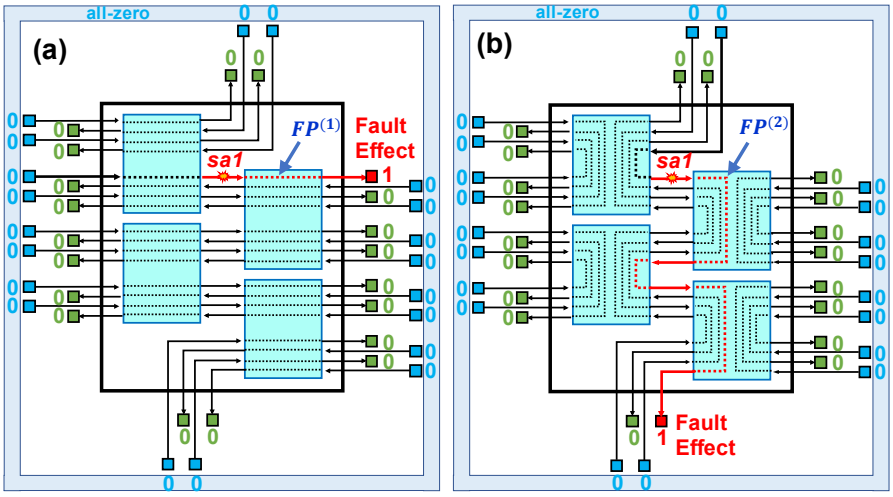


Figure 8.11 Apply *all-zero* to excite stuck-at-1 fault.

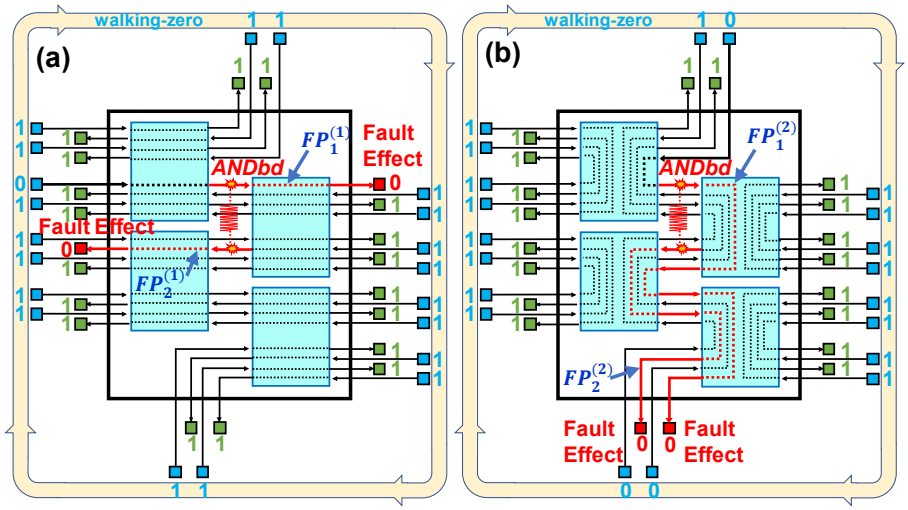


Figure 8.12 Apply *walking-zero* to excite AND-bridge fault.

8.3 Simulation Results

To verify the proposed test method, we performed logic simulations using *ModelSim* by injecting fault nodes to the netlist of the MPLD we designed.

The designed MPLD has 36 MLUTs arranged in a 6×6 array as shown in Figure 8.13. Its external logic IO ports include 48bit left/right IOs: *li[47:0]*, *lo[47:0]*, *ri[47:0]*, *ro[47:0]* and 20bit top/bottom IOs: *ti[19:0]*, *to[19:0]*, *bi[19:0]*, *bo[19:0]*. Each MLUT has *16-pair* AD interconnects ($A_{15:0}$, $D_{15:0}$) and consists of four *256word×16-bit* SRAMs including two asynchronous SRAMs: SRAM1, SRAM2, two synchronous SRAMs: SRAM3, SRAM4.

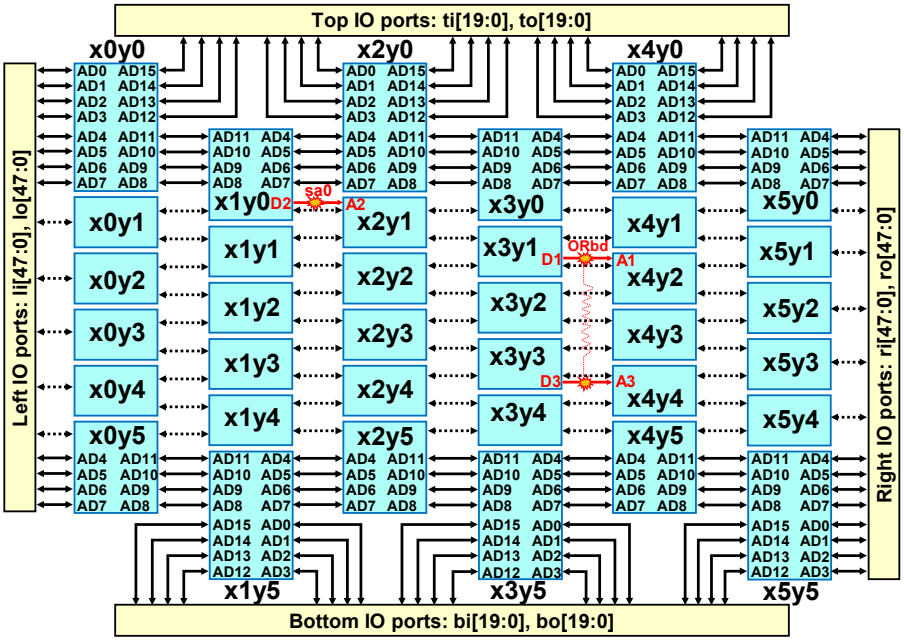


Figure 8.13 MPLD with 6×6 MLUTs array.

The processes of performing logic simulations are as the following.

1: We injected a stuck-at-0 (*sa0*) fault at $\mathbf{x_2y_1A_2}$ (address line A_2 of MLUT x_2y_1) and an OR-bridge (*ORbd*) fault between $\mathbf{x_4y_2A_1}$ and $\mathbf{x_4y_4A_3}$.

2: In the memory operation mode of the MPLD, we configured the test cubes into SRAMs of each MLUT to create the route maps.

3: In the logic operation mode, we applied external test patterns *all-one*(*walking-one*) *vector* to external logic input ports (*li*, *ri*, *ti*, *bi*) to excite the injected *sa0*(*ORbd*) fault.

4: We observed the fault effects at external output ports (*lo*, *ro*, *to*, *bo*) and located fault location through the fault propagation paths of the observed fault effects.

8.3.1 Verification of Testing to Stuck-at Interconnect Faults

The simulation results of testing the *sa0* fault under the rm_1(rm_2) are shown in Figure 8.14(Figure 8.15). Before we enable the fault injection *sa_fltinj_en* (=0), the MPLD is fault-free, and its output ports are *all-1*. When enabling *sa_fltinj_en* (=1), $\boldsymbol{x_2y_1A_2}$ is fixed

to 0 and the value 0 of the fault effect is propagated along the horizontal(vertical) route to the port *ro[6]*(*bo[14]*) (=0). The *sa0* propagation path set on the rm_1 and rm_2, $FP^{(1)}$ and $FP^{(2)}$, can be determined:

$$FP^{(1)} = \{ li[10] \rightarrow x_1y_0A_{13} \rightarrow \boldsymbol{x_2y_1A_2} \rightarrow x_3y_0A_{13} \rightarrow x_4y_1A_2 \rightarrow x_5y_0A_{13} \rightarrow ro[6]\},$$

$$FP^{(2)} = \{ ti[14] \rightarrow x_1y_0A_5 \rightarrow \boldsymbol{x_2y_1A_2} \rightarrow x_1y_1A_5 \rightarrow x_2y_2A_2 \rightarrow x_1y_2A_5 \rightarrow x_2y_3A_2 \rightarrow x_1y_3A_5 \rightarrow x_2y_4A_2 \rightarrow x_1y_4A_5 \rightarrow x_2y_5A_2 \rightarrow x_1y_5A_5 \rightarrow bo[14] \}.$$

The *sa0* can be located:

$$F_{loc} = \bigcap_{i=1}^{2} FP^{(i)} = FP^{(1)} \cap FP^{(2)} = \boldsymbol{x_2y_1A_2}.$$

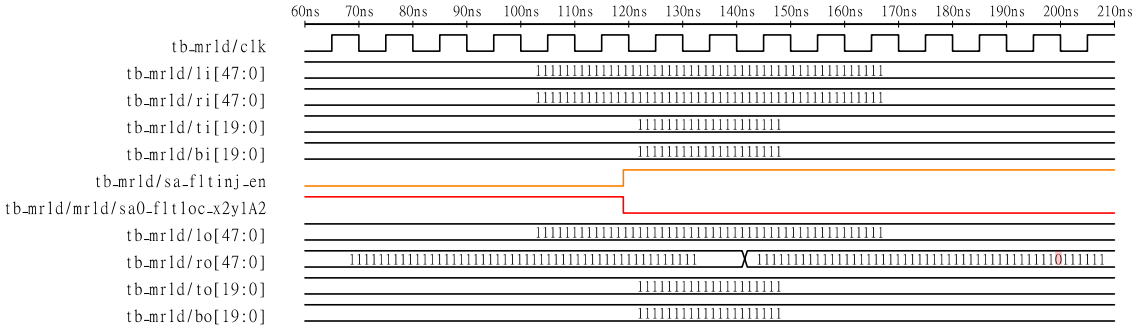


Figure 8.14 Simulation result of the test under rm_1 for *sa0*.

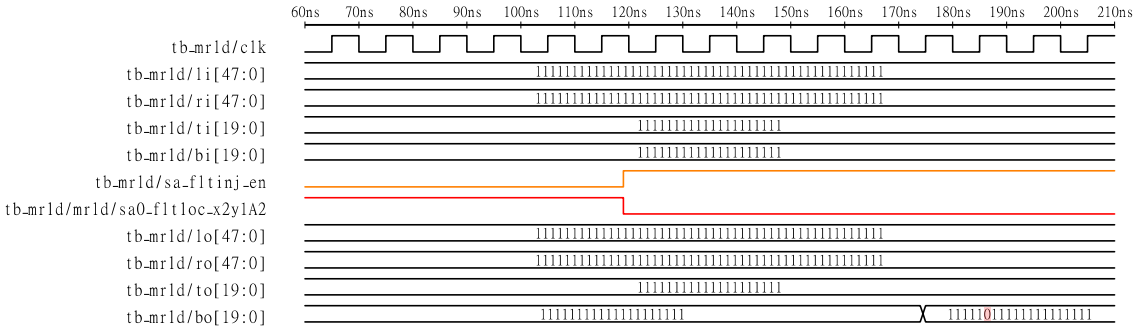


Figure 8.15 Simulation result of the test under rm_2 for *sa0*.

8.3.2 Verification of Testing to Bridge Interconnect Faults

The simulation results of testing the *ORbd* fault under the rm_1(rm_2) are shown in Figure 8.16(Figure 8.17). A bridge fault is injected into the MLUT array by setting *bd_fltinj_en* to 1. The $\boldsymbol{x_4y_2A_1}$ and $\boldsymbol{x_4y_4A_3}$ are bridged by the OR logic function:

$$x_4y_4A_3^{(0\rightarrow 1)} = x_4y_2A_1^{(=1)} = x_4y_2A_1^{(=1)} \vee x_4y_4A_3^{(=0)} = 1.$$

The fault effect value 1s are propagated along horizontal(vertical) route to respectively the port *ro[13]* and *ro[31](bo[5] and bo[7])* (=1). The *ORbd* propagation paths can be

obtained:

$$FP_1^{(1)} = \{\, li[17] \rightarrow x_1y_1A_{14} \rightarrow x_2y_2A_1 \rightarrow x_3y_1A_{14} \rightarrow \boldsymbol{x_4y_2A_1} \rightarrow x_5y_1A_{14} \rightarrow ro[13] \,\},$$

$$FP_2^{(1)} = \{\, li[35] \rightarrow x_1y_3A_{12} \rightarrow x_2y_4A_3 \rightarrow x_3y_3A_{12} \rightarrow \boldsymbol{x_4y_4A_3} \rightarrow x_5y_3A_{12} \rightarrow ro[31] \,\},$$

$$FP_1^{(2)} = \{\, ti[5] \rightarrow x_3y_0A_6 \rightarrow x_4y_1A_1 \rightarrow x_3y_1A_6 \rightarrow \boldsymbol{x_4y_2A_1} \rightarrow x_3y_2A_6 \rightarrow x_4y_3A_1 \rightarrow x_3y_3A_6 \rightarrow x_4y_4A_1 \rightarrow x_3y_4A_6 \rightarrow x_4y_5A_1 \rightarrow x_3y_5A_6 \rightarrow bo[5] \,\},$$

$$FP_2^{(2)} = \{\, ti[7] \rightarrow x_3y_0A_4 \rightarrow x_4y_1A_3 \rightarrow x_3y_1A_4 \rightarrow x_4y_2A_3 \rightarrow x_3y_2A_4 \rightarrow x_4y_3A_3 \rightarrow x_3y_3A_4 \rightarrow \boldsymbol{x_4y_4A_3} \rightarrow x_3y_4A_4 \rightarrow x_4y_5A_3 \rightarrow x_3y_5A_4 \rightarrow bo[7] \,\}.$$

The *ORbd* can be identified as:

$$F_{loc} = \bigcap_{i=1}^{2} FP^{(i)} = \bigcap_{i=1}^{2} \left(\bigcup_{k=1}^{2} FP_k^{(i)} \right) = \left(FP_1^{(1)} \cup FP_2^{(1)} \right) \cap \left(FP_1^{(2)} \cup FP_2^{(2)} \right) = \{\, \boldsymbol{x_4y_2A_1}, \boldsymbol{x_4y_4A_3} \,\}.$$

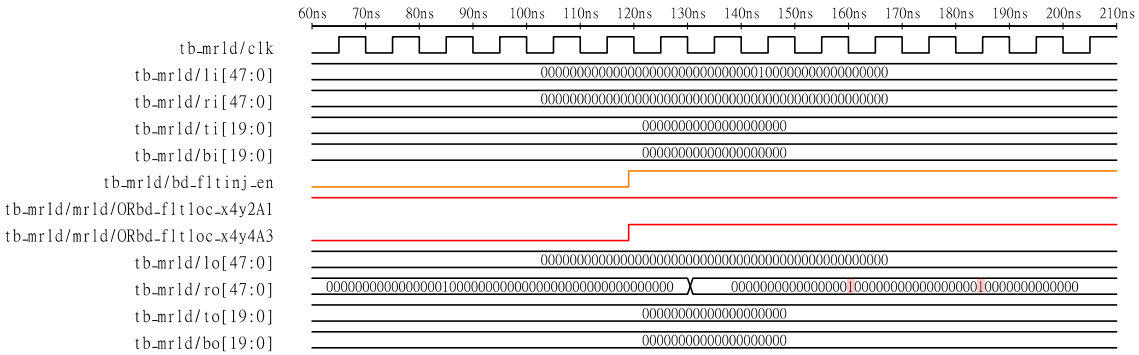


Figure 8.16 Simulation result of the test under rm_1 for *ORbd*.

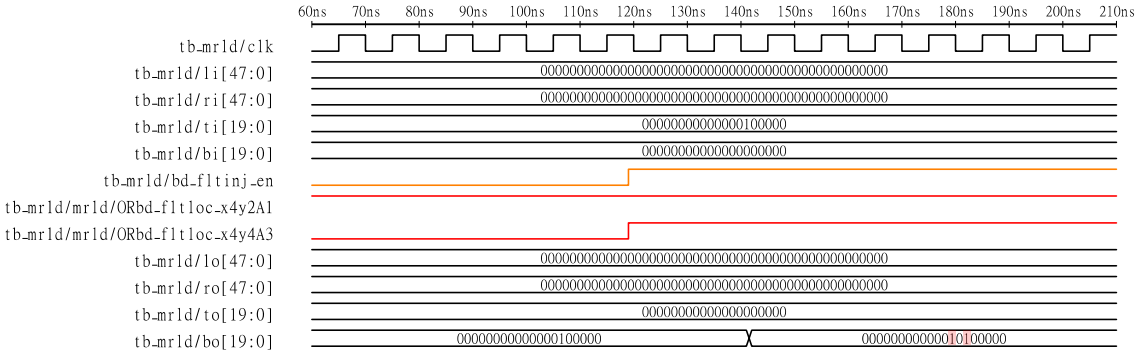


Figure 8.17 Simulation result of the test under rm_2 for *ORbd*.

8.4 Discussion

8.4.1 Test Effectivity for Interconnect Faults

The proposed test method can detect and locate all single interconnect faults at any

AD interconnects. Table 8.3 shows the test effectivity to an MPLD with the size of $x \times y$ MLUTs (having m-pair AD interconnects) array. For each type of fault, the most number of fault lines that might exist in the MPLD is $((x+1)y + (x-1)/2)m$, and all can be located by the proposed test method. Locating the stuck-at (*sa*) fault only requires two-time reconfigurations (*config.*) and external test patterns (*EP*); for the bridge (*bd*) fault, it only involves two-time or three-time (depending on the location distribution of the two bridged interconnects).

Table 8.3 Test effectivity for all single AD interconnect faults.

MPLD size	locatable (=total) fault numbers	config. (times)		EP (times)	
MLUT:$x \times y$	$\left((x+1)y+\frac{x-1}{2}\right)m$	*sa*	*bd*	*sa*	*bd*
AD-pair: m		2	2 or 3	2	2 or 3

8.4.2 Time Complexity of the Test Procedure

The time complexity of the test is $O(N_{mlut})$, where N_{mlut} denotes the number of MLUTs in the MPLD. The test procedure includes two phases:

1) *Configuration phase*: Write test cubes into memory to configure the test route map (routing logic) in memory operation mode.

2) *Logic phase*: Run the configured logic by applying test patterns to the external logic input ports in logic operation mode.

Let t_{conf} denote the time to configure the test cube into an MLUT, and t_{logic} denote the time to run the configured logic. N_{rm} represents the number of route maps (for stuck-at fault N_{rm} =2, for bridge fault N_{rm} = 2 or 3).

For an MPLD with N_{mlut} MLUTs, the total test time is:

$$t_{test} = (N_{mlut} \times t_{conf} + t_{logic}) \times N_{rm}$$

Since time t_{conf} and t_{logic} depend on the memory access speed, which is common for all MLUTs, the total testing time is determined by the size of the MLUTs array. Therefore, the time complexity of the test is $O(N_{mlut})$.

8.4.3 Test Availability for Multiple Interconnect Fault

The proposed test method is available for multiple faults at the AD interconnects.

Definition 8.5 *N-faults*. N faults, which exist at the different AD interconnects on the MLUT array, where $N > 1$.

Testing mechanisms of multiple faults:

Let, the range of the multiple AD interconnect faults that occur be $N_F{}^R$, the number of faults in the range $N_F{}^R$ be N_F, and the number of faulty effects observed on the testing under the route map i is denoted by $N_{FE}{}^{(i)}$, where $i = 1, 2, 3, \ldots$, denote the horizontal route map, vertical route map, diagonal route map, ..., respectively.

For N-faults existing at any site on the MLUT array, the fault range can be determined by the following equation:

$$N_F \in N_F{}^R = [max(N_{FE}{}^{(1)}, N_{FE}{}^{(2)}), N_{FE}{}^{(1)} \times N_{FE}{}^{(2)}] \tag{8.4}$$

And they can be detected and located by executing i-times tests with i route maps until the following equation holds.

$$N_F = N_{\cap_{i=1}^{N_{rm}} FP^{(i)}} \text{ , } if\ max(N_{FE}{}^{(1)}, \cdots, N_{FE}{}^{(i)}) == N_{\cap_{i=1}^{N_{rm}} FP^{(i)}} \tag{8.5}$$

$$F_{loc} = \cap_{i=1}^{N_{rm}} FP^{(i)} \text{ , } if\ max(N_{FE}{}^{(1)}, \cdots, N_{FE}{}^{(i)}) == N_{\cap_{i=1}^{N_{rm}} FP^{(i)}} \tag{8.6}$$

For an instance: to identify *3-faults* (stack-at-1) at interconnect A, B, and C on the MLUT, as shown in Figure 8.18.

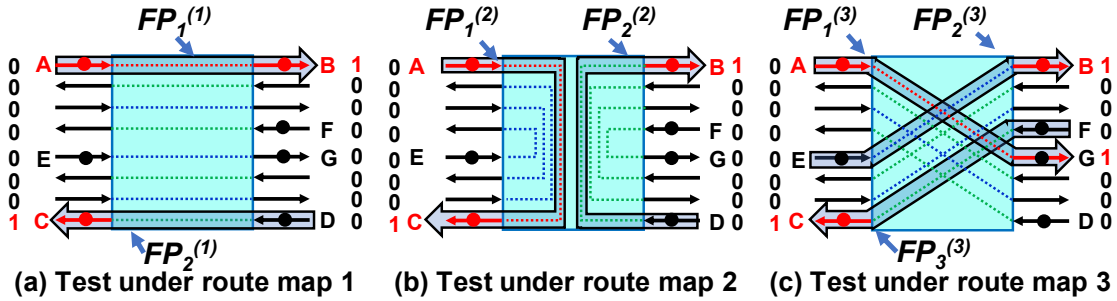


Figure 8.18 Example to identify multiple faults.

By executing the test under two route maps (1: horizontal route map, 2: vertical route map), we can determine the range $N_F{}^R$ of the number of multiple faults N_F:

$N_{FE}{}^{(1)} = 2,$

$N_{FE}{}^{(2)} = 2,$

$max(N_{FE}{}^{(1)}, N_{FE}{}^{(2)}) = 2,$

$N_F \in N_F{}^R = [max(N_{FE}{}^{(1)}, N_{FE}{}^{(2)}), N_{FE}{}^{(1)} \times N_{FE}{}^{(2)}] = [2,4],$

$FP^{(1)} = \cup_{k=1}^{N_{FE}{}^{(1)}} FP_k^{(1)} = FP_1^{(1)} \cup FP_2^{(1)} = \{A,B\} \cup \{C,D\} = \{A,B,C,D\},$

$FP^{(2)} = \cup_{k=1}^{N_{FE}{}^{(2)}} FP_k^{(2)} = FP_1^{(2)} \cup FP_2^{(2)} = \{A,C\} \cup \{D,B\} = \{A,B,C,D\},$

$\cap_{i=1}^{2} FP^{(i)} = FP^{(1)} \cap FP^{(2)} = \{A,B,C,D\}.$

Since the

$(\ max(N_{FE}{}^{(1)}, N_{FE}{}^{(2)}) = 2\ ) \neq (\ N_{\cap_{i=1}^{2} FP^{(i)}} = 4\ ),$

we need to perform the test using route map 3 (diagonal route map):

$N_{FE}^{(3)} = 3$,
$max(N_{FE}^{(1)}, N_{FE}^{(2)}, N_{FE}^{(3)}) = 3$,
$FP^{(3)} = \bigcup_{k=1}^{N_{FE}^{(3)}} FP_k^{(3)} = FP_1^{(3)} \cup FP_2^{(3)} \cup FP_3^{(3)} = \{A,G\} \cup \{E,B\} \cup \{F,C\} = \{A,B,C,E,F,G\}$.
$\bigcap_{i=1}^{3} FP^{(i)} = FP^{(1)} \cap FP^{(2)} \cap FP^{(3)} = \{A,B,C,D\} \cap \{A,B,C,E,F,G\} = \{A,B,C\}$.

Then, the
$(max(N_{FE}^{(1)}, N_{FE}^{(2)}, N_{FE}^{(3)}) = 3) == (N_{\bigcap_{i=1}^{3} FP^{(i)}} = 3)$,
we can determine the number of multiple faults and fault locations:

$N_F = N_{\bigcap_{i=1}^{3} FP^{(3)}} = 3$,
$F_{loc} = \bigcap_{i=1}^{3} FP^{(i)} = \{A,B,C\}$.

8.5 Conclusions

In order to improve the yield and guarantee the reliability of the MPLD device, in manufacturing the MPLD, identifying the AD interconnect faults in the MLUTs array is beneficial to improving the process. In addition, when the MPLD is put to actual use in the field, identifying the AD interconnect faults is helpful to avoid configuring the logic into a faulty MLUT block for high reliability.

In this chapter, we proposed a test method to identify the stuck-at and bridge faults at the AD interconnect between MLUTs of the MPLD device. The proposed test method consists of two phases: the configuration phase and the logic phase. The configuration phase creates the route map in the MLUTs array for fault propagation paths by configuring the pre-generated internal test data into the SRAMs of the MLUTs. The logic phase applies the pre-generated external test data to the logic external output ports of the MPLD to excite the target faults, observe the faulty effects at the external logic output ports of MPLD (fault detection), and obtain the fault propagation path set through the observed faulty effects (for fault location). The coordinate of the target interconnect fault can be determined by performing the two phases under route maps. The main contribution of this test method is to address not only the fault detection but also the fault diagnosis of MPLD.

To evaluate the proposed test method, we design an MPLD with a 6×6 MLUTs array and perform the logic simulation experiments by injecting the stuck-at and bridge fault node to the netlist of the MPLD. The results confirmed the effectiveness of the proposed test method which can diagnose the location of the injected stuck-at and bridge fault.

The proposed test method can detect and locate all single interconnect faults at any AD interconnects. In addition, it is available for multiple faults at the AD interconnects.

In our future work, we will explore the test generation of the internal test data and external data to identify other interconnect faults (such as the open fault, etc.) in the MPLD device, and explore the methods such as the design for testability and built-in self-test to the MPLD device.

Chapter 9

9. Aging monitoring for MPLD

To improve the reliability of the MPLD device, in *Chapter 8*, we described the test approaches for identifying the production defects referred to as stuck-at and bridge faults at the AD interconnects of the MPLD device. On the other hand, when a good MPLD device is put actual such as the uses of IoT and AI systems for a long time or works in a severe environment, various aging phenomena such as HCI (Hot carrier injection), BTI (Bias Temperature Instability) [50], [51] would cause delay degradation that threatens the in-field reliability [52] of the MPLD.

This chapter aims to present a method to periodically detect the degradation state of MLUTs in MPLDs operating in the field by monitoring the delay induced by aging.

The rest of this chapter is organized as follows: *Section 9.1* points out the necessity of delay monitoring techniques for improving the in-field reliability of MPLD. *Section 9.2* presents the implementation method of the RO-based delay monitor for the MPLD device. *Section 9.3* performs logic simulation to evaluate the proposed methods. *Section 9.4* discuss the proposed methods. Finally, the chapter concludes in *Section 9.5*.

9.1 Delay-Monitoring technologies

This section points out the necessity of delay monitoring techniques for improving the in-field reliability of MPLD.

Conventionally, the aging-induced extra delay can be relaxed by manufacturing tests (burn-in tests or stress tests), redundancy design, or by setting a certain timing margin in the operating frequency of the device at the design phase [62], [63]. However, it is difficult to optimize the timing margin for a device due to the variations in the fabrication process, workload, and operational environment. And there is a pessimistic prediction that the timing margin usually results in performance sacrifice although it can improve the reliability of the device [63].

For an MPLD device, it is composed of a large number of MLUT arranged in an array. During the operation, the progress of the aging at each single MLUT is different. When configuring a logic circuit into MPLD, the progress of aging at the often-used MLUTs would be faster which causes more extra delay. The variety of the aging-induced delay at MLUTs would affect the performance of the constructed logic circuit.

Commonly, a certain timing margin is pre-designed in the operating frequency of the MPLD device could cover the aging-induced delay of the MLUTs during most lifetimes. However, as the aging progresses, the delay at the MLUTs with faster aging progression would exceed the timing margin earlier which could cause a sudden system failure. On the other hand, it is getting difficult to design the timing margin for a device due to the variations in the fabrication process, workload, and operational environment.

Delay-monitoring techniques [62], [64] are one of the effective ways to ensure the in-filed reliability of the device, they can measure the delay variation of the circuit affected by the process, voltage, temperature (PVT) in real-time by implementing some special timing-measurement circuits such as the RO (Ring oscillator) [65], TDC (time-to-digital converter) [66] into the target device. Figure 9.1 shows the concept of delay monitoring. The delay of the device is measured periodically during the in-filed operation. When the delay value is getting exceed the timing margin (or a safe delay boundary), an early warning/report can be issued to the upper system to avoid a system failure or call for maintenance like repair/diagnosis.

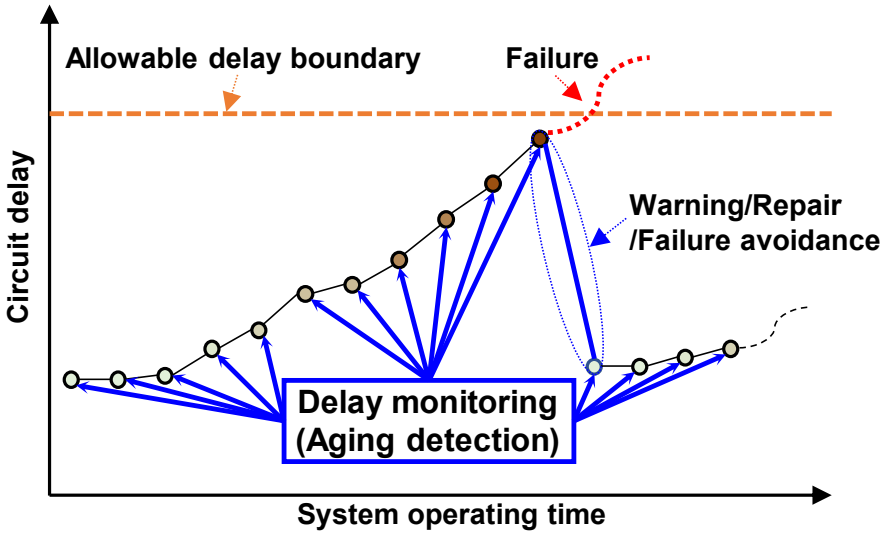


Figure 9.1 Concept of delay monitoring techniques.

Therefore, it is necessary to the delay monitoring techniques for improving the in-field reliability of MPLD.

9.2 Delay Monitoring in MPLD

This section presents the implementation method of the RO-based delay monitor for the MPLD device.

RO is commonly used as a sensor to monitor the delay variation of a circuit affected

by temperature, voltage, process, or aging on the circuit. To measure the delay of a circuit, it is effective to implement a RO in the device. In [67][68], the authors proposed an on-chip digital delay sensor using ROs to monitor the aging-induced delay of application-specific integrated circuit (ASIC) devices. Additionally, in [69], the authors integrated the on-chip digital delay sensor into the field-programmable gate array (FPGA) with the goal of enhancing the reliability of logic reconfigurable devices. As a result, we incorporate ROs into MPLDs for delay monitoring purposes.

9.2.1 Ring Oscillator (RO)

Figure 9.2 shows a general RO structure, a ring circuit composed of a 2-input *NAND* gate in series with an even number of inverters. One of the inputs of NAND is the oscillation control signal *EN*. While setting *EN* to 0, the RO is initial to a stable state; when setting *EN* to 1, the RO operates in the oscillation mode and generates an oscillation signal at a specific frequency. The delay (transmission time) D_{RO} of RO's entire ring routing path is half of the oscillation period T_{RO} of RO; we can calculate it through the oscillation number $N_{osc}^{t_{RO}}$ within a certain oscillation operation time t_{RO}:

$$D_{RO} = \frac{T_{RO}}{2} = \frac{t_{RO}}{2N_{osc}^{t_{RO}}} \qquad (9.1)$$

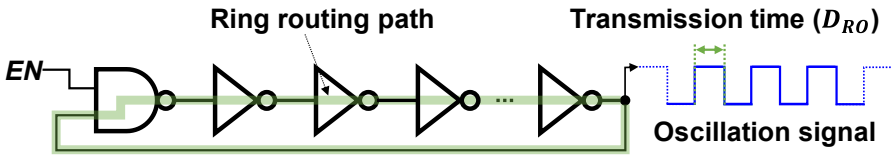


Figure 9.2 Ring oscillator.

9.2.2 Delay Monitor Design Using RO

In MPLD, we can deploy RO into specified measurement areas (partial MLUTs or all MLUTs) to measure the average delay (local delay or global delay) of MLUTs within the area.

Deploy RO in MLUTs:

We first specify the measurement area (MLUTs to be measured for the delay), then design the RO structure according to the following deployment rules:

Rule 9.1 *RO elements*. A NAND gate and an even number of inverters must route in series in a ring.

Rule 9.2 *Deployment area*. All elements must lie within the measurement area, and the ring routing path must pass through each MLUT within rather than outside the area.

Rule 9.1 is required to satisfy that the circuit can oscillate. *Rule 9.2* is required to ensure the delay measured is exactly the delay of the measurement area.

Figure 9.3(a) shows an example of deploying a RO circuit into MLUTs. RO elements are placed in the MLUTs to be measured and routed in series in a ring through AD interconnects of the MLUTs. We can calculate the average delay D_{MLUT} of the MLUTs through the transmission time D_{RO} of the ring routing path and the number N_{AD} of AD interconnects passed by the ring routing path:

$$D_{MLUT} = \frac{D_{RO}}{N_{AD}} = \frac{t_{RO}}{2\mathrm{N}_{\mathrm{osc}}^{t_{RO}} N_{AD}} \qquad (9.2)$$

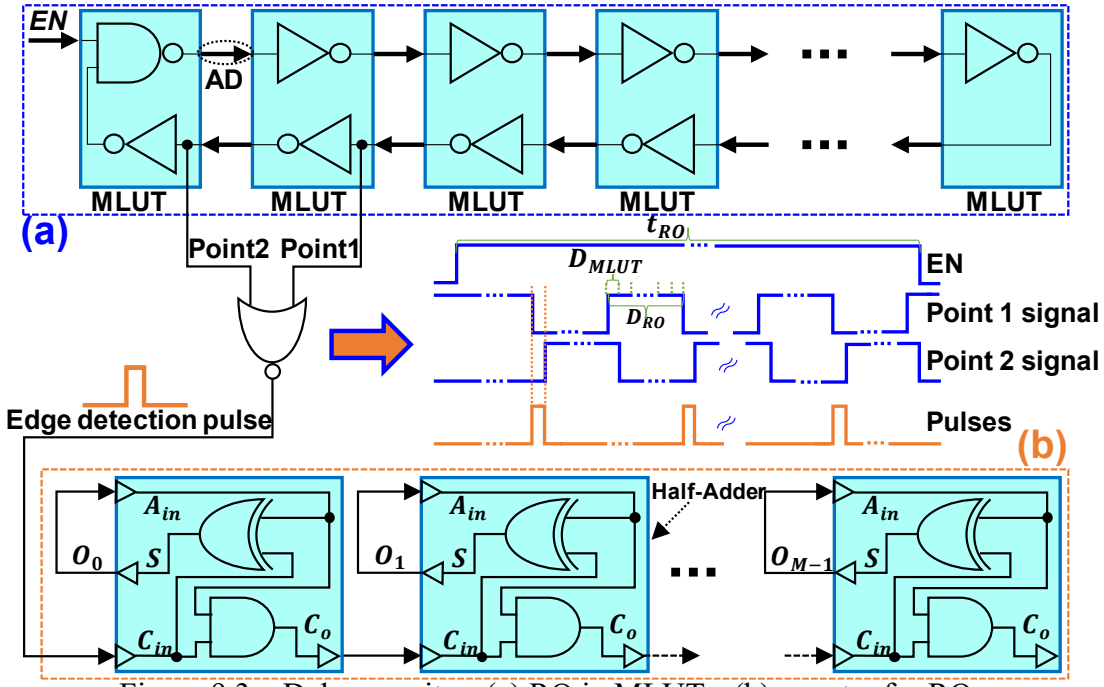


Figure 9.3 Delay monitor; (a) RO in MLUTs, (b) counter for RO.

Deploy Counter for RO:

Here we describe the design of the counter to calculate the oscillation number ($\mathrm{N}_{\mathrm{osc}}^{t_{RO}}$). Versus the conventional counter design composed of synchronous Flip-Flops, in this study, we proposed a new counter circuit design that is even more adapted and simpler to implement in the MPLD, as shown in Figure 9.3(b). The proposed counter consists of M half-adders connecting in series. When setting the RO oscillation mode, pulses can be outputted by a signal edge detection gate (here, we used NOR gate) by comparing the signals of two neighbor AD interconnects on the ring routing path. The $\mathrm{N}_{\mathrm{osc}}^{t_{RO}}$, i.e., the number of pulses, can be calculated by the counter by performing an addition carry operation for the pulses:

$$N_{osc}^{t_{RO}} = (O_{M-1} \cdots O_1 O_0)_2 \qquad (9.3)$$

The procedure of implementing RO and counter for measuring the delay of MLUTs is as follows.

Implementation Procedure

Step 1: Select measurement area (MLUTs);

Step 2: Deploy RO and counter;

Step 3: Create the truth tables for each MLUT in the area;

Step 4: Write the truth tables into corresponding MLUTs;

Step 5: Set the MPLD to logic operation mode;

Step 6: Set oscillation operation time (EN=1);

Step 7: Observe the oscillation number (counter outputs).

9.3 Simulation Results

To evaluate the proposed delay monitor method, as shown in Figure 9.4, we configured a RO with 11 elements (a NAND and 10 inverters) and a counter with 8 half-adders into the measurement area (MLUTs: x_0y_0, x_1y_0, x_2y_0, x_3y_0, x_4y_0) of the designed MPLD with 6×6 MLUTs array. We performed a logic simulation experiment using *ModelSim*.

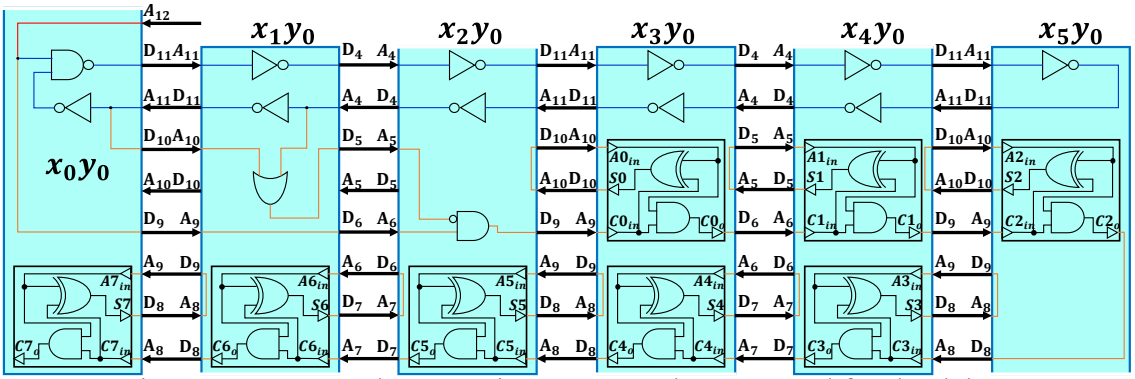


Figure 9.4 RO and counter in MLUTs to be measured for the delay.

1: We routed the RO pass through 10 AD interconnects in the measurement area (N_{AD}=*10*).

2: We set the delay of the ATD circuit for each MLUT to *5.5ns* and the overall oscillation operation time of the RO to *2000ns* (t_{RO}).

The waveform of the RO oscillation and the counter is shown in Figure 9.5. When setting the oscillation control signal to 1, the RO begins oscillating while the counter counts the detected pulse until the oscillation control signal becomes 0. The pulse number

(RO oscillation number) counted by the counter is 18:

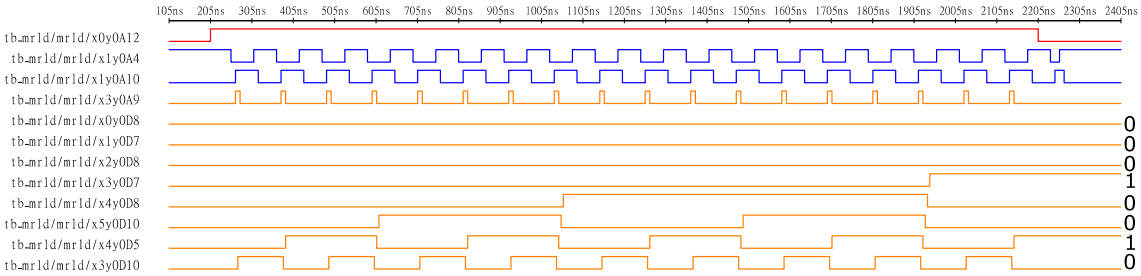


Figure 9.5 Simulation waveform to measure delay for MLUT.

$$N_{osc}^{t_{RO}} = (00010010)_2 = 18$$

Thus, the average delay of MLUTs in the area can be calculated by equation 9.2:

$$D_{MLUT} = \frac{t_{RO}}{2N_{osc}^{t_{RO}} N_{AD}} = \frac{2000ns}{2\times18\times10} = 5.\dot{5}ns$$

Comparing the set delay ($5.5ns$) of the ATD circuit with the D_{MLUT} ($5.\dot{5}ns$), the result confirms the effectiveness of the proposed delay monitor method.

9.4 Discussion

9.4.1 Overhead of Inserting Delay Monitor

It is worth noting that the proposed delay monitor is configured in the measured MLUTs as the truth tables without incurring logical gates and routing costs. The only additional overhead is the time cost of configuring the truth tables of the delay monitor into the MLUTs.

9.4.2 Work Scope of Delay Monitor

The ATD circuit, which detects address changes in the asynchronous SRAM to drive the logic operation, is the most sensitive component to the aging-induced delays in the MPLD. The proposed delay monitor aims to detect delays occurring in the ATD logic for each MLUT. As for the SRAM read/write delay, existing memory delay testing methods can be employed to detect it during the memory operation mode of the MPLD, which is beyond the scope of this work.

9.4.3 Locating Abnormal MLUTs

The proposed delay monitor aims to periodically detect the degradation state of MLUTs in MPLDs operating in the field. To detect degraded MLUTs in an MPLD with an $M\times N$ array of MLUTs, as shown in Figure 9.6, the total number and location of delay monitors are determined according to the detection method described below.

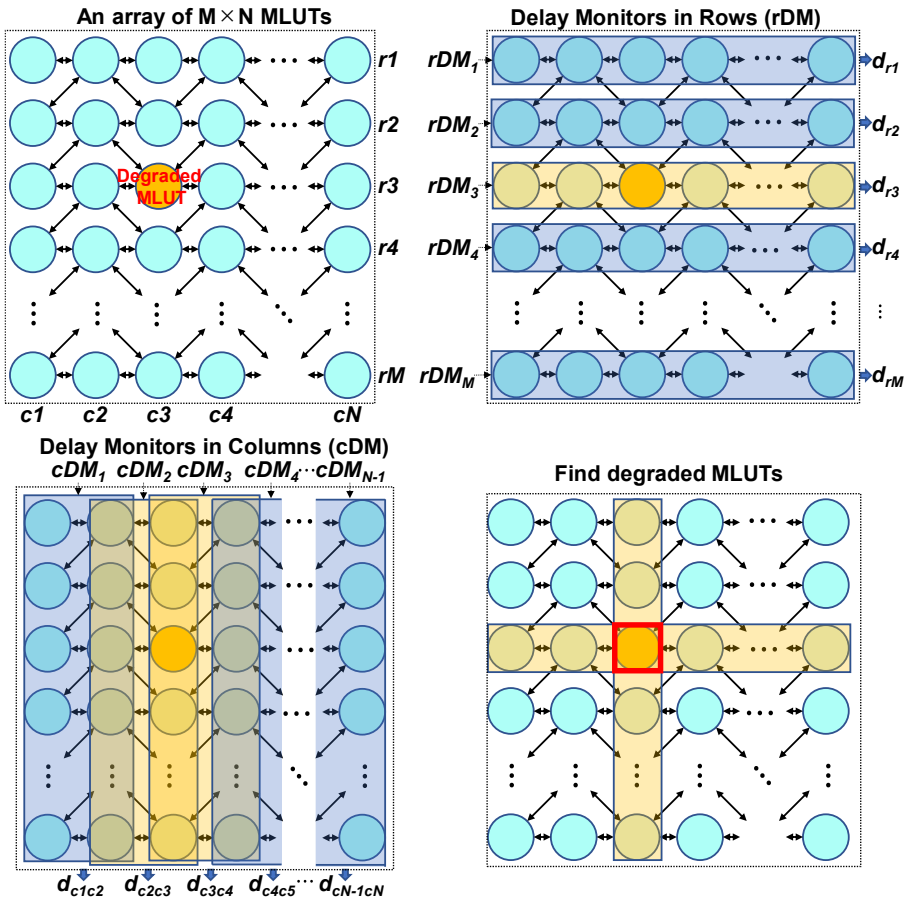


Figure 9.6 Delay-monitors deploying method.

(1) *row detection*: M delay monitors are configured in M rows (r_1, ..., r_M) to detect the average delay of MLUTs in each row (d_{r1}, ..., d_{rM}).

(2) *column detection*: N-1 delay monitors are configured in N columns (c_1, ..., c_N), where two adjacent columns are required to configure a delay monitor. This detects the average delay of MLUTs in two adjacent columns ($d_{c1,c2}$, ..., $d_{cN-1,cN}$).

(3) *locating degraded MLUTs*: MLUTs in the intersection region of rows with delays in M rows and columns with delays in N rows. For example, the MLUT at row 3 and column 3, is determined by d_{r3}, $d_{c2,c3}$, and $d_{c3,c4}$.

In this setup, row detection and column detection are configured simultaneously and work in parallel.

For achieving a certain or higher level of in-field reliability, we have examined this issue during our research, such as how to detect the details of the delay for each degraded MLUT. It is a challenging task that may require more complex models to build a finely designed monitor. This will be the subject of our future research. Nevertheless, locating abnormal MLUTs with large delays still represents a significant contribution to logic

designers as a reference.

9.5 Conclusions

In this chapter, to detect and report the aging state of MPLD devices during field operation, we have proposed an approach that uses a ring oscillator circuit for monitoring the aging by periodically measuring the delay of MLUTs in the field during MPLD's operation.

To configure the ring oscillator circuit into MPLD, we have proposed the design and implementation method of a ring oscillator circuit suitable for the structure of the MPLD device and designed a counter to store the RO oscillation frequency.

The proposed method can measure the Global Delay (of all MLUTs) and the Local Delay (of specified MLUTs) in the MPLD device.

To evaluate the proposed methods, we designed an MPLD with a 6×6 MLUTs array and performed logic simulations by injecting delay into MPLD. From the results of the logic simulation performed as an evaluation experiment, we confirmed that the proposed method can effectively measure the delay of the MLUT with a very small error.

In our future work, we will make a quantitative analysis of the aging phenomena, develop a precise simulation method as well as an on-chip test method, and explore the methods to determine the total number and locations of delay monitors for achieving a certain or higher level of in-field reliability.

Part IV: Application of MPLD

Chapter 10

10. A Solution to Implement Neural Networks in MPLD

With the rapid spread of *artificial intelligence* (AI) applications, the *neural networks* (NNs) algorithm has achieved significant successes at the machine learning domains including computer vision [70], speech recognition [71], and robotics [72]. In a practical intelligence application, NNs usually consist of millions of parameters involving multiply-accumulation operations, which requires high-performance computing equipment. In addition, with the rapid spread of IoT (*Internet of Things*) technology in both the industrial and consumer fields, NNs are widely applied into various edge terminals, e.g.: battery-powered mobile devices, robots, electric vehicle etc.. In such systems, real-time processing, low power consumption and low cost are the main concerns with the computing device used for NNs [73]. In order to achieve high performance and energy efficiency for AI application, hardware design for NNs is gaining great attentions [74].

Over the past few years, the strategy of hardware design for NNs application can mainly be classified into three types: 1) Use GPUs *(Graphics Processing Units*) to accelerate NN training. 2) ASICs (*Application Specific Integrated Circuits*) design for NNs. 3) FPGA-based accelerators of NNs. The GPUs apply single-instruction-multiple-data in parallel processing that can significantly speed up the training process of complicate NNs [75][76][77][78], however, usually accomplished with huge energy cost (e.g.: NVIDIA A100 Tensor Core GPU, the *thermal design power* (TDP) is 400W [79]) that is not suitable for edge device. The ASIC design for NNs is another key strategy for achieving high performance and energy efficiency for NNs application, such as Google edge TPU (*tensor processing unit*), NVIDIA Xavier, and NovuTensor achieved good energy efficiency [80]. However, the extremely high development cost might obstruct the application of ASICs for IoT system. Compared to ASIC design, reconfigurable devices such as FPGAs allow the user to reprogram the functionality and routing in field that can provide a flexible and scalable platform for implementing the NNs application with high-performance and low power consuming [81], however, the large area, delay and power issues due to the programmable interconnect resources prevent the use of FPGAs, and high cost is not friendly to the end user of edge devices.

In MPLD, functions (arithmetic logic, wiring logic) are expressed in the form of truth tables pre-stored in the SRAMs of MLUT. Since large amount of interconnect resources

like in FPGA are not needed anymore, a large number of SRAMs can be integrated that provides a chance to implement large and complex functions in a single MPLD by truth tables, such like a LUT-based neuron activation function [82] and the *LUT-based Neural Networks* (L-NNs) instead of implementing an accelerator in FPGA (due to the limited memory size of LUT). Since the *LUT-based neuron model* [82] only operates memory, it thus would work much faster and low power than a traditional accelerator which has to perform the multiply-accumulation operations every cycle even though with acceleration circuits. Therefore, we believe that MPLD would be a promising alternative edge AI device for NNs application.

On the other hands, due to the special structure of MPLD, implementing a neural network with fully-connected structure is an impossible task. There is an issue to implement an NNs application in MPLD, it needs a newly designed NN structure to adapt to the MPLD special structure.

In this chapter, we suggest a *LUT-based neuron model* to realize neuron functions in truth table and propose a novel neural network structure named MNN (*MPLD-based Neural Network*) to adapt the special connection structure of MLUTs for implementing a neural network into MPLD. To confirm the LUT-based neuron model, we design a logic simulation experiment in an MPLD with 6×6 MLUTs array. The simulation results confirm the feasibility of LUT-based neuron function expression are the same as the results of the theoretical analysis. To evaluate the effectiveness of MNN, we also perform an experiment by training MNN with the MNIST dataset. The experimental results show that the MNN can get almost the same accuracy and loss for MNIST data recognition compared to a *fully-connected neural network* (FNN).

The main contributions of this study are as follows.

1) A LUT-based neuron model is introduced.

2) A novel network structure named MNN is proposed.

The chapter is organized as follows. *Section 10.1* suggests a LUT-based neuron model. *Section 10.2* proposes an MNN (MPLD-based Neural Network) for implementing the NNs application into an MPLD device and describes the characteristics and wiring connection way of the proposed MNN in MPLD device. *Section 10.3* performs two experiments for confirming the LUT-based neuron model and evaluating the effectiveness of the proposed MNN, respectively. *Section 10.4* concludes the chapter.

10.1 LUT-based neuron model

In this section, we suggest a LUT-based neuron model to realize neuron functions in truth tables in MPLD.

According to the operating principle of MPLD, any functions (including wiring logic) can be written into the MLUT in the form of a truth table that provides a new computing model for neuron activation functions in MPLD. In addition, a large number of MLUTs make it possible to implement a complete LUT-based NN into a single MPLD device.

Figure 10.1 shows a basic neuron of neural network (NN). The neuron function is expressed in formula: $u = \sum_{i=0}^{N} w_i \times x_i + b$, $y = f(u)$, where f is an activate function for u. To compute the value of y for u, a traditional approach has to perform multiply-accumulate operation and activate operation in many cycles and requires large memory (buffer) to store the weights and input/output vectors.

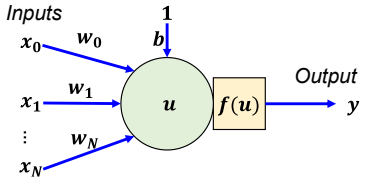


Figure 10.1 A NN neuron.

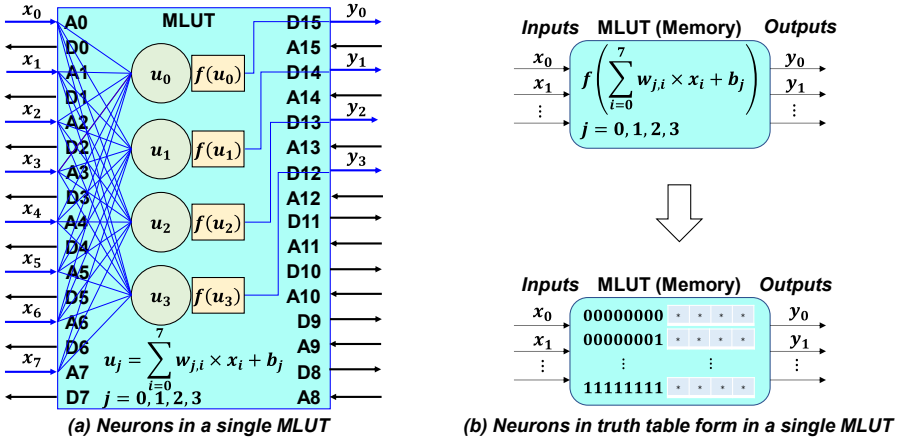


Figure 10.2 LUT-based neuron model in a single MLUT.

The main idea of this study is that a neuron function can be expressed in a truth table in MPLD. Figure 10.2 (a) shows an example to implement four neurons function in one MLUT. The correspondence between inputs x and outputs y of the neurons can be computed by pre-learning and formed in a truth table in the MLUT between the address-inputs and the data-outputs. Therefore, as shown in Figure 10.2 (b), when calculate the output y for a given input pattern x, it only needs to access the memory and readout the

prestored results of y. It is thus much faster and low power than a traditional accelerator which has to perform the multiply-accumulation operations every cycle even though with acceleration circuits.

Note that here we are discussing the binarization forms of x and y. For specific binarization methods of x and y, existing binarization methods are utilizable [83]; we will also conduct future research to explore other binarization methods valid for MNN.

10.2 MPLD-based Neural Network (MNN)

In this section, we explain and propose an MNN (MPLD-based Neural Network) for the aim of implementing a neural network into an MPLD device. we also describe the characteristics of the MNN and introduce the implementation way of the MNN neurons in MPLD.

10.2.1 A sparse neural network: MNN

A fully connected neural network (FNN) cannot be constructed directly into the MPLD. As shown in Figure **10.3**, all neurons of each layer are fully connected with the preceding layer. For the MPLD structure, as shown in Figure 10.4, each MLUT (e.g., MLUT5) can only connect up to four adjacent MLUTs (e.g., MLUT1, MLUT2, MLUT6, MLUT7), and the data outputs of other MLUTs (e.g., MLUT3, MLUT4) cannot be connected to the MLUT (MLUT5). Therefore, it is impossible to construct a fully connected NN into the MPLD due to such connection limits between MLUTs.

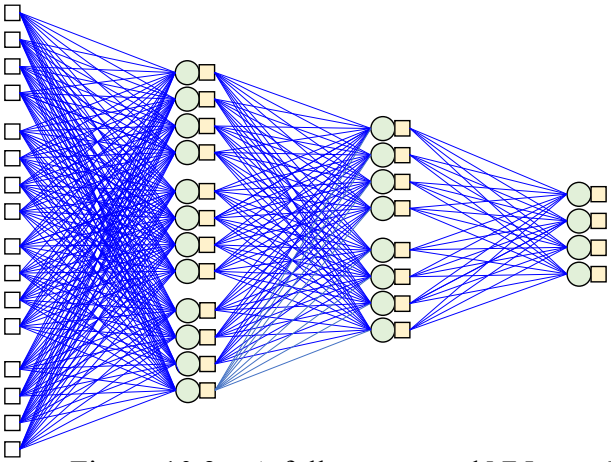


Figure 10.3 A fully connected NN.

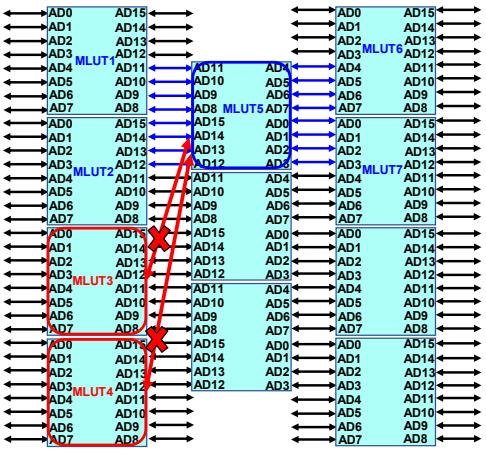


Figure 10.4 Connection limit in MPLD.

To implement a NN into MPLD, we propose a sparse neural network based on the MPLD structure named MNN (MPLD-based Neural Network) in this study. As shown in Figure 10.5, according to the structure of MPLD, we suggest sparsely connecting the neural network in units of MLUTs in MPLD. We call such a sparse neural network based

on the structure of MPLD an MNN (MPLD-based Neural Network), and its network structure is shown in Figure 10.6. The proposed MNN has inward gradual convergence and association characteristics to adapt the connection structure of MLUTs. In the input layer, the data input of each MLUT is independent of another MLUT, and the feature of these data will be converged and associated in the middle layer.

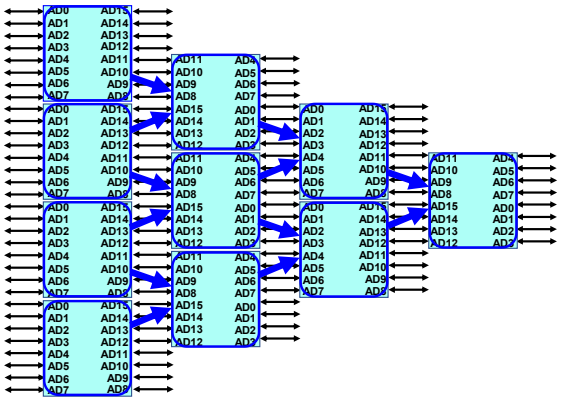


Figure 10.5 Sparse connection in unit of MLUT in MPLD.

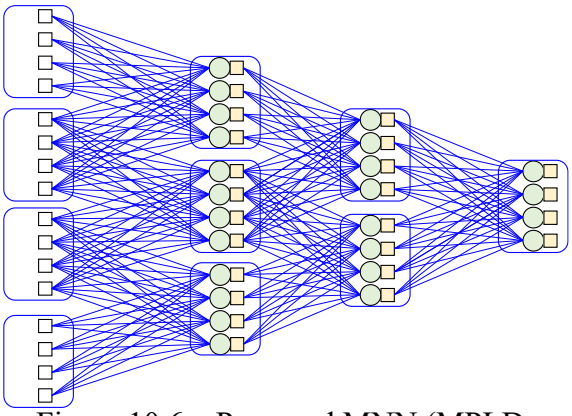


Figure 10.6 Proposed MNN (MPLD-based Neural Network)

Figure 10.7 shows an example of using MNN for a very simple image recognition application. Where, a 4×4 bit image of *O* and *Z* is given respectively, and the vector of each row (4bit) is applied to the address input of an MULT at the first column of the MLUT array, respectively. Throughout the hidden layers, the feature of each row vector will be converged inwardly and gradually, and associated until the output layer, where all features will be extracted and recognized.

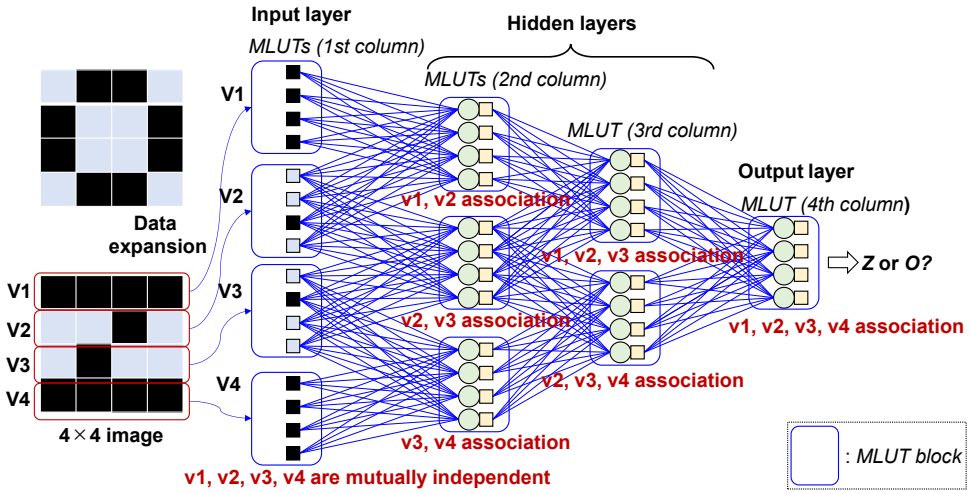


Figure 10.7 Feature extraction in MNN.

10.2.2 Implementing MNN into MPLD

Figure 10.8 shows the wiring method to connect the neurons between adjacent layers in an MLUT array where each MLUT has 16 bits AD lines. The output of each neuron

function configured in an MLUT will be read out and propagated to the following adjacent MLUT through only one AD (address-data) line. Then, the value of the address input from the preceding MLUT will be connected to the inputs of all neurons by configuring the branch logic (or wiring logic), e.g.: *AD11* in MLUT *x0y0* of Figure 10.8. Each neuron configured in an MLUT can connect to at most 8 neurons which are configured in the preceding adjacent MLUTs.

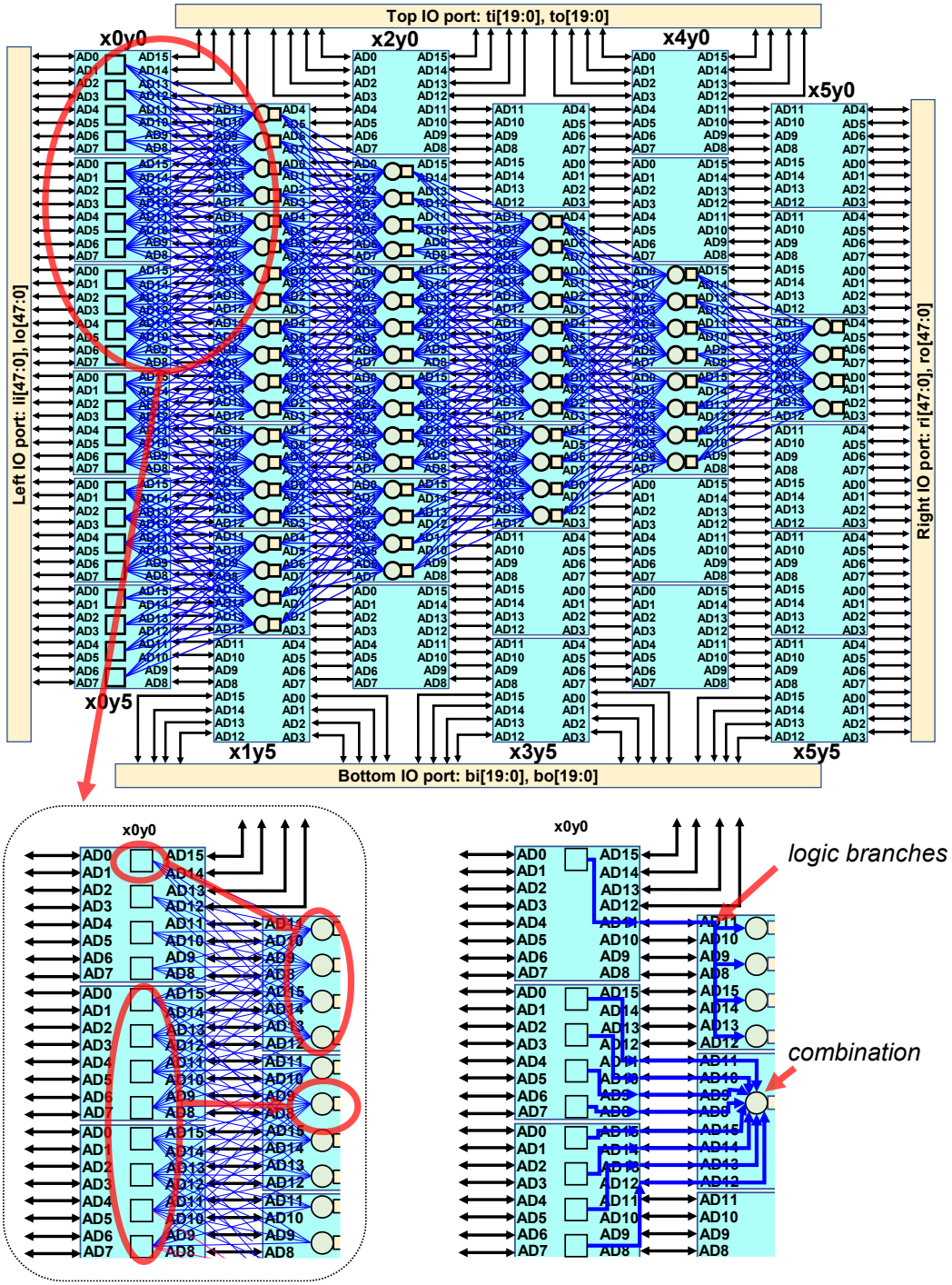


Figure 10.8 MNN wiring connection way in MPLD.

10.3 Experimental Results

In this section, we describe the performed the experiments. First, we design an experiment to confirm the LUT-based neuron model as described in section 10.1. Then, we also show the experimental results to confirm the effectiveness of the proposed MNN in the section 10.2 by the training using the MNIST dataset.

10.3.1 Confirm LUT-based Neuron Model

As shown in Figure 10.9, here a size of 4×4×4 NN is given, and each layer (Hidden-layer1, Hidden-layer2, Output-layer1) is constructed to MLUT x0y1, MLUT x1y0, MLUT x2y1, respectively. For simplicity in this experiment, for this NN we using the *Heaviside Step* (*Binary step*) as an activation function to calculate each neuron:

$$f(u) = \begin{cases} 1 & ,u \geq 1 \\ 0 & ,u < 1 \end{cases}$$

$$u = \sum_{i=0}^{N} w_i \times x_i + b$$

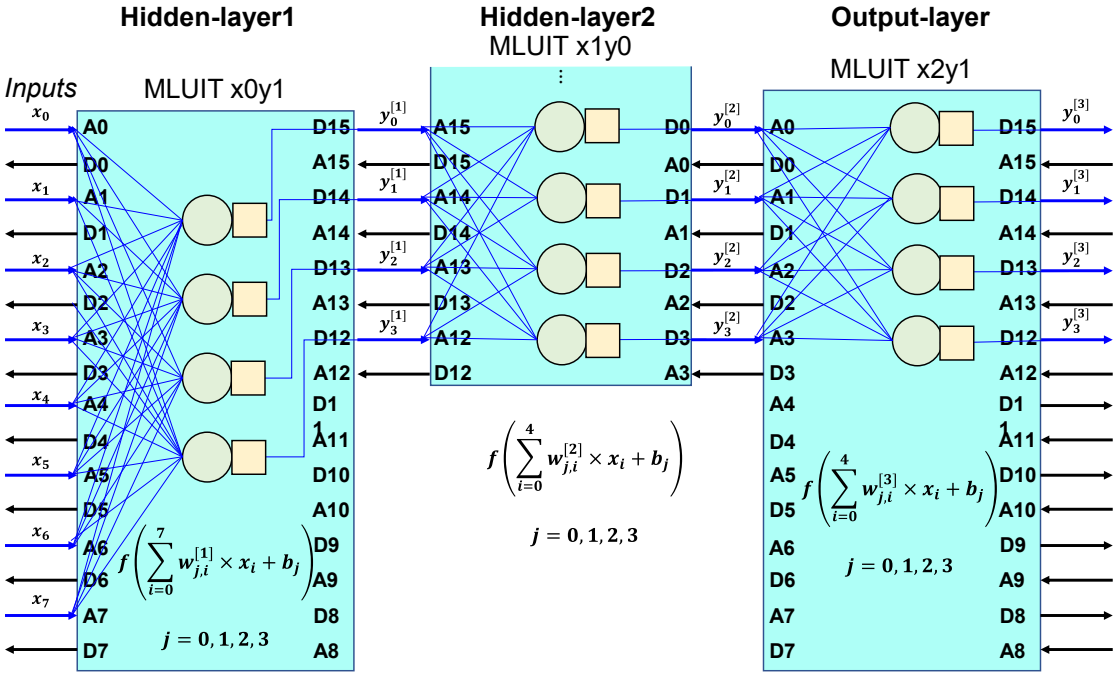


Figure 10.9 A size of 4×4×4 NN constructed in 3 MLUTs.

As shown in Figure 10.10, we give the value of the weights for each layer and assign the value of *b* to *0*. Each layer of given weights NN is calculated to a truth table stored in an MLUT to realize the neuron functions. Where, such as there are inputs 01010000, through the MLUT x0y1, MLUT x1y0, MLUT x2y1, theoretically the outputs of Hidden-layer1, Hidden-layer2, Output-layer is 0011, 1011, 0111, respectively.

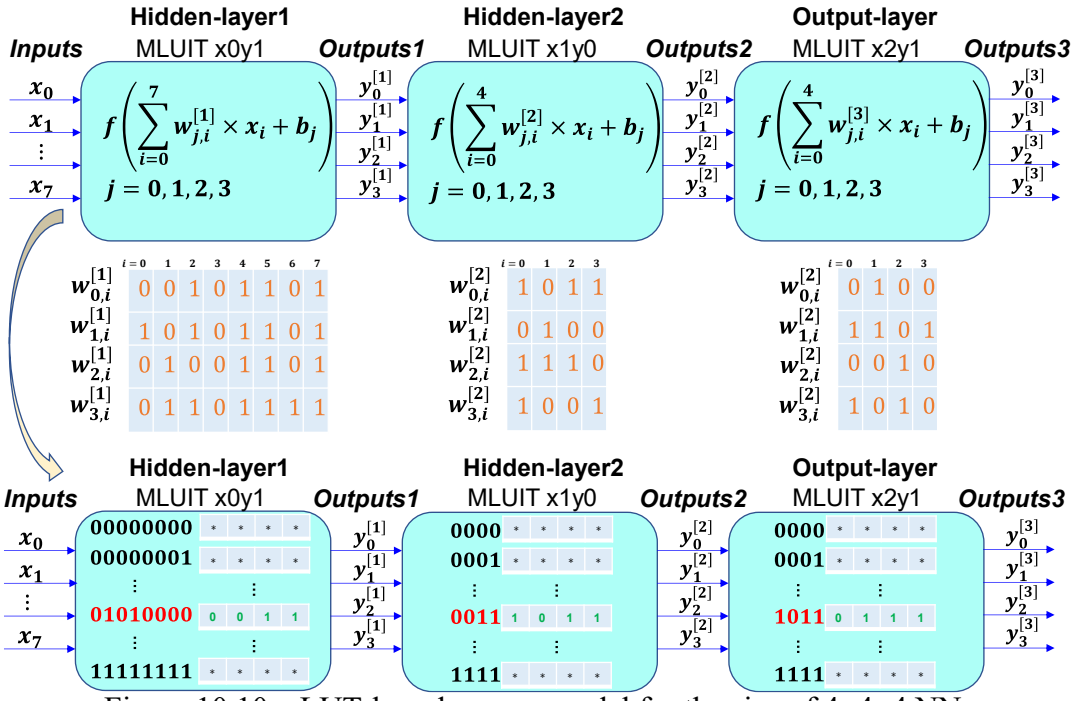


Figure 10.10 LUT-based neuron model for the size of 4×4×4 NN.

Figure 10.11 shows an performed logic simulation experiment in an MPLD with 16-bits 6×6 MLUTs array. the experimental result shows that the operating results of the LUT-based neuron model in MPLD are the same as the results of the above theoretical analysis.

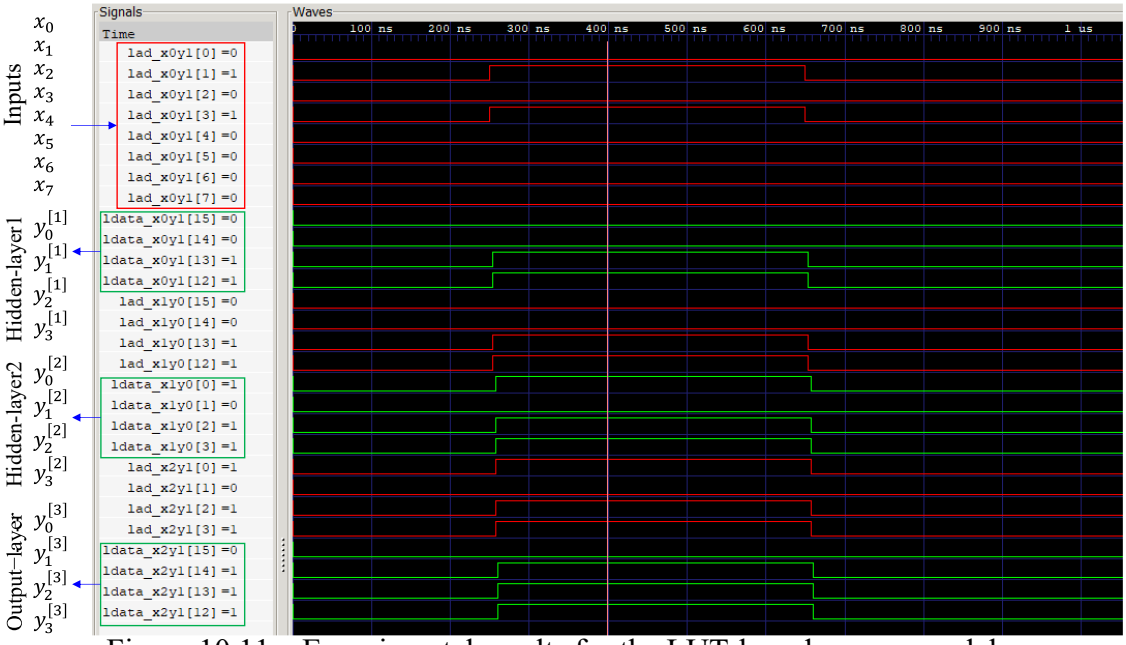


Figure 10.11 Experimental results for the LUT-based neuron model.

10.3.2 Confirm Proposed MNN

In this experiment, for comparing with FNN, we first designed the same size of FNN and the MNN. we used the MNIST dataset (60,000 handwritten number training images and 10,000 test images.) to make training the MNN and the FNN, respectively. Figure 10.12 shows the training results in 50 epochs, the results show that the MNN is an effective neural network that can get well accuracy and loss as same as the FNN. Figure 10.13 shows the MNN has been 150 epochs trained, and it can obtain the training accuracy and testing accuracy up to 0.99 and 0.96, respectively.

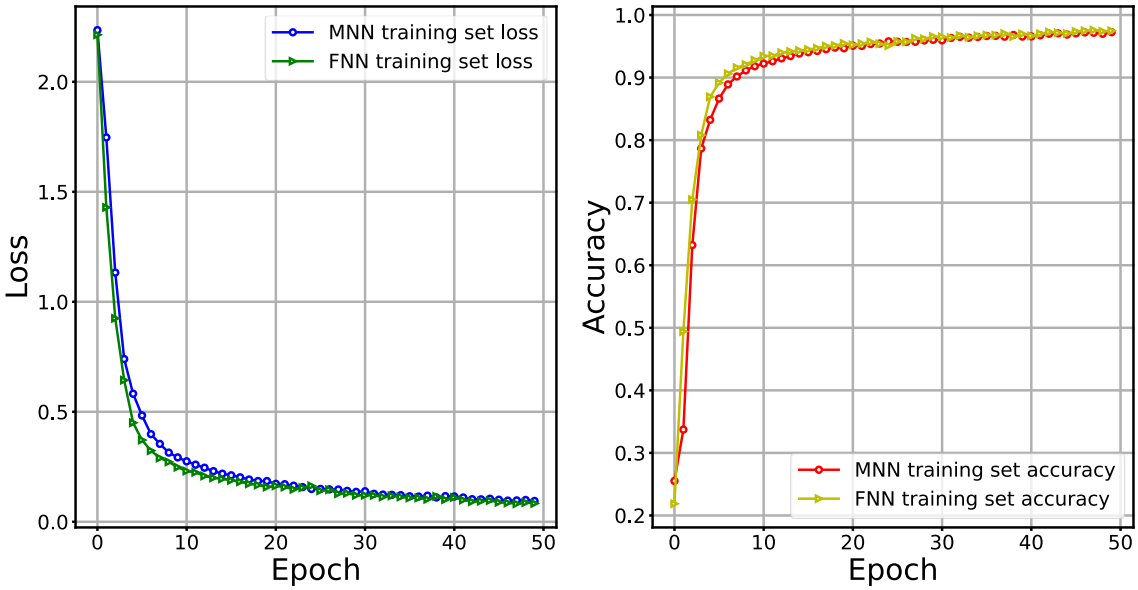


Figure 10.12 MNN and FNN training result in 50 epochs.

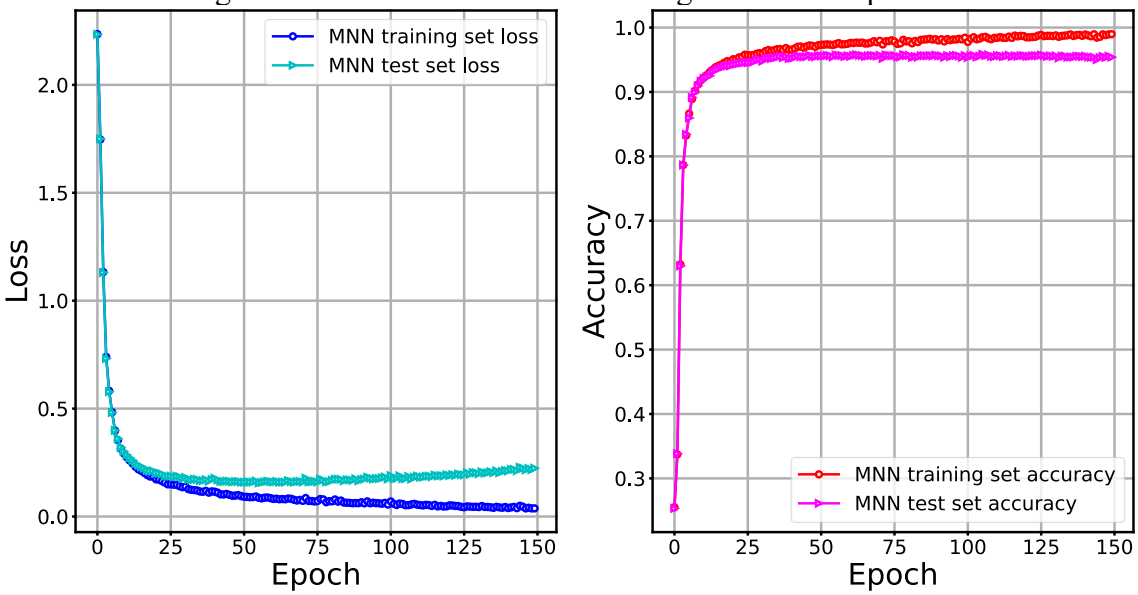


Figure 10.13 MNN training result in 150 epochs.

10.4 Conclusions

In this chapter, we suggest a LUT-based neuron model to implement a NNs into MPLD device. The NN neuron's operation can be calculated into truth table form pre-stored in MLUT of MPLD. In MPLD, due to the special interconnection structure of MLUTs, it is difficult to construct a NN with fully connection into the MPLD. Therefore, we proposed a novel network structure MNN (MPLD-based Neural Network) to adapt the MPLD structure. To confirm the LUT-based neuron model, we design a logic simulation experiment by implementing a 4×4×4 LUT-based neural network. We confirm that the simulation results are the same as the results of the theoretical analysis. To evaluate the effectiveness of the MNN, we also performed a recognition training experiment using the MNIST dataset. The experimental results show the MNN is an effective neural network which can get well accuracy and loss for MNIST data recognition.

In our future work, we will explore binarization methods for MNNs and analyze design approaches for MNNs that can recognize images and data of any size in MPLD.

Chapter 11

11. Summary

This study focuses on enhancing the reliability of IoT (internet of things) and AI (artificial intelligence) edge devices to advance the development of an ultra-smart society. Specifically, we concentrated on ECUs (electronic control units) employed in self-driving vehicle systems, which demand high functional safety, and on a new reconfigurable device, the MPLD (memory-based programmable logic device), currently under development for IoT and AI edge computing. We proposed testing methodologies designed to improve the reliability of these edge devices.

Initially, for automotive ECU edge devices, we proposed a technique for *test point insertion and selection for multi-cycle BIST (built-in self-test)*, designed to improve test quality and reduce test time.

Multi-cycle BIST has the potential to decrease the volume of scan-in patterns. This study meticulously examined the stuck-at-fault detection model in the time-expanded circuit. We found that the incongruity between controllability and observability of signal lines, exacerbated by increasing capture cycles, incites issues of fault masking and fault detection degradation. These issues hinder the effect of multi-cycle tests on test pattern reduction. To address this problem, we introduced a test point insertion (TPI) technique into a multi-cycle LBIST (logic BIST) scheme aimed at decreasing the volume of scan-in patterns for target fault coverage. The proposed TPI method involves replacing partial scan cells with fault detection scan flip-flops (FDS-FF), also known as observation point insertion (OPI), to enhance observability. It also incorporates self-flipping control logic into the combinational logic, termed control point insertion (CPI), to alleviate the controllability bias of signal lines of the circuit under test (CUT) at the intermediate capture cycles. We further propose a TPI procedure, which includes control point insertion and observation point pruning, to identify effective test points leading to maximum scan-in pattern reduction. Experimental results on ISCAS89 and ITC99 benchmarks demonstrate an average pattern reduction of 24.4X, thus validating the proposed TPI's effectiveness in reducing the test application time of power-on self-test (POST). Future work aims to implement the proposed test point selection algorithm in an industrial design to evaluate the effectiveness of the multi-cycle LBIST scheme on commercial automotive ECUs.

Subsequently, to ensure the reliability of MPLD devices, we proposed a high-quality *interconnect defect test method* to improve the reliability during the manufacturing process, and *an aging monitoring technique* for field reliability.

The proposed interconnect defect test method can identify stuck-at and bridge faults at the address-data (AD) interconnects between MLUTs (multiple look-up tables) within the MPLD device. This method also holds potential for actual field use as it can help avoid configuring the logic into a faulty MLUT block, thereby ensuring higher reliability. The test method consists of a configuration phase, which configures pre-generated internal test data to create the route map in the MLUTs array for fault propagation paths, and a logic phase, which applies pre-generated external test data to the MPLD's external logic output ports to excite target faults, observe faulty effects, and acquire the fault propagation path set for fault location. This test method addresses both fault detection and fault diagnosis in MPLDs. Our proposed test method has been validated through logic simulation experiments on the designed MPLD with a 6×6 MLUTs array. The results confirm its effectiveness in diagnosing the location of the injected stuck-at and bridge faults. Future work will explore the test generation of internal and external test data to identify other interconnect faults in the MPLD device, and consider methods such as design for testability and built-in self-tests for the MPLD device.

The proposed aging monitoring technique aims to detect and report the aging state of MPLD devices during field operation. The method involves periodically measuring the delay of MLUTs (multiple look-up tables) during the operation of the MPLD devices, using a specially designed delay monitor. This delay monitor is implemented using a ring oscillator circuit compatible with the MPLD device structure. Furthermore, we designed a new counter circuit, adapted to the MPLD structure, to store the ring oscillator's oscillation frequency for delay calculation. This method enables the measurement of both the global delay (across all MLUTs) and the local delay (of specified MLUTs) within the MPLD device. To evaluate the proposed methods, we designed an MPLD with a 6×6 MLUTs array and conducted logic simulations by injecting delay into the MPLD. The logic simulation results confirmed that the proposed method can effectively measure the delay of the MLUTs with minor error. In future work, we aim to conduct a quantitative analysis of aging phenomena and develop a precise simulation method along with an on-chip test method. Furthermore, we intend to explore strategies to determine the total number and locations of delay monitors needed to achieve a specific or higher level of in-field reliability.

References

[1] H. Fujiwara, Logic Testing and Design for Testability, The MIT Press, 1985.

[2] M. Michael Vai, VLSI Design, CRC press, 2000.

[3] M. L. Bushnell and V. D. Agrawal, Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits, Kluwer Academic Publishers, 2000.

[4] Laung-Terng Wang, Cheng-Wen Wu, Xiaoqing Wen, VLSI Test Principles and Architectures: Design for Testability (Systems on Silicon), Morgan Kaufmann Publishers Inc., San Francisco, CA, 2006.

[5] E.R. Hnatek. Integrated circuit quality and reliability, 2nd edition. Marcell Dekker Inc, 1995.

[6] A Charles E. Stroud, Designer's Guide to Built-in Self-Test, Kluwer Academic Publishers, 2002.

[7] A. Krstić and K.-T. Cheng, Delay Fault Testing for VLSI Circuits, Kluwer Academic Publishers, 1998.

[8] ISO26262-5:2011, "Road vehicles - Functional Safety - Part 5: Product development at the hardware level," Online Browsing Platform, https://www.iso.org/obp/ui/#iso:std:iso:26262:-5:ed-1:v1:en, accessed Jun. 18. 2023.

[9] P. Girard, N. Bicolici, and X. Wen, Power-Aware Testing and Test Strategies for Low Power Devices, Springer, ISBN 978-1-4419-0927-5, New York, 2010.

[10] K. Ichino, T. Asakawa, S. Fukumoto, K. Iwasaki, and S. Kajihara, "Hybrid BIST using partially rotational scan," in Proc. ATS, 2001, pp. 379-384.

[11] S. Narayanan, R. Gupta, M. A. Breuer: "Optimal Configuring of Multiple Scan Chains", IEEE Trans. on Comp., Sep. 1993, pp.1121-1131

[12] R. Kapur, S. Patil, T.J. Snethen, and T.W. Williams, "A weighted random pattern test generation system," IEEE Trans. CAD, vol. 15, no. 8, pp.1020-1025, Aug. 1996.

[13] A. Jas, C.V. Krishna, and N.A. Touba, "Weighted pseudorandom hybrid BIST," IEEE Trans. VLSI, vol. 12, no. 12, pp. 1277-1283, Dec. 2004.

[14] N.A. Touba and E.J. McCluskey, "Bit-fixing in pseudo-random sequences for scan BIST," IEEE Trans. CAD, vol. 20, no. 4, pp. 545-555, April 2001.

[15] H.-J. Wunderlich and G. Kiefer, "Bit-flipping BIST," in Proc. ICCAD, 1996, pp. 337-343.

[16] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, and B. Courtois, "Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers," IEEE Trans. Comput., vol. 44, no.2, pp. 223-233, Feb. 1995.

[17] C. V. Krishna, A. Jas, and N. A. Touba, "Test Vector Encoding Using Partial LFSR Reseeding," in Proc. Intl. Test Conf. (ITC), Baltimore, MD, USA, Nov. 2001, pp. 885–893.

[18] I. Pomeranz, "Computation of Seeds for LFSR-Based n-Detection Test Generation," ACM Trans. Des. Autom. Electron. Syst., vol. 22, no. 2, pp. 29:1–29:13, Jan. 2017.

[19] J. P. Hayes and A. D. Friedman, "Test Point Placement to Simplify Fault Detection," IEEE Trans.

Comput., vol. C–23, no. 7, pp. 727–735, Jul.1974.

[20] A. J. Briers and K. A. E. Totton, “Random Pattern Testability by Fast Fault Simulation,” in Proc. Intl. Test Conf. (ITC), Washington, DC, USA, Sep. 1986, pp. 274–281.

[21] H. Vranken, F. S. Sapei, and H.-J. Wunderlich, “Impact of Test Point Insertion on Silicon Area and Timing During Layout,” in Proc. Design, Automation, and Test in Europe Conf. (DATE), Paris, France, Feb. 2004.

[22] O. Novak and J. Nosek, “Test-per-clock testing of the circuits with scan,” Proc. Int. On-Line Test Workshop, 2001, pp. 90-92.

[23] S. Milewski, N. Mukherjee, J. Rajski, J. Solecki, J. Tyszer, and J. Zawada, “Full-scan LBIST with capture-per-cycle hybrid test points”, Proc. ITC, 2017, paper 10.3.

[24] F. Zhang, D. Hwong, Y. Sun, A. Garcia, S. Alhelaly, G. Shofner, L. Winemberg, and J. Dworak, "Putting wasted clock cycles to use: Enhancing fortuitous cell-aware fault detection with scan shift capture,” Proc. ITC, 2016, paper 2.3.

[25] G. Mrugalski, J. Rajski, J. Solecki, J. Tyszer, and C. Wang, “Trimodal scan-based test paradigm,” IEEE Trans. VLSI Systems, vol. 25, no. 3, pp. 1112-1125, March 2017.

[26] Y. Liu, N. Mukherjee, J. Rajski, S. M. Reddy and J. Tyszer, "Deterministic Stellar BIST for Automotive ICs," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 8, pp. 1699-1710, Aug. 2020, doi: 10.1109/TCAD.2019.2925353.

[27] I. Pomeranz and S. M. Reddy, "Static test compaction for scan-based designs to reduce test application time", Proc. 7th Asian Test Symp. ATS, pp. 198-203, 1998.

[28] X. Lin and R. Thompson, "Test generation for designs with multiple clocks", Proc. Design Autom. Conf., pp. 662-667, Jun. 2003.

[29] Kajihara, M. Matsuzono, H. Yamaguchi, Y. Sato, K. Miyase, and X. Wen, “On Test Pattern Compaction with Multi-Cycle and Multi-Observation Scan Test,” Proc. Int’l. Symposium on Com. and Inf. Tech. (ISCIT), Tokyo, pp. 723-726, Oct. 2010. DOI: 10.1109/ISCIT.2010.5665084

[30] Y. Huang, I. Pomeranz, S. M. Reddy and J. Rajski, “Improving the proportion of At-Speed Tests in Scan BIST,” Int’l. Conf. on Computer Aided Design, San Jose, pp. 459-463, Nov. 2000. DOI: 10.1109/ICCAD.2000.896514

[31] E. K. Moghaddam, J. Rajski, S. M. Reddy and M. Kassab, “At-Speed Scan Test with Low Switching Activity,” Proc. IEEE 28th VLSI Test Symposium, Santa Cruz, pp.177-182, April 2010. DOI: 10.1109/VTS.2010.5469580

[32] Y. Sato, S. Wang, T. Kato, K. Miyase and S. Kajihara, "Low Power BIST for Scan-Shift and Capture Power," Proc. IEEE Asian Test Symposium, Niigata, pp. 173-178, 2012. DOI: 10.1109/ATS.2012.27

[33] I. Pomeranz, "Multicycle Tests with Fault Detection Test Data for Improved Logic Diagnosis," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, doi: 10.1109/TCAD.2021.3079146.

[34] N. Devtaprasanna, A. Gunda, P. Krishnamurthy, S. M. Reddy, and I. Pomeranz, "Methods for

improving transition delay fault coverage using broadside tests," in Proceedings of IEEE International Conference on Test 2005, pp.256-265. doi: 10.1109/TEST.2005.1583983.

[35] I. Pomeranz, "Design-for-testability for multi-cycle broadside tests by holding of state variables," in ACM Transactions on Design Automation of Electronic Systems, vol. 19, issue 2, article 19, pp 1-20.

[36] I. Pomeranz, “Enhanced Test Compaction for Multi-Cycle Broadside Tests By Using State Complementation”, in ACM Transactions on Design Automation, November 2015, vol. 21, no. 1, article 13.

[37] S. Wang et al., "Structure-Based Methods for Selecting Fault-Detection-Strengthened FF under Multi-cycle Test with Sequential Observation," Proc. IEEE Asian Test Symposium, Hiroshima, pp. 209-214, Nov. 2016. DOI: 10.1109/ATS.2016.40

[38] S. Wang, Y. Higami, H. Iwata, J. Matsushima and H. Takahashi, "Automotive Functional Safety Assurance by POST with Sequential Observation," IEEE Design & Test Magazine. Vol.35, no.3, pp.39-45, June 2018. DOI: 10.1109/MDAT.2018.2799801

[39] S. Wang, Y. Higami, H. Takahashi, H. Iwata, Y. Maeda and J. Matsushima, "Fault-detection-strengthened method to enable the POST for very-large automotive MCU in compliance with ISO26262," Proc. IEEE 23rd European Test Symposium, Bremen, pp. 1-2, 2018. DOI: 10.1109/ETS.2018.8400707

[40] S. Wang, et al., "Capture-Pattern-Control to Address the Fault Detection Degradation Problem of Multi-Cycle Test in Logic BIST," Proc. IEEE Asian Test Symposium, Hefei, pp.155-160, 2018. DOI:10.1109/ats.2018.00038

[41] H.T. Al-Awadihi, T. Aono, S. Wang, Y. Higami, H. Takahashi, H. Iwata, Y. Maeda, J. Matsushima, “FF-Control Point Insertion (FF-CPI) to Overcome the Degradation of Fault Detection under Multi-Cycle Test for POST,” IEICE Transactions on Information and Systems, 2020, Vol. E103.D, No. 11, pp. 2289-2301, DOI:10.1587/transinf.2019EDP7235.

[42] P. K. Datla Jagannadha et al., “Special session: In-System-Test (IST) Architecture for NVIDIA Drive-AGX Platforms,” In IEEE 37th VLSI Test Symposium (VTS), Monterey, CA, 1–8. 2019. https://doi.org/10.1109/VTS.2019.8758636

[43]] G. Mrugalski, J. Rajski, J. Tyszer, and B. Włodarczak, “X-Masking for In-System deterministic test,” In IEEE European Test Symposium (ETS), Barcelona, 1–6. 2022. https://doi.org/10.1109/ETS54262.2022.9810407

[44] A. Rupani, D. Pandey and G. Sujediya, “Review and Study of FPGA Implementation of Internet of Things,” Int. J. of Sci. Technol. & Eng., Vol. 3, Issue. 02, August 2016.

[45] Nithin M.R, Raisa Basheer and Sreela Mohan “Advanced Driver Assistance System using FPGA,” QuEST Global Corp., May 2017.

[46] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie and X. Zhou, “DLAU: A Scalable Deep Learning Accelerator Unit on FPGA,” IEEE Trans. on Computer-Aided Design of Integrated Circuits and Syst., vol. 36, no. 3, pp. 513-517, March 2017.

[47] I. Kuon and J. Rose, “Measuring the Gap Between FPGAs and ASICs,” in EEE Transactions on

Computer-Aided Design of Integrated Circuits and Systems, vol. 26, no. 2, pp. 203-215, Feb. 2007.

[48] T. Tuan and B. Lai, "Leakage power analysis of a 90nm FPGA," Proceedings of the IEEE 2003 Custom Integrated Circuits Conference, 2003., 2003, pp. 57-60.

[49] TAIYO YUDEN CO LTD, “Reconfigurable semiconductor device”, Japan Patent JP2016208426A, Dec. 08, 2016.

[50] H. Puchner, L. Hinh, “NBTI reliability analysis for a 90nm CMOS technology,” in 30th Eur. Solid-State Circuits Conf., Sept. 2004, pp.257-260.

[51] F. Chen, M. Shinosky, “Addressing Cu/Low-k Dielectric TDDBReliability Challenges for Advanced CMOS Technologies,” IEEE Trans. on Electron Devices, vol.56, no.1, pp.2-12, Jan. 2009.

[52] D. Rossi, “The Effects of Ageing on the Reliability and Performance of Integrated Circuits,” in Ageing of Integrated Circuits: Causes, Effects and Mitigation Techniques, B. Halak, Ed., Springer International Publishing, 2020, pp. 35-64.

[53] S. Sapatnekar, “What happens when circuits grow old: Aging issues in CMOS design,” in 2013 Int. Symp. on VLSI Technol., Syst. and Appl. (VLSI-TSA), 2013, pp. 1-2.

[54] M.S. Mispan (et al.), “Ageing Mitigation Techniques for SRAM Memories,” in Ageing of Integrated Circuits: Causes, Effects and Mitigation Techniques, B. Halak, Ed., Springer International Publishing, 2020, pp. 91-111.

[55] T. Q. Bui, L. D. Pham, H. M. Nguyen, V. T. Nguyen, T. C. Le, and T. Hoang, “An Effective Architecture of Memory Built-In Self-Test for Wide Range of SRAM,” in 2016 Int. Conf. Adv. Comput. Appl., pp. 121–124, 2016, doi: 10.1109/ACOMP.2016.026.

[56] A. Sharma and V. Ravi, “Built in self-test scheme for SRAM memories,” in 2016 Int. Conf. Adv. Comput. Commun. Informatics, pp. 1266–1270, 2016, doi: 10.1109/ICACCI.2016.7732220

[57] S. Wang, Y. Higami, H. Takahashi, M. Sato, M. Katsu, and S. Sekiguchi, “Testing of Interconnect Defects in Memory Based Reconfigurable Logic Device (MRLD),” in 2017 IEEE 26th Asian Test Symp., pp. 17–22, 2017.

[58] S. Wang et al., “Test Method for the Bridge Interconnect Faults in Memory Based Reconfigurable-Logic-Device (MRLD) Considering the Place-and-Route,” in 33th Int. Tech. Conf. Circuits/Syst., Comput. Commun., 2018.

[59] W. K. Huang, X. T. Chen, and F. Lombardi, “On the diagnosis of programmable interconnect systems: Theory and application,” in Proc. 14th VLSI Test Symp., pp. 204–209, 1996, doi: 10.1109/VTEST.1996.510859

[60] D. Das and N. A. Touba, “A low cost approach for detecting, locating, and avoiding interconnect faults in FPGA-based reconfigurable systems,” in Proc. Twelfth Int. Conf. VLSI Des. (Cat. No.PR00013), pp. 266–269, 1999, doi: 10.1109/ICVD.1999.745159.

[61] T. Inoue, S. Miyazaki, and H. Fujiwara, “Universal fault diagnosis for lookup table FPGAs,” IEEE Des. Test Comput., vol. 15, no. 1, pp. 39– 44, 1998, doi: 10.1109/54.655181.

[62] Y. Sato, Seiji Kajihara, Y. Miura, T. Yoneda, S. Ohtake, M. Inoue, H. Fujiwara, “A Circuit Failure

Prediction Mechanism (DART) for High Field Reliability," in 8th IEEE Int. Conf. on ASIC, Oct. 2009, pp. 581-584.

[63] M.S. Mispan (et al.), "Ageing Mitigation Techniques for SRAM Memories," in Ageing of Integrated Circuits: Causes, Effects and Mitigation Techniques, B. Halak, Ed., Springer International Publishing, 2020, pp. 91-111.

[64] Y. Tsugita, K.Ueno, T. Hirose, T. Asai, Y. Amemiya, "An on-chip PVT compensation technique with current monitoring circuit for low-voltage CMOS digital LSIs," IEICE Trans. on Electronics, vol. 93 , no. 6, pp. 835-841, 2010.

[65] M. Bhushan, A. Gattiker, M. B. Ketchen and K. K. Das, "Ring oscillators for CMOS process tuning and variability control," IEEE Trans. on Semicond. Manuf., vol. 19, no. 1, pp. 10-18, Feb. 2006.

[66] Poki Chen, Chun-Chi Chen, Chin-Chung Tsai and Wen-Fu Lu, "A time-to-digital-converter-based CMOS smart temperature sensor," IEEE J. of Solid-State Circuits, vol. 40, no. 8, pp. 1642-1648, Aug. 2005.

[67] S. Kajihara, Y. Miyake, Y. Sato and Y. Miura, "An On-Chip Digital Environment Monitor for Field Test," in 2014 IEEE 23rd Asian Test Symp., Nov. 2014, pp. 254-257.

[68] Y. Miyake, Y. Sato, S. Kajihara and Y. Miura, "Temperature and Voltage Measurement for Field Test Using an Aging-Tolerant Monitor," IEEE Trans. on Very Large Scale Integration (VLSI) Systems, vol. 24, no. 11, pp. 3282-3295, Nov. 2016.

[69] Y. Miyake, Y. Sato and S. Kajihara, "On-Chip Delay Measurement for In-Field Test of FPGAs," in 2019 IEEE 24th Pacific Rim Int. Symp. on Dependable Computing (PRDC), Kyoto, Japan, Dec. 2019, pp. 130-1307.

[70] K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," in Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), pp. 770-778, 2016.

[71] T. Ochiai, S. Watanabe, S. Katagiri, T. Hori, J. Hershey, "Speaker Adaptation for Multichannel End-to-End Speech Recognition," in Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), pp. 6707-6711, 2018.

[72] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," Int. J. Robot. Res., vol. 37, no. 4-5, pp. 421-436, Apr. 2018.

[73] A. Wheeldon, R. Shafik, T. Rahman, J. Lei, A. Yakovlev, and O.-C. Granmo, "Learning automata based energy-efficient AI hardware design for IoT applications," Philos. Trans. R. Soc. A Math. Phys. Eng. Sci., vol. 378, no. 2182, p. 20190593, Oct. 2020.

[74] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," Neurocomputing, vol. 74, no. 1, pp. 239-255, 2010.

[75] Z. Fan, F. Qiu, A. Kaufman, S. Yoakum-Stove, "GPU Cluster for High Performance Computing," in Proc. ACM/IEEE Conf. on Supercomputing, Nov. 2004.

[76] E. Mizell, R. Biery, Introduction to GPUs for Data Analytics Advances and Applications for Accelerated Computing, O'Reilly, 2017.

[77] N. Singh, S. P. Panda, “Enhancing the Proficiency of Artificial Neural Network on Prediction with GPU,” in Int. Conf. on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), Oct. 2019.

[78] Y. Kim, H. Choi, J. Lee, J. Kim, H. Jei, H. Roh, “Efficient Large-Scale Deep Learning Framework for Heterogeneous Multi-GPU Cluster,” in 2019 IEEE 4th Int. Workshops on Foundations and Applications of Self* Systems (FAS*W), Jun. 2019.

[79] NVIDIA, “NVIDIA A100 Tensor core GPU,” NVIDIA, 2020.

[80] Y Hui, J Lien, X Lu, “Three-Dimensional Characterization on Edge AI Processors with Object Detection Workloads,” in Int. Conf. for High Performance Computing, Networking, Storage, and Analysis, Nov. 2019.

[81] K. Guo, S. Zeng, J. Yu, Y. Wang, H. Yang, “A Survey of FPGA-based Neural Network Inference Accelerators,” J ACM Trans. Reconfigurable Technol. Syst., Vol. 12, No. 1, Article No.: 1, pp. 1-26, Apr. 2019.

[82] F. Piazza, A. Uncini and M. Zenobi, “Neural networks with digital LUT activation functions,” Proc. Int. Jt. Conf. Neural Networks (IJCNN), vol. 2, pp. 1401-1404, 1993.

[83] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe, “Binary neural networks: A survey. Pattern Recognition”, in Pattern Recognition, 2020.

List of Publication

Journal Publications

1. Senling Wang, Xihong Zhou, Yoshinobu Higami, Hiroshi Takahashi, Hiroyuki Iwata, Yoichi Maeda, Jun Matsushima, "Test Point Insertion for Multi-Cycle Power-On Self-Test," *ACM Transactions on Design Automation of Electronic Systems (ACM TODES)*, Vol. 28, No. 3, pp. 1-21, May 2023.

International Conferences Publications

1. Xihong Zhou, Senling Wang, Yoshinobu Higami, Hiroshi Takahashi, Mitsunori Katsu, Shoichi Sekiguchi, "MNN: A Solution to Implement Neural Networks into a Memory-based Reconfigurable Logic Device (MRLD)," *in 36th International Technical Conference on Circuits, Systems, Computers, and Communications (ITC-CSCC)*, Jun. 2021.

2. Xihong Zhou, Senling Wang, Yoshinobu Higami, Hiroshi Takahashi, "Aging Monitoring for Memory-based Reconfigurable Logic Device (MRLD)," *in 35th International Technical Conference on Circuits, Systems, Computers, and Communications (ITC-CSCC)*, Jul. 2020.

International Conferences Presentations

1. Xihong Zhou, "Study on the High Reliability of MPLD (Memory-based Programmable Logic Device)," *in Asian Test Symposium, Ph.D. Thesis Competition, Semi-Final of 2023 TTTC's E. J. McCluskey Doctoral Thesis Award (ATS Doctoral Thesis Award)*, Nov. 2022.

2. Xihong Zhou, Senling Wang, Yoshinobu Higami, Hiroshi Takahashi, "Diagnosis for Interconnect Faults in Memory-based Reconfigurable Logic Device," *in 22nd IEEE Workshop on RTL and High Level Testing (WRTLT)*, Nov. 2021.

National Conferences

1. Xihong ZHOU, Senling WANG, Yoshinobu HIGAMI, Hiroshi TAKAHASHI, Masayuki SATO, Mitsunori KATSU, Shoichi SEKIGUCHI, "Implementing Neural Networks on Memory-based Reconfigurable Logic Device (MRLD)," *in 30th Microelectronics Symposium (MES)*, Sep. 2020.

2. 周 細紅, 王 森レイ, 樋上 喜信, 高橋 寛, "メモリベース論理再構成デバイス(MRLD)における劣化状態検知のためのリングオシレータ実装," *第 34 回エレクトロニクス実装学会春季講演大会講演集 (JIEP)*, Mar. 2020.

3. 青野 智己, 中岡 典弘, 周 細紅, 王 森レイ, 樋上 喜信, 高橋 寛, 岩田 浩幸, 前田 洋一, 松嶋 潤, "マルチサイクルテストにおける故障検出強化のためのテストポイント挿入法," *電子情報通信学会技術研究報告 (IEICE-DC)*, vol. 119, no. 420, pp. 19-24, Feb. 2020.

4. 阿部 寛人, 畝山 勇一朗, 中岡 典弘, 渡辺 友希, 福本 真也, 森田 航平, 中本 裕大, 周 細紅, 河野 靖, 木下 浩二, 一色 正晴, 二宮 崇, 田村 晃裕, 甲斐 博, 高橋 寛, 王 森レイ, "Raspberry Pi を用いた画像処理と CNN による微小害虫の計数システムの構築," *令和元年度電気関係学会四国支部連合大会論文集 (CD-ROM) (SJCIEE)*, 2019.

5. 周 細紅, 王 森レイ, 高橋 寛, "サウンドコード技術を利用した電気錠システムの開発," *電気関係学会四国支部連合大会 (SJCIEE)*, 2018.