Implementation of Boundary-Scan Architecture and its Application to Module Interconnect Testing

A thesis submitted to
The Hong Kong University of Science and Technology
in partial fulfillment of the requirements for
the Degree of Master of Philosophy in
Electrical and Electronic Engineering

by

Lo Chun Keung

B.Eng.
Department of Electronic Engineering
The Chinese University of Hong Kong

July 1998
Implementation of Boundary-Scan Architecture and its Application to Module Interconnect Testing

by

Lo Chun Keung

Approved by:

Prof. Philip C. H. Chan
Thesis Supervisor

Dr. Ho-Chi Huang
Thesis Examination Committee Member (Chairman)

Dr. Jack Lau
Thesis Examination Committee Member

Prof. Philip C. H. Chan
Head of Department

Department of Electrical and Electronic Engineering
The Hong Kong University of Science and Technology

July 1998
Implementation of Boundary-Scan Architecture and its Application to Module Interconnect Testing

by

Lo Chun Keung

in partial fulfillment of the requirements for the degree of
Master of Philosophy in Electrical and Electronic Engineering
at The Hong Kong University of Science and Technology
July 1998

Abstract

With the advances in packaging technologies and increasing demand for high-speed and small size electronic circuits, Multichip Module (MCM) technology has been developed for high-speed and high-performance applications. However, due to the very limited accessibility to the module circuitry, testing for such a high density module is a complex and time consuming process. This thesis addresses the issues of assembled module interconnect testing in the context of boundary-scan architecture. The emphasis will be on the structural interconnect integrity testing as well as mechanical assembly defects detection.

In this thesis, the development of a low-cost test and diagnosis platform based on the IEEE Std 1149.1 boundary-scan testing methodology will be presented. Boundary-Scan is a structured design-for-testability (DFT) technique which can be used to simplify the testing of digital circuits, boards, and systems. A test chip containing the boundary-scan architecture has been fabricated using a 1.2μm n-well CMOS process. To demonstrate the prototype capabilities of the test system, a test evaluation module
containing the boundary-scan test chips has been implemented. In view of
interconnect faults detection and diagnosis, an efficient structural diagnostic approach
has been proposed. While existing diagnosis schemes assume only simple bridging
fault model, a more complex bridging short fault model in CMOS circuit environment
is considered. Simulation results show that our approach performs very well when the
short fault rate is very small. Limitations and further enhancements of the test system
will also be discussed. We have also investigated the application of boundary-scan
testing as part of the manufacturability assessment of the Flip Chip On Board (FCOB)
assembly process. Assembly defects such as solder bumps opens and shorts could be
detected and located.
Acknowledgments

First, I would like to thank my research advisor Prof. Philip Chan for his support and guidance of this thesis. I have benefited greatly from his valuable experiences and specialized knowledge. I would also like to thank Dr. Ho-Chi Huang and Dr. Jack Lau for being my thesis examination committee.

I would like to thank Siu-Fai Luk and Jack Chan in VLSI research laboratory for their technical assistance. I am also grateful to my friends and my colleagues, both in HKUST and CUHK, for their advice and encouragement.

Lastly, and most importantly, I would like to express my gratitude to my parents for their patience and encouragement.

LO CHUN KEUNG
# Contents

Abstract ii

Acknowledgments iv

List of Figures viii

List of Tables x

1 Introduction 1
  1.1 Board-Level Test Technology 2
  1.2 An Overview of Boundary-Scan 3
  1.3 Boundary-Scan Test Platform 5
  1.4 Thesis Outline 8

2 IEEE Std 1149.1 Architecture 9
  2.1 Background 10
  2.2 Basic Operation of IEEE Std 1149.1 11
  2.3 Test Logic Architecture 12
     2.3.1 The Test Access Port (TAP) 13
     2.3.2 The TAP Controller 15
     2.3.3 The Instruction Register 16
     2.3.4 Test Data Registers 18
     2.3.5 Instructions 20

3 Chip Design Implementing Boundary-Scan 22
  3.1 Cell Library 23
     3.1.1 Standard Cell Layout Format 23
     3.1.2 Pad Cell Layout Format 25
5.1.3 IC Internal Test ........................................... 76
5.2 Test and Diagnosis System Overview ...................... 78
5.3 Test Pattern Generation .................................... 81
  5.3.1 CAD Database ...................................... 81
  5.3.2 Generation of Test Stimulus Patterns ................ 82
  5.3.3 Logic Simulation .................................... 84
5.4 Test Execution ........................................... 84
5.5 Fault Diagnosis .......................................... 86
5.6 Module Implementation and Test Application .............. 87
  5.6.1 MUT Description ................................... 87
  5.6.2 Tester Setup ........................................ 89
  5.6.3 Results ............................................. 91
5.7 Limitations of BTDS .................................... 91
5.8 Summary ................................................ 94

6 Application to Multichip Module Testing .................... 95
  6.1 An Introduction to MCM Assembly and Testing Process ... 96
  6.2 Future Work on Flip Chip On Board Assembly Testing . .. 100
    6.2.1 Potential Solder Joint Defects .................... 100
    6.2.2 Test Vehicle Design ............................... 102

7 Conclusions ............................................... 104

A Cell Library Listing ....................................... 106

B Macrocell Synthesized Netlists ................................ 107

C SKILL Program of One-step Diagnosis Algorithm ............. 112

Bibliography .................................................. 118
# List of Figures

1.1 Inclusion of Boundary-Scan Cells in an IC .......................... 3
1.2 A Boundary-Scannable Board Design ................................. 4
1.3 MCM Production Flow ............................................. 5

2.1 A Conceptual Schematic of the Test Logic ......................... 12
2.2 Tap Controller State Diagram ..................................... 15
2.3 Tap Controller Output Signals .................................... 16
2.4 An Instruction Register Cell ...................................... 17
2.5 A Bypass Register Implementation ................................. 18
2.6 A Boundary-Scan Register Cell .................................. 19
2.7 Structure of the Device Identification Register .................. 20

3.1 Basic Standard Cell Layout Structure ............................ 24
3.2 Basic Pad Cell Layout Structure .................................. 25
3.3 Input Protection Circuit ........................................... 26
3.4 Implemented Boundary-Scan Architecture ....................... 28
3.5 2-bit Instruction Register ........................................ 31
3.6 An Implementation of a Boundary-Scan Register Cell ........... 33
3.7 Adding Delays to Prevent Simultaneous Switching of Outputs . 35
3.8 IC Design Flow .................................................... 37
3.9 Floorplan for the Chip Core ........................................ 40
3.10 Power Bus Distribution ........................................... 41
3.11 Test Chip Photomicrograph ....................................... 43
# List of Tables

2.1 Public Instructions ........................................ 21

3.1 Implemented Instructions and their Binary Codes ............. 30
3.2 Mode Signal Generation for the BSC. .......................... 33
3.3 Macrocell Transistor Count. ................................. 39

4.1 Counting Sequence Test Set .................................. 50
4.2 Modified Counting Sequence Test Set ........................ 50
4.3 Counting Sequence Plus Complement Test Set .................. 51
4.4 Walking-1 Test Set ........................................... 51
4.5 Walking-0 Test Set ........................................... 51
4.6 Test Results for Shorted Nets. ............................... 56
4.7 Board Examples. .............................................. 65
4.8 Simulation Results for Benchmark Layouts ....................... 65
4.9 Simulation Results for Random Interconnects .................. 66
4.10 Comparison of Diagnosis Approaches .......................... 69

5.1 Example Tests for Interconnect Faults ........................ 75
5.2 Example Tests for the NOR Gate. ............................. 77
Chapter 1

Introduction

With the emergence of surface mount technology and advances in packaging technologies, the packaging density of a printed circuit board (PCB) and multichip module (MCM) has been increasing. Today's need for denser packaging technologies is dictated by systems requiring smaller physical sizes and/or higher system performances. In particular, the MCM technology meets the current miniaturization and performance demands. Advanced consumer products, such as personal communicators, need MCMs primarily to achieve smaller size systems, whereas telecommunication equipment and high-speed computers are considered performance-driven applications and require MCMs to achieve operations with very high speeds. However, testing for such a high density board or module is a complex and time-consuming process. In this thesis, we will study some of the design-for-testability (DFT) issues in board or module level testing. The emphasis will be on the structural interconnect integrity testing based on the boundary-scan testing approach.
1.1 Board-Level Test Technology

In the earlier generation of board testers, the board was tested by applying test signals at the board edge connectors and by monitoring the output signals, again at the edge connectors. This board-level functional tests is simple. However, test generation, fault simulation and test application costs are excessive to achieve acceptable fault coverage.

Another technique in board testing is in-circuit testing. In order to achieve the required controllability and observability, the devices on a board are accessed by a bed-of-nail test fixture such that the probes can directly make contact with the device I/O pins from the pads on the surface of the circuit board. Although the in-circuit testing technique can be used to test complex boards, it fails with boards containing surface mounted integrated circuits or ICs with very small pitch of I/O pins. The same problem also applies to MCM assembly testing. It is difficult to access internal nodes in a module due to the chip density and small interconnection line dimensions.

As a result of the limited observability of the chip leads, these testing techniques using traditional automatic test equipment (ATE) become impractical. The introduction of IEEE Std 1149.1 Test Access Port and Boundary-Scan Architecture [1] provides access to internal module networks required for interconnect testing in cases where physical access is limited or not possible. In addition, this standard ensures that chips contain common circuitry that will make test software development and the testing of boards containing these chips significantly more effective and less costly. Today, various commercial chips including those high-performance microprocessors have already incorporated boundary-scan into their designs.
1.2 An Overview of Boundary-Scan

Boundary-scan is a design-for-test (DFT) philosophy. It is aimed at simplifying testing by modifying a design to improve the controllability and observability of internal signals and interconnections of devices on a PCB or MCM. The boundary-scan technique involves the inclusion of a shift-register stage contained in a boundary-scan cell (BSC) adjacent to each component pin so that signals at component boundaries can be controlled and observed using the scan testing principles. The BSCs for the pins of a component are interconnected so as to form a shift-register chain around the border of the circuit, and this path is provided with a serial input and output connections and appropriate clock and control signals. Figure 1.1 shows the inclusion of BSCs in an integrated circuit (IC). During normal IC operation, input and output signals pass freely through each BSC, from the system input pins, to the system output pins.

Figure 1.1 Inclusion of Boundary-Scan Cells in an IC
Within a board or module assembled from several integrated circuits, the boundary-scan registers for the individual components could be connected in series to form a single path through the complete design, as illustrated in Figure 1.2. As a consequence, if all the components used to construct a circuit have a boundary-scan register, then the resulting serial path through the complete design can be used to perform two major test functions.

First, the interconnections between the various components can be tested. Test data can be shifted into all the BSCs associated with component output pins and loaded in parallel through the component interconnections into those cells associated with input pins. Second, the boundary-scan register can be used as a means of isolating on-chip system logic from stimuli received from surrounding components. Thus, an internal self-test can be performed. It also permits a limited slow-speed static test of the on-chip system logic since it allows delivery of test data to the component and examination of the test results.
1.3 Boundary-Scan Test Platform

The objective of this thesis is to apply boundary-scan testing methodology to the module interconnects in advanced packaging and assembled processes. Module assembly testing is one of the test procedures in a typical MCM production flow as shown in Figure 1.3. Although IEEE Std 1149.1 provides other testability features like built-in-self-test (BIST) [1], [2], we will concentrate on the detection and diagnosis of structural interconnect faults, such as opens and shorts. The interconnect structures under consideration can be tap automated bonding, wire bonding, or solder bumped flip chip.

![Diagram of MCM Production Flow](image)

**Figure 1.3 MCM Production Flow**
Much has already been written concerning the application of boundary-scan testing to the detection of MCM manufacturing defects [3]-[10]. The motivation herein is to use the same test methodology and apply to the assembly process. The process under investigation is the flip-chip-on-board (FCOB) technology with solder bumps as the electrical and structural interconnects. FCOB is one of the direct-chip-attach technologies and is currently under active research in HKUST. In particular, the structural interconnect integrity can be checked by incorporating boundary-scan architecture into a test vehicle designed for this assembly process evaluation. However, before implementing this test strategy, the corresponding boundary-scan test tools must be developed.

In this work, we have successfully developed a boundary-scan test platform which is sufficient to perform the assembly defect detection and diagnosis operations under consideration. This platform includes three major parts. They are: (1) the design and implementation of a test chip, (2) the introduction of a new interconnect diagnosis approach, and (3) the test execution with a PC-based tester and the associated software development.

To evaluate the boundary-scan testing both at IC and module level, a chip containing the boundary-scan architecture has been designed and fabricated using a 1.2μm n-well, two metal layers CMOS process. The cell library [11] implemented is designed with the MOSIS SCMOS design rules [12]. We choose 1.2μm technology because it is compatible with the Microelectronics Fabrication Facility (MFF) base-line process. Basically, the boundary-scan test chip design contains all the basic components which conform to the minimum requirements of IEEE Std 1149.1 and the resulting test chip can be used as a chip sized test vehicle for assembly process evaluation.
Introduction

The miniaturization of digital circuits has made possible the manufacturing of high density layouts and boards. The large number of interconnects and their very high density have caused an increase in the occurrence of faults. Diagnosis consists of fault detection and location. As open and stuck-at faults are relatively simple to diagnose, the analysis is usually pursued with respect to bridging faults as the most likely faults to occur in an interconnect. Moreover, although the introduction of boundary-scan reduces the need for in-circuit testing, the time taken to load a test vector serially places a premium on minimizing the number of test vectors. We will examine these problems in the context of net bridging faults and describe an efficient structural diagnosis approach based on an enhanced fault model. Both one-step and two-step diagnosis algorithms are presented. They guarantee the complete diagnosis of multiple interconnect faults with no aliasing and confounding, while the test length is significantly reduced.

Boundary-scan based tests, while providing many internal “virtual” test points, require data to be formatted in serial form in order to shift in stimulus and shift out response data. This requires specific tester in order to apply the tests and analyze the responses. We have developed our own boundary-scan test and diagnosis system for generating the stimulus and performing fault diagnosis based on the measured circuit responses. We use an existing PC-based tester for test execution and a test module has been implemented to demonstrate the prototype capabilities of the test system. It should be noted that unlike other commercially available boundary-scan testers, there are several limitations for our test system. However, it is sufficient for our specific applications.
1.4 Thesis Outline

This thesis contains 7 chapters. Chapter 1 is the introduction. Chapter 2 gives an overview of the IEEE Std 1149.1 boundary-scan architecture. The design and implementation of the boundary-scan test chip is presented in Chapter 3. Also described are some of the design considerations as well as the IC design methodology. Chapter 4 describes a new structural diagnostic approach for board interconnect testing with boundary-scan. Algorithms are explained in detail and simulation results are also given. Chapter 5 presents the development of the boundary-scan test and diagnosis system, together with the associated test strategy. Limitations of the test system and further enhancements that can be made are also discussed. Chapter 6 gives a brief introduction to multichip module testing and looks at future directions. Lastly, Chapter 7 summarizes the work in this thesis.
Chapter 2

IEEE Std 1149.1 Architecture

The development of IEEE Std 1149.1 Test Access Port and Boundary-Scan Architecture has made an important change in electronic design and test methodologies. Over the years, it has been widely adopted as the design-for-test (DFT) method of choice for complex boards that use high density interconnect and packaging technologies. In Section 2.2 of this chapter, a general overview of the operation of IEEE Std 1149.1 is given. The test circuitry defined by the standard is briefly described in Section 2.3.

Most of the materials presented in this chapter are summarized from the official standard [1]. It is also important to note that the standard is fairly complex and is very detailed in many places that any summary may fail to describe important features of the full definition. This chapter only describes the basic concept of the standard and provides a background to the discussion in later chapters.
2.1 Background

With the advent of packaging technologies, such as surface mount technology (SMT) and multichip module (MCM), circuit boards become much denser and faster than ever before. As a result, the testing becomes very difficult because the internal circuits are hard to access through traditional bed-of-nail fixtures. At the same time, board-level probing is becoming increasingly difficult and costly. The IEEE Std 1149.1 Test Access Port and Boundary-Scan Architecture [1] was developed to offer an adequate alternative to the above mentioned test problems.

The process of developing the IEEE Std 1149.1 began in 1985 when the Joint Test Action Group (JTAG) was formed. Between 1986 and 1988, the JTAG Technical Subcommittee developed and published a series of proposals for a standardized form of boundary-scan. Finally, the IEEE Std 1149.1 was approved in February 1990. The standard defines a test access port and boundary-scan architecture for digital integrated circuits and for the digital portions of mixed analog/digital integrated circuits. The technique primarily seeks to provide a solution to the problem of testing assembled printed circuit boards and other products based on highly complex digital integrated circuits and high-density surface-mounting assembly techniques. They also provide a means of accessing and controlling design-for-test features built into the digital integrated circuits themselves.

Note that the standard only provides a document describing the basic rules to which any boundary-scan architecture has to comply. It includes possible embodiments for various boundary-scan components, but it does not consider any preferred architecture implementation. Specific architecture related details are defined by the user.
2.2 Basic Operation of IEEE Std 1149.1

The circuitry defined by the standard basically allows test instructions and associated test data to be fed into a component and, subsequently, allows the results of execution of such instructions to be read out. All information including instructions, test data and test results is communicated in a serial format. Starting from an initial state in which the test circuitry defined by the standard is inactive, a typical sequence of operations would be as follows.

The first step is to load serially into the component the instruction code for the particular operation to be performed. The test logic defined by the standard is designed such that the serial movement of instruction information is not apparent to those circuit blocks whose operation is controlled by the instruction. The instruction applied to these blocks changes only on completion of the shifting process.

Once the instruction has been loaded, the selected test circuitry is configured to respond. In some cases, however, it is necessary to load data into the selected test circuitry before a meaningful response can be made. Such data are loaded into the component serially in a manner similar to the process used previously to load the instruction. The movement of test data has no effect on the instruction present in the test circuitry. Following executing of the test instruction based on the supplied data, the results of the test can be examined by shifting data out of the component. In cases where the same test operation is to be repeated but with different data, new test data can be shifted into the component while the test results are shifted out. There is no need for the instruction to be reloaded. Operation of the test circuitry may proceed by loading and executing several further instructions in a manner similar to that described. The test concludes by returning the test circuitry to its initial state.
2.3 Test Logic Architecture

The IEEE Std 1149.1 test logic architecture contains four basic hardware elements: (1) a test interface referred to as the Test Access Port (TAP), (2) a TAP controller, (3) an instruction register, and (4) a group of test data registers. The group of test data registers include a bypass and a boundary-scan register. It may also include an optional device identification register and further optional test data registers. Figure 2.1 shows the conceptual view of the test logic architecture. The optional registers are shown in dotted line. Specifications for the various blocks contained within the test logic design are briefly described in the following subsections.

As shown in Figure 2.1, the instruction register and test data registers are separate shift-register-based paths that are connected in parallel and have a common test data input
(TDI) pin and a common test data output (TDO) pin. This architecture allows the TAP controller to control the selection between the alternative instruction and test data register paths between TDI and TDO. Moreover, the TAP controller, the instruction register and the associated circuitry necessary for control of the instruction and test data registers are dedicated test logic. In other words, these test logic blocks do not perform any system function.

2.3.1 The Test Access Port (TAP)

The TAP is a general-purpose port that can provide access to the many test support functions built into a component. For the IEEE Std 1149.1 test logic, the TAP contains a minimum of three input connections (TCK, TMS, TDI) and one output connection (TDO). An optional fourth input connection (TRST*) provides for asynchronous initialization of the test logic.

The Test Clock Input — TCK

TCK provides the clock for the test logic and is included so that the serial test data path between components can be used independently of built-in system clocks. Moreover, the provision of an independent clock ensures that test data can be moved to or from a chip without changing the state of the on-chip system logic.

The Test Mode Select Input — TMS

Signal values presented at TMS are sampled by the test logic on the rising edge of TCK and are decoded by the TAP controller to control the test operations. When TMS is not
driven, it must be held at a logic 1. This can be accomplished by including a pull-up resistor in the component's TMS input circuitry.

**The Test Data Input — TDI**

Serial test instructions and data are received by the test logic at TDI. Values presented at TDI are clocked into either the instruction register or a test data register on the rising edge of TCK. When TMS is not driven, it must be held at a logic 1. This can be accomplished by including a pull-up resistor in the component's TDI input circuitry.

**The Test Data Output — TDO**

TDO is the serial output for test instructions and data from the test logic. The contents of either the instruction register or a test data register are serially shifted out of TDO on the falling edge of TCK. When data are not being shifted through the chip, TDO is set to an inactive state. This requirement can be met through the use of a three-state output buffer.

**The Test Reset Input — TRST***

The optional TRST* input provides for asynchronous initialization of the TAP controller. When a logic 0 is applied to TRST*, the test logic can be reset independent of the on-chip system logic and the TAP controller is asynchronously forced into its reset state independently of TCK and TMS signals. Due to its resetting behavior, a pull-up resistor is included in the TRST* input circuitry of the component such that an undriven input produces a response identical to the application of a logic 1.
2.3.2 The TAP Controller

The TAP controller is a synchronous finite state machine that responds to changes at the TMS and TCK signals of the TAP. Its basic function is to generate clock and control signals required for the correct sequence of operations of the instruction register, test data registers, and the associated circuitry within the test logic. The state diagram for the TAP controller is shown in Figure 2.2. It contains sixteen states and the value shown adjacent to each state transition in this figure represents the signal present at TMS at the time of a rising edge of TCK.

![Figure 2.2 Tap Controller State Diagram](image)

---

*IEEE Std 1149.1 Architecture* 15
Note that in the Test-Logic-Reset state, the TAP controller issues a reset signal that places all test logic in a condition that does not disturb the normal on-chip system logic operation. The TAP controller can be asynchronously forced into this reset state by the use of TRST* signal. Besides, the TAP controller can be set synchronously into the reset state following five rising edges at TCK, provided TMS is held high, regardless of its current position in the state diagram.

Figure 2.3 shows the TAP controller output control signals, along with the instruction register and test data registers interconnects. Detailed design requirements for the TAP controller are contained in Chapter 3.

2.3.3 The Instruction Register

The instruction register allows an instruction to be shifted into the design. The instruction is used to select the test to be performed or the test data register to be accessed or
both. Basically, it is a shift-register-based design with a shift portion and a parallel hold portion. The shift portion can be loaded in parallel at Capture-IR controller state, shifted between TDI and TDO at Shift-IR, and the contents of the shift portion are transferred to the hold portion at Update-IR.

The size of the instruction register dictates the size of the instruction codes that can be used. As defined by the standard, the instruction register must contain at least two register cells which can hold instruction data and the two least significant register cells must load a fixed binary "01" pattern in the Capture-IR controller state.

Figure 2.4 shows an implementation of an instruction register cell that operates in response to the signals generated by the TAP controller (see Figure 2.3). Each instruction register cell has a shift-register flip-flop controlled by the ClockIR signal and a parallel output latch controlled by the UpdateIR signal. The shift-register flip-flop holds the instruction bit moving through the instruction register. The instruction is latched onto the parallel output during the Update-IR controller state. Thus, the values latched onto the parallel outputs define the test mode to be entered and the test data register to be accessed.

Figure 2.4 An Instruction Register Cell
2.3.4 Test Data Registers

The test logic architecture contains a minimum of two test data registers — the bypass and boundary-scan registers. In addition, the design of a third, optional, device identification register is defined. Additional user-defined data registers can be included. The test data registers are arranged in parallel from TDI to TDO, forming a serial scan path. Each test data register has a fixed length and can be accessed through one or more instructions that can be shifted into the instruction register. When one scan path is being accessed, all the other scan paths remain in their present state.

Bypass Register

The bypass register consists of a single shift-register stage. When selected, the bypass register provides a single bit scan path for the movement of test data between TDI and TDO. This path can be selected when no other test data register needs to be accessed during a board-level test operation. Use of the bypass register in a component speeds access to test data registers in other components on a board-level test data path. The bypass register can be implemented as shown in Figure 2.5.

![Figure 2.5 A Bypass Register Implementation](image-url)
Boundary-Scan Register

The mandatory boundary-scan register (BSR) consists of a number of boundary-scan register cells (BSCs), equal to the number of shift-register stages contained in the BSR. These cells are positioned around the on-chip system logic of a component (see Section 1.2). They are connected to form a single shift-register-based path that is connected between TDI and TDO in the Shift-DR controller state when an appropriate instruction is selected. The BSCs provide the controllability and observability features required to perform boundary-scan testing and they must be dedicated to the test logic only.

Figure 2.6 shows an implementation of a BSC which can be used for both input and output pins. Depending on the control signals applied to the multiplexers, data can either be loaded into the scan register from the Signal-in port, or driven from the register through the Signal-out port of the cell. The second flip-flop controlled by the UpdateDR signal is provided to ensure that the signals driven out of the cell are held while new data is shifted into the cell using the ClockDR signal.

![Figure 2.6 A Boundary-Scan Register Cell](image_url)
Device Identification Register

The device identification register is an optional test data register that allows the manufacturer, part number, and version of a component to be determined through the TAP. One application of the device identification register is to distinguish the manufacturer(s) of components on a board when multiple sourcing is used. Figure 2.7 shows the bit assignments defined for the device identification register.

<table>
<thead>
<tr>
<th>MSB</th>
<th>31</th>
<th>28</th>
<th>27</th>
<th>12</th>
<th>11</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Version</td>
<td>Part number</td>
<td>Manufacturer identity</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(4 bits)</td>
<td>(16 bits)</td>
<td>(11 bits)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.7 Structure of the Device Identification Register

Design-Specific Test Data Registers

The optional design-specific test data registers are provided to allow access to design-specific test support features in the integrated circuit such as self-tests, scan-path, etc. These test data registers can be a part of the on-chip system logic and can have both system and test functions.

2.3.5 Instructions

The instructions are serially loaded into the test logic during an instruction register scan cycle and are decoded to achieve two basic functions. First, each instruction defines the set of test data registers that can operate while the instruction is current. Second, an instruction defined the serial test data register path that is used to shift data between TDI and
TDO during data register scanning.

The standard offers two groups of instructions: the Public and Private instructions. Public instructions are well documented and provide the user with access to the test features, while Private instructions are intended solely for the use of the component manufacturer. All components that claim conformance to the standard have to support a minimum set of three public instructions. Six other optional public instructions are also provided. Table 2.1 summarizes the public instructions. Descriptions of the instructions are included in the test chip implementation in Chapter 3.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYPASS</td>
<td>Mandatory</td>
</tr>
<tr>
<td>SAMPLE/PRELOAD</td>
<td>Mandatory</td>
</tr>
<tr>
<td>EXTEST</td>
<td>Mandatory</td>
</tr>
<tr>
<td>INTEST</td>
<td>Optional</td>
</tr>
<tr>
<td>RUNBIST</td>
<td>Optional</td>
</tr>
<tr>
<td>CLAMP</td>
<td>Optional</td>
</tr>
<tr>
<td>IDCODE</td>
<td>Optional</td>
</tr>
<tr>
<td>USERCODE</td>
<td>Optional</td>
</tr>
<tr>
<td>HIGHZ</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Table 2.1 Public Instructions
Chapter 3

Chip Design Implementing Boundary-Scan

The insertion of test circuitry defined by IEEE Std 1149.1 in an integrated circuit allows boundary-scan testing, however, this design-for-testability technique benefits accrue at the board or module level. To evaluate the boundary-scan testing both at IC and board level, a chip containing the boundary-scan architecture has been designed and fabricated using a 1.2μm n-well CMOS process. This chapter describes the design and implementation of the boundary-scan test chip using the standard cell-based design approach. In Section 3.1, the cell library is introduced. Section 3.2 gives a general overview of the implemented boundary-scan architecture. The design specifications are also presented. Section 3.3 discusses the design issues of the test chip. The chip implementation and layout considerations are given in Section 3.4. This chapter concludes with a brief summary in Section 3.5.
3.1 Cell Library

A set of CMOS standard cells and pad cells for 1.2\textmu m n-well fabrication process is developed for the test chip implementation. We choose 1.2\textmu m technology because it is compatible with the Microelectronics Fabrication Facility (MFF) base-line process. These cells are scaled from the existing CMOSN standard cell library [11], which has been designed for high-speed and high-density random logic applications. Only those cells required for implementation are scaled to 1.2\textmu m and mapped to the MOSIS Scalable CMOS (SCMOS) layers [12]. A complete listing of the cell library resides in Appendix A.

3.1.1 Standard Cell Layout Format

Standard cells are mainly used for most random logic circuits. Figure 3.1 shows a summary of the basic layout rules for the standard cell. All standard cell I/O terminals are designed in metal2 and are double entry so that they are available on the top and bottom edges of the cell. The terminals must have 10\lambda center-to-center spacing along the horizontal axis, with an offset of 5\lambda from either side of the cell. As a result, the overall width of the cell is a multiple of 10\lambda. Whenever possible, the unused metal2 slots are used for routing channels over the cell. These signal feedthroughs provide a means of routing across the cell rows without the need for feedthrough cells and reduce the capacitive loading on large nets. Each cell is designed with three metal1 busses which run horizontally through the cell. The top and bottom busses are VDD and are 20\lambda in width, while the center bus is VSS and is 30\lambda in width. This structure allows the designs to be accommodated within the same standard cell height of 250\lambda. No metal1 or metal2 can be placed outside the power rails, as this would
conflict with the router. For a n-well fabrication process, the well is extended 6λ outside of the VDD busses along the vertical axis and is extended 5λ beyond the cell edge along the horizontal axis. The cells are tiled horizontally, so there must be no design rule violations, when cells are abutted to form rows.

The library contains 13 standard cells. Cells present include simple gates – inverters, nand, nor and mux. There are 2-state and 3-state buffers and three different types of flip-flops. In addition, pull-up and pull-down resistors are provided for on-chip purpose. The maximum and minimum width for a single PMOS transistor are 43.2μm and 13.8μm respectively, and that for NMOS transistor are 45.6μm and 9.6μm respectively. While the
PMOS/NMOS width ratio is around 2 to 3 for most of the standard cells, the actual size of the transistors depends on the layout structure and logic function.

3.1.2 Pad Cell Layout Format

Pad cells are needed for interfacing to the outside world. The pad cells are abutted to form a rectangular pad frame, which defines the size of the silicon die. The inside of the pad frame contains the usable IC design area, and channels used for routing to the pads. The basic I/O pad layout structure is shown in Figure 3.2. The basic cell size is set by the metal2 pad size of $170\lambda \times 170\lambda$ ($102 \times 102\mu m^2$), and the required pad-to-pad spacing of $101.4\mu m$. As a result, the I/O pads remain on a minimum 8 mil pitch (203.4\mu m). Note that the pad size is

![Figure 3.2 Basic Pad Cell Layout Structure](image-url)
fixed and does not scale with process. For each cell, there are two metal2 busses which run horizontally on one side of the pad. The one which is closer to the pad is VDD and the other is VSS. Both are 127\(\lambda\) in width and separated by 42\(\lambda\). All the input protection circuits, pad driver circuits and other required logic are placed in the I/O circuitry region as shown in Figure 3.2. These I/O circuitry are connected to the pads via metal1 and should be placed at least 30\(\mu\)m from the perimeter of the pad, as pad bonding can distort the pad contact by exerting pressure on it.

Input protection circuit is included in the pad cell to protect against damage caused by electrostatic discharge (ESD). The input protection circuit is shown in Figure 3.3. The input resistor is composed of 10 squares of polysilicon having a resistance of approximately 250 ohms. After the input resistor the signal line is connected in metal to a P+/N- diode and an N+/P- diode. The diode areas are maximized in order to enhance their capability to handle current. The P+/N- diode is surrounded by an N+ ring tied to VDD while the N+/P- diode is surrounded by a P+ ring tied to VSS. To reduce the possibility of latch-up, the N-channel transistors are surrounded by a P+ active area guard ring and the P-channel transistors are surrounded by a N+ active area guard ring [11].

![Diagram of Input Protection Circuit](image)

Figure 3.3 Input Protection Circuit
There are 5 different pad cells, including an input pad (X12IPD), an output pad (OPAD12) and a tri-state pad (X12TRI). PWR12 and GND12 are two power pads which connect to the VDD and VSS power busses respectively. The input, output and tri-state pads are configured in a two stage buffer with stage ratio ranged from 3 to 7.

3.2 Chip Overview

The implemented boundary-scan architecture is shown in Figure 3.4. Basically, the boundary-scan test chip design contains all the basic components which conform to the minimum requirement of the standard. This basic configuration is sufficient for the IC user to examine the operation of a prototype system and to test assembled products for assembled-induced defects during manufacturing, which is also the main interest in our work. The system logic contains six inverters. Each inverter is connected to a system input pin and a system output pin. In normal mode, the chip is functionally equivalent to a hex inverters with pull-up circuitry included at each system input pin. In the test mode, the normal operation of the hex inverters is inhibited and the test circuitry is enabled to perform boundary-scan test operations. The I/O boundary of the chip can then be observed and controlled.

In addition to the four dedicated TAP connections, the optional TRST* input is provided for asynchronous initialization of the TAP controller. The instruction register (IR) and boundary-scan register (BSR) are two bits and twelve bits long respectively. All signals generated by the TAP controller except the Select and the Enable signal are distributed to the instruction register or the test data registers. The Select signal is used to control the
multiplexer labeled G2, while the Enable signal is used for 3-state control of the TDO output. Note that a flip-flop is included before the 3-state output buffer so that changes at TDO are only occur on the falling edge of TCK.

Figure 3.4 Implemented Boundary-Scan Architecture
3.2.1 Instruction Set

Four instructions are supported by our implemented architecture. In addition to the three mandatory instructions (BYPASS, EXTEST and SAMPLE/PRELOAD), the optional instruction INTEST is included to provide on-chip testability feature. Each of these instructions are briefly described as follows. The detailed test application with these instructions is contained in Chapter 5.

The BYPASS Instruction The mandatory BYPASS instruction selects the bypass register to be connected between TDI and TDO. It allows serial data to be transferred through the component from TDI to TDO without affecting the operation of the on-chip system logic. The binary code for the BYPASS instruction is \{111 ... 1\}, i.e., a logic 1 is loaded into every instruction register cell.

The SAMPLE/PRELOAD Instruction The mandatory SAMPLE/PRELOAD instruction selects only the boundary-scan register to be connected for serial access between TDI and TDO. It allows scanning of the boundary-scan register without interfering with the normal operation of the on-chip system logic. During this instruction, data entering and leaving the on-chip system logic can be sampled via a data register scan operation. Besides, this instruction is used to preload an initial data pattern at the latched parallel outputs of boundary-scan register cells prior to selection of another boundary-scan operation. Note that the shifting of data for SAMPLE and PRELOAD phases can occur concurrently, while data captured is shifted out, the preload data can be shifted in. The binary code for the SAMPLE/PRELOAD instruction is defined by the user.
The \textit{EXTEST} Instruction \ The mandatory \textit{EXTEST} instruction selects only the boundary-scan register to be connected for serial access between TDI and TDO. It is used to allow testing of off-chip circuitry and board level interconnections. During this instruction, the boundary-scan register cells at output pins apply test stimuli, while those at input pins capture test results. The binary code for the \textit{EXTEST} instruction is \{000 \ldots 0\}, i.e., a logic 0 is loaded into every instruction register cell.

The \textit{INTEST} Instruction \ The optional \textit{INTEST} instruction selects only the boundary-scan register to be connected for serial access between TDI and TDO. It allows static, low-speed testing of the on-chip system logic. Using this instruction, test stimuli are shifted in one at a time and applied to the on-chip system logic. The test results are captured into the boundary-scan register and are examined by subsequent shifting. The binary code for the \textit{INTEST} instruction is defined by the user.

Since four instructions are supported by the implemented architecture, a 2-bit instruction register is sufficient for loading the instructions. In fact, the instruction register must contain at least two register cells as defined by the standard. The binary codes for the instructions are listed in Table 3.1.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Binary Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYPASS</td>
<td>11</td>
</tr>
<tr>
<td>SAMPLE/PRELOAD</td>
<td>01</td>
</tr>
<tr>
<td>EXTEST</td>
<td>00</td>
</tr>
<tr>
<td>INTEST</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 3.1 Implemented Instructions and their Binary Codes
3.2.2 Instruction Register

Figure 3.5 shows the 2-bit instruction register. It contains two instruction register cells with the same type of parallel output latch, which is provided with a set input. For each instruction register cell (see also Figure 2.4), the parallel output is set high in the Test-Logic-Reset controller state as a result of a logic 0 received at the Reset* or TRST* input of the cell. Notice that a low Reset* signal will be generated after entry into the Test-Logic-Reset controller state under control of TMS and TCK, provided that TRST* is held at logic 1. Thus, the parallel output of the instruction register cell will change on the falling edge of TCK. In contrast, when a logic 0 is applied to TRST*, the change at the parallel output occurs immediately, irrespective of the state of TMS or TCK. As a consequence, when either one of the above actions are executed, the Bypass instruction, which has a instruction code of \{11\}, would be latched onto the parallel outputs of the instruction register.

In addition, the standard defines that the parallel data inputs of the two least significant instruction register cells must be tied to fixed logic levels, so that a fixed binary

![Figure 3.5 2-bit Instruction Register](image-url)
'01' pattern is loaded in the *Capture-IR* controller state. In our case, the Data port of the least significant instruction register cell (see Figure 3.5) is tied to VDD and that of the most significant cell is tied to VSS.

Another design feature related to *BYPASS* instruction is described as follows. Since the TDI input is designed with pull-up circuitry, an open circuit fault in the serial board-level test data path would cause a logic 1 to be shifted into the test logic. As a result, the *BYPASS* instruction becomes active following an instruction-scan cycle and the bypass register is being selected. Therefore, no unwanted interference with the operation of the on-chip system logic can occur.

### 3.2.3 Control Signals

The instruction decode logic design is determined by the chosen instruction codes and boundary-scan circuitry. In our design, it generates two control signals for the test data registers. The first control signal defines the serial test data register path that is used to shift data between TDI and TDO. Since we have two test data registers and only the *BYPASS* instruction selects the bypass register to be connected between TDI and TDO, while other instructions select the boundary-scan register, this control signal is simply generated by logical *and*ing the two latched parallel outputs of the instruction register and connected to the selection pin of the MUX labeled G1 as shown in Figure 3.4.

The second control signal controls the routing of data through each boundary-scan register cell (BSC). Figure 3.6 shows an implementation of a BSC (see also Figure 2.6). When a logic 0 is applied to the Mode port, the on-chip system logic can be normally
operated. Table 3.2 shows how the Mode signal for the BSC is derived for each of the boundary-scan register instructions.

In case when the BYPASS instruction is selected, the Mode signal is set to be equal to logic 0. As a result, the Mode signal can be obtained by inverting the latched parallel output signal of the instruction register cell that is placed closest to TDO. It then connects to the Mode signal network which links to all BSCs. As was discussed in Section 3.2.2, the BYPASS instruction is forced into the latches at the parallel outputs of the instruction register during the Test-Logic-Reset controller state, thus the Mode signal is equal to 0 in this state. In other words, the above Mode signal assignment allows normal operation of the on-chip system logic as long as neither the EXTEST nor the INTEST instruction is selected.
3.2.4 System Pins

Basically, four different boundary-scan cell designs are supported to meet the specific requirements for different types of system pins [1]. These are: (1) general BSC for 2-state input and output pins, (2) observe-only BSC for high performance inputs, (3) BSC for bidirectional pins and (4) BSC for tri-state output pins. For our test chip implementation, we only consider 2-state system input and output pins. For 2-state output pins where signals can only be at the high or low logic level at any given instant, one BSC is sufficient to allow the state of the pin to be controlled or observed. Clock inputs to the on-chip system logic are not considered. Since our main interest is to test board-level or module-level interconnects using boundary-scan, the inclusion of other types of system pins and clock inputs would make the chip design complicated and the subsequent testing difficult.

3.3 Design Considerations

In this section, several design issues other than those presented in the previous section are discussed. They are not part of the standard, however, the inclusion of these design features in our test chip facilitates the boundary-scan testing.

3.3.1 Output Switching Limitation

As was discussed in Section 3.2.1, the EXTEST instruction is used to allow testing of wiring interconnects between multiple chips on a loaded board. When this instruction is selected, the pins of a chip that does not normally support simultaneous switching or enabling
of all outputs are controlled from the boundary-scan path, rather than from the on-chip system logic. It is probable that the tests applied through the boundary-scan path will cause all outputs to change state and/or be active simultaneously. This would cause interference to the core logic as well as the boundary-scan cells and TAP.

In our design, delay elements are added to prevent simultaneous switching of outputs when pins are driven from the boundary-scan register. The delay element is implemented with a buffer cell (BUFF). Figure 3.7 shows how this is implemented. The added delay should be small in comparison with the minimum period of TCK, but should be sufficient to ensure that the power-current demand arising from the change of state at one pin does not
overlap with that from another. Note that the added delay impact only changes at the pin due
to a change of instruction (the delays in the Mode signal network) and a change in test pattern
(the delays in the UpdateDR signal network). Signals received from the on-chip system logic
propagate through the boundary-scan cells without added delay. For symmetry, the delay
elements are also added to the UpdateDR and Mode signal networks connected to the BSCs
that associated with the system input pins. Although it is not necessary, the buffers can
provide a means of enhancing the drive strength.

3.3.2 Internal Buffers

The signals (ShiftDR, ClockDR and UpdateDR) generated by the TAP controller (see
Figure 3.4) are connected to each BSC to control its operation. The output loadings for these
signals increase substantially as the number of BSC increases. Thus, internal buffers are
inserted at these TAP controller outputs in order to enhance the drive strength. In addition, a
buffer is placed after the instruction decoding logic to drive the Mode signal to all the BSCs.

3.3.3 Pull-up Resistors

Pull-up resistor is included in each system input pin. This can simplify diagnosis since
any open circuit faults in the board-level interconnects associated with these pins would
result a defined logic value presented at these pins. Note that on-chip pull-up resistors for
system input pins are not included in the test chip, only external resistors are connected.
3.4 IC Implementation

After specifying the design requirements, the architecture can then be implemented in an integrated circuit. Figure 3.8 illustrates the design flow for our IC implementation. The flow starts initially from the design architecture and finally to the IC fabrication. Sophisticated computer aided design (CAD) tools are extensively utilized in the design process in order to cope with the high complexity of the IC. Design verification is also done in various steps to verify the performance and functionality of the design.

![IC Design Flow](image)

Figure 3.8 IC Design Flow
3.4.1 Macrocell Generation

Instead of placing all the individual standard cells directly into the chip layout. The layout of several standard components are first generated to form macrocells or blocks. The standard components include bypass register, instruction register cell, boundary-scan register cell, and the TAP controller. The use of these macrocells facilitate the subsequent floor-planning so that the silicon area can be well utilized. Besides, these macrocells can be reused in the same chip or other chip designs as they are mandatory components for implementing IEEE Std 1149.1.

The design starts at the top level by writing the VHDL codes for each macrocell. The codes are then compiled and synthesized to the gate-level netlists using Synopsys VHDL Compiler® and Synopsys Design Compiler®. The resulting netlists are then transferred to the Cadence database. Note that the synthesized netlists are mapped to the existing 0.8μm technology library, as 1.2μm technology library is not available in the Synopsys environment. Thus, it is necessary to convert these 0.8μm cells to 1.2μm cells before the final layout can be generated.

The VHDL codes and synthesized netlists are given in Appendix B. The TAP controller design contains 36 gates, 3 multiplexers and 8 flip-flops. One internal buffer is included to drive a large capacitive net. Table 3.3 summarizes the transistor count for each macrocell. Generated from the gate-level schematics, the macrocell layouts are then automatically created by the Cadence Cell Ensemble® place and route tool. In this case, two metal layers are used for routing. Pre- and post-layout timing analysis are done on these macrocells to make sure they meet the design constraints. Note that a gate-level Verilog file for each
<table>
<thead>
<tr>
<th>Macrocell</th>
<th>Transistors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bypass register</td>
<td>36</td>
</tr>
<tr>
<td>Instruction register cell</td>
<td>82</td>
</tr>
<tr>
<td>Boundary-scan register cell</td>
<td>76</td>
</tr>
<tr>
<td>TAP controller</td>
<td>518</td>
</tr>
</tbody>
</table>

Table 3.3 Macrocell Transistor Count

A macrocell is also created for subsequent logic simulation. After creating the macrocells, the schematics of the entire chip design is then captured. To verify the performance of the design, we use Verilog-XL® for logic simulation and HSPICE for pre-layout circuit simulation.

### 3.4.2 Chip Floorplan

Floorplanning is the task of placing the components in the design so as to optimize the final chip layout with respect to a set of constraints defined on the interconnections and chip dimensions. Since our design is ‘core limited’, the placement of the blocks and cells for the core portion of the chip are made as compact as possible, so that the overall chip area is kept within bounds. The components are placed interactively such that the length of interconnects is minimized. The floorplan for the chip core is shown in Figure 3.9. The interconnects are then routed using the Cadence Cell Ensemble® place and route tool with the given netlist. Note that this tool allows both blocks and individual standard cell to be routed on the same plane.
3.4.3 Layout Considerations

After creating the chip core layout, the remaining task is to add power busses and I/O pads to the design. The design quality of these circuits is a critical factor that determines the reliability, signal integrity, and interchip communication speed of the chip in a system environment. Several layout issues are considered in laying out these circuits.

**Power Distribution**  In our design, the chip power distribution is routed in a way such that the pad power bus are isolated from the cell power bus in order to reduce power and ground noise in the functional cells [13], [14]. Two VDD and two VSS pads are dedicated to both I/O
pad power busses and cell power busses. As a matter of fact, the more the pins dedicated to VDD and VSS, the less the power and ground noise induced in the pad area. Figure 3.10 illustrates the power bus distribution in our chip design. In general, these power busses are much wider than that in standard cells. Guard rings are strapped in the cell power bus to further reduce power noise and prevent latch-up [13], [15].

I/O Pads As was mentioned in Section 3.4.2, our design is ‘core limited’, the pad frame is designed such that the chip core layout just fits inside it. For the sake of regularity, the pads for system input pins and output pins are placed at the left side and right side of the pad frame respectively, while the TAP connections are located at the bottom of the pad frame. Moreover, the relative position of the I/O pads are placed such that the length of interconnects routed to the chip core is minimized.
3.4.4 Chip Fabrication

The entire chip layout is created after connecting the chip core to the I/O pads. Design rule check (DRC) and layout versus schematic (LVS) are performed. Finally, the layout of the entire chip is extracted and post-layout timing analysis is done using HSPICE to verify the performance. The output signals and control signals are carefully examined, especially for those nodes which drive large capacitive nets.

The test chip was fabricated using MOSIS HP_CMOS34 process, which is a 1.2μm, 5-V, 2-metal-layer, n-well CMOS process. A photomicrograph of the boundary-scan test chip is shown in Figure 3.11. The chip integrates 2287 transistors, including those found in the I/O buffers, in an area of 2.04mm × 2.04mm. The chip was packaged in a 40-pin DIP. Each packaged chip was first tested in its normal operation mode before it is mounted on a board for boundary-scan testing. Board-level testing will be discussed in Chapter 5.

3.5 Summary

We have successfully implemented a boundary-scan test chip which is complied with IEEE Std 1149.1. The chip can be used as a standalone device or a boundary-scan test device. In the latter case, it performs as a chip sized test vehicle for assembled-induced defects, such as opens or shorts, of an assembled product. Some comments on the design and suggestions for future work are discussed below.

Numerous benefits of boundary-scan testing have been well documented in test literature [2], [16], [17]. However, when we consider only at the IC level, the penalties of
boundary-scan testing may appear to outweigh the benefits. This is actually what we have learnt in the design and implementation process, although the chip is designed solely for the purpose of boundary-scan testing. First of all, boundary-scan introduces a fair amount of circuit overhead penalty, as can be seen in our chip layout (see Figure 3.11). Together with the addition of dedicated test pins to the chip, boundary-scan is not suitable for small integrated circuit design. It also creates a performance trade-off, since a multiplexer is inserted in the functional path of the I/O signals (see Figure 2.6 and Figure 3.6).

Another cost of boundary-scan is the increased design time. From a circuit designer perspective, the insertion of boundary-scan circuitry into an integrated circuit simply means that some form of additional design effort will be required. The design time can be reduced if
this circuitry is automatically placed around the core design by the use of CAD tools. As a suggestion for future work, a comprehensive library which contains all the boundary-scan components can be built into the Synopsys Test Compiler® tool. Given a top-level core design, the Test Compiler® can automatically add the boundary-scan path and associated test logic to the design. Optimization and verification can also be done to ensure the performance of the design. Furthermore, a set of boundary-scan I/O pads can be designed such that the final chip layout is optimized, especially for those with high transistor and I/O pin count.

It can be concluded that although there is trade-off in applying boundary-scan testing, the benefits usually far outweigh the penalties when we consider a more comprehensive analysis spanning all levels of assembly from chip to board, and to system. In later chapters, we will find out how boundary-scan can be applied as a DFT strategy in board-level testing – from simple circuit board to complex multichip module.
Chapter 4

Interconnect Diagnosis with Boundary-Scan

4.1 Introduction

The packaging density of a printed circuit board (PCB) or multichip module (MCM) has been increasing due to the miniaturization of digital circuits and current technologies. With such a high density and complexity, shorts are more likely to happen among interconnect nets. As a result, testing and diagnosis for interconnect faults is an important problem in VLSI, PCB and MCM manufacturing. However, due to the limited observability of the chip leads, traditional testing technique such as in-circuit test approach is no longer applicable.
With IEEE Std 1149.1 boundary-scan architecture [1], these interconnect faults can be tested and diagnosed under full controllability and observability conditions for MCMs and PCBs. Figure 4.1 shows an example circuit that contains a short-to-ground fault, an open circuit fault and a bridging short fault in the board interconnect. By executing the EXTEST instruction, these faults can be detected and located. In a boundary-scan environment, test sets are scanned in, the saving in the number of tests and the test application time is particular important. A variety of approaches [18]-[27] have been proposed for generating diagnostic test sets based on boundary-scan architecture. However, they only assume a simple interconnect short fault model. It can be shown later that it is insufficient for full diagnosis if the system is implemented with CMOS technology, for which a more complicated bridging short fault model is required [28]. Although a new complete diagnosis patterns was developed recently [29], the large number of test vectors generated made boundary-scan testing

Figure 4.1 Testing for Interconnect Faults using EXTEST
inefficient.

In this chapter, an enhanced structural diagnostic approach for board interconnect testing with boundary-scan is proposed. This approach not only tackles multiple faults, but also deals with more complex issues such as CMOS bridging faults. The test set is generated by a simple yet effective algorithm based on graph theoretic technique. The adjacency fault model in which two nets can be shorted only if they terminate at adjacent pins or their tracks are adjacent within a predetermined distance, is adopted [24]. A two-step algorithm is further proposed for test generation and diagnosis. Compared with one-step diagnosis, the two-step diagnosis algorithm can further reduce the number of test vectors while retaining the same level of diagnostic resolution. Both algorithms guarantee the complete diagnosis of multiple interconnect faults with no aliasing and confounding. Simulation results for benchmark layouts and randomly generated layouts show a 18% to 45% savings in the number of tests for one-step diagnosis. For two-step diagnosis, more than 50% reduction in the test length can be achieved when the short fault rate is very small. We shall also show the adjacency fault model has a major impact on the efficiency of the two-step diagnosis.

In Section 4.2, a brief review of the fault model and previous diagnosis algorithms is introduced. Section 4.3 addresses the deficiency of previous approaches. In Section 4.4, the preliminaries for the proposed diagnosis approach are given. Both the one-step and two-step diagnosis algorithms are also described. Section 4.5 presents the simulation results. Summary is given in Section 4.6.
4.2 Review of Fault Model and Diagnosis Algorithm

4.2.1 Notation and Definitions

We shall use the following notation and definitions:

**Parallel Test Vector (PTV):** the vector applied to all nets of a wiring network in parallel.

**Sequential Test Vector (STV):** the vector applied to a net, over a period time, by a sequence of PTVs.

**Test Set S:** the collection of all STVs. Each column of S is a PTV and each row of S is a STV.

**Sequential Respond Vector (SRV):** the response of a net to a STV.

**Syndrome:** if the SRV of a net differs from its STV, then this vector is referred to as a fault syndrome.

**Aliasing syndrome:** if a syndrome in the presence of a fault is the same as the fault free response of a net, then it is impossible to determine whether or not this net is also part of the short. The response in this case is referred to as an aliasing syndrome.

**Confounding syndrome:** the bridge fault between a pair of nets may produce the same syndrome as between another net pair, so it is impossible to determine whether or not there is a short between which pair of nets. The response is called a confounding syndrome.

4.2.2 Fault Model

There are three types of faults commonly associated with nets on a PCB. They are stuck-at-faults, open faults, and bridging faults. A net shorted to a power line VCC (GND) will exhibit a stuck-at-1 (stuck-at-0) behavior. If a net contains an open, the logic value
interpreted by all the floating input pins of the opened net will be the same. And the open fault can be modeled as either a pull-up or pull-down circuit, dependent on the circuit design and technology used. A bridging fault creates a short between two or more nets. The resulting behavior can be modeled as either (1) a wired-OR short, where logic 1 dominates, (2) a wired-AND short, where logic 0 dominates, or (3) a strong-driver short, where one driver dominates the resulting behavior. In all cases, all nets involved in a short will have the same resulting logic value.

4.2.3 Previous Diagnosis Algorithms

A variety of approaches [18]-[27] have been proposed for generating diagnostic test sets for boundary-scan testing. They differ with each other in their diagnostic resolution. In general, these approaches fall in two basic categories: the behavioral testing and the structural testing strategy.

In behavioral testing, it is assumed that every net can be shorted to any other net, regardless of their placement in the layout. The Counting Sequence Algorithm [18] has been proposed to detect all bridging faults with a test length of \( \lceil \log_2(n) \rceil \), where \( n \) is the number of nets in the interconnect. Table 4.1 shows the test set for \( n = 8 \) generated by counting sequence. Since the patterns include STVs with all zeros and all ones, the stuck-at-zero or stuck-at-one fault on these nets cannot be detected. By excluding the all zeros and all ones, the modified counting sequence algorithm has been proposed with a test length of \( \lceil \log_2(n+2) \rceil \). The test set is shown in Table 4.2. By applying these test sets to the nets, the stuck-at faults can be detected. If the SRV of a net differs from its STV, then a shorted net
fault resulted. However, these two counting sequence methods have limitations such as aliasing and confounding syndromes. For example, in Table 4.2, a wired-OR short between n2 and n4 will have a syndrome ‘0110’, which is the fault-free response of n6. Therefore, aliasing syndrome resulted. Similarly, a wired-OR short between n3 and n4 is indistinguishable from the wired-OR short between n5, n6 and n7, since both faults would produce the syndrome ‘0111’. This is a confounding syndrome.

The Complement Counting Sequence Algorithm [19] has been proposed to avoid aliasing. This test set consists of a counting sequence and its complement, as shown in Table 4.3. The size of the test set is $2 \times \lceil \log_2(n) \rceil$. Although aliasing syndromes are eliminated, confounding may still occur.
<table>
<thead>
<tr>
<th>Nets</th>
<th>$PTV_1$</th>
<th>$PTV_2$</th>
<th>$PTV_3$</th>
<th>$PTV_4$</th>
<th>$PTV_5$</th>
<th>$PTV_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>n2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>n3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>n4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>n5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>n6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>n7</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>n8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.3 Counting Sequence Plus Complement Test Set

<table>
<thead>
<tr>
<th>Nets</th>
<th>$PTV_1$</th>
<th>$PTV_2$</th>
<th>$PTV_3$</th>
<th>$PTV_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>n2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>n3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>n4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.4 Walking-1 Test Set

<table>
<thead>
<tr>
<th>Nets</th>
<th>$PTV_1$</th>
<th>$PTV_2$</th>
<th>$PTV_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>n2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>n3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>n4</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.5 Walking-0 Test Set

The walking-1 or walking-0 Algorithm [20], [21] can be used to avoid aliasing and confounding. Diagonal independence of this test set guarantees complete detection and diagnosis. Table 4.4 shows the walking-1 test set for $n = 4$ with wired-OR short fault model, while Table 4.5 shows the walking-0 test set with wired-AND short fault model. The drawback of this algorithm however, is that large number of PTVs are generated as a sequence of length $n$ is required.

In order to reduce the number of PTVs for diagnosis, structural testing can be employed. In structural testing, the wiring layout must be specified. The Pin-Adjacency Detection and Diagnosis Algorithm [24] has been proposed, which considered that a short can occur only in a restricted, yet realistic manner in which two nets can be shorted only if they terminate at adjacent pins, or their tracks are adjacent within a predetermined distance. This is referred to as the adjacency fault model. The nets are represented by an adjacency
graph and diagnosis is shown to be equivalent to the well-known graph coloring problem [24]-[27].

4.3 Deficiency in Previous Approaches

It is important to note that all the above mentioned diagnosis approaches assume only wired-AND or wired-OR bridging fault model, no matter the fault involves two nets or multiple nets. However, in a CMOS circuit environment, the fault model for bridging faults is rather complex than the wired-OR or the wired-AND model [28]. This is because a shorted net may settle to an indeterminate voltage level which makes prediction of the resulting logic levels very difficult. Suppose there is a short between two CMOS pin drivers or output buffers, each attempting to drive their respective nets to opposing logic values as shown in Figure 4.2. The logic values obtained by the receivers or input buffers depend on (1) the resistance of the short, (2) the switching threshold of the receivers and (3) the relative drive strength of the pull-up and the pull-down networks of the two drivers.

![Figure 4.2 CMOS Bridging Fault](image-url)
In our boundary-scan test chip implementation, the measured resulting voltage level with such a short is about 2.3V with a power supply of 5V. The input buffers are designed with a switching threshold above 2.5V, thus a logic 0 will be obtained by the receiver and a wired-AND short can be assumed. If a multiple fault involves three adjacent IC pins short together, then the strong-driver short fault model is assumed. Suppose two of the nets are driving a logic 1 and the other is driving a logic 0, then the resulting voltage of the shorted net will be closer to VCC than to GND and the logic 1 will dominate. The converse argument holds when a logic 0 dominates.

Recently, a new ‘n+1’ Algorithm [29] has been proposed to overcome this deficiency. It can completely diagnose any multiple interconnect faults, regardless of the type of bridging faults. The resulting test set $S$ consists of $n$ rows and $n+1$ columns. Let $b_j$ be an element of $S$, where $0 \leq i \leq n-1$, $0 \leq j \leq n$, then $S$ can be formed by

$$b_j = \begin{cases} 0 & \text{for all } i \geq n - j \\ 1 & \text{for all } i < n - j \end{cases}$$

For $n = 4$, $S$ becomes

$$S = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

As the same with the walking test sequence, the drawback of this method is the large number of PTVs. This problem is especially severe if boundary-scan testing is employed, in which each PTV has to be serially scanned into the board under test, resulting in a total test time of $n \times (n+1)$. 
4.4 Proposed Diagnosis Approach

This section will first give the preliminaries for the proposed diagnosis approach, and then both the one-step and two-step diagnosis algorithms are described in detail.

4.4.1 Preliminaries

Given the layout of nets and the criteria for the short faults, the structure of the interconnect can be modeled by using an adjacency graph $G_{ad}$ to reflect all the possible faults under consideration. The adjacency graph is given by $G_{ad} = (V,E)$, where each node in the set $V$ identifies a net and an edge $e_{ij} \in E$ if a short can exist between $n_i$ and $n_j$ as they are adjacent in the layout of the interconnect [30]. For each node $v \in V$, its degree with respect to $G_{ad}$ is defined to be the number of other nodes adjacent to it in $G_{ad}$. Let $D$ be the maximum degree of the nodes in $G_{ad}$. The edge-distance between two nodes in $G_{ad}$ is the number of edges on the shortest edge path between the two nodes. A $k$-coloring of an arbitrary graph $G_{ad} = (V,E)$ is a mapping $f: V \rightarrow \{C_1, C_2, ..., C_k\}$ which assigns a color $C_i$ to each node in such a way that no two adjacent nodes receive the same color [31]. A primary net is defined as the particular net under consideration, while Primary Shorting Nets (PSNs) are the nets which are likely to be shorted to the primary net, i.e., for a node $n_i$ in $G_{ad}$ all the nodes $n_j$ such that $e_{ij} \in E$. The set of Secondary Shorting Nets (SSNs) is defined as the primary shorting nets to the PSNs, but excluding the primary net itself.

The bridging short fault model under consideration follows what has been discussed in the previous section. It is summarized as follows. For a given short fault, if the number of
nets driving a logic 1 is larger than that driving a logic 0, then logic 1 dominates. Similarly, if the number of nets driving a logic 0 is larger than that driving a logic 1, then logic 0 dominates. As a matter of fact, this assumption is valid for most of the CMOS circuits. In the case where half of the shorted nets driving a logic 1 and the other half driving a logic 0, it reduces to a wired-AND short fault model. It is also assumed that both open and short faults do not exist on the same net.

4.4.2 Description of the One-step Diagnosis Algorithm

The basic idea behind the one-step algorithm is to reduce the number of test vectors by exploiting the net adjacency relationships in the layout. A crucial part of the algorithm is to assign test vector to each net in a one-step process such that no aliasing and confounding syndrome exist. Unlike the conventional graph coloring problem [25]-[27], [31] where each node is assigned a color in such a way that no two adjacent nodes receive the same color, we made the problem more restricted in our algorithm in order to diagnosis all the multiple interconnect faults under our fault model.

Basically, the algorithm utilizes the test set from the "n+1" algorithm [29]. Instead of using n+1 PTVs, the number of PTVs can be reduced to k+1, where k is the number of colors used when our algorithm is applied to $G_{ad}$ and k is generally less than n. Each color $C_i$ is first associated with a unique k+1 bits STV $\{b_{k+1}, b_k, ..., b_2, b_1\}$, where $b_j = 0$ when $j \leq i$ and $b_j = 1$ when $j > i$. As a consequence, the net with color $C_1$ is assigned with a STV $\{111...110\}$, while the net with color $C_k$ is assigned with a STV $\{100...000\}$, where $C_k$ denotes the largest color to be assigned. The test set $S$ will be equal to $(STV_1, ..., STV_n)^T$. With this test set, all stuck-at-
faults can be detected since all STVs are different and contain at least one zero and one. Open faults can also be detected as this test set includes PTVs with all zeros and all ones. In addition, every STV contains different number of ones with a covering relationship. We say net a covers net b if and only if the STV for net a contains ones wherever STV for net b has ones. In other words, net with a lower color covers net with a higher color. Due to this covering relationship, two important behaviors with respect to this test set are observed under our fault model.

First, if two nets are wired-AND shorted, the resulting fault syndrome will be the same as the STV of one of the faulty nets that assigned with the higher color. Suppose for $k = 8$, if net with $C_2$ and net with $C_5$ are shorted, then the SRV for both nets will be equal to $\{111111100\} \cap \{111100000\}$, where $\cap$ denotes the wired-AND operation. As a result, the fault syndrome becomes $\{111100000\}$.

Second, for a short involving $m$ nets, $m > 2$, if we have a list contains the colors of these $m$ nets in ascending order, then the resulting fault syndrome will be the same as the STV of the net with the $\lceil m/2 \rceil + 1)$th color in the list. For example in Table 4.6, if nets with colors $C_1$, $C_3$ and $C_4$ are shorted, the syndrome becomes the STV of the net with the second color in

<table>
<thead>
<tr>
<th>Nets</th>
<th>Colors $(k=8)$</th>
<th>STVs if $n1+n2+n3$ are shorted</th>
<th>SRVs if $n1+n2+n3+n4$ are shorted</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>$C_1$</td>
<td>111111110</td>
<td>111110000</td>
</tr>
<tr>
<td>n2</td>
<td>$C_3$</td>
<td>111111100</td>
<td>111110000</td>
</tr>
<tr>
<td>n3</td>
<td>$C_4$</td>
<td>111111000</td>
<td>111110000</td>
</tr>
<tr>
<td>n4</td>
<td>$C_6$</td>
<td>111000000</td>
<td>111000000</td>
</tr>
</tbody>
</table>

Table 4.6 Test Results for Shorted Nets
the list \(\{1 \ 3 \ 4\}\) and is equal to \(\{11111000\}\), i.e., the STV assigned to the net with \(C_3\). Similarly, if nets with colors \(C_1, C_3, C_4\) and \(C_6\) are shorted, the color list is \(\\{1 \ 3 \ 4 \ 6\}\) and the syndrome will then become \(\{111110000\}\).

Based on these two behaviors, we come up with the following two corollaries which form the basic elements in our algorithm.

**Corollary 1** For a graph \(G_{ad}\), color \(C_1\) can be reassigned to a node which has at least an edge-distance of 3 from those nodes already assigned with \(C_1\) such that no syndrome equal to the STV assigned to the net with \(C_1\) exists.

**Proof:** Since any two nodes with \(C_1\) are separated by at least two other nodes with different colors, a short involving \(p\) \((p \geq 2)\) nodes with \(C_1\) contains at least \(p\) other nodes with different colors. By the above test set behavior, no resulting syndrome is equal to the STV of net with \(C_1\). Thus, no aliasing and confounding.

**Corollary 2** For a graph \(G_{ad}\), color \(C_k\) (the largest assigned color) can be reassigned to a node which has at least an edge-distance of 4 from those nodes already assigned with \(C_k\) such that no syndrome equal to the STV assigned to the net with \(C_k\) exists.

**Proof:** Since any two nodes with \(C_k\) are separated by at least three other nodes with different colors, a short involving \(q\) \((q \geq 2)\) nodes with \(C_k\) contains at least \((q + 1)\) other nodes with different colors. By the above test set behavior, the resulting syndrome is not equal to the STV of net with \(C_k\). Even in the case where there are only two nets short, each involving a node with \(C_k\), aliasing and confounding do not exist since there always exists at least one node separating these sets of shorted nodes.
With these two corollaries, the number of colors used can be reduced by reassigning $C_1$ and $C_k$ to the nodes in $G_{ad}$. However, colors between the lowest and the highest color cannot be reassigned since it cannot guarantee aliasing and confounding free due to the complicated short fault model. In addition, although these two corollaries are sufficient for full diagnosis, they may not be necessary.

In some cases, there may be one set of nets which are not shorted to another set of nets under the adjacency fault model. Thus, it is necessary to find (if any) the disjoint graph component $G_{ad}$ from the adjacency list. There are three phases in the one-step algorithm. Phase 1, 2 and 3 are repeated for each connected adjacency graph $G_{ad}$ until all the nets are assigned with colors and the number of colors or test vectors is minimized. The one-step diagnosis algorithm is summarized in Figure 4.3. It is important to note that the one-step algorithm can also be applied when the wired-OR two nets short fault model is assumed, by taking the complement of the STVs in the above test set.

In Phase 1, the nodes in $G_{ad}$ are sorted according to their degrees (see Section 4.4.1). The node with the highest degree will be first assigned with a color and Phase 1 will exit if all nodes have been assigned with colors. For each node, the algorithm will check all its PSNs (Primary Shorting Nets) and SSNs (Secondary Shorting Nets). A node is assigned with $C_1$ if and only if its PSNs and SSNs are not already assigned with $C_1$. Otherwise, the next lowest color is assigned. With this assignment scheme, the edge-distance for every pair of nodes with $C_1$ will be at least equal to 3 after Phase 1. This follows Corollary 1 and no aliasing and confounding exist. The time complexity of Phase 1 is $O(n \times D + n \times D^2) = O(n \times D^2)$. 
The goal of Phase 2 is to reduce the number of colors by assigning each node originally with the highest color $k$ to the next highest color $k-1$, but with the constraint that all the edge-distances between nodes with $C_k$ and nodes with $C_{k-1}$ should be at least equal to 4. If this condition fails, Phase 2 will exit. The edge-distance between such a pair of nodes is computed by the breadth-first search algorithm for shortest path problem [31]. After each successful iteration, the number of colors used is reduced by one, while the number of nodes with the updated highest color $k$ is increased by one. The time complexity of Phase 2 is $O(m \times n^2)$, where $m$ is the total number of edges $|E|$ in $G_{ad}$. In case where $G_{ad}$ is a complete graph, the time complexity becomes $O(n^4)$. However, $m \ll n(n-1)/2$ under our adjacency fault model since $G_{ad}$ is relatively sparse as $D \ll n$.

The number of colors can be further reduced by going through Phase 3 of the algorithm. In Phase 3, the edge-distance between each node with color $C_k$ and node with color other than $C_1$ and $C_{k-1}$, say $C_j$, is computed. $j$ is first selected to be $k-2$. If all the computed edge-distances are at least equal to 4, then nodes with $C_{k-1}$ will be assigned with $C_j$ while nodes with $C_j$ will be assigned with $C_{k-1}$, otherwise, $j$ is reduced by one and the edge-distance computation procedure is processed again. If there is a successful color swapping, the edge-distance among all the nodes with $C_k$ and $C_{k-1}$ will then be at least equal to 4 and node with $C_k$ can be assigned with color $C_{k-1}$. Thus, the number of colors can be reduced by one. The entire procedure is repeated again with the current $j$ and updated $k$ until there is no more node with color other than $C_1$ and $C_{k-1}$ satisfying the above requirements. The time complexity of Phase 3 is $O(m \times n^3)$. 
**Algorithm (1) One-step Diagnosis Algorithm**

**Input:** Adjacency Graph $G_{ad}$

/* Phase 1 */
Let $k = 2$

**Step1.** Sort the adjacency list $D \times |V|$

**Step2.** For $q = D$ down to 1

- select a current uncolored node $w$ that has the degree of $q$

  - Let $X = \{ u \in V : u$ is the PSN of $w$ \}$
  - If $\exists u_i \in X$ such that $f(u_i) = C_1$ then
    - assign color $k$ to $w$
    - $k = k + 1$
  - else
    - foreach $u_i \in X$, let $Y = \{ v \in V : v$ is the PSN of $u_i \}$
      - if $\exists v_j \in Y$ such that $f(v_j) = C_1$ then
        - assign color $k$ to $w$
        - $k = k + 1$ and goto **Step 3**
      - endif
    - end foreach
  - endif
  - if node $w$ is uncolored then
    - assign color $C_1$ to $w$, i.e., $f(w) \leftarrow C_1$
  - endif
endif

**Step3.** If there is any uncolored node, then goto **Step2**; else exit

/* Phase 2 */
Let $V_i = \{ u \in V : f(u) = C_i \}$, $i \geq 1$

**Step1.** For each $u_i \in V_k$

  - find the edge distance $d_i$ with the vertex $v \in V_{k-1}$

**Step2.** If $\min_i \{d_i\} \geq 4$ then

  - for all $u \in V_k, f(u) \leftarrow C_{k-1}$
  - $k = k - 1$
    - if $k > 1$ then goto **Step 1**, else exit

  else exit
endif

/* Phase 3 */
Let $V_j = \{ u \in V : f(u) = C_j \}$, $j \geq 1$

**Step1.** After Phase 2, if $k > 3$ then let $j = k - 2$, else exit

**Step2.** For each $u_i \in V_k$

  - find the edge distance $d_i$ with the vertex $v \in V_j$

**Step3.** If $\min_i \{d_i\} \geq 4$ then

  - swap($j, k-1$) such that
    - for all $u \in V_j, f(u) = C_{k-1}$
    - and for all $u \in V_{k-1}, f(u) = C_j$
    - for all $u \in V_k, f(u) \leftarrow C_{k-1}$
    - $k = k - 1$
    - if $k > 3$ then goto **Step2**, else exit
  - else $j = j - 1$

**Step4.** If $j > 1$ then goto **Step 2**, else exit

---

Figure 4.3 One-step Diagnosis Algorithm
Note that the time complexity of executing Phase 3 appears to be high for a large number of nets under test. But the test generation process is a one-time off-line computation for a specific board or module. Its cost becomes of little concern if the resulted compact test sequence can save the cost in on-line testing. One alternative approach for testing thousands of nets is to execute Phase 1 and Phase 2 only. Test length can be reduced to a reasonable time.

4.4.3 An Example

Consider the adjacency graph $G_{ad}$ as shown in Figure 4.4. There are 12 nodes with $D$ equal to 4. Figure 4.4(a) shows the assigned color associated with each node after executing Phase 1. The highest color $k$ is equal to 10. Node n2 and n3 can be reassigned with color $C_1$ according to our algorithm. In Phase 2, the edge-distance between n11 and n7 is computed, since their edge-distance is equal to 4, then the color of n11 is changed to 9 as shown in Figure 4.4(b). Next, the edge-distances of (n11, n6) and (n7, n6) are computed. However, the edge-distance between n7 and n6 is less than 4, thus Phase 2 exit. In Phase 3, the edge-distances of (n11, n12) and (n7, n12) are both found to be equal to 4, the colors for n12 and n6 are then swapped as shown in Figure 4.4(c). Finally, the colors of n11 and n7 are changed to 8 as shown in Figure 4.4(d). The process stops as the number of colors cannot be further reduced.
4.4.4 Description of the Two-step Diagnosis Algorithm

Instead of diagnosing the set of all interconnect faults using the one-step diagnosis algorithm, we have proposed another algorithm for fault diagnosis based on a two-step process. Two-step diagnosis refers to the process that diagnosis is done by applying two test sequences. The results of the first test sequence is used in generating the second test sequence. Two-step diagnosis approaches have been proposed in [21], [23]. However, the fault model they assumed is inadequate for CMOS circuits. This problem can be solved with our two-step diagnosis algorithm. As the same with one-step diagnosis, the adjacency fault model is assumed. The two-step diagnosis algorithm is shown in Figure 4.5.
In the first step, test set is generated to detect the presence of faults as well as identifying if each individual net is either fault free or faulty. Besides, both open faults and stuck-at-faults can be detected. Conventional graph coloring is applied to $G_{ad}$ such that no two adjacent nodes receive the same color. The STV assignment is the same as that in one-step diagnosis. It has been shown that for an arbitrary graph $G = (V,E)$ with maximum degree $D$, the number of colors used is less than or equal to $(D + 1)$ [30]. Thus, the first step yields a low test length as $D \ll n$ under our adjacency fault model.

With this simple coloring, if there is a short between two adjacent nodes in $G_{ad}$, then one of the nodes will receive the fault syndrome as these two nodes are assigned with different STVs. In other words, if a node receives a syndrome after applying the first test set to the interconnects, then all the edges associated with this node are susceptible to faulty and all the nodes associated with these edges may involve in a short. A fault list is built to store all these nodes. As a result, a edge in $G_{ad}$ is considered to be fault free only when the two nodes associated with this edge are not in the fault list. The remaining task in the first step is to delete all the fault free edges from $G_{ad}$ and find (if any) the disjoint graph components of the new $G_{ad}$.

In the second step, Algorithm (1) (see Figure 4.3) is applied to the resulted graph components for the full diagnosis of interconnect faults. With two-step diagnosis, the test length can be further reduced as $G_{ad}$ is decomposed into a number of disjoint graph components with reduced number of nodes, especially when the fault rate is low.
Algorithm (2) Two-step Diagnosis Algorithm

**Input:** Adjacency Graph $G_{ad}$

1. **Task1.** Sort the adjacency list $D \times |V|$
2. **Task2.** For $q = D$ down to 1,
   - select a current uncolored node $w$ that has the degree of $q$
3. **Task3.** Check all the PSNs of $w$
   - assign the lowest color number to $w$ as possible
4. **Task4.** If there is any uncolored node, then goto Task2
5. **Task5.** Assign STV to each node
   - apply the test and analyze the responds
   - if there is no faulty responds, then exit
6. **Task6.** Let $\{ F \}$ be a list with nodes that are susceptible to faulty
   - foreach faulty node $u \in V$
     - $\{ F \} \leftarrow u$
     - find all $v \in V$ such that $(u, v) \in E$
     - $\{ F \} \leftarrow v$
7. **Task7.** Remove all edges $e_i$ from $G_{ad}$
   - iff $n_i \notin \{ F \}$ and $n_j \notin \{ F \}$
8. **Task8.** Find the disjoint graph components $G_{ad}$
9. **Task9.** For each $G_{ad}$ execute Algorithm (1)

---

Figure 4.5 Two-step Diagnosis Algorithm

### 4.5 Simulation Results

The proposed one-step diagnosis algorithm (see Figure 4.3) was implemented using Cadence SKILL programming language [32]. The source code is listed in Appendix C. The program is tested on a set of real PCB layouts and a MCM routing benchmark\(^1\).

Table 4.7 outlines the characteristics of the PCB layouts whose adjacency graphs\(^2\) are known. Each board layout contains one power and one ground net, for which no test vectors

---

\(^{1}\)Benchmark released by the Microelectronics and Computer Technology Corporation (MCC).

\(^{2}\)Layout information provided by J.T. de Sousa, EEE, Imperial College, London, UK.
<table>
<thead>
<tr>
<th>Board</th>
<th>no. of components</th>
<th>no. of nets</th>
<th>no. of pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>ir232</td>
<td>24</td>
<td>22</td>
<td>80</td>
</tr>
<tr>
<td>68hc11</td>
<td>54</td>
<td>93</td>
<td>332</td>
</tr>
<tr>
<td>sram8</td>
<td>47</td>
<td>120</td>
<td>680</td>
</tr>
<tr>
<td>cperi24</td>
<td>79</td>
<td>271</td>
<td>952</td>
</tr>
</tbody>
</table>

Table 4.7  Board Examples

<table>
<thead>
<tr>
<th>Example</th>
<th>no. of net</th>
<th>test length of ( n+1 ) algorithm</th>
<th>test length of proposed approach</th>
<th>% reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>ir232</td>
<td>20</td>
<td>21</td>
<td>16</td>
<td>24%</td>
</tr>
<tr>
<td>68hc11</td>
<td>91</td>
<td>92</td>
<td>74</td>
<td>20%</td>
</tr>
<tr>
<td>sram8</td>
<td>118</td>
<td>119</td>
<td>92</td>
<td>23%</td>
</tr>
<tr>
<td>cperi24</td>
<td>269</td>
<td>270</td>
<td>227</td>
<td>16%</td>
</tr>
<tr>
<td>MCC1</td>
<td>799</td>
<td>800</td>
<td>581</td>
<td>27%</td>
</tr>
</tbody>
</table>

Table 4.8  Simulation Results for Benchmark Layouts

are needed. The MCM routing benchmark MCC1 consists of 6 chips, 765 I/O pins, 799 signal nets, two power and one ground net. There are 2496 pins total, 2043 of which are signal pins. There are numerous 3 to 7 pin nets. The adjacency graph \( G_{ad} \) for this benchmark is generated by the assumption that two nets are shorted only if they terminate at adjacent pins. The simulation results are shown in Table 4.8. Note that the power and ground nets are excluded in the test vector generation. The proposed one-step diagnosis approach has also been evaluated by simulation on different size random interconnects with varying maximum degree \( D \). For example, referring to Table 4.9, \( r_{100} \_4 \) is a randomly generated adjacency graph with 100 nets and \( D \) equal to 4, while \( r_{200} \_8 \) contains 200 nets and \( D \) equal to 8. Table 4.9 shows the simulation results.

For the PCB layouts and the MCM routing benchmark, the percentage reduction of the test length are less than 30%. One major reason is that their adjacency graphs \( G_{ad} \) are
<table>
<thead>
<tr>
<th>Example</th>
<th>test length of ('n+1') algorithm</th>
<th>test length of proposed approach</th>
<th>% reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>r100_4</td>
<td>101</td>
<td>56</td>
<td>45%</td>
</tr>
<tr>
<td>r200_4</td>
<td>201</td>
<td>118</td>
<td>41%</td>
</tr>
<tr>
<td>r500_4</td>
<td>501</td>
<td>277</td>
<td>45%</td>
</tr>
<tr>
<td>r1000_4</td>
<td>1001</td>
<td>771</td>
<td>23%</td>
</tr>
<tr>
<td>r100_8</td>
<td>101</td>
<td>83</td>
<td>18%</td>
</tr>
<tr>
<td>r200_8</td>
<td>201</td>
<td>164</td>
<td>18%</td>
</tr>
<tr>
<td>r500_8</td>
<td>501</td>
<td>406</td>
<td>19%</td>
</tr>
<tr>
<td>r1000_8</td>
<td>1001</td>
<td>813</td>
<td>19%</td>
</tr>
</tbody>
</table>

Table 4.9 Simulation Results for Random Interconnects

relatively dense with high number of edges and high \(D\). For randomly generated adjacency graph with \(D=4\), the test length can be reduced by more than 40%. However, for random \(G_{ad}\) with \(D=8\), they show a relatively lower percentage reduction. This is due to the fact that in general, adjacency graph constructed with low \(D\) is relatively sparse as compared to that with high \(D\). Moreover, the execution time is different. As a result, the adjacency fault model has a large impact on the complexity of the proposed approach and the generated test sequence length.

To evaluate the proposed two-step diagnosis algorithm (see Figure 4.5), we perform the simulations by randomly injecting faults to the four PCB layouts and random interconnects. The short fault rate is defined as the probability of the existence of a short fault for an edge in the adjacency graph \(G_{ad}\). Figure 4.6 and Figure 4.7 show the simulation results for the test length versus short fault rate for the four PCB layouts and the random interconnects respectively. Note that the number of tests for detection required for ir232, 68hc11, sram8 and cperi24 in the first step of the two-step diagnosis are 4, 7, 5 and 8 respectively.
Figure 4.6 Test Length vs. Short Fault Rate in Two-step Diagnosis for PCB Layouts

Figure 4.7 Test Length vs. Short Fault Rate in Two-step Diagnosis for Random Interconnects
The following conclusions can be drawn from the simulation results. The number of tests using the proposed two-step process shows a direct dependency with the short fault rate for diagnosis. With a short fault rate above 20%, it is shown that the test length for all the PCB layouts and the random interconnects with D equal to 8 are almost equal to that obtained by one-step diagnosis. Thus, two-step diagnosis is not necessary in this case for test length reduction. However, when the short fault rate is very low (<5%), at least a 50% reduction in the test length compared with [29] can be achieved. It can be shown in Figure 4.7 that random interconnects with different D exhibit different characteristics. Simulation results show that a 50% reduction in the test length can be achieved with a short fault rate of as high as 60% for random interconnects with D=4. Again, this is related to the fact that random G_{ad} with low D is relatively sparse as compared to that with high D. As a result, in addition to the short fault rate, the adjacency fault model affects the performance of the two-step diagnosis process.

Table 4.10 summarizes the comparison for our diagnosis approach with previous algorithms. From this table, it can be seen that our approach provides both detection as well as diagnosis, while avoiding aliasing and confounding. Compared with Park's algorithm [29], our approach has the advantage of using layout information to reduce the test length for full diagnosis of the interconnect. On the other hand, structural diagnosis approaches proposed in [24], [25] produce shorter test sets than the behavioral test sets. However, the short fault model they assumed is inadequate for CMOS circuits.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Detection</th>
<th>Diagnosis</th>
<th>Aliasing</th>
<th>Confounding</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kautz [18]</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Guarantee detection only</td>
</tr>
<tr>
<td>Wagner [19]</td>
<td>Yes</td>
<td>Partial</td>
<td>No</td>
<td>Yes</td>
<td>Guarantee detection only</td>
</tr>
<tr>
<td>Walking Sequence [20]</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Inadequate short fault model</td>
</tr>
<tr>
<td>Self-Diagnosis [23]</td>
<td>Yes</td>
<td>Partial</td>
<td>No</td>
<td>Yes</td>
<td>Guarantee detection only</td>
</tr>
<tr>
<td>Pin-Adjacency Method [24]</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Inadequate short fault model</td>
</tr>
<tr>
<td>Feng, Huang, and Lombardi [25]</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Inadequate short fault model</td>
</tr>
<tr>
<td>Park [29]</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Long test sequence length</td>
</tr>
<tr>
<td>Our Approach</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Complete diagnosis with reduced test length; Applicable to both one-step and two-step diagnosis</td>
</tr>
</tbody>
</table>

Table 4.10 Comparison of Diagnosis Approaches

4.6 Summary

We have described a new approach for test vector generation and diagnosis of interconnects with boundary-scan testing. By considering a more complex bridging short fault model in CMOS circuit environment, the test set is generated using graph coloring technique based on the net adjacency relationships in the layout. Our proposed structural approach guarantees the complete diagnosis of multiple interconnect faults and is applicable to both one-step and two-step diagnosis. Simulation results show that the two-step diagnosis approach performs very well when the short fault rate is very small. Besides, for a board in a matured product line, the number of possible primary shorting nets for each net can be reduced under the adjacency fault model. As a result, the test length can be further decreased. This can save the diagnosis cost for boundary-scan testing.
Chapter 5

Board-Level Test System Development

With the emerging trends in physical interconnect and packaging technologies, functional testing and in-circuit testing using traditional automatic test equipment (ATE) become impractical. As boundary-scan test became the new approach to PCB or MCM testing, test and diagnosis tools must be adapted to support this DFT technique. This chapter describes a boundary-scan test and diagnosis system that has been developed for detecting and locating defective board interconnects. The test strategy is first discussed in Section 5.1. An overview of the test system is given in Section 5.2, while major phases in the system flow including test pattern generation, test execution and fault diagnosis are presented in separate sections. Section 5.6 describes the implementation of a test module for evaluating the test system. Limitations of the test system as well as further enhancements that can be made are discussed in Section 5.7. This chapter concludes with a brief summary in Section 5.8.
5.1 Boundary-Scan Test Strategy

An efficient test scheme is necessary for boundary-scan testing in order to achieve a high fault coverage as well as a low defect level for a board or module containing a number of chips. A boundary-scan test strategy, which is incorporated in our developed boundary-scan test and diagnosis system, is described in this section. This strategy is generic in the sense that it can apply to any boundary-scan test system.

5.1.1 Testing the Integrity of the BST Chain

We are using the connectivity of the board and the boundary-scan infrastructure to test the integrity of the board interconnects. In particular, the integrity of the boundary-scan test (BST) chain should be tested first. As was discussed in Chapter 2, a simple BST chain is a collection of BST ICs with common TCK, TMS and TRST* signals, and with their shift paths linked together by connecting a TDO pin of one IC to the TDI pin of the next IC.

In order to test the BST infrastructure, a design independent test sequence is employed [33]. This test sequence can be generated for any board that has BST capabilities, regardless of what kind of BST infrastructure is used. The main aim of the integrity test is to have a fast and easy means to check the BST infrastructure and testability. It is not our intention to fully test the BST functionality inside the BST ICs, as they are expected to be tested already. The integrity test sequence consists of three steps. Each successive step relies on a successful completion of the previous step. The three steps are: (1) Reset, (2) Instruction register shift, and (3) Identification register shift. Each step is described as follows.
Reset Starting a test with a reset for the test logic is necessary because not all ICs have an internal or external power up reset facility. The test prescribed in IEEE Std 1149.1 [1] is performed, for which TMS is held high for at least five TCK cycles (see Section 2.3.2). If a TRST* line is provided, the highest coverage for reset test is obtained by combining two tests: the TRST* line kept at logic 0 while resetting according to the 'TMS high' method. After application of this first step, it is assumed that no faults are detected. Then the next step can be taken.

Instruction Register Shift The main objective of this test step is to check the integrity of the external test data path connecting all the BST ICs and the internal test data path going through the instruction register of each BST IC. If this second step is successfully completed, then it can be concluded that the TCK, TMS, TDI and TDO nets are connected properly. IEEE Std 1149.1 specifies that the two least significant instruction register cells shall load a fixed binary "01" pattern in the Capture-IR controller state (see Section 2.3.3), such that each IC causes TDO to toggle to both logic states in the Shift-IR controller state. As a result, open circuit fault occurred on the TDO-TDI connecting net can be detected. Besides, the first TDI connection as seen from the tester to the board is tested by shifting in an addition "10" string. These two bits are deliberately made the opposite of the mandatory instruction capture pattern, they will be shifted out again in the TDO signal. Assuming that three ICs are connected in series and have a five bit instruction register (IR), then the expected values shifted out of TDO are:

10xxx01xxx01xxx01

(the rightmost bit is the LSB and is shifted out first, x = don’t care).
In addition, it is shown in [17] that a short between the TDI and TDO pin of any BST IC may remain undetected after the instruction register shift operation. A possible algorithm to check such a short between TDI and TDO pin of an IC is to provide a trailing flag that is different from all IR contents. In this work, an additional sequence "...11110000..." is being shifted in. The numbers of ones and zeros are equal and are the same as the longest IR in the BST chain on the board. So for the longest IR of length L, a sequence of 2L bits is needed.

**Identification Register Shift** Shifting out all identification codes provides information on the position and type of the ICs in the BST chain. According to the standard, those ICs that have no IDCODE instruction implemented will shift out a "0", coming from the bypass register (see Section 3.2.2). Note that the test results of this step detect most of the faults that appeared in the previous test step.

### 5.1.2 Interconnect Test

After the BST chain verification, testing of board-level interconnects can proceed by the use of EXTEST instruction (see Section 3.2.1). The flow of data through the boundary-scan register cells (BSCs) while the EXTEST instruction is selected is shown by the bold paths in Figure 5.1. Test patterns are shifted into the BSCs at component output pins and driven into the board-level interconnects by setting their Mode port to logic 1. The responses that arrive at chip input pins are loaded into their BSCs with ShiftDR set to 0 and shifted out for examination with ShiftDR set to 1 (see Section 2.3.4). By using the test patterns generated by the diagnosis algorithms proposed in Chapter 4, the board interconnects can be tested for
opens, shorts and stuck-at faults. Besides, defective interconnects can also be located. Figure 5.2 shows an example circuit that contains a short-to-ground fault, an open circuit fault and a wired-OR short fault in the board interconnect. It is assumed that the system input pins are equipped with pull-up circuitry. Table 5.1 shows some test vectors for these faults. Note that the rightmost bit of the data values is shifted into the serial input, or out of the serial output, first. Bold type is used to highlight the output data that are changed by the faults.
<table>
<thead>
<tr>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1x0x11xxxxxx</td>
<td>xxxxxxx1x10x1</td>
<td>xxxxxxx1x11x0</td>
</tr>
<tr>
<td>x0x1x00xxxxxx</td>
<td>xxxxxxx0x01x0</td>
<td>xxxxxxx1x11x0</td>
</tr>
</tbody>
</table>

Table 5.1 Example Tests for Interconnect Faults

The following steps describe the basic test procedure for interconnect testing [16]. Note that in this description, the "stimulus pattern" is applied to the component driver pins and the "response pattern" is received at the component input pins. This procedure forms an essential part in the test strategy and the subsequent test pattern generation process.

**Test Procedure for Interconnect Testing**

Step 1: Initialize the TAP to the *Test-Logic-Reset* controller state.

Step 2: Load the instruction register (IR) with `SAMPLE/PRELOAD` instruction. This makes the boundary-scan register (BSR) connected between TDI and TDO.

Step 3: Shift the first stimulus pattern into the BSR.

Step 4: Load the IR with `EXTEST` instruction. This puts the BSR between TDI and TDO. Upon passing the *Update-IR* controller state, the Mode ports in the BSCs are set to 1, this applies (writes) the first stimulus pattern.

Step 5: Capture (read) the response pattern into the shift portion of the BSCs.

Step 6: Shift the captured response pattern out while shifting in the next stimulus pattern.

Step 7: Update (write) the next stimulus pattern.

Step 8: If the last stimulus pattern has been written, then go to Step 9, otherwise go to Step 5.

Step 9: Capture (read) the last response pattern.

Step 10: Shift in a "safe" pattern while shifting out the last captured response pattern.

Step 11: Update (write) the "safe" pattern.

Step 12: Go to the *Test-Logic-Reset* controller state.
5.1.3 IC Internal Test

In addition to the interconnect test, the test and diagnosis system also supports IC internal test by executing the \textit{INTEST} instruction (see Section 3.2.1). This instruction allows static testing of the on-chip system logic after the component is mounted on the board. Following this instruction, test stimuli are shifted in one at a time and applied to the system logic. The flow of data through the BSCs while the \textit{INTEST} instruction is selected is shown by the bold paths in Figure 5.3. Test vectors are shifted into the boundary-scan path and applied to the system logic by setting the Mode port signals for the BSCs at input pins (i.e., pins that drive into the system logic) to logic 1. The result is then loaded into the BSCs at the output pins with ShiftDR set to 0 and shifted out for examination with ShiftDR set to 1. For example, Figure 5.4 shows a simple BST IC that contains a NOR gate, test vectors is shown in Table 5.2. Note that the rightmost bit of the data values is shifted into the serial input, or out of the serial output, first.

For IC internal test, the same test procedure as described for interconnect testing is applicable, but with the following modifications:

(1) The "stimulus pattern" is being written to the system logic inputs rather than to the component output pins.

(2) The captured (read) response is that of the system logic outputs.

(3) \textit{INTEST} is loaded instead of \textit{EXTEST} in step 4.

(4) All stimulus and response patterns are organized with respect to the system logic I/Os rather than the component I/O pins.
Figure 5.3 Test Data Flow while the INTEST Instruction is Selected

Figure 5.4 Testing On-Chip System Logic

<table>
<thead>
<tr>
<th>Input</th>
<th>Expected Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>x00xxxxx</td>
<td>xxxxx1xx</td>
</tr>
<tr>
<td>x01xxxxx</td>
<td>xxxxx0xx</td>
</tr>
<tr>
<td>x10xxxxx</td>
<td>xxxxx0xx</td>
</tr>
<tr>
<td>x11xxxxx</td>
<td>xxxxx0xx</td>
</tr>
</tbody>
</table>

Table 5.2 Example Tests for the NOR Gate
Note that this test procedure applies only to one chip on the board at a time. However, each chip can be tested consecutively by repeating the test procedure. In case where each chip is tested individually, all other chips are configured in bypass mode in order to save the test time (see Section 3.2.1).

5.2 Test and Diagnosis System Overview

The introduction of IEEE Std 1149.1 gives test engineer the potential to test assemblies for structural faults in circuits that lack physical access to internal nets. However, the standard only provides specifications at the chip level, no specific information is given as to how this approach is used for board or system test, particularly fault diagnosis. In order to fully utilize the benefits of the standard, it becomes imperative to implement a high performance board test and diagnosis system at low-cost [34]-[37].

In this work, we have developed our own test and diagnosis system instead of using those commercially available ones for boundary-scan testing [38], [39]. The Boundary-Scan Test and Diagnosis System, BTDS for short, provides a low-cost solution to detect and locate defective chips and interconnects on a board or module. However, due to limited resources, BTDS cannot handle loaded printed circuit boards with large number of interconnects and ICs. So we will emphasize on the discussion of testing module with a fairly simple configuration. We will use the term “module” in place of “board” for the discussion in the rest of this chapter. It is also assumed that the module under test (MUT) is digital and each chip on the module has been designed according to our implemented IEEE Std 1149.1
architecture as specified in Chapter 3.

Although a certain level of manual interaction is unavoidable, the system is made as automated as possible. Basically, test patterns for the BST chain and the module’s interconnect are generated automatically based on a description of the components forming the module and their interconnects. The diagnosis of faults is also performed automatically based on the measured circuit responses to the tests applied.

Figure 5.5 illustrates the flow diagram of BTDS. The BTDS flow can be partitioned into three phases. They are:

- the test pattern generation
- the test execution phase in which the MUT is being tested
- the result analysis and fault diagnosis

Note that except for the test execution phase in which on-line testing is performed, the other two phases are off-line processes. Each phase is described in detail, both in its software and hardware implementation, in the subsequent sections. The source codes for test pattern generation and fault diagnosis as well as other associated procedures were written in Cadence SKILL programming language [32].
Figure 5.5 Flow Diagram of BTDS
5.3 Test Pattern Generation

The test pattern generation (TPG) process is implemented with the Cadence Simulation and Test Language (STL) [40], which is based on the Cadence SKILL language. One advantage of using this tool is that its Verilog code generation module can automatically generate stimulus file in Verilog format for subsequent logic simulation. With STL, the stimulus patterns are generated in a sequential manner in BTDS, for which each stimulus pattern represents the test to be applied in one TCK cycle and contains the TDI, TMS and TRST* input signals. In brief, the TPG process focuses on providing test stimulus patterns, and expected responses and diagnostic data for BTDS. The corresponding test generation program is written based on the boundary-scan test strategy as described in Section 5.1. As a result, BTDS supports three test modes: (1) BST chain integrity test, (2) Interconnect test, and (3) IC internal test. Dependent on the complexity of the MUT, these three modes may not be executed in one test step. It is due to the limited resources of the tester, which will be discussed in later section. Thus, the test mode should be specified as the input to the TPG process. For instance, the first test to be applied may include BST chain integrity test and interconnect test, while the second test is solely for IC internal test.

5.3.1 CAD Database

In addition to the test mode selection, a CAD database is also required in BTDS. The data required to feed the TPG process include module netlists, parts list, chip-level boundary-scan specifications, and information on how the module-level scan connectivity is implemented [36], [37]. In particular, the following information must be provided by the user as
the standard input to the STL program:

- reference designator for each BST device
- pin type (input or output)
- number of I/Os
- pin name for each BST device
- network number associated with each pin (with boundary-scan cell)
- scan order for both BST devices and module

Besides, the adjacency graph $G_{ad}$ (see Chapter 4) should be present prior to the test generation process for generating interconnect test patterns for the MUT. Moreover, the module netlist in Verilog format is required for subsequent logic simulation. It is assumed the Verilog file for each BST IC has been already given, either from the library or vendor, only the module-level interconnect description is needed.

### 5.3.2 Generation of Test Stimulus Patterns

According to the user input and the information from the CAD database, the *stimulus patterns* as defined in the test procedure for interconnect testing (see Section 5.1.2) and IC internal testing (see Section 5.1.3) are generated.

For interconnect testing, the test set is generated based on the proposed one-step diagnosis algorithm using the adjacency graph $G_{ad}$ (see Chapter 4). In case where there is no adjacency graph provided, the default ‘$n+1’ algorithm (see Section 4.3) would be used for test generation and diagnosis. Sequential Test Vector (STV) is assigned to each pin that
drives an module interconnect. The STVs are then transposed into Parallel Test Vectors (PTVs). Each PTV corresponds to one stimulus pattern and the test pattern sequences are generated by the STL according to the test procedure as described in Section 5.1.2, as well as by taking care of the bit position of network in the boundary-scan chain.

BTDS allows user to specify which BST device to perform internal test. In this case, all other devices are configured in bypass mode. Note that BTDS only supports BST ICs with combinational system logic, no sequential system logic is allowed. Thus, an exhaustive test with $2^n$ PTVs is used for IC internal test, where $n$ is the number of system input pins of the device under test. However, it becomes impractical for large $n$. If $n > 8$, BTDS randomly generates 256 PTVs for the test. The remaining task is the same as that for interconnect testing, but following the test procedure as described in Section 5.1.3.

Note that at this point in the TPG process, the stimulus patterns have been serialized and they reflect the boundary-scan chain as specified in the CAD database. Moreover, the resulting test pattern sequences integrate the appropriate IEEE Std 1149.1 protocol as defined by the test. The user commands used for generating the sequences are listed below:

```c
async_reset()
/* asynchronous reset */

chain_integrity()
/* testing the integrity of BST chain */

bypass_all()
/* select the bypass registers for all BST devices */
```
extest()
/* interconnect testing */

intest()
/* IC internal testing */

5.3.3 Logic Simulation

Based on the module netlist from the CAD database, logic simulation using the Verilog-XL\textsuperscript{®} logic simulator is carried out. In brief, the stimulus patterns generated in the preceding test generation step are simulated against the module netlist to produce good responses and diagnostic data. These expected vectors are necessary not only in the fault analysis and diagnosis but also for the Go/No-Go test by comparing with the acquired responses during on-line testing.

5.4 Test Execution

After the TPG process, the resulting ASCII stimulus patterns file and expected vectors from logic simulation are compiled into tester-compatible format. Basically, BTDS uses a PC-based tester for test execution. The test hardware consists of a simple PC connected to the Logic Master ST tester. The Logic Master ST is an interactive test and verification system for digital designs and is manufactured by the Integrated Measurement Systems, Inc. (IMS) [41]. The translated test data file is then uploaded to the host PC and the MUT is connected to the tester via a socket card for boundary-scan testing. Actually, the test data file contains sequences of stimulus pattern, for which each sequence corresponds to a slice of time. The
duration of the sequence is controlled by the System Clock defined in the Logic Master ST operating setting. For BTDS, this system clock is defined to be the TCK signal.

The installed host interface package is first invoked. Then, after assigning the tester channels and setting up the operating conditions, the test sequences are applied to the MUT by issuing appropriate commands through the host PC. During test execution, the Logic Master ST sends verification data to the MUT and performs real-time comparison of expected data against acquired data. It recognizes differences between expected and acquired data as compare errors, which are highlighted on the PC graphic display for easy recognition. As a result, a Go/No-Go test can be done during on-line testing, or a test results file can be downloaded for off-line fault analysis and diagnosis.

Several comments on using the Logic Master ST test system are made as follows. First, as the Logic Master ST is not originally developed for boundary-scan testing, it does not has the functionality of generating boundary-scan test data and cannot keep trace of the IEEE Std 1149.1 protocol during on-line testing. As a result, serial test data as well as the corresponding control protocol in the tester-compatible format should be prepared in advance so that the tester performs the boundary-scan testing properly. Second, the test sequences applied to the MUT must be synchronized with the operation of the tester, otherwise enormous errors would occur, which make the diagnosis procedure difficult or even impossible. Moreover, errors occur during on-line testing may give a wrong diagnosis due to the improper operating conditions setting, such as timing parameters. Thus, care should be taken during tester setup. However, the use of this currently available test system provides a low-cost alternative to boundary-scan testing.
5.5 Fault Diagnosis

Fault diagnosis is performed by analyzing the test results file downloaded from the host PC. Based on the netlist description present in the CAD database, BTDS can check the integrity of the BST chain as well as detect and locate module interconnect faults including stuck-at-zero, opens and shorts. It also has the capabilities of diagnosing multiple interconnect faults and locating those components that failed the IC internal test. All the mentioned faults are summarized in a diagnostic report. A sample report is shown in Figure 5.6.

Finally it should be noted that the diagnosis algorithm as presented in Chapter 4 assumes only one fault in one net. However, in real situation, combined faults can be existed on the same net. As a result, it may appear that the faults on a net cannot be fully diagnosed in only one test cycle [17]. The only way out is to first repair one or more of the faults, followed by diagnosing the remaining fault causes. Thus a multiple test cycle is required.

<table>
<thead>
<tr>
<th>DIAGNOSTIC REPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of devices: 4</td>
</tr>
<tr>
<td>Number of nets: 17</td>
</tr>
<tr>
<td>Number of BSCs: 48</td>
</tr>
<tr>
<td>Chain integrity test: PASS</td>
</tr>
<tr>
<td>Net stuck-at zero faults: none</td>
</tr>
<tr>
<td>Net open faults: (2)</td>
</tr>
<tr>
<td>(16)</td>
</tr>
<tr>
<td>Net bridging faults: (3 4)</td>
</tr>
<tr>
<td>(6 14 17)</td>
</tr>
<tr>
<td>(9 10 11)</td>
</tr>
<tr>
<td>INTEST for component u1: PASS</td>
</tr>
<tr>
<td>INTEST for component u2: FAIL</td>
</tr>
<tr>
<td>INTEST for component u3: FAIL</td>
</tr>
<tr>
<td>INTEST for component u4: PASS</td>
</tr>
</tbody>
</table>

Figure 5.6 Diagnostic Report
5.6 Module Implementation and Test Application

5.6.1 MUT Description

A simple test module has been implemented to demonstrate the prototype capabilities of BTDS. This evaluation module contains a boundary-scan chain which links four identical BST ICs. These ICs are those designed and implemented as described in Chapter 3 and are packaged in the 40-pin DIP. Since the main objective of building this test module is to evaluate the test and diagnosis capability of BTDS, instead of testing the assembly defects. Thus, the ICs are simply mounted on a board while the interconnections are made by wire-wrapping. Figure 5.7 shows the MUT configuration. Note that I/O pins on the left side of the package are system inputs and those on the right side are system outputs. The primary inputs and primary outputs are deliberately made at the left and right side of the MUT respectively. Besides, the tester has a physical access to the Test Access Port (TAP) interface in order to control the operation of the TAP and to transfer data between the tester and the MUT. For the purpose of this application, the TAP signals and primary I/Os are brought out to the MUT edge connector.

Figure 5.8 shows the photo of the MUT. Note that both external pull-up resistors and input buffers are added to the system input pins of each IC. Since bridging short faults would be injected to the module interconnects, unwanted noise is induced. Thus, the buffers are included in order to increase the drive strength and suppress the noise in those interconnects. Moreover, the interconnects are subjected to large capacitance as they are hard-wired. It should be noted that the inclusion of external input buffers to a BST IC does not violate IEEE Std 1149.1.
Figure 5.7 MUT Configuration

Figure 5.8 Photo of the MUT


```plaintext
declare(u[5])

setplist('u1')(in 6 out 6 inpin (PI1 PI2 PI3 PI4 PI5 PI6)
  outpin (1 2 3 4 5 6)
  IC_scan_order (I1 I2 I3 I4 I5 I6 O6 O5 O4 O3 O2 O1)
)
setplist('u2')(in 6 out 6 inpin (1 2 3 4 17 6)
  outpin (7 PO7 8 9 10 11)
  IC_scan_order (I1 I2 I3 I4 I5 I6 O6 O5 O4 O3 O2 O1)
)
setplist('u3')(in 6 out 6 inpin (7 14 8 9 10 11)
  outpin (12 13 14 15 16 17)
  IC_scan_order (I1 I2 I3 I4 I5 I6 O6 O5 O4 O3 O2 O1)
)
setplist('u4')(in 6 out 6 inpin (12 13 5 15 16 5)
  outpin (PO1 PO2 PO3 PO4 PO5 PO6)
  IC_scan_order (I1 I2 I3 I4 I5 I6 O6 O5 O4 O3 O2 O1)
)
PI = '(PI1 PI2 PI3 PI4 PI5 PI6)
PO = '(PO1 PO2 PO3 PO4 PO5 PO6 PO7)
module_scan_order = list('u1 'u2 'u3 'u4)
for(i 1 4 u[i]=nthelem(i module_scan_order)
```

Figure 5.9 Manually Generated BTDS Input File

Once the module has been fixed, its netlist and scan connectivity are then captured to form the CAD database. For the MUT, the SKILL code statements as shown in Figure 5.9 are included in the BTDS input file. The resulting stimulus patterns and expected vectors are then compiled into a test data file.

5.6.2 Tester Setup

Prior to loading the test data file and executing the IEEE Std 1149.1 protocol to perform boundary-scan testing, the appropriate Logic Master ST parameters must be assigned before the correct responses can be obtained. These parameters include power supply levels,
logic levels and timing values, and the assignment of tester scan resources to the appropriate MUT signals. In particular, the timing parameters should be set properly for data scanning. As was discussed in Section 5.4, each stimulus pattern represents the test to be applied in one TCK cycle. Thus, the TCK is set to a RZ (return-to-zero) data format with a 50% duty cycle. And the test is run with a TCK frequency of 10MHz. On the other hand, the TDI, TMS, TRST* and the primary input signals are in a NRZ (non-return-to-zero) data format. Figure 5.10 illustrates the timing waveforms for the applied TAP signals and the acquired TDO signal for a sample test sequence.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>TCK</th>
<th>TRST*</th>
<th>TMS</th>
<th>TDI</th>
<th>TDO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>z</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>z</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>z</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>z</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>z</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>z</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>z</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

![Timing Diagram for a Sample Test Sequence](image-url)

Figure 5.10  Timing Diagram for a Sample Test Sequence
From the timing diagram, it can be seen that the signals applied to TMS and TDI change state on the falling edge of TCK. This guarantees that the setup and hold requirements of TMS and TDI can be met as they are sampled into the test logic on the rising edge of TCK. On the other hand, the standard defines that the TDO change state on the falling edge of TCK. As a result, the sample time for this signal is deliberately made at the rising edge of TCK in order to fulfill the timing requirements. Finally, it should be noted that the test logic of each BST IC can also be reset by asserting the TRST* irrespective of the TCK.

5.6.3 Results

The test module has been successfully tested with BTDS at the first step. All the mentioned test functions have been applied (see Section 5.1) and no errors occurred during on-line testing. Next, interconnect faults including opens and shorts are injected to the MUT using switches (see Figure 5.8). Results show that all these faults can be diagnosed accurately and indicated in the diagnostic report. We have also demonstrated the diagnosis capability for IC internal testing by altering several logic values of the expected vectors in the test data file. This is done because all the BST ICs have been already proved functioning properly, so it would give the correct responses.

5.7 Limitations of BTDS

This section discusses several limitations of BTDS with respect to the CAD/CAE environment, hardware configuration and system integration. Suggested further enhance-
ments are also given. First of all, the major difference between BTDS and other commercially available boundary-scan test systems is that BTDS supports only BST ICs with the specific IEEE Std 1149.1 architecture as discussed in Chapter 3. Although the number of I/O signal pins can be varied, they must be 2-state unidirectional ports. Besides, BTDS supports only four instruction codes, other testability features like built-in-self-test (BIST) and user-defined instructions are not included. In other words, BTDS is intentionally developed for testing and diagnosing board or module built with our implemented BST chip. However, this basic IEEE Std 1149.1 chip-level requirement and test system development are sufficient for providing a low-cost test platform for testing the advanced packaging and assembly processes like direct-chip-attach (DCA) or chip-on-board (COB) technology, which are actively under research in HKUST. In particular, the solder joint integrity can be tested using the boundary-scan test methodology. Discussion on testing advanced packaging is contained in Chapter 6.

In view of the CAD/CAE environment, the module netlist and scan connectivity information have to be manually provided by the user. This implies that the test engineer should have the knowledge about the circuit designs. Moreover, it is a error-prone process as the number of module interconnects could become very large. Additional development can be done to ensure the completeness and integrity of CAD data, particularly module netlists and scan data within the chip specifications. By providing correct and complete CAD data in a standard format, the TPG and fault diagnosis process would become significantly more automated. One possibility is to use the Boundary-Scan Description Language (BSDL) to describe the testability features of the BST devices [16], [42]. BSDL is written within a subset of VHDL and it can be complied using a VHDL analyzer and the resulting logic can be synthesized. In addition, the module netlists can be extracted automatically with the use of
existing CAD tools and translated into standard EDIF file.

The resources limitation posed by the Logic Master ST seems to affect the most to the performance of BTDS. The currently available PC-based tester can only support 4096 test pattern sequences during one test cycle. It becomes insufficient when modules with over hundreds of nets are being tested. The Logic Master ST was originally developed for conventional IC testing, in which parallel test patterns are applied. However, IEEE Std 1149.1 requires these patterns to be serialized, merged with other control protocol and then shifted into the MUT. It is also important to note that as the number of scan locations used by a test increases, the number of parallel test vectors generated may not increase, but the serial vector length may increase resulting in potentially greater execution time and data requirements. In this situation, the number of interconnects to be tested can be reduced or the test is separated into several test steps as discussed in Section 5.3. Additionally, the tester is a low-speed one, although we are not concerned about the speed of the test in this work.

As mentioned in Section 5.2, the MUT must be designed with BST ICs conforming to IEEE Std 1149.1. In other words, BTDS cannot handle BST and non-BST devices on the same module. However, due to cost considerations, it is believed that for many years PCBs or MCMs will contain both boundary-scan and conventional components without BST facilities. This requires enhancements to the TPG and fault diagnosis process [17], [43], [44]. As a result, a hybrid test strategy is needed for such a mixed-technology module. However, this is beyond the scope of this work.

Finally, there is limitation in performing the test in one operating system environment. Currently in BTDS, off-line TPG and fault diagnosis are done in a workstation
environment. The test data is transferred to and from the Logic Master ST in each test cycle. This becomes inefficient and time-consuming when there are changes on the configuration of MUT or the test functions. As a suggestion of future work, the PC-based tester can be replaced with a workstation-based one and BTDS would then be migrated to a workstation-based application development platform, in which all the necessary CAD tools, TPG and fault diagnosis software, and the tester interface package are included. As compared to the PC-based tester, much more test pattern sequences can be stored for test execution in the workstation-based test system.

5.8 Summary

We have developed a board-level Boundary-Scan Test and Diagnosis System. We successfully demonstrated its capability with a test module. Based on the description of the components forming the module and their interconnects, test stimulus patterns are generated automatically. A complete diagnosis of multiple interconnect faults can be achieved by using our proposed one-step diagnosis algorithm. We use an existing PC-based tester for test execution. In view of fault analysis, BTDS provides not only an immediate pass/fail response but also storage of test results data for more detailed diagnosis. Although BTDS is not developed as a generalized test system and has several limitations, it does provide a low-cost test platform for testing and diagnosing assembly defects in advanced packaging, such as flip-chip-on-board.
Chapter 6

Application to Multichip Module Testing

The introduction of IEEE Std 1149.1 has led to many researches on the application of boundary-scan testing to multichip modules (MCMs). This offers a cost effective approach for fault detection and diagnosis of manufacturing defects on MCMs. The first section of this chapter gives a brief introduction to the current practice of MCM assembly and testing. The last section presents directions for future work. Based on the materials we have studied and developed in this thesis, future development and extension can be made to the flip chip assembly testing. Structural interconnect integrity can be checked by incorporating the boundary-scan architecture into a test vehicle designed for assembly process evaluation, despite of the limited accessibility posed by the high-density MCMs.
6.1 An Introduction to MCM Assembly and Testing Process

Products motivated by performance-driven and/or density-driven goals have started to use MCM technology, which offers the ability to place multiple integrated circuit chips upon a single substrate. MCMs offer many advantages, including size and weight reduction, an increase in reliability, and increased performance [45]-[47]. However, MCM technology still has several challenging problems such as to achieve acceptable MCM assembly yields and to meet product quality requirements. Both of these problems can be reduced by adopting adequate testing approaches throughout the MCM fabrication process. One such approach is the Boundary-Scan testing.

A typical MCM production flow can be divided into four major processes. They are: (1) the process of fabricating the wafer, (2) the production of individual bare dies, (3) the fabrication of substrates, and (4) the assembly of bare dies and substrates to compose MCMs. As shown in Figure 6.1, MCM testing takes place throughout the production process including bare die test, substrate test and assembled module test and rework [3]. Moreover, the assembled MCM test and diagnosis procedure contains an interconnect test, a full functional test and a performance test. In the remainder of this section, we will emphasize on the description of MCM assembly process and assembled module interconnect testing.

Although we are concentrating on the study of assembled module interconnect test and diagnosis in this thesis, both the bare die test and substrate test are important [3]. If a bare die is found defective after assembly onto the MCM substrate, either the substrate is scrapped with the rest of the chips, or the MCM is repaired by removing the bad chips and replacing them with good ones. Both alternatives are expensive and undesirable. On the other hand, a
substrate defect hurts MCM yield and substrate repair is often not possible and components are sometimes damaged during the removal process. Thus, failure to detect a substrate defect can be extremely expensive.

Figure 6.1 MCM Production Flow
The tested dies are then mounted and bonded to the tested substrate to form the MCM. Known good dies (KGD) and known good substrate (KGS) are used for MCM assembly to provide high module yield. MCMs today consist of single packages containing multiple bare chips and/or discrete components built with different configurations and sizes connected to substrates using different types of chip-to-substrate attachment techniques, or chip-on-board (COB) techniques. Typically, there are three such methods including tape automated bonding (TAB), wire bonding, and flip chip attachment [45], [47]. These are illustrated in Figure 6.2 and are briefly described as follows:

- **Tape automated bonding** — whereby bare chips are supplied on a tape carrier on which the I/O pads of the individual chips are prebonded to a flexible lead frame structure which is transferred and bonded to the substrate.

- **Wire bonding** — whereby fine wires bond the bare chip I/O pads to the substrate in much the same way as a chip is connected to the lead frame in a packaged chip.

- **Flip chip** — whereby bare chip I/O pads are bumped and the chips are mounted face down on the substrate. Note that the chip bond pads are distributed all over the active area of the bare chip.

![Figure 6.2 The Three Major COB Assembly Techniques](image-url)
Compared between these three COB techniques, the chip-to-substrate interconnects of both TAB and wire bonding take up a certain amount of board space, thus the total area is larger than the chip itself. In contrast, flip chip COB technique mounts chips face down on the substrate, which is an advantage in terms of mounting density. With the use of these packaging technologies, high-performance and high-density MCMs can be manufactured. However, this poses problems for MCM testing after assembly. As a result of the high chip density and small interconnect line dimensions, accessing internal nodes in a module becomes difficult. Moreover, due to the very limited accessibility to the MCM circuitries, it is impossible to test the interconnects and the chips with conventional test techniques, such as in-circuit testing. Additionally, the module level test approach needs to provide diagnostic capabilities in order to isolate the defective components [3].

The above mentioned problems can be overcome by incorporating boundary-scan architecture in the chip design [3]-[10]. Other testability approaches are also addressed for assembled module test and diagnosis [3], [6]-[9]. In this thesis, we have studied the verification and fault diagnosis of an assembled module's structural interconnects using boundary-scan. It should be noted that interconnect testing not only checks module interconnect continuity but also detects mechanical assembly defects, such as flip chip solder joint integrity. This is important since one of the predominant fault mechanism encountered during production testing of MCMs is the failure of the IC bonds. In the smallest geometry only 30% to 50% yield after assembly is typical, with the failures almost attributed to structural opens and shorts [4]. By following the test strategy and procedures mentioned earlier in Chapter 5, together with the proposed diagnosis algorithms, these structural defects can be detected and isolated.
6.2 Future Work on Flip Chip On Board Assembly Testing

The work so far discussed in this thesis can be extended to Flip Chip On Board (FCOB) assembly testing. The emphasis will be on the structural interconnect integrity testing based on the boundary-scan testing approach discussed in this thesis.

The FCOB process specifically refers to the interconnection of unpackaged integrated circuit chips directly to an organic substrate [48]. This technology offers minimization of board area requirements in conventional, low cost assemblies, as well as a reduction in both weight and height profile. However, in order to develop a reliable and efficient FCOB assembly process, the reliability and performance of the corresponding FCOB interconnect system are critical. As the FCOB assembly under investigation uses solder bumps as the interconnects, a degradation in the structural integrity or other assembly defects of the solder bumps can affect the assembly yield or reliability performance of the FCOB assembly. Moreover, the solder bumps are not easy to inspect [46], [47]. One possible approach for testing and diagnosing such defects, such as solder joint shorts or opens, is to use boundary-scan testing.

6.2.1 Potential Solder Joint Defects [46]

Basically, the solder bumps provide three functions. First, the solder joint is the electrical connection between the chip and the substrate. In some instances, the solder joint may also serve as a path for heat dissipation from the chip. Finally, the solder joint often provides the structural link between the chip and the substrate. As a result, the structural integrity of the solder joint affects both the electrical and thermal performance of the flip chip
interconnect system [47].

Typical solder joint defects that can be tested using boundary-scan are solder bridging and opens. Figure 6.3 illustrates these types of potential solder joint defects. Bridging to the adjacent solder ball is possible when there is too much solder present. Small solder balls caused by material ejected from the solder during reflow can also short two solder bumps. Opens are caused by excessive bond wrap and poor wetting. Poor pad wetting causes the solder to flow up the side of the ball while poor ball wetting prevents the solder from flowing up the ball. Note that other solder joint defects such as voiding, contamination, insufficient strength and misregistration can be inspected by using X-ray laminography or scanned beam laminography [46].

![Figure 6.3 Potential Solder Joint Defects](image-url)
6.2.2 Test Vehicle Design

In order to evaluate the FCOB assembly process and access its reliability. A test vehicle with simple structures can be designed. The test vehicle contains a substrate and test dies with specific test structures [47]. Instead of using conventional test structures like continuity patterns or serpentine and comb patterns, the test structures under consideration include the boundary-scan architecture. Actually, the test chip designed and implemented in this work (see Chapter 3) can be used as the test die for this test vehicle. Figure 6.4 shows an example test vehicle that can be used for testing the solder joint defects as discussed earlier. An example of substrate design for a single layer PCB is shown in Figure 6.4(b). The PCB contains 4 FCOB sites and the signal tracks are all routed to the peripheral for external electrical connections. Other configurations are possible provided that the pad sizes and positions for both the test dies and the substrate are matched. It is also important to note that as the test die is flipped, the I/O signal connections on the PCB should also be adjusted.

![Image of Test Vehicle](image_url)

(a) Test Die  
(b) Substrate

Figure 6.4 An Example Test Vehicle
accordingly. Furthermore, care should be taken when routing the signals on the PCB since boundary-scan requires a common Test Access Port (see Chapter 2) to be connected to all the test dies.

The test to be applied basically follows the methodology as presented in Section 5.6, except that the MUT is replaced by the FCOB module. Test vectors are generated in our developed boundary-scan test system to test the module interconnects as well as the solder joint integrity. To further evaluate the interconnect reliability, the test vehicle is subjected to thermal cycle testing. In this case, the test program can be reused in the test system. It is also important to note that the testing operations will not require sophisticated test equipments. A low cost test platform will be sufficient to perform the assembly defect detection and diagnosis operations. In conclusion, the use of boundary-scan testing as part of the reliability assessment of the FCOB assembly process is cost effective and independent of silicon and substrate technologies.
Chapter 7

Conclusions

Boundary-Scan is a structured design-for-testability technique which can be used to simplify the testing of digital circuits, boards, and systems. In this research, a low-cost Boundary-Scan Test and Diagnosis System (BTDS) has been successfully developed and used to test an evaluation module. The BTDS flow consists of three phases. They are: (1) the test pattern generation, (2) the test execution phase in which the MUT is being tested using a PC-based tester, and (3) the result analysis and fault diagnosis. The evaluation module contains boundary-scan test chips which are complied with IEEE Std 1149.1. These chips are designed using the standard cell-based approach and fabricated using a 1.2μm n-well, two metal layers CMOS process. The test chip integrates 2287 transistors in an area of 2.04mm × 2.04mm and was packaged in a 40-pin DIP.

It is important to note that there are several limitations for BTDS, as it is intentionally developed for testing and diagnosing board or module built with our implemented test chip.
Suggested further enhancements have been given in view of the CAD/CAE environment, hardware configuration and system integration.

In order to have a complete diagnosis of multiple interconnect faults, an efficient structural approach for diagnosing board interconnects has been proposed and included in BTDS. A more complex bridging short fault model in CMOS circuit environment is considered. The diagnostic test set is generated based on graph theoretic technique and the adjacency fault model is adopted. By using the structural information of the wiring layout, the test length can be reduced. Both one-step and two-step diagnosis algorithms are given. They have been evaluated by simulation on several benchmark layouts and randomly generated layouts. Simulation results show that more than 50% reduction in the number of tests can be achieved for two-step diagnosis when the fault rate is very small, such as in a matured product line. This can significantly save the diagnosis cost for boundary-scan testing.

Lastly, future development and extension can be made to the flip-chip assembly testing using our developed boundary-scan test system. The aim is to evaluate the flip-chip-on-board (FCOB) assembly process and access its reliability. A test vehicle incorporated with the boundary-scan architecture can be designed to check the structural interconnect integrity, including both the detection and location of assembly defects such as solder bump opens and shorts. This kind of reliability assessment using boundary-scan testing has the advantage of being silicon and substrate technologies independent.
# Appendix A

## Cell Library Listing

<table>
<thead>
<tr>
<th>Cell Name</th>
<th>Cell Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INV2</td>
<td>Inverter (2x drive)</td>
</tr>
<tr>
<td>ND2X2</td>
<td>2-Input NAND (2x drive)</td>
</tr>
<tr>
<td>ND3</td>
<td>3-Input NAND</td>
</tr>
<tr>
<td>ND4</td>
<td>4-Input NAND</td>
</tr>
<tr>
<td>NR2</td>
<td>2-Input NOR</td>
</tr>
<tr>
<td>BUFF</td>
<td>Buffer</td>
</tr>
<tr>
<td>TRIB2</td>
<td>Tri-State Buffer (2x drive)</td>
</tr>
<tr>
<td>DSEL</td>
<td>2:1 Data Select Multiplexer</td>
</tr>
<tr>
<td>DFF</td>
<td>D - Flip Flop</td>
</tr>
<tr>
<td>DPFR</td>
<td>D - Flip Flop with Asynchronous Reset</td>
</tr>
<tr>
<td>DFFSR</td>
<td>D - Flip Flop with Asynchronous Set, Reset</td>
</tr>
<tr>
<td>PULU</td>
<td>Pull-up Resistor</td>
</tr>
<tr>
<td>PULG</td>
<td>Pull-down Resistor</td>
</tr>
<tr>
<td>X12IPD</td>
<td>Input Pad</td>
</tr>
<tr>
<td>OPAD12</td>
<td>Output Buffer Pad</td>
</tr>
<tr>
<td>X12TRI</td>
<td>Tri-State I/O Pad</td>
</tr>
<tr>
<td>PWR12</td>
<td>Power Pad - Connects to VDD Pad Ring</td>
</tr>
<tr>
<td>GND12</td>
<td>Power Pad - Connects to VSS Pad Ring</td>
</tr>
</tbody>
</table>
Appendix B

Macroccll Synthesized Netlists

B.1 Bypass Register

**VHDL Code**

Library IEEE;
use IEEE.std_logic_1164.all;

ENTITY br_cell IS
  PORT (clockDR, shiftDR, scan_in : IN STD_LOGIC;
        scan_out : OUT STD_LOGIC);
END br_cell;

ARCHITECTURE behavioral OF br_cell IS
  SIGNAL temp : STD_LOGIC;
BEGIN
  temp <= (shiftDR AND scan_in);
  dff : PROCESS (clockDR)
  BEGIN
    IF (clockDR = '1') THEN scan_out <= temp; END IF;
  END PROCESS dff;
END behavioral;

**Gate-Level Netlist**

![Netlist Diagram]

Figure B.1 Netlist of Bypass Register
B.2 Instruction Register Cell

VHDL Code

Library IEEE;
use IEEE.std_logic_1164.all;

ENTITY ir_cell IS
  PORT (shiftIR, data_in, scan_in, clockIR, updateIR, trst_bar,
       reset_bar : IN STD_LOGIC;
       data_out, scan_out : OUT STD_LOGIC);
END ir_cell;

ARCHITECTURE behavioral OF ir_cell IS
  SIGNAL ff1 : STD_LOGIC;
BEGIN
  dff : PROCESS (clockIR)
  BEGIN
    IF (clockIR = '1') THEN
      IF shiftIR = '0' THEN
        ff1 <= data_in;
      ELSE
        ff1 <= scan_in;
      END IF;
    END IF;
  END PROCESS dff;
  scan_out <= ff1;

  dffr : PROCESS (updateIR, trst_bar, reset_bar)
  BEGIN
    IF (trst_bar = '0' or reset_bar = '0') THEN
      data_out <= '1';
    ELSIF (updateIR = '1' AND updateIR'EVENT) THEN
      data_out <= ff1;
    END IF;
  END PROCESS dffr;
END behavioral;

Gate-Level Netlist

![Figure B.2 Netlist of Instruction Register Cell](image-url)
B.3 Boundary-Scan Register Cell

VHDL Code

Library IEEE;
use IEEE.std_logic_1164.all;

ENTITY dr_cell IS
  PORT ( mode, data_in, shiftDR, scan_in, clockDR, updateDR : IN STD_LOGIC;
         data_out, scan_out : OUT STD_LOGIC);
END dr_cell;

ARCHITECTURE behavioral OF dr_cell IS
  SIGNAL ff1, ff2 : STD_LOGIC;
BEGIN
 dff1 : PROCESS (clockDR)
  BEGIN
    IF (clockDR = '1') THEN
      IF shiftDR = '0' THEN
        ff1 <= data_in;
      ELSE
        ff1 <= scan_in;
      END IF;
    END IF;
  END PROCESS dff1;

dff2 : PROCESS (updateDR)
  BEGIN
    IF (updateDR = '1') THEN
      ff2 <= ff1;
    END IF;
  END PROCESS dff2;

  data_out <= data_in WHEN (mode = '0') ELSE ff2;
  scan_out <= ff1;
END behavioral;

Gate-Level Netlist

![Netlist of Boundary-Scan Register Cell](image)

Figure B.3 Netlist of Boundary-Scan Register Cell
B.4 TAP Controller

**VHDL Code**

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY tap_controller IS
  PORT (tms, tck, trst_bar : IN STD_LOGIC;
        reset_bar, select1, enable, shiftR, clockR, updateR,
        shiftDR, clockDR, updateDR : OUT STD_LOGIC);
END tap_controller;

ARCHITECTURE behavioral OF tap_controller IS
  SIGNAL A, B, C, D : STD_LOGIC := '1';
  SIGNAL NA, NB, NC, ND : STD_LOGIC;
BEGIN
  PROCESS (tck, trst_bar)
  BEGIN
    IF (trst_bar='0') THEN
      A <= '1';
      B <= '1';
      C <= '1';
      D <= '1';
    ELSIF ((tck = '1') AND NOT tck'STABLE) THEN
      A <= NA;
      B <= NB;
      C <= NC;
      D <= ND;
    END IF;
  END PROCESS;

  PROCESS (tck, trst_bar)
  BEGIN
    IF (trst_bar='0') THEN
      reset_bar<= '0';
      enable<= '0';
      shiftR<= '0';
      shiftDR<= '0';
    ELSIF ((tck = '0') AND tck'EVEN) THEN
      IF (D and C and B and A) = '1' THEN reset_bar <= '0';
      ELSE reset_bar <= '1';
      END IF;
      IF ((D and not C and B and not A) OR (not D and not C and B and not A)) = '1'
      THEN enable <= '1';
      ELSE enable <= '0';
      END IF;
      IF (D and not C and B and not A) = '1'
      THEN shiftR <= '1';
      ELSE shiftR <= '0';
      END IF;
      IF (not D and not C and B and not A) = '1'
      THEN shiftDR <= '1';
      ELSE shiftDR <= '0';
      END IF;
    END IF;
  END PROCESS;

  ND <= (D and (not C)) or (D and B) or ((not tms) and C and (not B))
       or ((not D) and C and (not B) and (not A));
  NC <= (C and not B) or (C and A) or (tms and not B);
```


NB <= (not tms and B and not A) or (not tms and not C) or (not tms and not D and B) or 
(not tms and not D and not A) or (tms and C and not B) or (tms and D and C and A);
NA <= (not tms and not C and A) or (tms and not B) or (tms and not A) or (tms and D and C);

clockIR <= not (not tck and ((D and C and B and not A) or (D and not C and B and not A)));
updateIR <= not tck and (D and C and not B and A);
clockDR <= not (not tck and ((not D and C and B and not A) or (not D and not C and B and not A)));
updateDR <= not tck and (not D and C and not B and A);

select1 <= D;
END behavioral;

Gate-Level Netlist

Figure B.4 Netlist of TAP Controller
Appendix C

SKILL Program of One-step Diagnosis Algorithm

; This SKILL program is written according to the one-step diagnosis algorithm
; described in Section 4.4.2. The input is the adjacency list of the interconnect
; network under consideration. The output is the color assignment for each node.

start=getCurrentTime()
load("adj_list") ; load adjacency list from file
; OR
; load("random.ii") ; load random graph
degree = makeTable("degree_table" nil)
color = makeTable("color_table" nil)
dis = makeTable("distance_table" nil)

procedure( arrange_degree( n1 n2 )
    cond(
        ( degree[n1] > degree[n2] t )
        ( t nil )
    ); end cond
)

procedure( arrange_number( n1 n2 )
    cond(
        ( n1 > n2 t )
        ( t nil )
    ); end cond
)

/* Sorting the degree of the nodes in descending order */
procedure( SortDegree( aList )
    aList = sort( aList 'arrange_number )
    sort( aList 'arrange_degree )
)
procedure( union( list1 list2 )
    nconc
        setof( element list1
            !member( element list2 )
        ); end setof
    list2
    ); end nconc
)

/* Build disjoint graph components */
procedure( makegraph( table )
    alist = list()
    graph = list()
    foreach( key table
        alist = cons( key alist
            degree[ key ] = length( table[ key ] )
        ); end foreach
    
    while( alist != nil
        blist = list( car( alist )
            graph = cons( SortDegree( subgraph( car( alist ) ) graph)
        ); end while
    
    graph_list = graph
    println( length(graph_list) )
)

procedure( subgraph( seed )
    if( member( seed alist ) ) then
        foreach( element1 adj[ seed ]
            blist = union( adj[ element1 ] blist )
            alist = remq( seed alist )
            subgraph( element1 )
        ); end foreach
        blist
    ); end if
)

/* Find out the edge distance between two nodes */
/* Return 0 if they are not in the same graph component */
procedure( edge_distance( a b )
    prog( ()
        for( i 1 length(graph_list)
            if( memq( a nthelem( graph_list ) ) &&
                memq( b nthelem( graph_list ) ) &&
                a != b then
                return( shortest_path( a b ) )
            ); end if
        ); end for
    return( 0 )
)

/* Find out the shortest path using the breadth first search algorithm */
procedure( shortest_path( a b )
    flag = t
    count = 0
    alist = list()
    foreach( key adj
        alist = cons( key alist )
    )
    blist = list(a)
    alist = remq( a alist )
    while( flag
        clist = list()
        foreach( element1 blist
            if( flag then
                flag = nconc( c...
foreach( element2 adj[element1]
    if( flag then
        if( memq( element2 alist ) then
            if( element2==b then flag=nil
                else clist = cons( element2 clist )
            );end if
        );end if
    );end foreach
);end if
);end foreach

blist = clist
count++

);end while
count

/* Color swapping */
procedure( swap( a b )
    temp_list = color_list[a]
color_list[a] = color_list[b]
color_list[b] = temp_list
)

procedure( rebuild()
    foreach( key color_list
        if( color_list[key]==nil then
            remove( key color_list )
        else
            foreach( element color_list[key]
                color[element] = key
            )
        );end if
    );end foreach
)

/* The three phases of color assigning procedure */
procedure( assign_color()
    nilist = list(0)
    nilist = list(0)

    for(i 1 length(graph_list)
        color_list = makeTable("color_list" nil)
        color_count = 2
        mc1 = 0
        current1 = car(graph)

        /* Phase 1 */
        while( car(current1)!=nil
            current_vertex = car(current1)
            flag = t

            foreach( element1 adj[current_vertex]
                if( flag then
                    if( color[element1]==1 then flag = nil
                else
                    foreach( element2 adj[element1]
                        if( flag then
                            if( color[element2]==1 then flag = nil )
                        );end if
                    );end foreach
                );end if
            );end foreach
        );end while
    );end for(i

)
if (flag then
    color[current_vertex] = 1
else
    color[current_vertex] = color_count
    color_count++
); end if

current1 = cdr(current1)
); end while

foreach (key color
    if (memq(key car(graph)) then
        n = color[key]
        color_list[n] = cons(key color_list[n])
    ); end if
); end foreach

mc = length(color_list)
println(mc)

/* Phase 2 */
while (mc > 1
    sp = 0
    element2 = car(color_list[mc-1])
    foreach (element1 color_list[mc]
        if (sp==0 || sp>4 then
            if (assq(element2 dis[element1]) then
                sp = cdr(assq(element2 dis[element1]))
            else
                sp = edge_distance(element1 element2)
                dis[element1] = cons(list(element2 sp) dis[element1])
                dis[element2] = cons(list(element1 sp) dis[element2])
            ); end if
        ); end if
    ); end foreach

    if (sp==4 then
        foreach (vertex color_list[mc]
            color_list[mc-1] = cons(vertex color_list[mc-1])
        ); end foreach
        color_list[mc] = nil
        mc--
    else
        mc1 = mc
        mc = 0
    ); end if
); end while

println(mc1)

/* Phase 3 */
current_index = mc1-2
while (mc1 > 3
    index1 = current_index
    while (index1 > 1
        ylist = color_list[index1]
        sp = 0
        element2 = car(color_list[index1])
        while (sp==0 || sp>4 && car(ylist)=nil
            element1 = car(ylist)
        ); end while
        current_index = index1
        index1 = current_index
    ); end while
    mc1 = mc1-1
); end while

println(mc1)
if (assq(element2 dis[element1])) then
    sp = cadr(assq(element2 dis[element1]))
else
    sp = edge_distance(element1 element2)
    dis[element1] = cons(list(element2 sp) dis[element1])
    dis[element2] = cons(list(element1 sp) dis[element2])
end if
ylist = cdr(ylist)
end while
if (sp >= 4 then
    swap(index1 mc1-1)
    foreach(vertex color_list[mc1]
        color_list[mc1-1] = cons(vertex color_list[mc1-1])
    )
    end foreach
    color_list[mc1] = nil
    mc1--
    current_index = index1
    index1 = 0
else
    index1--
end if
end while
if (index1==1 then mc1=0 )
end while
mlist = cons(length(color_list) mlist )
rebuild()
nlist = cons(length(color_list) nlist )
grah = cdr(graph)
end for i from 1 to length(graph_list)

/* Procedures call */
makegraph(adj)
assign_color()
: An example of adjacency list

/* File "adj_list" */

adj = makeTable("adj_table" nil)
adj[1] = '4 8
adj[2] = '18
adj[3] = '16
adj[4] = '1 11 20
adj[5] = '9
adj[6] = '7 10 13
adj[7] = '6 8 16
adj[8] = '1 7 15
adj[9] = '5 16 18 20
adj[10] = '6 12
adj[12] = '10
adj[13] = '6 14
adj[14] = '13 19
adj[15] = '8
adj[16] = '3 7 9
adj[17] = '11
adj[18] = '2 9
adj[19] = '14
adj[20] = '4 9

: SKILL code to generate the random graph

/* File "random.s" */

adj = makeTable("adj_table" nil)

net=200 ; Number of nodes
D=4 ; Maximum degree
for i=1 net
  k=2
  while( k > 0
    l = random(net)+1
    if( length(adj[l])=D then
      if( length(adj[l])=D then
        if( l=i && !member(l adj[i]) then
          adj[i] = cons( i adj[i] )
          adj[l] = cons( i adj[l] )
        
        k--
        
      )
    else
      k--
    ) ;end if
  ) ;end while
) ;end for

; write into a file
out1 = outfile("./adj_table")
fprintf(out1 "adj = makeTable("adj_table" nil)\n")
for(i 1 length(adj)
  if( length(adj[i]) >=2 then
    adj[i] = sort( adj[i] \lessp )
  )
  fprintf(out1 "adj[\%d] = %L\n" i adj[i])
) ;end for

close(out1)
Bibliography


