A Design System To Support Built-In Self-Test of VLSI Circuits Using Bilbo-Oriented Test Methodologies

Sen-Pin Lin

CENG Technical Report 94-11

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213)740-2269

May 1994

A DESIGN SYSTEM TO SUPPORT BUILT-IN SELF-TEST OF VLSI CIRCUITS USING BILBO-ORIENTED TEST METHODOLOGIES

by

Sen-Pin Lin

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Electrical Engineering)

May 1994

Copyright 1994 Sen-Pin Lin

Dedication

To my wife, my parents, my brother, and my sister.

Acknowledgements

I am grateful to Professor Melvin Breuer for the guidance, inspiration, and support I received from him over the years. I would like to thank him for the time he has spent on reading my research articles. Much of this thesis is due much to his numerous comments and suggestions. I also want to thank Professor Sandeep Gupta for serving on my dissertation committee and inspiring a great part of my research work. I benefited greatly from the comments I received from him. I am thankful to my colleagues and friends in USC for working and interacting in the test group. In particular I would like to mention Dr. Charles Njinda, Dr. Rajesh Gupta, Dr. Kuen-Jong Lee, Dr. Jung-Cheun Lien, Dr. Amit Majumdar, Mr. Rajagopalan Srinivasan, Mr. Sridhar Narayanan, Mr. Debaditya Mukherjee, Mr. Mody Lempel, and Mr. Ishwar Parulkar. Thanks also go to Professor Alice Parker who helped coordinate the interaction between the synthesis and test experiments, and Mr. Chih-Tung Chen and Mr. Pavil Gupta for providing the test circuits and helpful comments. Finally I wish to thank Professor Ming-Deh Huang for serving on my dissertation committee.

I would like to acknowledge the financial support provided by the Advanced Research Projects Agency through Contract No. J-FBI-90092 (monitored by the Federal Bureau of Investigation).

Contents

Dedication ii

Acknowledgements iii

List Of Tables viii

List Of Figures ix

Abstract xiii

1 Introduction 1
- 1.1 Built-In Self-Test 1
- 1.2 Testable Design Methodologies 3
- 1.3 Test Strategies 6
- 1.4 Overview of the BITS System and its Circuit Model 9
- 1.5 Outline of the Dissertation 12

2 Background 16
- 2.1 Kernel Identification Problem 16
- 2.2 Test Hardware Allocation Problem 18
- 2.3 Test Scheduling Problem 18
- 2.4 Design Space Exploration Problem 20
- 2.5 Design Selection Problem 21

3 The BIBS TDM 22
- 3.1 Introduction 22
- 3.2 Motivation 23
- 3.3 Balanced BISTable Kernels and the BIBS TDM 38
  - 3.3.1 Circuit Graph Model 38
  - 3.3.2 Definition of Balanced BISTable Kernels 40
  - 3.3.3 Properties of Balanced BISTable Kernels 43
- 3.4 Comparison of the BIBS TDM and the K&A TDM 44
- 3.5 Summary 49

4 TPG Design for Balanced BISTable Kernels **51**
4.1 Introduction 51
4.2 Uni-cone and Multi-cone Kernels 53
4.3 TPG Design for Uni-cone Kernels 55
4.4 TPG Design for Multi-cone Kernels 62
4.5 Functionally Pseudo-Exhaustive Testing 69
4.6 Performance Evaluation 72
4.7 Remark 74
4.8 Summary 79

5 Kernel Identification **80**
5.1 Introduction 80
5.2 BISTable Circuit Design with I-paths 81
5.2.1 I-path and I-mode Definition 81
5.2.2 I-path Identification 82
5.2.3 I-path Creation 84
5.3 BISTable Circuit Design Using Balanced BISTable Kernels 86
5.3.1 Series-Parallel Structure(SPS) 86
5.3.2 BIBS Design for SPSs 87
5.3.3 BIBS Design for non-SPSs 107
5.3.4 BIBS Design with Cone Size Constraint 111
5.3.5 BIBS Design with Switches and I-paths 112
5.4 Summary 116

6 Kernel Embedding Enumeration **118**
6.1 Introduction 118
6.2 I-path Conflicts 119
6.2.1 Classification of I-path Conflicts 119
6.2.2 Conflict Detection 120
6.3 Embedding Enumeration 127
6.4 Embedding Compatibility Analysis 134
6.4.1 Classification of Compatibility 134
6.4.2 Compatibility Analysis 134
6.5 Summary 138

7 Test Scheduling **140**
7.1 Introduction 140
7.2 Test Scheduling Problem 141
7.2.1 Problem Statement 141
7.2.2 Classification of Test Scheduling Strategies 141
7.2.3 Incremental Test Scheduling vs. Non-incremental Test Scheduling 144

7.3 Test Scheduling Procedure 145
7.3.1 Graph Model for a Test Schedule 145
7.3.2 Scheduling Procedure 147
7.3.3 Test Time Lower Bound Estimation 153
7.4 Performance Evaluation 155
7.5 Summary 158

8 Design Space Exploration **161**
8.1 Introduction 161
8.2 Design Space and Representative BISTable circuits 162
8.3 Design Space Exploration Procedure 163
8.4 Complexity Issues 166
8.4.1 Using Dominating I-paths 167
8.4.2 Selective Exploration 169
8.5 SA Merging to Reduce Aliasing Probability 171
8.6 Test Plan Generation 174
8.6.1 Test Plan Specification 175
8.6.2 From Test Schedule to Test Plan 176
8.7 Summary 177

9 Selection Problem **180**
9.1 Introduction 180
9.2 Problem Statement 181
9.3 Selection System Overview 182
9.4 Initial Requirement Specification 183
9.4.1 A Probability Model 184
9.4.1.1 Preliminaries 184
9.4.1.2 Correlation between random variables 185
9.4.1.3 Correlation between attributes 186
9.4.1.4 Conditional Probabilities and Likelihood Subinterval Set 187
9.4.2 Procedure for Initial Requirement Specification 190
9.5 Evaluation of Objects and the Score Function 191
9.5.1 Score Function 192
9.5.2 Customizing PCFs 195
9.6 First Phase Comparison and Self Modification Process I 198
9.7 Second Phase Comparison and Self Modification Process II 202
9.8 A Case Study 204
9.9 Summary 215

10 BITS Implementation and Experiments **218**
10.1 BITS Implementation 218

10.2 Experimental Results 230
10.2.1 The DCT Circuit 230
10.2.2 The AR Filter Circuits 231
10.2.3 Miscellaneous Circuits 235

11 Conclusion **241**
11.1 BIBS TDM and its Associated TPG Design 241
11.2 Test Scheduling and BISTable Design Space Exploration 244
11.3 BISTable Design Selection 245

Reference List **247**

List Of Tables

1.1 Summary of test strategies 7
1.2 Summary of TDMs and test strategies supported by BITS 8

3.1 Circuit characteristics of the experiment 27
3.2 Simulation results for circuits N1.1 and N1.2 28
3.3 Simulation results for circuits N2.1 and N2.2 28
3.4 Simulation results for circuits N3.1 and N3.2 29
3.5 Simulation results for circuits N4.1 and N4.2 29
3.6 Results of testing S_1, S_2, and S_3 36
3.7 Summary of the data path circuits 47
3.8 Summary of the experimental results 48
3.9 Effects of adding extra BILBO registers to BIBS testable circuits . 49

4.1 Results of testing $K_{1,2}$ and $K_{2,1}$ 75

7.1 Summary of the experimental results 157
7.2 Comparisons between the two scheduling schemes and the test time lower bound 159

8.1 An execution trace of Procedure *TCF2TSG* 178

9.1 Number of subintervals in each attribute 187
9.2 Absolute values of correlations 187
9.3 Example BISTable circuits 216

10.1 BISTable designs for the DCT circuit 232
10.2 Characteristics of the AR filter circuits 232
10.3 BISTable designs for circuit AR2 233
10.4 BISTable designs for circuit AR4 234
10.5 BISTable designs for circuit AR6 234
10.6 BISTable designs for circuit AR8 234
10.7 BISTable designs for circuit `ex1` 235
10.8 BISTable designs for circuit `sps3` 239

4.4 (a) a reduced bipartite graph representation of a uni-cone kernel; (b) a TPG design for the kernel 55
4.5 A TPG design for the balanced BISTable kernel in Figure 4.2(a) . . 56
4.6 TPG for a balanced BISTable kernel 58
4.7 A TPG design for Example 4.2 61
4.8 (a) the bipartite graph representation of an example kernel; (b) TPG design for (a) 62
4.9 (a) an example multi-cone kernel; (b) TPG design for (a) 63
4.10 An output cone that depends on R_{x_1}, R_{x_2}, ..., R_{x_p} 66
4.11 (a) a multi-cone kernel for Example 4.5; (b) TPG design for (a) . . 68
4.12 A reconfigurable TPG design for kernel in Figure 4.11(a) 68
4.13 (a) a three cones kernel; (b) a TPG design for (a); (c) an alternative TPG design for (a) 70
4.14 (a) an example circuit; (b) S_1; (c) S_2 73
4.15 Bipartite graph representations for (a) $K_{1,2}$; and (b) $K_{2,1}$ 73
4.16 TPG for (a) $K_{1,2}$; (b) $K_{2,1}$ 74
4.17 (a) a bipartite graph representation of a balanced BISTable kernel; (b) a counting sequence and the sub-cycles generated by a counter; (c) patterns observed at O_1 in (a) when $d_{1,1} - d_{2,1} = 1$ 77
4.18 (a) a bipartite graph representation of a balanced BISTable kernel; (b) a TPG employing a pure LFSR; (c) patterns generated by TPG in (b) and patterns observed at O; (d) an alternative TPG; (c) patterns generated by TPG in (d) and patterns observed at O . . . 78

5.1 A MUX with four I-modes 82
5.2 (a) A circuit with no available test hardware for port P; (b) adding a transparent register; (c) adding an I-path 85
5.3 (a) S-reduction; (b) P-reduction; (c) C-reduction 87
5.4 (a)-(f) maximal reduction of the circuit graph in Figure 3.3(b) . . . 88
5.5 A simple cycle 89
5.6 (a) an example circuit graph having non-simple cycles; (b) two BILBO edges (a, b) and (b, c); (c) another two BILBO edges (c, d) and (d, e) 90
5.7 A non-primitive UBS Y contains X 91
5.8 (a) One BILBO edge in every path between u and v; (b) two BILBO edges in all but one path between u and v 92
5.9 Series paths in a non-primitive UBS 92
5.10 (a)-(g) an execution trace of Procedure *ckt_simplify* 97
5.11 (a) UBSs X and Y; (b) partition of X using $S_0(X)$; (c) partition of X using $S_1(X)$ 99
5.12 (a) A tree; (b) a reverse tree 104

5.13 (a) An example circuit graph; (b) simplified circuit graph; (c) partitions of the circuit graph . . . 105
5.14 TPG design for kernel represented by $P2$ . . . 106
5.15 (a) a non-SPS $G1$; (b) a partition of $X1$ that modifies a non-SPS ($G1$) to be a SPS ($G1'$) as in (c) . . . 110
5.16 Partitioned graph after cone size reduction . . . 113
5.17 (a) An example data path circuit with switches; (b) circuit graph for (a); (c) partitioned graph without using switches; (d) partitioned graph using switches . . . 115

6.1 (a) Rule 1.2(a); (b) Rule 1.2(b); (c) Rule 1.2(c); (d) Rule 1.3(a); (e) Rule 1.3(b); (f) Rule 1.4(a); (g) Rule 1.4(b).i; (h) Rule 1.4(b).ii . . 122
6.2 (a) Rule 2.2(a); (b) Rule 2.2(b); (c) Rule 2.3(a); (d) Rule 2.3(b); (e) Rule 2.4(a); (f) Rule 2.4(b) . . . 124
6.3 (a) Rule 3.1(a); (b) Rule 3.1(b); (c) Rule 3.2(a); (d) Rule 3.2(b); (e) Rule 3.3(a); (f) Rule 3.3(b).i; (g) Rule 3.3(b).ii . . . 125
6.4 (a) An embedding for C_6; (b) an execution of the embedding in (a); (c) reservation table for (b); (d) modified execution of the embedding to resolve SC; (e) reservation table for (d) . . . 129
6.5 Examples of employing COMBILBO TDM . . . 133
6.6 An example circuit . . . 135
6.7 (a) an embedding for K_1; (b) an execution of embedding in (a); (c) reservation table for (b) . . . 136
6.8 (a) an embedding for K_2; (b) an execution of embedding in (a); (c) reservation table for (b) . . . 137
6.9 (a) a concurrent execution of embeddings for K_1 and K_2; (b) reservation table for (a) . . . 138

7.1 An example circuit . . . 142
7.2 A TCG for the tests in Figure 7.1 . . . 142
7.3 An example search tree . . . 145
7.4 (a) an example TCT; (b) its corresponding test schedule . . . 146
7.5 The process of Procedure *genTCF* (steps (a) through (f)) . . . 151
7.6 (a) test schedule for the BISTable circuit in Figure 7.1; (b) a modified test schedule from (a) to simplify test controller . . . 152
7.7 The process of Procedure *genTCF* for Example 7.3 . . . 153
7.8 The TIG for tests in Example 7.2 . . . 155

8.1 (a) a typical design space; (b) a design space with design constraints 163
8.2 Equivalent I-paths . . . 168
8.3 An example circuit to illustrate selective exploration . . . 170

9.1 An overview of SAESS . . . 182

9.2 An example PCF 193
9.3 Illustration of calculating a normalized PC value 194
9.4 Process of determining number of subintervals 196
9.5 Effects of score function modification 201
9.6 An example bar-chart diagram 202
9.7 Flowchart for the first phase comparison and self modification process I 203
9.8 Flowchart for the second phase comparison and self modification process II 205

10.1 The top level menu and pop-up window of BITS 219
10.2 Illustration of the CRIPI option in ToolBox 221
10.3 Circuit graph generated by the *dot* program 222
10.4 BISTable circuit preview - first test session 226
10.5 BISTable circuit preview - second test session 227
10.6 Balanced version of `ex1` generated by the BIBS TDM 236
10.7 Test circuit `sps3` 238
10.8 Balanced version of `sps3` generated by the BIBS TDM 240

Abstract

Built-in self-test (BIST) is an approach to solve the complex problem of testing a digital circuit by modifying the circuit so that it can test itself. Numerous BIST techniques have been proposed. Each technique has its advantages and disadvantages, and there is no universally superior technique to meet every designer's requirements. This thesis presents an integrated CAD system, called BITS, that employs a variety of BIST techniques to generate a family of self-testable circuits having a range of values in terms of a list of BIST parameters, including area overhead, test time, and performance impact. A novel BIST methodology, called *BIBS*, that leads to low hardware cost and performance impact while guaranteeing comprehensive fault coverage is proposed. The design of a family of test pattern generators that achieve comprehensive fault coverage when using the BIBS methodology is presented. Algorithms are presented to allocate test hardware and resolve conflicts between the employed test hardware. A test scheduling problem is discussed and an efficient procedure is developed to minimize the test application time of a circuit. A systematic way to explore the design space is presented that is practical and efficient in producing a family of self-testable circuits for a design. Optimal designs in terms of various BIST parameters such as area overhead and test time can also be generated. A knowledge base selection system is implemented to help a designer trade-off different design costs and make an intelligent choice among the generated circuits. BITS has been implemented and experiments performed to make many data path circuits self-testable. The results demonstrate that BITS is able to efficiently generate a family of designs, and trade-offs exist between the designs generated. BITS is a valuable CAD tool that helps a circuit designer generate a self-testable version of a design that meets the design constraints.

Chapter 1

Introduction

The objective of this research is to develop an integrated system that makes a VLSI circuit under consideration(CUC) testable using built-in self-test(BIST) techniques. Various BIST techniques have been proposed to make a CUC testable. However, it remains an open problem to select and implement the most appropriate technique based on the characteristics of a CUC. In addition, there are often many different ways to implement a selected technique, where each technique has its own merits and drawbacks. Therefore it is useful to have an integrated system that automatically generates a family of implementations and helps a designer make an intelligent choice based on the design constraints.

In the following sections we will first discuss the need for BIST and present some general concepts in employing BIST. The BIST design parameters such as area overhead and test application time are also discussed. Various techniques including testable design methodologies(TDM)[1] and test strategies will then be studied. An overview of the developed system will be presented where the TDMs and test strategies supported and the circuit model employed by the system are described. Finally the organization of this dissertation will be briefly outlined.

1.1 Built-In Self-Test

The high circuit density and the limited I/O inherent to VLSI chips make the problem of testing physical faults very complicated and time consuming. Design for testability (DFT) is a technique that improves the testability of a circuit by

adding test hardware such as test points or scan cells. This often results in shorter test application time, higher fault coverage, and easier automatic test pattern generation (ATPG). However, external automatic test equipment (ATE) is required to apply test patterns and control signals, and to receive and analyze output responses. ATE is expensive and the application of tests is time consuming due to two reasons. First, ATE usually operates at a slower clock rate than state of the art high performance circuits. Second, test patterns cannot be applied to the circuit every clock cycle since they need to be shifted into the circuit through a scan chain. This makes at speed testing impossible. These problems have led to the development of BIST techniques where parts of a circuit are used to test the circuit itself. To a great extent this alleviates the reliance on an ATE and testing can be carried out at normal functional speed. In some cases this not only substantially reduces the test application time, but also enables the detection of timing related faults.

BIST techniques are classified into two categories, namely on-line BIST and off-line BIST. In on-line BIST testing occurs during normal functional operation. On the other hand, off-line BIST deals with testing a circuit when it is not performing its normal functions. A circuit under test(CUT) is configured in a test mode during the application of BIST. Only off-line BIST is dealt with in this research. A circuit is called *BISTable* if it is made testable in a BIST environment. Most practical circuits are too complicated to be tested as one entity. Therefore, a circuit is usually partitioned into sub-circuits, where each sub-circuit is called a *kernel*. A kernel is a test primitive in the sense that test patterns for this kernel are generated and output responses from this kernel are compressed outside of the kernel. To make a kernel K BISTable, every input port of K is directly or indirectly fed by a test pattern generator(TPG) and every output port of K directly or indirectly feeds a signature analyzer(SA). These TPGs and SAs are said be to associated with K. TPGs and SAs are often configured as a linear feedback shift register(LFSR). During the test of K, the associated TPGs and SAs are first initialized to known states, then a sufficient number of test patterns are generated by the TPGs and applied to K. Outputs from K are compressed in the SAs to form a signature. After all patterns have been applied to K, the final signature is

shifted out of the SAs and compared with a fault-free signature. Due to the test hardware required by TPGs and SAs, a BISTable circuit has a greater layout area than the original circuit. This extra layout area of a BISTable circuit is referred to as its *area overhead*. Test hardware often increases circuit delays that may lead to *performance degradation*.

A circuit usually consists of a set of kernels and depending on the test hardware allocation, some kernels may be tested at the same time while others cannot. A *test schedule* specifies the order of testing all the kernels, and is divided into several *test sessions*, where in each test session one or more kernels are tested. A *test plan* specifies how to initialize test hardware, perform tests, and observe the final signature for each test session. A test plan is executed by a *test controller* which can be on-chip or off-chip. The *test time* of a BISTable circuit is the time to execute its test plan.

In addition to the area overhead, performance degradation and test time, other BIST parameters to be considered include fault coverage, design complexity and extra I/O pins. The BISTable circuit design problem is complicated since there are many BIST TDMs available and numerous ways to implement each TDM for a CUC. There are usually a family of BISTable versions of the original circuit and each may have different BIST parameters. Some BIST parameters may be more significant than the others in one design environment but less important in another environment. The relative importance and the requirements for BIST parameters will be decided by a designer based on the design constraints. As can be observed from the above discussion, generating an optimal BISTable design for a given circuit is a non-trivial process and an integrated system that can efficiently explore the BIST design space and produce an optimal design is desired.

1.2 Testable Design Methodologies

Numerous BIST TDMs have been proposed in recent years. A summary of many of the TDMs can be found in [2]. Among them the BILBO(Built-In Logic-Block Observation) TDM[3] and its variations have been widely employed[4, 5, 6, 7, 8, 9].

The basic BILBO architecture consists of partitioning a circuit into a set of registers and blocks of combinational logic, where some normal registers are replaced by BILBO registers. BILBO registers in a circuit are linked as a scan chain. In addition, the inputs to any block of logic K are driven by a BILBO register and the outputs of K drive another BILBO register (see Figure 1.1). A BILBO register can perform any one of four functions, namely parallel load (normal mode), TPG, SA and shift (scan). The function performed by a BILBO register is selected by the values applied to the control lines of the register. When a BILBO register is driven by a combinational logic block and operates as a SA, it compresses the output responses of the block to form a signature. When a BILBO register drives a block of logic and operates as a TPG with a non-zero initial value, it performs as a pseudo-random pattern generator. A BILBO *embedding* consists of a kernel under test and its associated BILBO registers as shown in Figure 1.1.

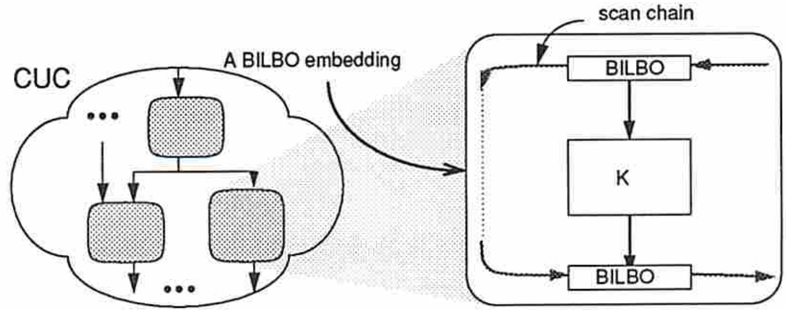


Figure 1.1: The basic BILBO TDM and a BILBO embedding

A BILBO register may not be required to operate in all four modes during the testing of a circuit. In such a case, a simpler TPG or SA design can be employed that reduces the area overhead[10]. A data path circuit often contains combinational logic blocks, such as MUXes and busses, that can be used as switching devices. These blocks are called *switches* in this dissertation. A switch has a set of *I-modes*[4] where each I-mode is represented as a pair of input and output ports, (P_i, P_o), of the switch. Data can be transferred unaltered from P_i to P_o in each I-mode by setting proper values on the control lines of the switch. Other common circuit structures having I-modes are registers, adders, multipliers, and ALUs. If a data path only consists of blocks having I-modes and the connections between these

blocks, then it is called an *I-path*[4] since data can be transferred unaltered from the source of the path to its destination. Clearly test patterns and output responses can be transferred to and from a kernel if I-paths exist between the kernel and its associated TPGs and SAs. The methodology that enhances the basic BILBO TDM by employing the concept of I-path is called *extended-BILBO(EXTBILBO)*[9]. The EXTBILBO TDM often allows for sharing of TPG and SA among different kernels and leads to less area overhead. An EXTBILBO embedding consists of a kernel under test, the associated TPGs and SAs, and the I-paths connecting these TPGs and SAs to the kernel (see Figure 1.2(a)). As can be seen from Figure 1.2(a), the register R_1 can be shared by both the embeddings for K_1 and K_2 due to the switch BUS.

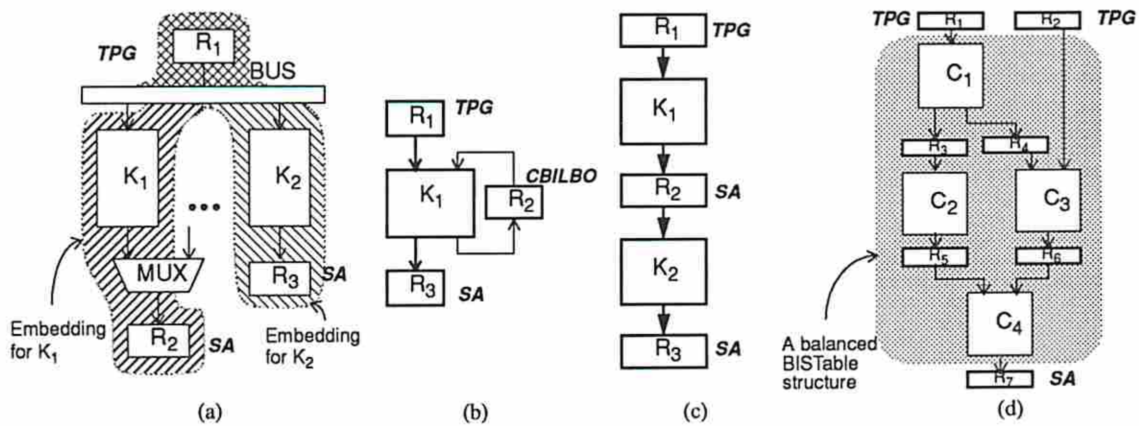


Figure 1.2: (a) extended BILBO(EXTBILBO); (b) concurrent BILBO(CBILBO); (c) combined BILBO(COMBILBO); (d) balanced BILBO(BIBS)

Wang and McCluskey proposed a test register design that can simultaneously operate as a TPG and SA[11]. Such registers are called *CBILBO registers* and the methodology that employs CBILBO registers is called the *Concurrent BILBO(CBILBO)* TDM (see Figure 1.2(b)). CBILBO TDM is often used in circuit with *self loops*, where a self loop consists of a combinational logic block that feeds itself through a single register. For example, there exists a self loop in Figure 1.2(b) since the kernel K_1 feeds itself through register R_2. R_2 has to be a CBILBO register so that it can operate simultaneously as both a TPG and a SA. Due to the extra hardware required in a CBILBO register, the area overhead of

such a register is approximately twice of that of a normal BILBO register[10].

Kim *et al.* proposed a TDM in which a BILBO register that operates as a SA is also used to simultaneously apply test patterns to the combinational logic block fed by the register[12]. This TDM is illustrated in Figure 1.2(c) where register R_2 operates as a SA and its outputs are used as test patterns for K_2. Both analytical and experimental results presented in [12] showed that for this configuration, after an initial transient period, the characteristics of the SA register are similar to a random pattern generator. However, theoretically the effectiveness of this technique depends on the functionality of the combinational logic block (K_1) that drives the SA register (R_2). In this thesis this methodology is called the *combined BILBO(COMBILBO)*.

Lin *et al.* proposed a low cost BIST methodology, called the *BIBS* TDM, that employs *balanced BISTable structures* as kernels[13]. A kernel is balanced BISTable if (1) it does not contain cycles; (2) all paths between a pair of combinational logic blocks in the kernel contain the same number of registers; and (3) registers feeding the inputs of the kernel and registers fed by the outputs of the kernel are distinct. Figure 1.2(d) shows a balanced BISTable structure and its embedding where the input of the structure is fed by R_1 and R_2, and the output of the structure feeds R_7. A detailed discussion on balanced BISTable structures and the BIBS TDM will be presented in Chapter 4. The BIBS TDM usually requires less test hardware and leads to lower performance degradation than the other TDMs described above.

1.3 Test Strategies

Once a BIST TDM is chosen for a kernel, there are often several test pattern generation methods applicable to the kernel. Each method is called a *test strategy*. The test strategies considered in this work are summarized in Table 1.1.

For a given kernel, different test strategies may achieve different fault coverage and require different test application time and area overhead. The most appropriate test strategy for a kernel usually depends on the design constraints specified by a designer and the logic inside the kernel. For example, exhaustive testing guarantees the detection of every irredundant stuck-at fault, thus achieves a high fault

Test Strategy	Description	Generic Circuit Representation	
Exhaustive	Apply 2^n distinct patterns,K is combinational	Combinational	Balanced sequential
Pseudo-exhaustive	Apply 2^k distinct patterns, $k \leq n$, and guarantee detection of all detectable stuck-at faults in a combinational K		TPG n Normal registers
Pseudo-random	Apply p distinct patterns, $p < 2^n$	TPG n	
Random	Apply q patterns. Patterns may repeat		
Functionally exhaustive	Apply 2^n distinct patterns, K is balanced sequential	K	K
Functionally pseudo-exhaustive	Apply 2^j distinct patterns, $j \leq n$, and guarantee detection of all detectable stuck-at faults in a balanced sequential K		

Table 1.1: Summary of test strategies

coverage. However, when the input width of a kernel is large, the test application time may be excessive and exceed the designer's constraint. On the other hand, similar fault coverage may be achieved with significantly less test time when the *maximal cone size* of the kernel is small compared with the input width of the kernel, where a *cone* is the logic associated with an output of the kernel and the maximal cone size is the maximum among the number of inputs on which every cone in the kernel depends. For combinational logic blocks with regular structures, such as adders and multipliers, a high fault coverage can often be achieved by using pseudo-random testing in a small fraction of the time required by exhaustive or pseudo-exhaustive testing. Functionally exhaustive testing guarantees the detection of every stuck-at fault that interferes with normal circuit function. It can be employed when balanced sequential kernels are allowed and a comprehensive fault coverage is desired. In some circuits, similar fault coverage can be achieved with less test time using functionally pseudo-exhaustive testing when output cones of a balanced sequential kernel do not depend on every inputs. In practice, a CUC often has a set of kernels, each having different sizes (i.e. number of gates), input widths and logic complexity. Therefore, it is advantageous to employ a variety of test strategies to test a circuit so that each kernel is tested in the most efficient fashion.

It should be noted that not every test strategy can be applied to each TDM

Test Strategy \ TDM	BILBO & EXTBILBO	CBILBO	COMBILBO	BIBS
Exhaustive	YES	YES	YES	N.A.
Pseudo-exhaustive	YES	YES	YES	N.A.
Pseudo-random	YES	YES	YES	YES
Random	N.S.	N.S.	YES	N.S.
Functionally exhaustive	YES	YES	N.A.	YES
Functionally pseudo-exhaustive	YES	YES	N.A.	YES

N.A. - not applicable N.S. - not supported YES - supported

Table 1.2: Summary of TDMs and test strategies supported by BITS

selected. The relationship between TDMs and test strategies are summarized in Table 1.2. For each test strategy Y, a 'N.A.' under TDM X implies that this test strategy is not applicable to TDM X; a 'YES' indicates that the BITS system will support this TDM/test strategy combination; otherwise it is not supported, indicated by a 'N.S.'. Notice that when a kernel is combinational, exhaustive testing is the same as functionally exhaustive testing, and pseudo-exhaustive testing is the same as functionally pseudo-exhaustive testing. Therefore when BILBO, EXTBILBO, or CBILBO TDM is employed, exhaustive testing and functionally exhaustive testing are equivalent. Similarly, under the above TDM pseudo-exhaustive testing and functionally pseudo-exhaustive testing are equivalent. When COMBILBO TDM is employed, only those kernels that are driven by a SA register are tested randomly and the remaining kernels are tested either exhaustively, pseudo-exhaustively, or pseudo-randomly.

1.4 Overview of the BITS System and its Circuit Model

BITS(Built-In Test System) is an integrated system developed by the USC Test Group that directs a designer in specifying design constraints, generates a family of BISTable designs, helps the designer select a BISTable design, and physically modifies the CUC to be BISTable. The input to BITS is a CUC stored in Cbase (an object-oriented database developed by the USC Test Group[14]). The input circuit may be generated by a designer using `ui` (Cbase user interface), translated from a structural VHDL description, or generated by a high-level synthesis system such as ADAM[15] using a behavioral VHDL description as input. A circuit in Cbase is represented in a hierarchical fashion. In the top level of the hierarchy, the circuit is represented at register transfer level (RTL), where a circuit is composed of cells and the connections between cells. A cell in an RTL description of the circuit can be a combinational logic block, a sequential logic block, a register, or a fanout. Some combinational logic blocks have special functions such as MUXes or busses that can be used as switches. Each cell is made up of gate level constructs such as AND gates, OR gates, F/Fs, etc. These gate level components are stored in the lower level of the hierarchy. The output of BITS is a BISTable circuit with test hardware and test controller embedded and this circuit is stored in Cbase for future processing or layout.

An overview of the BITS system is illustrated in Figure 1.3. The major modules that constitute the system are described below.

- **CLARION**[16]. The CLARION system partitions and clusters a circuit to identify combinational logic blocks, fanouts and registers. The objective of the CLARION system is to provide a view of the CUC that is more appropriate than the original view for the test purpose.
- **Kernel Identification Module**. This module identifies kernels in a CUC based on the specified design constrains and the selected TDM. When a large combinational logic block exists that requires an excessively long test time, it

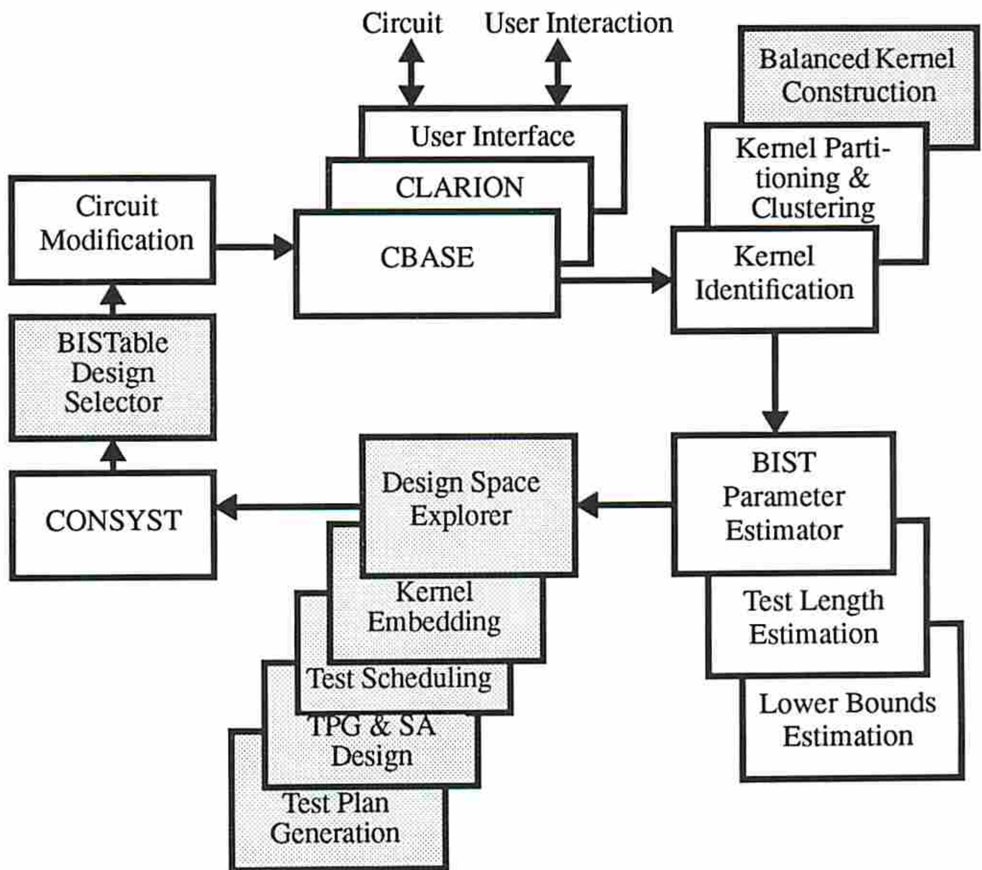


Figure 1.3: System overview

may be necessary to partition the block to reduce the test time. Partitioning a combinational logic block may also be required when pseudo-exhaustive testing is employed. On the other hand, when the constraints on the area overhead and/or performance degradation are tight, it may be desirable to employ the BIBS TDM by clustering blocks of logic into complex kernels so that the amount of test hardware required and the performance degradation are reduced.

- **BIST Parameter Estimator.** This module estimates the test lengths of each kernel for various test strategies, analyzes the compatibility of the kernels and calculates lower bounds on the test time and area overhead. When pseudo-exhaustive or pseudo-random testing is employed, gate level description of a kernel is required to estimate its test length. The objective of the estimator is to determine the test length of each kernel and help a designer select the appropriate TDM and test strategies, and modify the design constraints.

- **Design Space Explorer.** This module first constructs a set of embeddings for each kernel by allocating test hardware for the kernel. Embeddings for different kernels may not be compatible if they share test hardware. Compatible embeddings can be executed simultaneously to reduce test time. An incremental test scheduler is employed to explore the concurrency of executing embeddings. A family of BISTable circuits each having different values of BIST parameters is generated by the design space explorer. A test plan is generated and associated with each BISTable circuit. The LFSR designs for the TPGs and SAs employed in the BISTable circuits are also produced.

- **CONSYST (Test Controller Synthesizer).** This module produces a test controller for each BISTable circuit based on the test plan provided by the design space explorer.

- **BIST Design Selector.** This module allows the designer to make tradeoffs between different BIST parameters and select a BISTable circuit that satisfies the design constraints.

- **Circuit Modification Module.** This module physically modifies a CUC to embed the selected test hardware and test controller in the circuit.

It should be noted that each module in Figure 1.3 may be executed more than once to obtain a satisfactory BISTable circuit. Therefore, backtracking is allowed in the system to fully explore the design space. This dissertation deals with a substantial part of the BITS system as illustrated by the shaded blocks in Figure 1.3.

1.5 Outline of the Dissertation

As can be seen from the previous sections, the BISTable circuit design problem and the implementation of the BITS system is complex. The remainder of this dissertation is organized as follows.

Chapter 2 provides the background of this study and surveys previous research that dealt with various aspects of the BISTable circuit design problem. The major problems considered in this study will also be summarized.

Chapter 3 introduces a new BIST TDM, called BIBS, that requires less area overhead and performance degradation than other BILBO-oriented TDMs, and ensures comprehensive fault coverage. The concepts of k-step functionally detectable faults and k-step functional testing are introduced. A class of circuits, called *balanced BISTable structure*, will be defined which is employed in the BIBS TDM. We will study the property associated with a balanced BISTable structure and show that the BIBS TDM guarantees the detection of every 1-step functionally detectable fault.

Chapter 4 presents a novel TPG design scheme that can be used to test a balanced BISTable structure. A TPG thus designed is able to provide 1-step functionally exhaustive test patterns when a comprehensive fault coverage is desired. When a balanced BISTable structure has only one output port, a procedure for designing a TPG with minimal test time to achieve 1-step functionally exhaustive testing is proposed. For a structure having multiple output ports and every output port not depending on all input registers, a new test strategy called *functionally pseudo-exhaustive testing* is invented that can achieve the same fault coverage as

functionally exhaustive testing with lower test time. A procedure for designing a TPG for this test strategy is also proposed.

Chapter 5 deals with the process of identifying kernels under test. Since I-paths in a circuit allow for sharing test hardware among different kernels, I-paths are identified and employed to reduce area overhead. In the I-path identification process, all the I-paths that drive an input port of a kernel or are driven by an output port of a kernel are constructed by a circuit traversal. When no I-path exists for a port of a kernel, I-paths must be created by adding extra registers or MUXes to a circuit. The procedure for identifying and creating I-paths is presented. For a complex circuit the area overhead of a BISTable circuit may be excessive when only combinational kernels are allowed. In such cases balanced kernels may be desired, where a balanced kernel is simply a balanced BISTable structure. The procedure for finding balanced kernels in a circuit is presented, which consists of two steps, namely circuit reduction and circuit partitioning. In circuit reduction step a set of reduction rules are applied to a CUC to reduce the complexity of the circuit. The quality of a solution in the balanced kernel identification process is not affected by the circuit reduction. In circuit partitioning step the reduced circuit is partitioned into disconnected circuit components so that each component represents a balanced kernel. A class of circuits, called *series-parallel structure(SPS)*, is introduced. A polynomial time procedure will be presented for finding balanced kernels that requires minimal test hardware for SPS. For a non-SPS, a branch and bound procedure is described for generating balanced kernels that require a minimal amount of test hardware. In the presence of switches, test hardware may be shared by different kernels. We extend the above procedures to deal with circuits with switches. When the maximal allowable test time for a circuit is constrained, some restrictions must be applied to the kernels and the procedures are extended to deal with this problem.

Chapter 6 discusses the process of kernel embedding enumeration. To embed a kernel into a TDM, an I-path is associated with each port of the kernel. When two I-paths have common circuit components, such as MUXes or registers, contentions may occur in these common circuit components if they are simultaneously used by both I-paths. This is referred to as an I-path conflict. The I-paths associated with

a kernel must be free of I-path conflicts or the I-path conflicts between them must be resolved. The I-path conflict resolution is considered and the *execution* of an embedding is discussed. A procedure to enumerate embeddings for each kernel and determine how to execute each embedding is presented. Each embedding consists of a kernel under test, the test hardware required to test the kernel, the I-paths to transfer test patterns and output responses between the kernel and its associated test hardware, the test length required by the embedding, and an execution of the embedding. Finally the compatibility between embeddings to be used by the test scheduler to minimize the test time is analyzed.

Chapter 7 presents an incremental test scheduler that is capable of finding a near-optimal test schedule for a given BISTable design in polynomial time. The test scheduler is incremental in the sense that it does not require a fully specified BISTable circuit to start its execution. Therefore, a partial test schedule can be obtained for any partial design. This is an important feature since during the design space exploration process, it is desirable to reject a partial design that cannot lead to a satisfactory BISTable circuit early in the process. We also develop a procedure to estimate a lower bound for the test time of a BISTable design. This lower bound can be used by a designer to specify a reasonable constraint on test time. It can also be used to evaluate the quality of the test schedule generated by the test scheduler.

Chapter 8 discusses the design space exploration process. The procedure employed in this process will be presented. The design space explorer generates a family of BISTable designs for a CUC. For each design, a test plan is automatically generated that will be used by the test controller synthesizer to produce a test controller for the design. The LFSR designs for the TPGs and SAs employed in each design will also be generated.

Chapter 9 presents a self-adaptive expert selection system(SAESS) that is a generic system applicable to almost every selection problem, including the problem of selecting a BISTable circuit. SAESS is a knowledge-based system that evaluates the candidates of a selection problem using pre-stored expert knowledge. However, making a selection is often a subjective process since every user has his/her own preferences. Therefore, the knowledge in SAESS is dynamically updated during a

selection process interactively to reflect a user's preferences. This machine learning mechanism is formulated as a linear programming problem.

Chapter 10 discusses the implementation of the BITS system and experimental results obtained by using the system. A set of test circuits are considered. For each test circuit a variety of TDMs are employed to illustrate the design space for making a circuit BISTable.

Chapter 11 concludes this dissertation with a summary, proposed extensions of the BITS system, and a list of future research topics.

Chapter 2

Background

In this chapter previous research on various problems and related systems dealing with the development of a BISTable circuit design system is reviewed. These problems include kernel identification, test hardware allocation, test scheduling, design space exploration, and design selection.

2.1 Kernel Identification Problem

The complexity of VLSI circuits makes it necessary to partition a circuit into smaller and manageable sub-circuits, each of them being a kernel. Several approaches have been proposed to partition a circuit into kernels as described below.

The CRETE system developed by Gupta *et al.*[17] reorganizes a hierarchically described circuit and transforms it into a circuit consisting of clouds and registers. A *cloud* is a maximally connected combinational logic block that has either PIs or registers at its inputs, and either POs or registers at its outputs. Each cloud is simply a kernel if it is not too complex. Otherwise further partitioning such as the technique presented in [18] can be applied to reduce its complexity.

When a circuit contains combinational blocks with special functions such as MUXes, busses, adders or ALUs, it is usually desirable not to combine them with other combinational blocks to form clouds since they provide I-modes and/or special functional testability. The CLARION system developed by Parulkar and Breuer[16] can exclude such combinational blocks from being combined in a cloud. In addition, fanouts can be identified that often lead to less complex kernels. As

described in the previous chapter, CLARION is employed in the BITS system to identify combinational kernels.

Krasniewski and Albicki[8] proposed a kernel identification approach where sequential kernels are allowed. A sequential kernel employed in [8] must satisfy the following requirements.

1. It does not contain a sequential feedback loop.

2. Every input port of a combinational logic block in the kernel must be driven directly by a BILBO register if the combinational block has more than one input port.

This approach usually leads to less area overhead than the approach where only combinational kernels are allowed. In addition, single pattern testability[19] of a kernel is guaranteed. However, the second requirement restricts each kernel to be a simple pipeline structure without fan-ins. In the next chapter we will present a novel kernel identification approach that also guarantees the single pattern testability of kernel and is more general than the approach in [8].

Abadir *et al.*[20] proposed a heuristic-based approach to partition and cluster a circuit into kernels suitable for a variety of TDMs. A hierarchical VHDL description of a circuit is first flattened and clustered into RTL components such as registers, RAMs, ROMs, MUXes, and combinational logic blocks. Adjacent RTL components are then clustered into larger components so that they can be tested more efficiently. Each RTL component may be clustered with other components in various ways, thus it can be contained in more than one cluster. Certain clustering rules, such as a cluster cannot contain a register at its input, are applied during the clustering process. In addition, constraints are placed on the size and input width of each cluster so that complex clusters are not generated. Several costs are then estimated for each cluster based on its area/performance overhead and test generation complexity. These costs are employed in the cluster selection process where a set of clusters is selected so that each RTL component is covered by exactly one cluster. The selected clusters are the kernels under test. The drawback of this approach is the heuristic nature of the cost function employed in the cluster selection process. In addition, it may not be computationally feasible to explore

all possible clusters for a complex circuit. The lack of an efficient and systematic way of generating clusters makes this approach impractical.

2.2 Test Hardware Allocation Problem

Once kernels in a circuit are identified, test hardware is allocated so that each kernel can obtain test patterns and its output responses can be observed during test. Test hardware contributes to the area overhead of a BISTable circuit. In addition, it increases circuit delays and may degrade the circuit performance. Therefore, it is desirable to minimizes the amount of test hardware employed in a BISTable circuit.

Abadir and Breuer[4] developed the pioneering TDES system to employ the I-path concept in allocating test hardware. Kernels in a circuit are processed sequentially. During the processing of the ith kernel in a circuit ($i > 1$), the system tries to minimize test hardware by reusing the test hardware already allocated to kernels $1, 2, \ldots, (i-1)$. Bhawmik[5] developed an integrated system for designing BISTable circuits using the BILBO TDM and the I-path concept. In this system a global approach is employed to minimize test hardware. This approach formulates the minimization problem as an integer linear programming (ILP) problem and heuristics are employed in solving the ILP formulation. The only objective in [5] is to minimize the area overhead, which may not always be the best criterion.

2.3 Test Scheduling Problem

Given a BISTable circuit and its associated test hardware, it is necessary to generate a test schedule for the circuit so that no test hardware conflicts occur during the testing of the circuit. The test scheduling problem to minimize test time has been proven to be NP-complete[21] and several heuristic approaches have been proposed.

Craig *et al.*[21] formulated the minimal test time scheduling problem as a clique covering problem, where kernels have equal test lengths. Due to the complexity of solving a clique covering problem, a heuristic procedure employing a graph coloring

technique is presented. For the case of unequal test lengths, a transformation is used that partitions the tests for the kernels into equal length sub-tests, and the procedure for the equal test length problem is then employed. It is obvious that if the longest test length is much greater than the shortest test length, the problem becomes intractable. For example, if kernel a requires 2^{20} test patterns and kernel b requires 2^{10} test patterns, then the test for a has to be partitioned into at least 2^{10} sub-tests.

Chen[22] employed a weighted graph partitioning approach to solve the minimal test time scheduling problem. The optimal graph partitioning problem is NP-complete[23] and a search procedure is employed in this approach. The assumption that every test must start its execution at the same time within a test session is employed which often leads to solutions that are far from optimum. The graph partitioning approach was later extended by Chen and Yuen[24] to reduce test time. However, due to the NP-completeness of the problem of graph partitioning, the extended approach is not efficient. No experimental results and computational complexity of the presented procedure are reported.

Jone *et al.*[25] employed a 2-step procedure in solving the test scheduling problem. The first step of the procedure generates cliques from a test compatibility graph. The second step of the procedure produces a scheduling tree from the cliques and the tree is then translated into a test schedule. Since clique enumeration is a NP-complete problem[23], a heuristic approach is employed in the first step. The scheduling tree is a binary tree that restricts the applicability of this test scheduler to only test lengths of 2^k, where k is an integer. The complexity of the procedure presented in [25] is $O(ne)$, where n is the number of nodes (i.e. tests) in a test compatibility graph and e is the number of edges in the same graph. Garg *et al.*[26] proposed a heuristic test scheduling procedure similar to that in [25]. The results obtained are also similar to those obtained by using the procedure in [25]. However, the complexity of the procedure presented in [26] is $O(n^3)$, which is more complex than the procedure presented in [25].

All the existing test schedulers assume a fully specified BISTable circuit is given, namely every kernel has been made BISTable by allocating the required test hardware. This assumption implies that they can only be used as a post processor.

In other words, test hardware must be allocated first for every kernel in a circuit before generating a test schedule. Therefore it is not possible to determine if the test time of a BISTable circuit exceeds a designer's constraint before the entire design is generated.

2.4 Design Space Exploration Problem

As mentioned above, the TDES[4] system explores the design space of a CUC by sequentially processing each kernel in the circuit. For each kernel, all possible embeddings are first constructed and evaluated by a score function. The best embedding for the kernel under consideration according to the score function is recommended by the system. The designer can either accept this embedding or select a different embedding based on the design constraints, and repeat this process for the next kernel. If there is no acceptable embedding for this kernel, the exploration process backtracks to the previous kernel considered to select a different embedding. The sequential nature of this process does not have a global view of the design space, hence it is often difficult and inefficient to find the best BISTable design.

Abadir[27] extended TDES to the more complicated TIGER (Testability Insertion Guidance Expert) system. The kernel identification feature[20] discussed in the previous section has been included in the TIGER system. A test plan is automatically generated for each testable design so that a test controller or ATE can execute the test using the test plan generated. However, the TIGER system adopts the same philosophy employed in the TDES system, namely kernels are processed sequentially in a local exploration fashion. Therefore the system does not have a global view of the design space, and thus it has similar problems as those encountered by the TDES system.

Kim *et al.*[7] developed the BIDES system where BISTable circuits are generated using a variety of TDMs and test strategies. The BISTable circuit design process consists of an initial design and subsequent redesign steps if the initial design is not acceptable. Techniques in AI planning are employed for backtracking in the redesign steps. Similar to the TDES system, kernels in a circuit are also

processed in sequence and a time consuming search procedure is required in the redesign steps. The BIDES system requires frequent user interactions to direct its search procedure and it is not clear if the best BISTable circuit can be generated in this manner.

2.5 Design Selection Problem

During the process of designing a testable circuit, there are often many choices in various steps. For example, a TDM has to be selected so that kernels in a CUC can be embedded into this TDM. Once a TDM is selected there are usually many different ways to embed each kernel into the selected TDM. This leads to many testable versions of the original circuit. When the number of choices is large and trade-offs exist between choices, to select the best choice is usually not a straightforward process.

Zhu and Breuer[28, 29, 30] developed an expert selection system (PLA-ESS) to help a designer select a TDM for an arbitrary PLA(programmable logic array). TDMs are evaluated using a score function to convert incommeasurable attribute values such as fault coverage, area overhead and extra I/O pins to a unified measure. A reason-directed backtracking mechanism is employed to explore a selection space. Expert knowledge such as score functions and various TDMs are acquired from test engineers and stored in a knowledge base. During a selection process a score function can be customized by a designer based on his/her preferences. However, PLA-ESS is a dedicated system that can only be used to select TDMs for PLAs. In addition, the score function customization process is very complicated. Thus it is not easy for a non-expert user to meaningfully tailor a score function.

A simplified selection sub-system is employed in both the TDES and TIGER systems to help a user evaluate different testable designs and select the best design based on the design constraints. The selection sub-system in either of the above two systems is less sophisticated but easier to use than PLA-ESS.

Chapter 3

The BIBS TDM

3.1 Introduction

Area overhead and performance degradation are two of the most significant costs incurred in designing BISTable circuits. Conventional BIST TDMs usually partition a circuit into disjoint combinational logic blocks, each being a kernel during the test. Therefore, every combinational logic block in a circuit has to be driven by a test register (TPG) and drive a test register (SA). Today's high performance circuits are usually optimized by increasing the layout density and the clock frequency. In such circuits, the extra test hardware required by a conventional BIST TDM often substantially increases the chip layout area and penalizes circuit performance. To reduce the amount of test hardware required by a BISTable circuit, sequential kernels are allowed in the techniques presented in [8, 31]. However, the technique presented in [31] may not achieve a satisfactory fault coverage or may require excessive test length to ensure a comprehensive fault coverage. On the other hand, the restrictions imposed by the technique in [8] on sequential kernels allowed often preclude the existence of sequential kernels in many data path circuits, thus leads to similar area overhead and performance degradation of conventional BIST TDMs.

In this chapter we present a novel BIST TDM, called *Built-In test for Balanced Structure* (BIBS), that can significantly reduce test hardware and performance degradation. In addition, high fault coverage and acceptable test time can usually be obtained. A class of sequential circuits, namely *balanced BISTable structures*,

are employed by the BIBS TDM. These structures can be found in many data path circuits. In this thesis the technique presented in [8] will be referred to as the K&A TDM. We will show that the K&A TDM is a special case of the BIBS TDM. CBILBO registers[11] are only used when necessary in this TDM since these registers introduce a significant amount of hardware overhead

In this chapter we will first present the motivation for developing the BIBS TDM. The concepts of k-pattern detectable faults and k-step functionally testable circuits are introduced. The circuit graph model employed is then described and the definitions of a balance BISTable structure and the BIBS TDM are presented. The properties associated with a balanced BISTable structure are also studied. Finally we will present both theoretical and experimental comparison between the BIBS and the K&A TDMs.

3.2 Motivation

Increasing the size of kernels usually leads to fewer kernels and thus less test hardware. However, as the size of a kernel increases and sequential kernels are formed, it becomes more difficult to excite faults and propagate fault effects to an observable output. This may lead to a decrease in fault coverage, an increase in test length, and an increase in TPG complexity. It is important to identify kernels that minimize the above problems.

A synchronous sequential circuit is said to be *balanced* if it is acyclic and the sequential lengths of all paths between every pair of combinational blocks in the circuit are the same. Testability of a sequential kernel may be reduced due to circuit imbalance. For example, consider the circuit shown in Figure 3.1, where C is a combinational logic block, F is a fanout block fed by a primary input PI, and R is a register. This circuit is unbalanced since the two paths from F to C have different sequential lengths. To detect a fault f in C may require that a test pattern consisting of two vectors, u and v, be applied at the inputs to C. This in turn requires that a test sequence of two vectors (v followed by u) be applied at PI. In general, some faults in an unbalanced circuit require a test sequence for their detection. On the other hand, it has been shown that all detectable stuck-at

faults in balanced circuits are *single pattern detectable*[19]. An arbitrary stuck-at fault in such a circuit is tested by applying a test pattern to the inputs of the circuit, clocking the circuit one or more times allowing data to propagate through the circuit, and finally observing the output response. Note that this property of single pattern testability assumes all registers consist of D-type flip-flops(F/F). Single pattern testability is usually no longer guaranteed when designs contain F/Fs that have a hold mode of operation, such as JK or SR F/Fs. In general, a fault is *k-pattern detectable* if there exists an input sequence of length k (or less) that will detect this fault. Every detectable stuck-at fault in the circuit shown in Figure 3.1 is 2-pattern detectable.

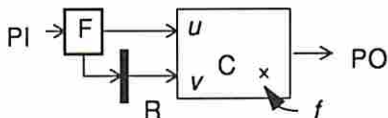


Figure 3.1: An unbalanced circuit

Consider the circuit shown in Figure 3.2 where C_1 and C_2 are combinational logic blocks and assume that R_1 and R_2 are n-bit registers. By applying all 2^n patterns at R_1 we say that C_2 is tested *1-step functionally exhaustively* even though the output of C_1 may not apply all possible 2^n patterns at the input of C_2. The concept of *1-step* follows because each detectable stuck-at fault only requires a single test pattern for its detection. The phrase *functionally exhaustive* is used since any pattern that can be applied under functional operation can also be applied during the test mode. In fact C_1 is tested both exhaustively and 1-step functionally exhaustively. This circuit is said to be *1-step functionally testable*. That is, by applying all possible test patterns at PI, all detectable stuck-at faults are detected. In practice, it may not be necessary or feasible to apply all possible test patterns to a 1-step functionally testable circuit. For example, in many data path circuits that are 1-step functionally testable, a small portion of the functionally exhaustive test set suffices to detect every detectable stuck-at fault. In addition, when the input width of a kernel is large, say $n = 40$ in Figure 3.2, it may not be feasible to apply all possible test patterns to the kernel.

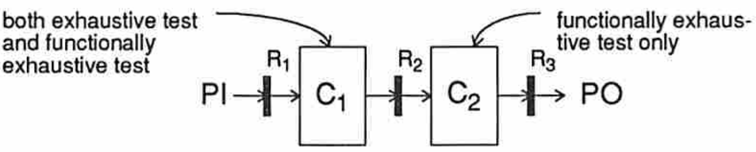


Figure 3.2: A 1-step functionally testable circuit

The circuit shown in Figure 3.1 is 2-step functionally testable since by applying all possible pairs of test patterns at PI, all detectable stuck-at faults are detected. In general, an acyclic circuit is said to be *k-step functionally testable* if for each detectable fault that does not modify the sequential aspects of the circuit, there exists a test sequence of length k (i.e. consists of k vectors) that detects this fault. Note that after applying the last test sequence it may be necessary to apply d clocks to flush the data through the circuit, where d is the sequential depth of the circuit.

A circuit that is not 1-step functionally testable may be considerably harder to test than a 1-step functionally testable circuit since for the former case sequences of test patterns are required to ensure a comprehensive fault coverage[19, 32]. The difficulty of testing k-step functionally testable circuits, where $k > 1$, is further increased if the circuit is to be self-testable. Consider the 1-step functionally testable circuit N_1 and the 2-step functionally testable circuit N_2 shown in Figures 3.3(a) and (b), respectively. The only difference between N_1 and N_2 is the absence of R_2 in N_2. Suppose a detectable stuck-at fault f in C_2 requires test vectors u_1 and u_2 at inputs of C_2 for its detection. Note that u_1 and u_2 need not be unique. Let v_1 be a test pattern generated by the TPG that produces u_1 and u_2 at the outputs of C_1. Again v_1 need not be unique. Let $\mathrm{S}(v_1)$ denote the set of such test patterns (see Figure 3.3(a)). Let $\mathrm{P}_1(f)$ be the probability that a random pattern generated by the TPG detects f in N_1. Then $\mathrm{P}_1(f) = |\mathrm{S}(v_1)| \,/\, 2^n$, where we ignore issues dealing with the all 0 test pattern. For the fault f in N_2 to be detected a sequence of test patterns v_2 and v_3 may be required as shown in Figure 3.3(b), where v_2 at time $(t-1)$ produces u_1 and d (don't care) at the outputs of C_1 and v_3 at time t produces d and u_2 at the outputs of C_1. Let $\mathrm{S}(v_2)$ denote the set of patterns that produce u_1 and d at the outputs of C_1, and $\mathrm{S}(v_3)$ denote the set of patterns that produce

d and u_2 at the outputs of C_1 (see Figure 3.3(b)). The probability that f in N_2 is detected, denoted by $P_2(f)$, is the conditional probability that the TPG generates $v_3 \in$ S(v_3P at time t, given that $v_2 \in S(v_2)$ is generated at time $(t-1)$. The set of test patterns that satisfy the above condition is illustrated in Figure 3.3(b) as $S(v_3|v_2)$. Note that $P_2(f)$ can be 0 even if f affects the normal operation of N_2 since sequence of test patterns generated by a TPG using a conventional LFSR only cover a small fraction of the pairs of patterns of the form $(a(t), b(t+1))$. Therefore if conventional LFSRs are used as TPGs, the fault coverage may be low due to faults that require sequence of test patterns for their detection. Special TPG designs that can generate a large subset of all the test sequences of length k are complicated, even for $k = 2$[33]. The area overhead required for such TPGs and the test application time are usually excessive and unacceptable. On the other hand, an LFSR configured using a primitive polynomial can generate all possible test patterns (except the all-0 pattern). Therefore $P_1(f) > 0$ if f is detectable, thus every fault that interferes with normal function is guaranteed to be detected when exhaustive patterns are applied to N_1. Similar analysis can be done to show that 1-step functionally testable circuits are superior to k-step $(k > 2)$ functionally testable circuits in terms of their fault coverage.

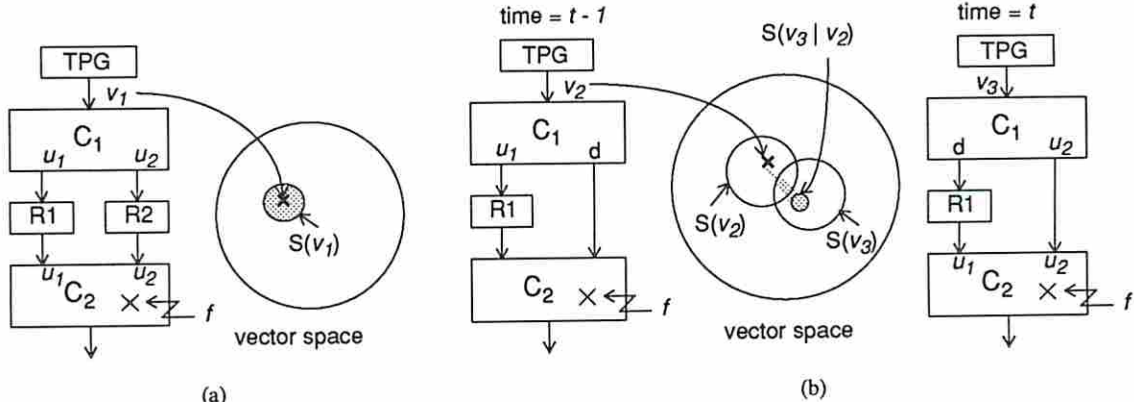


Figure 3.3: (a) a 1-step functionally testable circuit; (b) a 2-step functionally testable circuit

Next we perform a series of experiments to verify the above observation. In each experiment two circuits, a 1-step functionally testable circuit and a 2-step

Circuit	# PI	# PO	# gate	# F/F	C_2 circuit	% random resistant fault in C_2	# detectable fault
N1.1/N1.2	12	14	222	24/12	in5	12.2%	645/651
N2.1/N2.2	12	102	516	24/12	cps	21%	708/628
N3.1/N3.2	13	11	402	21/11	bc0	4.5%	887/798
N4.1/N4.2	13	25	943	33/16	c1908	not measured	688/639

Table 3.1: Circuit characteristics of the experiment

functionally testable circuit, are fault simulated using pseudo-random patterns. The circuit configurations in Figure 3.3 are employed in this experiment. The characteristics of the circuits are summarized in Table 3.1. Each row describes two circuits, one being 1-step functionally testable (denoted by N*i*.1) and one being 2-step functionally testable (denoted by N*i*.2). In each circuit C_1 is a random combinational logic block synthesized given the specification on the number of inputs (i.e. number of inputs of N*i*), the number of outputs (i.e. the number of inputs of the corresponding C_2), and a gate count limit of 50. C_2 is chosen from a set of benchmark circuits with or without random resistant faults. The identification of random resistant faults is based on the technique developed by Lempel *et al.*[34]. The number of F/Fs (i.e. F/Fs in R_1 and R_2) in each circuit is shown in column 6. Column 7 shows the percentage of random resistant faults in C_2, namely the number of random resistant faults over the total number of faults in C_2.

Simulation results for each experiment are summarized in Tables 3.2, 3.3, 3.4, and 3.5. For each circuit 10 fault simulation runs are performed for every given test length in the table, and the fault coverage obtained for a given test length is computed as the average of the results from these 10 simulation runs. The fault coverage vs. test length curves are shown in Figures 3.4, 3.5, 3.6, and 3.7.

It can be observed from the experimental results that when random resistant faults exist, the fault coverage for 2-step functionally testable circuits are significantly inferior to that for 1-step functionally testable circuits. Note that 100% fault coverage may not be achieved for 2-step functionally testable circuits even

# pattern	fault coverage (%) for N1.1	fault coverage (%) for N1.2
50	76.82	56.07
100	86.58	64.04
200	91.32	68.82
300	93.68	73.43
400	95.53	77.15
500	97.37	80.03
1,000	99.47	86.79
2,000	100	91.86
3,000	100	94.48
4,096	100	96.31

Table 3.2: Simulation results for circuits N1.1 and N1.2

# pattern	fault coverage (%) for N2.1	fault coverage (%) for N2.2
50	82.60	75.57
100	90.01	82.55
200	94.75	86.61
300	96.73	88.07
400	97.76	89.77
500	98.11	90.42
1,000	99.66	91.23
2,000	100	93.34
3,000	100	94.48
4,096	100	94.72

Table 3.3: Simulation results for circuits N2.1 and N2.2

# pattern	fault coverage (%) for N3.1	fault coverage (%) for N3.2
50	73.92	61.01
100	81.62	70.32
200	88.61	77.62
300	92.11	81.62
400	94.25	84.81
500	95.49	87.12
1,000	98.31	92.54
3,000	100	97.96
5,000	100	98.85
8,192	100	99.56

Table 3.4: Simulation results for circuits N3.1 and N3.2

# pattern	fault coverage (%) for N4.1	fault coverage (%) for N4.2
50	82.50	83.52
100	88.89	90.11
200	92.95	93.13
300	94.98	94.99
400	95.56	95.54
500	96.54	96.02
1,000	98.39	97.39
3,000	99.84	98.56
5,000	100	99.11
8,192	100	99.31

Table 3.5: Simulation results for circuits N4.1 and N4.2

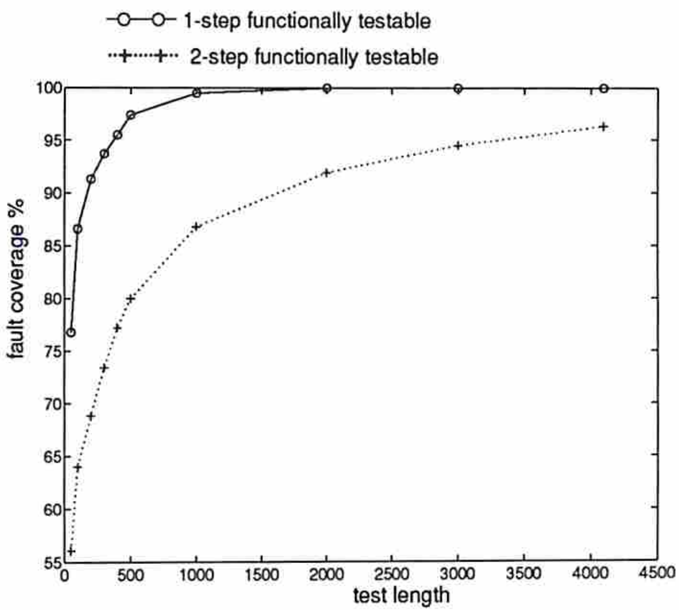


Figure 3.4: Fault coverage vs. test length curves for circuits N1.1 and N1.2

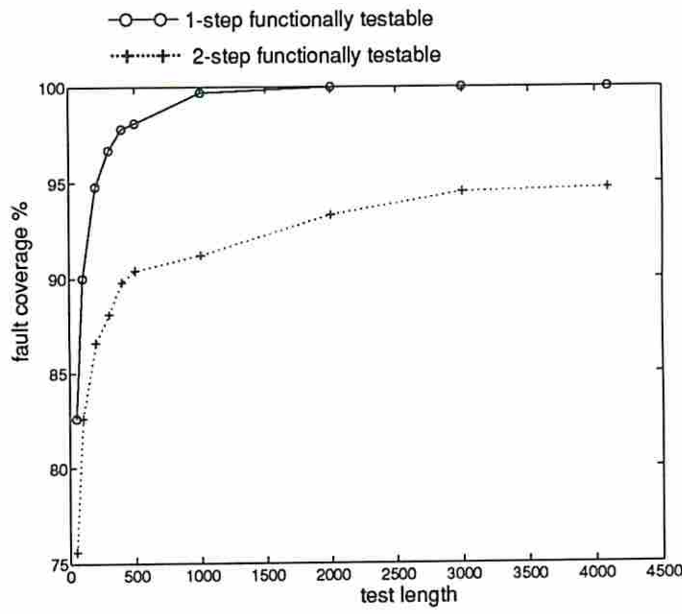


Figure 3.5: Fault coverage vs. test length curves for circuits N2.1 and N2.2

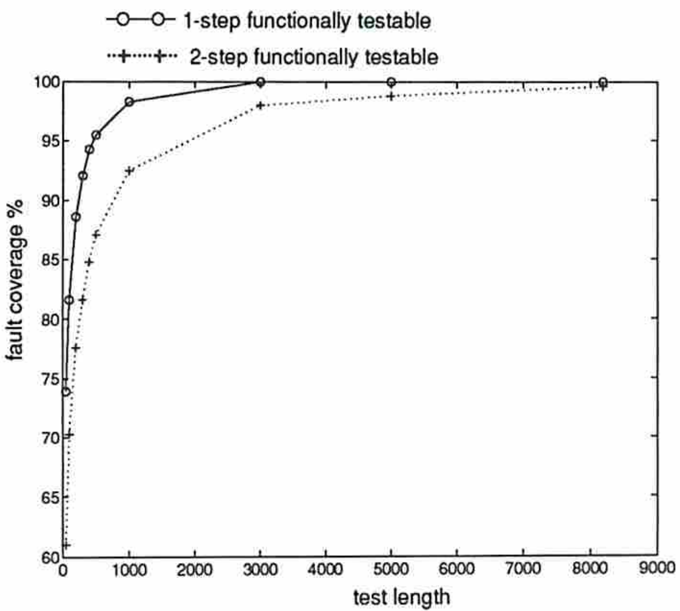


Figure 3.6: Fault coverage vs. test length curves for circuits N3.1 and N3.2

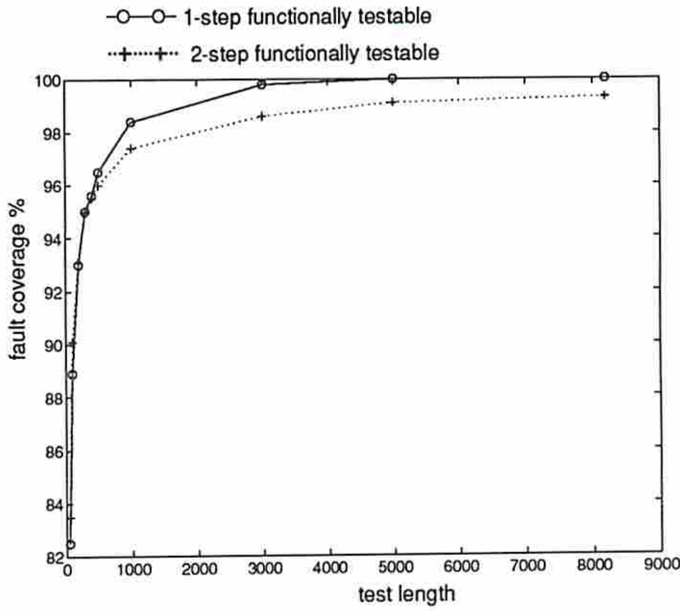


Figure 3.7: Fault coverage vs. test length curves for circuits N4.1 and N4.2

by applying exhaustive patterns. In addition, a good fault coverage, say greater than 99.5%, can often be obtained by applying a small subset of exhaustive patterns to 1-step functionally testable circuits. The difference in fault coverage is less significant when no random resistant faults exist. Several experiments using different circuit configurations and benchmark circuits have also been performed and similar results are obtained. More complex circuit configurations are shown in Figure 3.8. The circuits shown in Figures 3.8(a), (b), and (c) are 1-step, 2-step, and 3-step functionally testable respectively. C_1 and C_2 in these circuits are synthesized with the specifications of number of inputs, number of outputs, and a gate count limit of 50. *in*5 and *cps* are the same as described above. The fault coverage vs. test length curves for these three circuits are shown in Figure 3.9. Again the fault coverage for the 1-step functionally testable circuit grows much faster than that of the other two circuits. In addition, neither the 2-step functionally testable circuit nor the 3-step functionally testable circuit achieve 100% fault coverage after exhaustive test patterns have been applied.

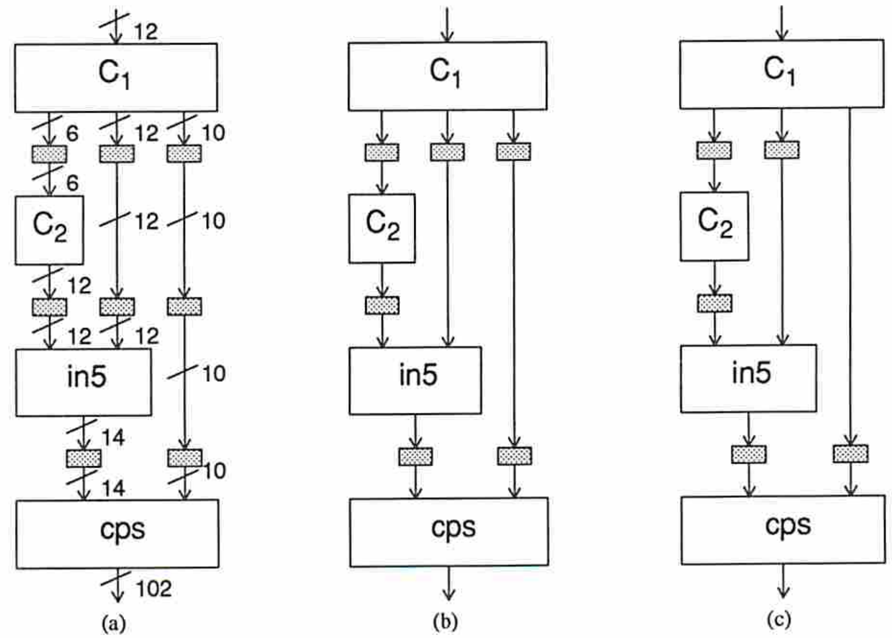


Figure 3.8: (a) 1-step; (b) 2-step; (c) 3-step functional testable circuits

Most data path circuits are too complex to be tested as a single kernel. In

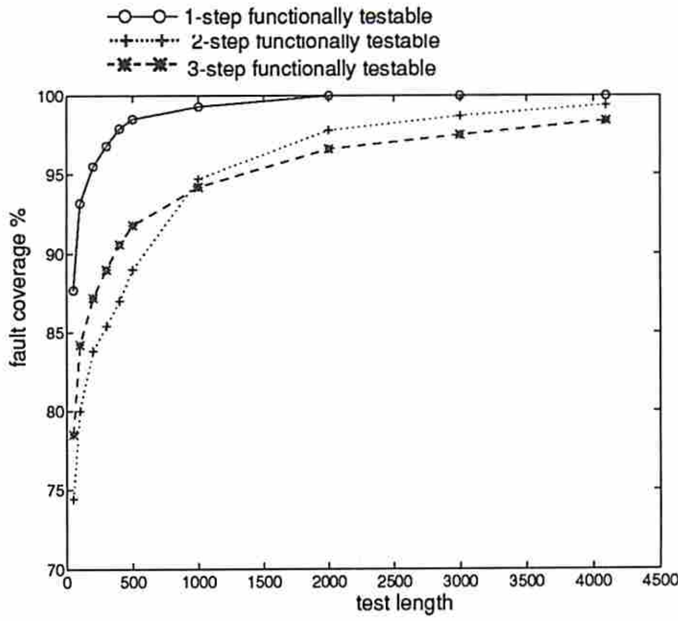


Figure 3.9: Fault coverage vs. test length curves for circuits in Figure 3.8

such cases, certain registers in a circuit can be converted to be BILBO registers, which essentially partitions the circuit into smaller kernels. Next we will show that when converting registers in a complex circuit to BILBO registers, it can be advantageous to select these registers so that the resulting kernels are 1-step functionally testable. Consider the circuit shown in Figure 3.10(a), where the number in parentheses after each register indicates its width. `in5` and `cps` are the same as before. 8.6% of the detectable stuck-at faults in the combinational logic block `exep` have been identified as being random pattern resistant[34]. Consider three different circuit configurations, S_1, S_2 and S_3, that are shown in Figures 3.11(a), (b), and (c). In each case two registers (in addition to PI/PO registers) are converted to BILBO registers. In S_1, R_4 and R_5 are BILBO registers; in S_2, R_3 and R_5 are BILBO registers; and in S_3, R_6 and R_7 are BILBO registers. Each configuration partitions the original circuit into two kernels as illustrated by the shaded blocks in Figure 3.11. Kernels $K_{1,1}$, $K_{1,2}$, $K_{2,1}$, $K_{2,2}$, and $K_{3,2}$ are 1-step functionally testable; and kernel $K_{3,1}$ is 2-step functionally testable.

Each circuit configuration is tested using pseudo-random test sequences of a

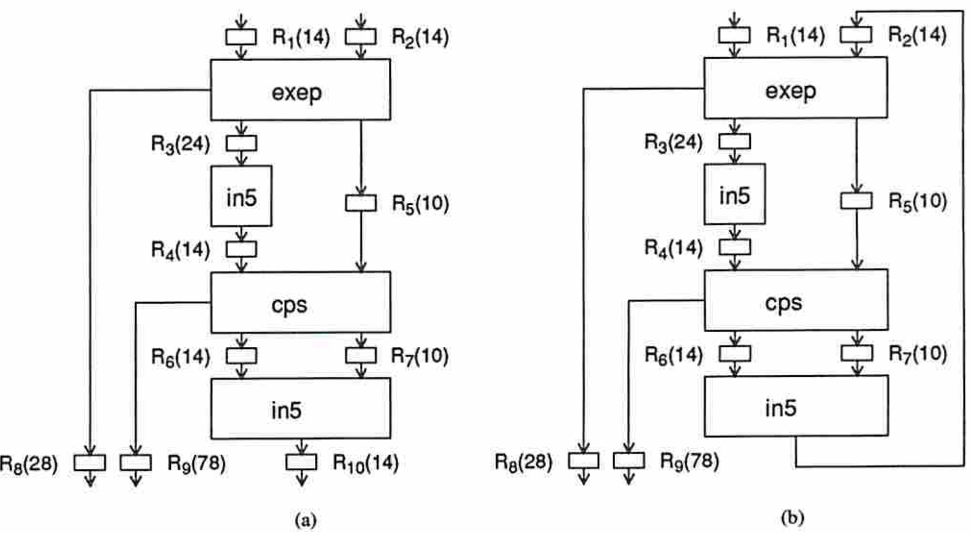


Figure 3.10: (a) and (b) example complex circuits

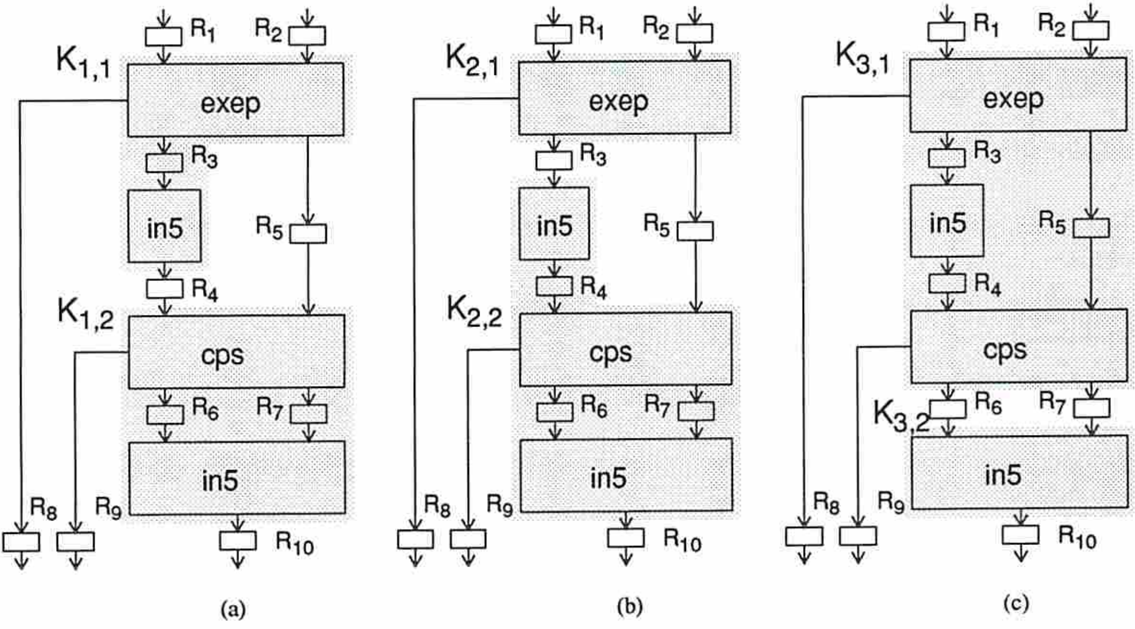


Figure 3.11: (a) S_1; (b) S_2; (c) S_3

variety of test lengths. For each test length five different sequences of pseudo-random patterns are applied to each configuration and the results are summarized in Table 3.6. In Table 3.6 for each circuit configuration, the first column shows the average (over five experiments) number of total faults, denoted by nf, that are detected by applying the designated number of pseudo-random patterns to each kernel in the configuration. We ignore aliasing in this experiment. That is, the output responses from each kernel are assumed to be observed after applying every test pattern. In the second column under each configuration, nf is normalized by dividing it by the total number of detectable stuck-at faults in the individual combinational logic blocks. The normalized number, in percent, is denoted by FC_1. The numbers of detectable stuck-at faults in `exep`, `in5`, and `cps` are 995, 630, and 1823, respectively. These, for example, would be the number of faults detected in a full scan version of the original circuit. Thus the total number of detectable stuck-at faults in these blocks is $995+630+630+1823 = 4078$. Note that FC_1 represents the fault coverage of a circuit configuration assuming that every detectable stuck-at fault in the individual combinational logic blocks remains detectable in the BIST version of this circuit. This is usually not true due to the circuit redundancy introduced by such factors as reconvergent fanout, feedback, and feedforward paths. Therefore, the actual coverage of single stuck-at faults that can affect the normal operation of the original circuit is normally greater than FC_1. Due to the lack of a powerful sequential ATPG program, we are unable to obtain an accurate count of the number of detectable faults in the original circuit. As can be observed from the table, significantly more faults can be detected in S_1 and S_2 than in S_3, which may lead to higher coverage of detectable faults. The hardware cost of S_1 and S_3 are similar. In most cases S_2 leads to better fault detection than S_1, while the hardware cost of S_2 is greater than that of S_1.

In the third column under each configuration, nf is normalized by dividing it by the number of detectable faults in each circuit configuration. FC_2 is thus the coverage of the detectable faults in each configuration. Clearly the numbers of detectable faults between different configurations can be different. To calculate the number of detectable faults in each configuration, ATPG programs are used to test each kernel in the configuration. Note that the kernels in both S_1 and S_2 and the

K	S_1			S_2			S_3		
($\log_2$ test length)	nf	FC_1(%)	FC_2(%)	nf	FC_1(%)	FC_2(%)	nf	FC_1(%)	FC_2*(%)
5	1227.8	30.11	41.83	1197.8	29.37	39.04	935.2	22.93	46.34
6	1444.6	35.42	49.22	1419.0	34.80	46.25	1055.6	25.89	52.31
7	1620.0	39.73	55.20	1548.0	37.96	50.46	1173.4	28.77	58.15
8	1764.6	43.27	60.12	1749.8	42.91	57.03	1285.2	31.52	63.69
9	1886.0	46.25	64.26	1921.8	47.13	62.64	1402.2	34.38	69.48
10	1959.2	48.04	66.75	2107.8	51.69	68.70	1533.0	37.59	75.97
11	2031.6	49.82	69.22	2300.0	56.40	74.97	1625.2	39.85	80.54
12	2114.4	51.85	72.04	2460.8	60.34	80.21	1731.0	42.45	85.78
13	2205.2	54.08	75.13	2526.8	61.96	82.36	1794.0	43.99	88/90
14	2309.2	56.63	78.68	2656.6	65.14	86.59	1860.6	45.63	92.20
15	2391.2	58.64	81.47	2763.6	67.66	90.08	1900.8	46.61	94.19
16	2515.6	61.69	85.71	2811.0	68.93	91.62	1932.4	47.39	95.76
17	2636.8	64.66	89.84	2887.0	70.79	94.10	1958.2	48.02	97.04
18	2728.0	66.90	92.95	2961.4	72.62	96.53	1958.8	48.03	97.07
19	2801.2	68.69	95.43	3029.5	74.29	97.41	1962.4	48.12	97.24
20	2851.0	69.91	97.14	3075.5	75.42	98.89	1963.0	48.14	97.27
21	2876.6	70.54	98.01	3080.2	75.53	99.04	1963.0	48.14	97.27

* Estimated values, may be greater than actual values

Table 3.6: Results of testing S_1, S_2, and S_3

kernel $K_{3,2}$ in S_3 are 1-step functionally testable. Their combinational equivalents can be obtained by ignoring the registers (if any) in the kernels. A combinational ATPG program is then used to test these kernels and the numbers of detectable faults can be obtained. The kernel $K_{3,1}$ in S_3 is 2-step functionally testable and thus a sequential ATPG program has to be employed to compute the number of detectable faults. Due to the large number of faults that were aborted during the ATPG process, the number of faults detected by the sequential ATPG program is substantially less than the actual number of detectable faults. In fact, the number of faults detected by the ATPG program we used is less than the number of faults detected by using a sequential fault simulator and applying a large number (2^{21}) of pseudo-random patterns. Therefore, an estimate of the number of detectable faults in $K_{3,1}$ is obtained by repeatedly using a sequential fault simulator and increasing the lengths of pseudo-random test sequences until no significant increase in the number of detected faults is observed. Note that this estimated number may still be inaccurate since certain faults in $K_{3,1}$ may require specific pairs of patterns for their detection. The accuracy of this number can be guaranteed only if all possible (2^{56}) pairs of patterns are applied to $K_{3,1}$, which is infeasible. The computed numbers of detectable faults in S_1 and S_2 are 2935 and 3110, respectively. The estimated number of detectable faults in S_3 is 2018. It can be observed that good fault coverage can be obtained by a test set much smaller than the exhaustive test set for all three configurations. The estimated values of FC_2 for S_3 are better than the (actual) FC_2 values for S_1 and S_2 in most cases due to fewer detectable faults (estimated) in S_3. Note that the FC_2 values for S_3 may be greater than the actual values since the estimated number of detectable faults is a lower bound. Although S_3 has better FC_2 values for most test lengths in this experiment, the number of faults detected in S_3 is significantly less than that detected in S_1 or S_2. Therefore, we conjecture that S_1 and S_2 lead to better coverage of the faults that affect the normal operation of the circuit. In addition, complete coverage of such faults can be guaranteed for S_1 and S_2, but not for S_3, when a functionally exhaustive test set is applied to the circuit.

A circuit similar to that in Figure 3.10(a) is shown in Figure 3.10(b), where a feedback path exists from `in5` to `exep`. Three circuit configurations, S_1', S_2', and

S_3', are obtained by using the same BILBO registers (except R_{10} in Figure 3.10(a)) as in S_1, S_2, and S_3, respectively. Clearly the kernels thus partitioned are the same as the kernels in Figure 3.11, thus leading to the same results as in Table 3.6. In the next section the criteria for selecting BILBO registers to make a circuit 1-step functionally testable will be described.

Based on the following factors, 1-step functionally testable sequential kernels are conjectured to be desirable to ensure BISTable circuits with acceptable area overhead, test time and fault coverage.

- Each detectable stuck-at fault is single pattern testable.
- Provably complete fault coverage of all detectable faults when tested with an appropriate test pattern generator (to be discussed in Chapter 4).
- Efficient TPGs can be designed that will apply all possible inputs to circuit, but not necessarily a prescribed set of sequence of length k, for $k > 1$.
- Experiments comparing 1-step vs. 2 and 3-step functionally testable circuits using non-exhaustive test sets on circuits having random resistant faults.

3.3 Balanced BISTable Kernels and the BIBS TDM

In this section we will first present a circuit graph model that is employed throughout this dissertation. The definition of a balanced BISTable kernel and its properties are then described.

3.3.1 Circuit Graph Model

As mentioned in Chapter 1, a CUC consists of RTL components such as combinational logic cells, fanout cells, register cells, PIs/POs, and the connections between them. An additional class of cells, called *vacuous cells*, are introduced. A vacuous cell simply consists of wires connecting its inputs and outputs. A vacuous cell is present between a pair of register cells when one of the register cells directly feeds

the other register cell and there is no fanout. A CUC is modeled as a directed graph $G = (V, E, w)$. $v \in V$ represents a combinational cell, a PI/PO, a fanout cell, or a vacuous cell; $e \in E$ represents a connection between two vertices through a register cell or wires; and $w : E \rightarrow \mathcal{Z}^+$ defines the weight of each edge. A vertex is called a *logic vertex*, *I/O vertex*, *fanout vertex*, or *vacuous vertex* if it represents a combinational logic cell, a PI/PO, a fanout cell, or a vacuous cell, respectively. An edge is called a *register edge* or *wire edge* if it represents a connection between vertices through register or wires, respectively. When e is a register edge, $w(e)$ denotes the width of the register, otherwise $w(e)$ is ∞ (a large number in practice). An example circuit is shown in Figure 3.12(a) where every register is assumed to be 8 bits wide. The corresponding circuit graph is illustrated in Figure 3.12(b) where wire edges are represented as bold arcs and the weights for register edges are shown next to the edges in Figure 3.12(b).

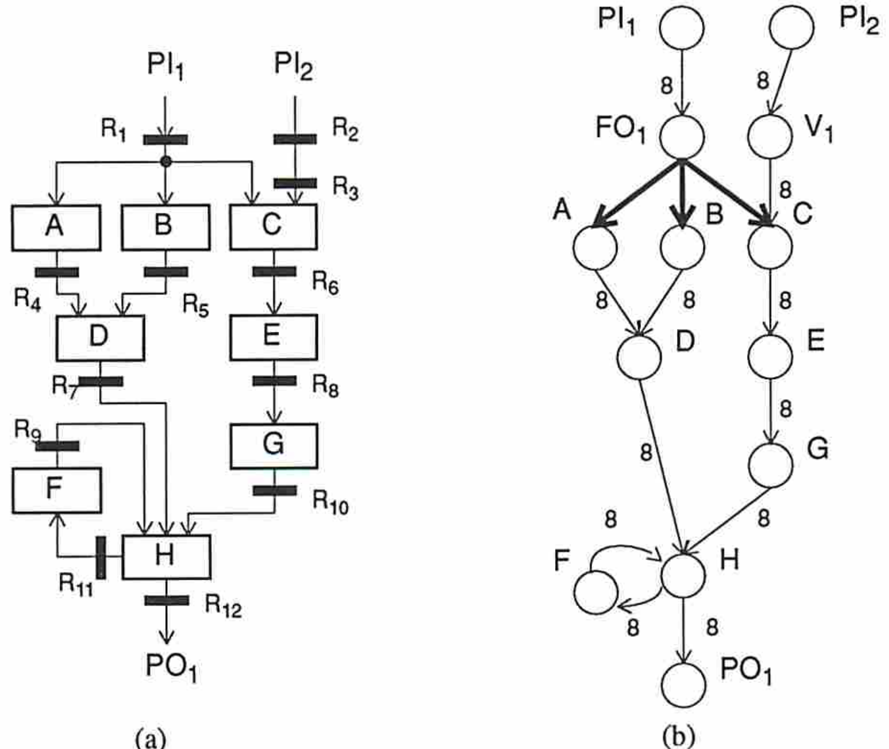


Figure 3.12: (a) An example circuit; (b) corresponding circuit graph

Notice that there is a fanout from the outputs of R_1 to the cells A, B and C in

the circuit, hence a fanout vertex FO_1 in required in the circuit graph. Between this fanout vertex and the logic vertices A, B and C are wire edges as denoted by the bold arcs. Furthermore, there is no logic between the two registers R_2 and R_3, thus a vacuous vertex V_1 is required in the circuit graph. A path in G is said to form a *cycle* if it contains at least one register edge and starts and ends at the same vertex. It should be noted that combinational cycles (i.e. cycles composed of vertices and wire edges only) are not allowed in this dissertation since they may cause a circuit to behave asynchronously. A subgraph of G is called an *unbalanced reconvergent-fanout structure (URFS)* if it contains two vertices where two or more paths between them have an unequal number of register edges. The subgraph ($V = \{F, H\}$, $E = \{(F, H), (H, F)\}$) of the circuit graph shown in Figure 3.12(b) is a cycle; the subgraph ($V = \{FO_1, A, C, D, E, G, H\}$, $E = \{(FO_1, A), (A, D), (D, H), (FO_1, C), (C, E), (E, G), (G, H)\}$) constitutes an URFS.

An *input port* of a cell C is a collection of inputs at C that are directly driven by logic in a common cell C'. Similarly an *output port* of C is a collection of outputs at C that directly drive logic in a common cell C''. Note that C must not be C' or C'', otherwise there exists a combinational cycle that is not allowed in this study. An input port is represented by an in-coming edge to a vertex in a circuit graph; and an output port is represented by an out-going edge from a vertex in a circuit graph. For example, there are two input ports and one output port on the combinational cell D in Figure 3.12(a). In its corresponding circuit graph shown in Figure 3.12(b), there are two in-coming edges and one out-going edge connecting to the vertex D. The sequential length of a path in a circuit graph is simply the number of register edges in the path.

3.3.2 Definition of Balanced BISTable Kernels

Let S be an arbitrary synchronous sequential circuit with a circuit graph $G = (V, E, w)$.

Definition 3.1 *S is said to be* **balanced BISTable** *if*

1. G is acyclic,

2. *$\forall v_1, v_2 \in V$, all directed paths (if any) from v_1 to v_2 are of equal sequential length, and*

3. *there does not exist a pair of input and output ports on S that is directly driven by and directly drives a common register.*

Clearly the first two requirements enforce S to be a *balanced structure* as defined in [35]. BALLAST is a design-for-test partial scan TDM that employs balanced structures. It has been shown in [19] that every balanced structure is 1-step functionally testable. Due to this property, only an ATPG for combinational logic is required to generate a test for each detectable fault. A procedure is presented in [35] to modify a CUC by converting a set of registers to scan registers so that the circuit becomes balanced. A scan register can operate as both a pseudo PI and a pseudo PO simultaneously during the testing of a kernel. On the other hand, a normal BILBO register can operate in either of the two modes, but not in both modes simultaneously, during the testing of a kernel. Due to this reason the third requirement is necessary to ensure that each pair of input and output ports of a kernel is driven by and drives distinct BILBO registers. This concept can be illustrated by the example below.

Example 3.1 Consider a circuit S and its corresponding circuit graph shown in Figures 3.13(a) and (b), respectively. S is not balanced since the sequential lengths of the paths from C_1 to C_3 are different. In a partial scan design a minimal cost solution to balance the circuit is to convert R_3 and R_9 to scan registers. The resulting circuit is illustrated in Figure 3.14. The BIST equivalence of this solution is to convert R_3 and R_9 (along with R_1 and R_6) to BILBO registers. However, the modified circuit is not balanced BISTable since R_3 and R_9 are used simultaneously as both a TPG and a SA. To make this circuit balanced BISTable, additional registers have to be converted to BILBO registers. One solution is to convert two additional registers, R_7 and R_8, to BILBO registers. This leads to two kernels, each being balanced BISTable, as shown by the shaded blocks in Figure 3.15. To test the circuit, two test sessions are required. In the first session kernel 1 is tested by configuring R_1 as a TPG and R_3, R_7, R_8 and R_9 as SAs. In the second session kernel 2 is tested by configuring R_3, R_7, R_8 and R_9 as TPGs and R_6 as a SA. □

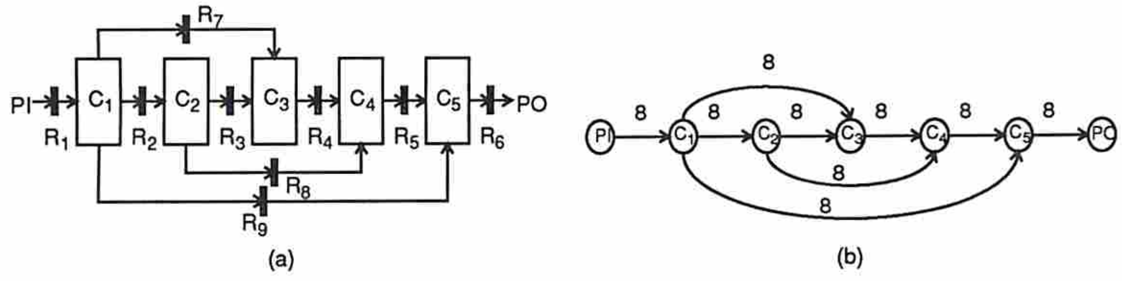


Figure 3.13: (a) Circuit for Example 3.1; (b) corresponding circuit graph

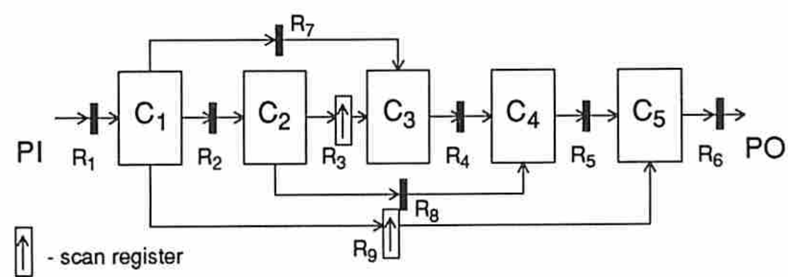


Figure 3.14: A partial scan design for circuit in Figure 3.4

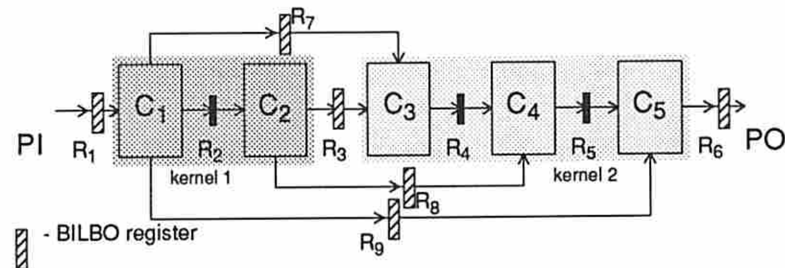


Figure 3.15: A BISTable design for circuit in Figure 3.4

BIBS is a BIST TDM that requires kernels under test to be balanced BISTable. Given a CUC, it is usually necessary to modify the circuit by converting a set of registers to BILBO registers in order to satisfy the requirements of the BIBS TDM. The modified circuit is said to be *BIBS testable*. Every edge in the circuit graph that represents a BILBO register is called a *BILBO edge*.

3.3.3 Properties of Balanced BISTable Kernels

Theorem 3.1 A balanced BISTable kernel is 1-step functionally testable.

Proof By definition a balanced BISTable kernel is a balanced structure. Therefore, the theorem follows from the results presented in [19]. □

Theorem 3.2 Given a circuit S with a circuit graph G, at least two BILBO edges are required in every cycle and URFS (if any) in G to make S BIBS testable.

Proof Clearly at least one BILBO edge is required in every cycle and URFS to balance G since a balanced circuit has to be free of cycles and URFSs. Let G' be the modified circuit graph (and S' the modified circuit) of G so that there is one BILBO edge in every cycle and URFS. Consider a cycle in G as shown in Figure 3.16(a) where (u, v) represents a BILBO edge. Let U and V be the combinational logic blocks represented by the vertices u and v, respectively, and R be the BILBO register represented by the edge (u, v). Clearly the input port of V is an input port of S' and the output port of U is an output port of S'. This pair of input and output ports of S' is fed by and feeds the same BILBO register R, thus S' is not balanced BISTable. Similarly, consider an URFS in G that contains only one BILBO edge, say (u, v), as shown in Figure 3.16(b). Again let U and V be the combinational logic blocks represented by the vertices u and v, respectively, and R be the BILBO register represented by the edge (u, v). The input port of V and output port of U form a pair of input/output ports of S' that is fed by and feeds the same BILBO register and thus S' is again not balanced BISTable. Consequently, at least two BILBO edges are required in every cycle and URFS in G to make S balanced BISTable. □

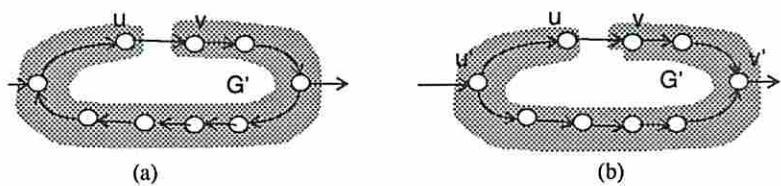


Figure 3.16: (a) A cycle with one BILBO edge; (b) an URFS with one BILBO edge

Note that if a cycle contains only one register edge, then either an extra register needs to be added in the circuit or a CBILBO register is required. In the former case, the extra register is transparent during normal functional mode and operates as an LFSR during test mode.

3.4 Comparison of the BIBS TDM and the K&A TDM

The problem of adding BILBO registers to a CUC so that the circuit can be tested *functionally exhaustively* has been addressed by Krasniewski and Albicki[8]. The criteria used in the K&A TDM for adding BILBO registers are:

1. a BILBO register is required for every input port of a combinational logic block if the block has more than one input port,

2. a BILBO register is required for every PI/PO port, and

3. at least two BILBO registers are required in any cycle of the circuit.

Theorem 3.3 The K&A TDM is a special case of the BIBS TDM.

Proof To prove this theorem it suffices to show that every circuit satisfying the three criteria in the K&A TDM consists of only balanced BISTable structures, and the converse is not always true. Assume that there exists a circuit satisfying the three criteria, but containing a non-balanced BISTable structure S. Note that S is connected. Based on the definition of a balanced BISTable structure, S must either (1) contain a cycle, (2) contains an URFS, or (3) feed and be fed by the same BILBO register.

Case 1 is clearly impossible since there will be at least two BILBO registers in any cycle of the original circuit, thus no cycles exist in the modified circuit. Since a BILBO register is required for every input port of a combinational logic block if the block has more than one input port, these BILBO registers essentially break all reconvergent fanout paths, therefore case 2 is not possible either. For case 3, suppose a BILBO register R feeds a combinational logic block V in S and is fed by another combinational logic block U in S (see Figure 3.17(a)). Since S is connected, U and V must be connected as shown in either Figure 3.17(b) or Figure 3.17(c). In Figure 3.17(b) there is a path from V to U. In other words a cycle exists before register R is converted to a BILBO register. It is necessary to convert another register in this path to a BILBO register as stated by the third criterion above. The extra BILBO register breaks the path and disconnects S, leading to a contradiction. In Figure 3.17(c) there exists a pair of combinational logic blocks U' and V' that can be the same blocks as U and V, respectively, that form a reconvergent fanout structure. Since V' has more than one input port, BILBO registers are required in front of both input ports of V'. This breaks the path between U' and V' and disconnects S. Thus the assumption that there exists a circuit satisfying the three criteria in the K&A TDM but containing a non-balanced BISTable structure is not possible. Therefore every circuit that satisfies the three criteria in the K&A TDM consists of only balanced BISTable structures.

Next consider the circuit shown in Figure 3.13(a). Clearly all nine registers in the circuit are required to be converted to BILBO registers according to the K&A TDM. On the other hand, only six BILBO registers are required by the BIBS TDM as shown in Example 3.1. Therefore a balanced BISTable structure need not satisfy the criteria in the K&A TDM and this completes the proof. □

Next we consider the example circuit employed in [8] which is shown in Figure 3.18(a). To make the circuit BISTable using the K&A TDM, 10 BILBO registers are required as shown in Figure 3.18(b). The total number of F/Fs that require modification is 52. On the other hand, only 8 BILBO registers that consists of a total of 43 F/Fs are required if the BIBS TDM is employed. Both TDMs partition the circuit into two kernels as illustrated by the shaded blocks in the figure.

A series of experiments on several data path circuits consisting of adders and

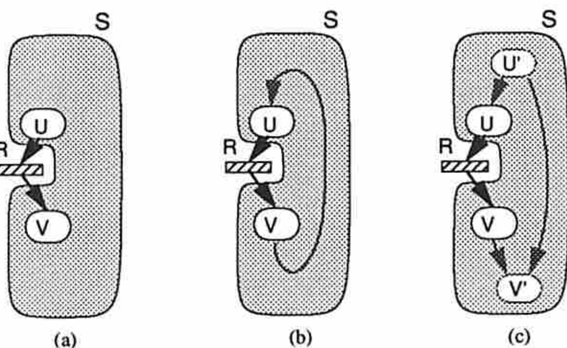


Figure 3.17: (a) A connected structure that feeds and is fed by a BILBO register R; (b) and (c) possible configurations

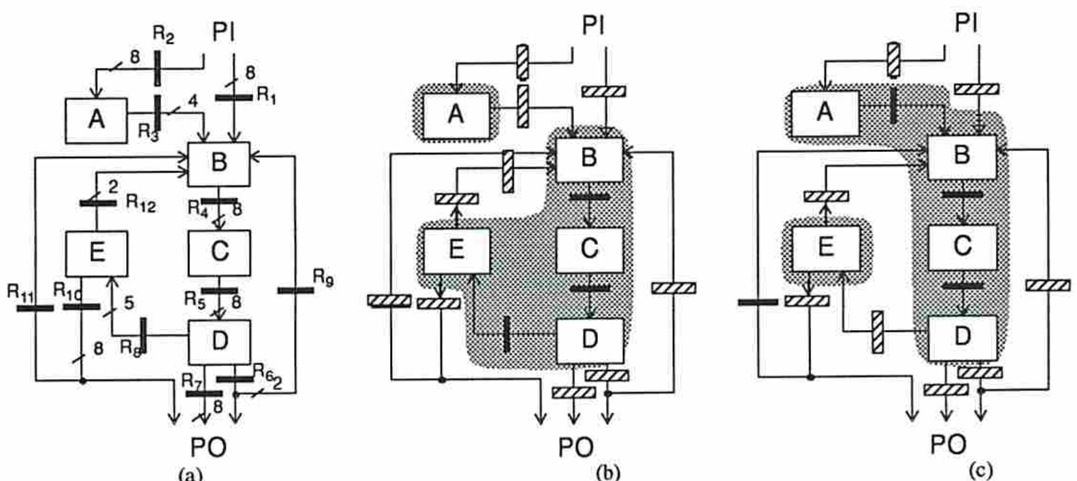


Figure 3.18: (a) The example circuit employed in [8]; (b) BISTable design using the K&A TDM; (c) BISTable design using BIBS TDM

Circuit	c5a2m	c3a2m	c4a4m
Function	o=(a+b)*(c+d)+(e+f)*(g+h)	o=((a+b)*c+d)*e+f	o=a*(f+g)+e*(b+c) p=d*(b+c)+h*(f+g)
Implementation	a b c d e f g h o	a b c d e f o	a b c d e f g h o p
# of gates	2,542	2,218	4,096

Table 3.7: Summary of the data path circuits

multipliers were performed. The circuit characteristics are summarized in Table 3.7. These circuits are portions of a digital filter design synthesized by MABAL, a CAD tool developed by the USC design automation group[36]. These data paths are all 8 bits wide.

Note that the circuits in Table 3.7 are inherently balanced BISTable. Thus only registers at the PI and PO need be converted to BILBO registers as shown by the shaded registers in the table when employing the BIBS TDM. This leads to a single kernel in the BISTable version of the original circuit. In general, a CUC may not be balanced BISTable or the test time to achieve a satisfactory fault coverage may be excessive if the input widths of kernels are large. For such cases additional BILBO registers are required. When the K&A TDM is employed BILBO registers are required to satisfy the criteria as discussed in Section 3. The results of employing the BIBS TDM and the K&A TDM are compared as summarized in Table 3.8.

In Table 3.8 the results for each circuit are divided into three columns. The first column shows the results of employing the BIBS TDM as described in this paper. The second column shows the results of employing the BIBS TDM under a kernel input width limit of 32 bits. The resulting circuits with this input width

Circuit	c5a2m			c3a2m			c4a4m		
	BIBS	BIBS*	K & A	BIBS	BIBS*	K & A	BIBS	BIBS*	K & A
# of BILBO registers	9	11	15	7	8	15	10	14	22
# of kernels	1	3	7	1	2	5	1	4	8
Input width of largest kernel	64	32	16	48	32	16	64	32	16
Test time to achieve 99.5% fault coverage	1,440	972	782	2,060	2,840	782	1,900	1,902	1,037
Test time to achieve 100% fault coverage	7,300	2,642	2,172	9,240	5,990	2,172	19,120	6,262	2,172

* - BIBS design with maximal kernel input width of 32

Table 3.8: Summary of the experimental results

constraint are shown in Table 3.9. The procedures to identify balanced BISTable kernels with or without input width constraint will be discussed in Chapter 5. The third column shows the results of applying the K&A TDM. Since the K&A TDM requires every register that feeds an input port of a multiple input combinational block to be a BILBO register, each adder and multiplier is a kernel by itself during the test. As can be seen from row 1, the number of BILBO registers required by the BIBS TDM with or without kernel input limit is much smaller (approximately 50% less) than that required by the K&A TDM. This leads to less area overhead and performance impact. In addition, the BIBS TDM leads to fewer number of kernels as shown in row 2, thus less complex test controller.

The fault coverage and test time of using these two TDMs are also compared as shown in rows 3 and 4. The fault coverage is computed using a fault simulator and applying pseudo-random patterns. Row 4 in the table shows that both TDMs achieve 100% fault coverage if a sufficient number of patterns are applied. The number of patterns required is substantially less than that required by functionally exhaustive testing. The number of patterns required by the K&A TDM is generally less than that needed by the BIBS TDM. In practice it is not always necessary to achieve 100% fault coverage. In such cases, the differences in test time between these two TDMs are substantially less as can be observed from row 3. The increase in test time for the BIBS TDM seems to be inevitable because larger and more

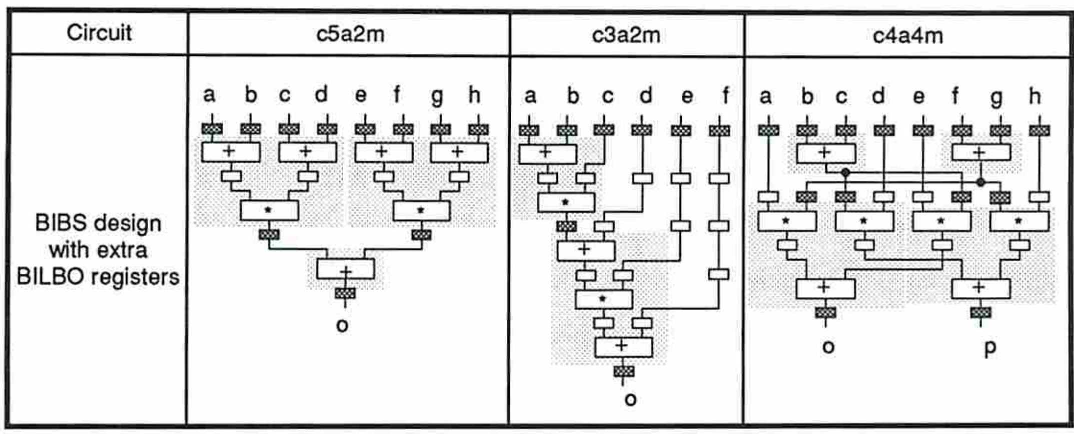


Table 3.9: Effects of adding extra BILBO registers to BIBS testable circuits

complex structures are tested as kernels. However, with the kernel input width constraint, the test time can often be substantially reduced by adding a few more BILBO registers. This implies that a designer can often trade off test time with test hardware and performance impact. In a high performance circuit where performance impact and area overhead usually have higher priority than test time, the BIBS TDM is an attractive option in making the design BISTable.

When functionally exhaustive testing is desired the differences in test time between these two TDMs may be greater since the input widths of the kernels in the BIBS TDM are usually larger. However, for the data path circuits considered, it is not necessary to apply functionally exhaustive testing to achieve 100% fault coverage. From the above experimental results it can be observed that the BIBS TDM is very attractive when the area overhead and performance impact are important attributes in a BISTable circuit. In addition, high fault coverage can be achieved if adequate test patterns are applied to a circuit under test.

3.5 Summary

In this chapter the BIBS TDM for designing BISTable circuits has been described. The BIBS TDM employs a class of synchronous sequential kernels called balanced BISTable structures. These structures can be found in many data path circuits.

The BIBS TDM requires significantly less test hardware than conventional BIST TDMs. Since less test hardware is required, the area overhead and performance degradation associated with the BIBS TDM are lower than for conventional BIST methodologies. The BIBS TDM ensures that every kernel under test is 1-step functionally testable, that is, every detectable stuck-at fault in a circuit is single pattern detectable. When all possible patterns are applied to a kernel under test, the technique guarantees the detection of all stuck-at faults in the circuit that would interfere with normal functional operation. This comprehensive fault coverage can seldom be achieved by other BIST TDMs that allow sequential kernels.

When the circuit under test mainly consists of regular structures such as adders and multipliers, which is the case in our experiments on several digital filter designs, fault coverage close to 100% can usually be obtained by applying pseudo-random patterns and the test time is often a small fraction of that required by functionally exhaustive testing. In general, the BIBS TDM provides a test engineer with a trade-off between test time and fault coverage. When functionally exhaustive patterns are applied, comprehensive fault coverage is guaranteed. On the other hand, pseudo-random patterns may be used to obtain satisfactory fault coverage in significantly shorter test length since every detectable fault in the circuit is single pattern detectable. The K&A TDM has been shown to be a special case of the BIBS TDM. Experimental results demonstrated that the BIBS TDM requires substantially less test hardware and results in less circuit delay than the K&A TDM.

The BIBS TDM employs balanced BISTable kernels that are 1-step functionally testable. When a balanced BISTable kernel has two or more input ports, a conventional LFSR may not be sufficient to provide an exhaustive test set to guarantee the above testability. Therefore, special TPGs may be required to test such kernels. The TPG design problem for balanced BISTable kernels will be considered in the next chapter.

Chapter 4

TPG Design for Balanced BISTable Kernels

4.1 Introduction

In the previous chapter we presented the BIBS TDM that ensures every kernel under test to be 1-step functionally testable. To guarantee comprehensive fault coverage, all possible patterns (i.e. functionally exhaustive patterns) should be applied to a kernel under test. When a kernel K under test is a combinational logic block with a single input port, a TPG design that configures the input register of K as a maximal length LFSR is sufficient to provide such an exhaustive test set. When K has n input ports and is driven by a set of registers R_1, R_2, ..., R_n, these registers need to be concatenated and configured as a single maximal length LFSR. When a kernel is sequential, the TPG design that can provide a functionally exhaustive test set when the sequential lengths of paths from the TPG to blocks in the kernel are unequal may be more complicated. We address the issue of TPG design for balanced BISTable kernels in this chapter.

Consider the balanced BISTable kernel shown in Figure 4.1(a). When R_1, R_2 and R_3 are concatenated and configured as a maximal length LFSR, a functionally exhaustive test set may not be obtained at the inputs of C_2 and C_3. This is due to the unequal sequential lengths of the paths from the input registers R_1, R_2 and R_3 to C_2 and C_3. To compensate for the imbalance in the sequential lengths of these paths, one and two clock cycles of delay can be added to the paths by inserting transparent registers to the circuit which balance the sequential lengths of these paths (see Figure 4.1(b)). This, however, would add significant area overhead and

the circuit performance may be adversely affected. On the other hand, the same balancing effect can be achieved by adding one D-type F/F before each of R_2 and R_3 if a type 1 LFSR[2] is employed (see Figure 4.1(c)). This modification works because in a type 1 LFSR L, the data present in the ith stage of L at time t is the same as the data present in the $(i-1)^{st}$ stage of L at time $t-1$ for $i > 1$. The area overhead for this approach is much smaller than the previous one and no additional performance degradation is introduced that would result if transparent registers were added.

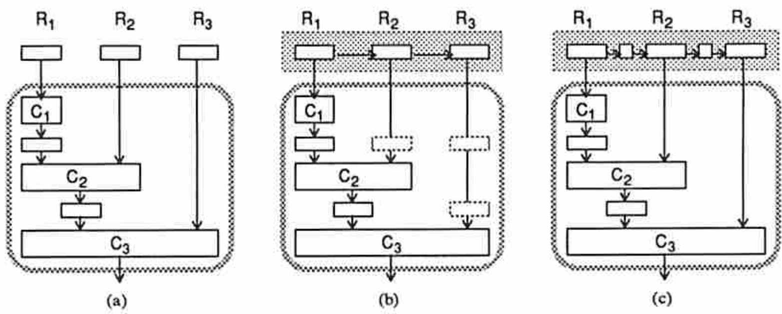


Figure 4.1: (a) a balanced BISTable kernel; (b) a simple approach to test kernel in (a); (c) proposed approach

As can be seen from the above example, more complicated TPG design may be required to balance the sequential lengths of paths in a sequential kernel. In this chapter we will first introduce a graph model to represent a balanced BISTable kernel. Balanced BISTable kernels are then classified into two categories, namely *uni-cone kernels* and *multi-cone* kernels. A novel TPG design scheme using a combination of LFSRs and shift registers(SRs) is then introduced. This scheme can be easily applied to a uni-cone balanced BISTable kernel. The TPG design scheme is then extended to deal with multi-cone balanced BISTable kernels. In multi-cone balanced BISTable kernels where the input width of each cone is less than the input width of the entire circuit, the test time for the circuits may be reduced by using a technique similar to *pseudo-exhaustive testing*[2]. A new concept, called *functionally pseudo-exhaustive testing*, will be introduced to design efficient TPGs for these circuits and the TPG design process will be presented. The proposed TPG designs will be compared with TPGs designed using simple maximal length LFSR

to demonstrate that the proposed TPGs achieve better fault coverage. Finally, we will remark on the necessity and optimality in terms of the test time and TPG hardware of the proposed TPG design scheme.

4.2 Uni-cone and Multi-cone Kernels

Given a balanced BISTable kernel K having m input ports, n output ports, and m input registers associated with the input ports. A bipartite graph (U, V, E) is employed to represent K. Each vertex $u \in U$ corresponds to an input register; each vertex $v \in V$ corresponds to an output port of K; and each edge $e \in E = U \times V$ corresponds to a data path or a set of parallel data paths from an input register to an output port of K. Note that $|U| = m$ and $|V| = n$. An integer $L(e)$ associated with the edge e is the sequential length of the data path represented by e. For example, consider the balanced BISTable kernels shown in Figures 4.2(a)-(d). The bipartite graphs representing these kernels are shown in Figure 4.3(a)-(d), respectively.

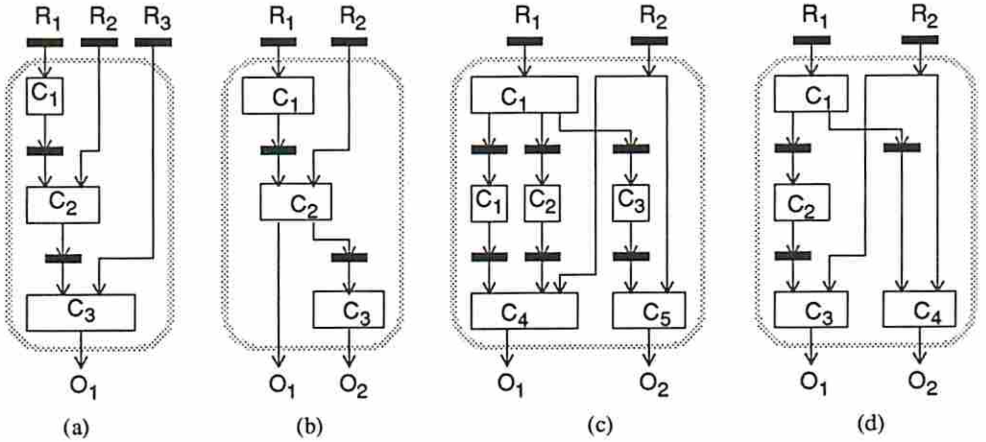


Figure 4.2: (a) - (d) example balanced BISTable kernels

A *cone* of a kernel consists of the logic associated with an output port of the kernel.

Definition 4.1 *A balanced BISTable kernel represented by a bipartite graph (U, V, E) is said to be* **uni-cone** *if*

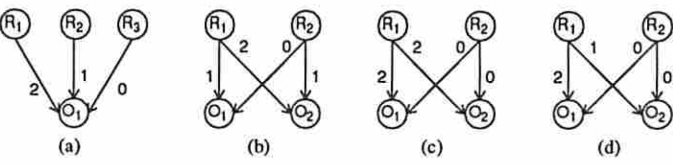


Figure 4.3: (a) - (d) bipartite graphs correspond to kernels in Figure 4.2

1. $|V| = 1$, *or*

2. for each pair of vertices v_1 *and* $v_2 \in V$*, if there exists a pair of vertices* u_1 *and* $u_2 \in U$ *so that* (u_1, v_1)*,* (u_2, v_1)*,* (u_1, v_2)*, and* $(u_2, v_2) \in E$*, then* $L((u_1, v_1)) - L((u_2, v_1)) = L((u_1, v_2)) - L((u_2, v_2))$.

Otherwise, it is a **multi-cone** *kernel.*

Clearly every balanced BISTable kernel having only one output port is a uni-cone kernel according to the first condition. For a kernel K having two or more output ports, if for every pair of output ports of K depending on the same pair of input registers, the differences in sequential lengths of the paths from these input registers to the output ports are identical, then K is also a uni-cone kernel. Based on this definition, the kernels in Figures 4.2(a), (b) and (c) are uni-cone kernels. On the other hand, the kernel in Figure 4.2(d) is a multi-cone kernel.

Lemma 4.1 Given a uni-cone kernel K that has two or more output ports. If a TPG can functionally exhaustively test an output cone of K, then it can also functionally exhaustively test the remaining output cones of K.

Proof The second condition for a uni-cone kernel along with the balance property of K ensure that the difference in sequential lengths of the paths from a pair of input registers to any combinational logic cell in K is a constant. If there exists a TPG that functionally exhaustively tests an output cone of K, then the combinational logic cells associated with the output cone are functionally exhaustively tested by this TPG. The same TPG functionally exhaustively tests every combinational logic cell in K due to the constant sequential length difference and the fact that every output cone depends on the same set of input registers. Consequently, every output cone of K is tested functionally exhaustively by this TPG. □

Based on Lemma 4.1, for every pair of output cones Ω_1 and Ω_2 (if exists) of a uni-cone kernel K, a TPG functionally exhaustively tests Ω_1 if and only if it functionally exhaustively tests Ω_2. Therefore the bipartite graph for K can be reduced by only considering an arbitrary vertex in V and its associated edges when generating a TPG for K. A reduced bipartite graph will be employed in the next section.

4.3 TPG Design for Uni-cone Kernels

As was discussed in the previous section, a uni-cone kernel K can be represented by a reduced bipartite graph as shown in Figure 4.4(a), where O is arbitrarily chosen from the output ports of K if K has more than one output port. $d_1, d_2, \ldots, d_n$ are the sequential lengths of the paths from $R_1, R_2, \ldots, R_n$, respectively, to O.

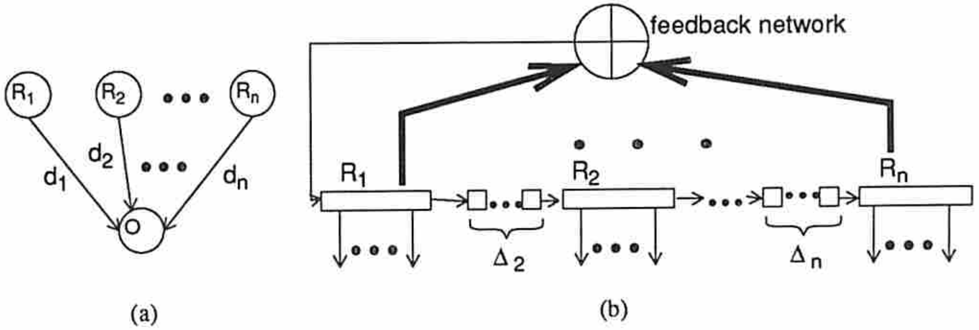


Figure 4.4: (a) a reduced bipartite graph representation of a uni-cone kernel; (b) a TPG design for the kernel

Without loss of generality assume $d_1 \geq d_2 \geq \ldots \geq d_n$. A TPG design for the uni-cone kernel employing type 1 LFSR is illustrated in Figure 4.4(b). Δ_i extra D-type F/Fs are added in front of R_i, where $\Delta_i = d_{i-1} - d_i$ for $1 < i \leq n$. Therefore, the total number of extra F/Fs in the TPG is $\sum_{i=2}^{n}(d_{i-1} - d_i) = d_1 - d_n$. In other words the TPG consists of $(M + d_1 - d_n)$ F/Fs, where $M = \sum_{i=1}^{n} |R_i|$ and $|R_i|$ is the width of R_i. Among the string of $(M + d_1 - d_n)$ F/Fs, the first M F/Fs are connected as a maximal length type 1 LFSR. In general, the number of extra F/Fs depends on the difference of sequential lengths between the longest path and the

shortest path from the TPG to O. For simplicity, we ignore the all-0 pattern in the following discussion. The all-0 pattern can be provided by modifying an LFSR to be a *complete LFSR*[37].

Example 4.1 Consider the balanced BISTable kernel shown in Figure 4.2(a). Suppose R_1, R_2 and R_3 are all 4-bit registers. A TPG design for this kernel is shown in Figure 4.5 where the primitive polynomial $x^{12} + x^7 + x^4 + x^3 + 1$ is employed in the LFSR configuration. Only 2 extra D-type F/Fs are required, thus adding 7.2% extra area to a 12-bit BILBO register based on the `magic` layout tool. The test time for the kernel is $2^{12} - 1 + 2$ clock cycles, where the extra 2 clock cycles are needed to flush the last test pattern through the circuit. □

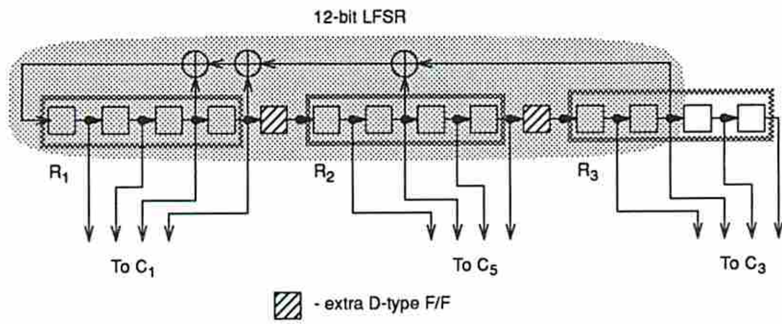


Figure 4.5: A TPG design for the balanced BISTable kernel in Figure 4.2(a)

Theorem 4.1 The TPG design scheme presented above provides a functionally exhaustive test set to a uni-cone balanced BISTable kernel.

Proof To show that a TPG provides a functionally exhaustive test set to a kernel we need to prove that every combinational logic cell in the kernel is tested functionally exhaustively when the TPG goes through all possible states of its state space. Given a uni-cone balanced BISTable kernel as represented by a reduced bipartite graph in Figure 4.4(a), the difference in sequential lengths of the paths from a pair of input registers, say R_i and R_j, to a combinational logic cell associated with the output port O is the constant $(d_j - d_i)$ due to the balance property. For example, the difference in sequential lengths of the paths from R_1 to C_3 and from R_2 to C_3 in Figure 4.2(a) is 1; and the difference in sequential lengths of the paths

from R_1 to C_2 and from R_2 to C_2 is also 1.

We first consider the combinational logic cell, say C_j, that drives the output port O. Suppose the ith input port of C_j depends on a set of input registers $\{R_{i_1}, \ldots, R_{i_k}\}$. Let $Q_i(t)$ be a vector at this input port of C_j that occurs during normal functional operation. Then there exists a set of vectors in $R_{i_1}, \ldots, R_{i_k}$, say $P_{i_1}(t - d_{i_1}), \ldots, P_{i_k}(t - d_{i_k})$, which when present at time $t - d_{i_1}, \ldots, t - d_{i_k}$, respectively, will produces $Q_i(t)$ at time t. In general, for every pattern at the inputs of C_j that occurs during normal functional operation, there exists a pattern in the form of $P_1(t-d_1) \ldots P_n(t-d_n)$ in the input registers $R_1, \ldots, R_n$ that produces such a pattern for C_j. To exercise all possible functional patterns at the input of C_j during test mode, all $(2^M - 1)$ possible patterns $P_1(t - d_1)P_2(t - d_2) \ldots P_n(t - d_n)$ should be applied to the circuit. Note that the M-stage LFSR in the TPG shown in Figure 4.4(b) can be divided into n segments where segment S_i contains $|R_i|$ F/Fs (see Figure 4.6). For each register R_i, $1 < i \leq n$, there exists a displacement of $(d_1 - d_i)$ F/Fs with respect to segment S_i due to the extra D F/Fs added in the TPG construction. This displacement is equivalent to a time shift between the vectors in R_i and S_i. In other words, the vector in R_i at time t is the same as the vector in S_i at time $t - (d_1 - d_i)$. Therefore, the pattern $P_1(t - d_1)P_2(t - d_2) \ldots P_n(t - d_n)$ in R_1, R_2, ..., R_n is equivalent to $P_1(t - d_1)P_2(t - d_1) \ldots P_n(t - d_1)$ in S_1, S_2, ..., S_n. Clearly the latter can go through all $(2^M - 1)$ combinations since S_1, S_2, ..., S_n are connected as a maximal length LFSR of degree M, thus functionally exhaustive patterns can be applied to C_j. As was seen above, the difference in the sequential lengths of the paths from R_i and R_j in the TPG to any block C_k is the constant $(d_j - d_i)$. Consequently, the same TPG will generate all patterns required to functionally exhaustively test C_k. According to Lemma 4.1, this TPG can also functionally exhaustively test other output cones (if any) of the kernel. Therefore, the proposed TPG can provide a functionally exhaustive test set to a uni-cone balanced BISTable kernel. □

Corollary 4.1 The test time to functionally exhaustively test a uni-cone balanced BISTable kernel is $2^M - 1 + d$, where M and d are the input width and sequential depth, respectively, of the kernel.

Proof It follows from Theorem 4.1 that a TPG containing a maximal length

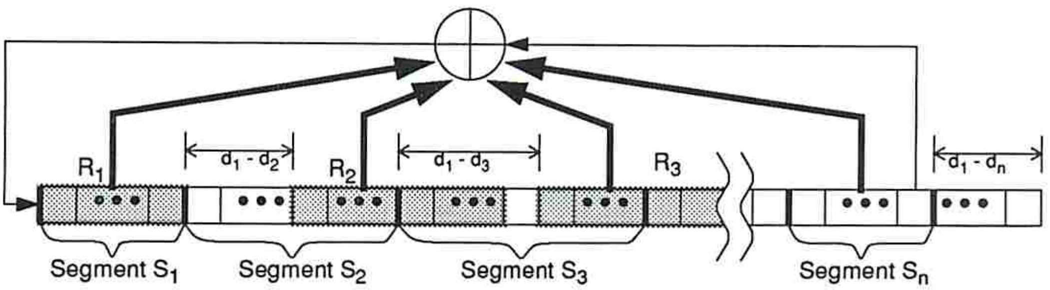


Figure 4.6: TPG for a balanced BISTable kernel

LFSR of degree M exists for each uni-cone balanced BISTable kernel. From the construction of the TPG, the TPG can apply one test pattern per clock cycle to the kernel. Therefore, it takes $(2^M - 1)$ clock cycles to generate a functionally exhaustive test set and d clock cycles to propagate the last pattern through the kernel. □

Note that even though extra F/Fs are needed in a TPG, they do not increase the test time required to functionally exhaustively test a balanced BISTable kernel. This scheme can be contrasted with the circular self-test path (CSTP) TDM proposed by Krasniewski and Palarski[31]. Here kernels can also be sequential and need not be balanced. It is estimated that to apply an exhaustive test set requires about $T \cdot 2^M$ test patterns, where T varies from 4 to 8. Since kernels need not be balanced, they may not be tested functionally exhaustively.

In general, the sequential lengths of the paths from input registers to an output cone may not be in descending order. We have developed a procedure described below to construct a TPG for a general uni-cone balanced BISTable kernel. The procedure first assigns a label to each register cell to indicate its corresponding LFSR stage in the TPG (steps 1 and 2). We denote the width of R_i as r_i and the jth cell of register R_i as $R_{i,j}$. Two variables, l_i and u_i, are associated with a register R_i to indicate the labels assigned to the first and the last cells, respectively, of R_i. The labels are assigned to the registers incrementally in the order of R_1, R_2, ..., R_n. Initially register cells in R_1 are labelled 1, ..., r_1, and thus $l_1 = 1$ and $u_1 = r_1$ (step 1). Subsequent labels are assigned based on the difference in the sequential

lengths of the paths from each pair of consecutive input registers to the output cone in step 2. Let $\Delta_i = d_{i-1} - d_i$, where d_i is the sequential length from R_i to the output cone. Let $P_{i-1}(t)$ and $P_i(t)$ be the vectors present in R_{i-1} and R_i, respectively, at time t. When $\Delta_i = 0$, the paths from R_{i-1} to the output and from R_i to the output are of equal sequential length. In this case, l_i is simply $u_{i-1}+1$ and R_{i-1} and R_i are physically adjacent in the TPG. If $\Delta_i > 0$, $P_i(t)$ will propagate to the output Δ_i clock cycles earlier than $P_{i-1}(t)$ does. To compensate for this imbalance so that functionally exhaustive patterns can be applied to the circuit, R_i should be shifted Δ_i stages away from R_{i-1} in the TPG. that is, $l_i = u_{i-1} + \Delta_i + 1$. R_{i-1} and R_i are said to be separated by Δ_i LFSR stages. If $\Delta_i < 0$ then $P_i(t)$ will propagate to the output $|\Delta_i|$ clock cycles later than $P_{i-1}(t)$ does. To compensate for this imbalance, R_i should be shifted $|\Delta_i|$ stages closer to R_{i-1} in the TPG, and thus l_i is also equal to $u_{i-1} + \Delta_i + 1$. R_{i-1} and R_i are said to share $|\Delta_i|$ LFSR stages. Since Δ_i determines the separation or sharing of R_i and R_{i-1} in the TPG, it is called the displacement of R_i with respect to R_{i-1}. Once register cells have been assigned labels, each register is first converted to a shift register (step 3). Then the connection between registers is done in steps 4 and 5. When two registers are physically adjacent, they are simply connected as a long shift register. When two registers are separated by Δ LFSR stages, a shift register consisting of Δ F/Fs is added between these two registers. When two registers share Δ LFSR stages, the register cells that correspond to the shared LFSR stages are fed by the same fanout stem. Step 6 adds additional LFSR stages to the tail of the TPG when the total number of LFSR stages after the above steps is insufficient, i.e. less than M. This may happen when most registers share LFSR stages. Finally in step 7 an M-stage LFSR is constructed. This procedure, known as *UC_TPG*, is presented next.

Procedure *UC_TPG*

/* INPUT: Input registers R_1, R_2, ..., R_n and the associated sequential lengths d_1, d_2, ..., d_n. */

/* OUTPUT: a TPG design. */

1. Let $l_1 = 1$ and $u_1 = r_1$. Label $R_{1,j}$ as j for $j = 1$ to r_1.

2. For $i = 2$ to n do

   (a) let $\Delta_i = d_{i-1} - d_i$, $l_i = u_{i-1} + 1 + \Delta_i$, and $u_i = l_i + r_i - 1$.

(b) label $R_{i,j}$ as $(l_i + j - 1)$ for $j = 1$ to r_i

3. For $i = 1$ to n, convert R_i to a shift register and mark R_i as `unprocessed`.

4. Let R_i be the register having the smallest l value. Mark R_i as `processed`.

5. While (`unprocessed` registers exist) do

   (a) let R_j be the `unprocessed` register having the smallest l value.
   - if $(l_j = u_i + 1)$, connect the output of R_{i,r_i} to the input of $R_{j,1}$.
   - else if $(l_j \leq u_i)$, let the register cell in R_i that is labelled $(l_j - 1)$ fanout to $R_{j,1}$.
   - else create a shift register SR consisting of $(l_j - u_i - 1)$ D F/Fs. Label the cells of SR as $(u_i + 1)$, ..., $(l_j - 1)$. Connect the output of R_{i,r_i} to the input of SR and the output of SR to the input of $R_{j,1}$.

   (b) mark R_j as `processed` and let i be j.

6. If $(u_i < M)$, create a shift register SR consists of $(M - u_i)$ D F/Fs. Label the cells of SR as $(u_i + 1)$, ..., M. Connect the output of R_{i,r_i} to the input of SR.

7. Connect register cells labelled 1, ..., M as a maximal length LFSR. If two or more F/Fs have the same label, only connect the last F/F.

Theorem 4.2 Procedure *UC_TPG* generates a minimal test time TPG to functionally exhaustively test a uni-cone balanced BISTable kernel.

Proof The TPG generated by Procedure *UC_TPG* employs an LFSR of M stages whose test time $(2^M - 1)$ is minimal for a circuit with M inputs. Therefore, what remains to be shown is the coverage of the patterns generated.

Given a uni-cone balanced kernel, the condition that a displacement $(d_j - d_i)$ of R_i with respect to R_j for every pair of input registers R_i and R_j $(j < i)$ is sufficient to functionally exhaustively test the kernel. Procedure *UC_TPG* assigns labels (LFSR stages) to register cells incrementally where a displacement $(d_{i-1} - d_i)$ of R_i with respect to R_{i-1} is ensured in each iteration. Consequently, the displacement of R_i with respect to R_j obtained by Procedure *UC_TPG* is $(d_{i-1} - d_i) + (d_{i-2} - d_{i-1}) + \ldots + (d_j - d_{j+1}) = d_j - d_i$, thus satisfying the above condition and the kernel can be tested functionally exhaustively. □

Example 4.2 Consider a kernel represented by the bipartite graph shown in Figure 4.3(a). Assume that each register is 4 bits wide and the sequential lengths from R_1, R_2 and R_3 to O_1 are 1, 2, and 0, respectively. The TPG generated by Procedure *UC_TPG* is shown in Figure 4.7, where the primitive polynomial $x^{12} + x^7 + x^4 + x^3 + 1$ is employed. The line below the register cells illustrates the labels assigned to them. The displacement of R_2 with respect to R_1 is -1. Therefore, R_1 and R_2 share one LFSR stage (stage 4). On the other hand, R_2 and R_3 are separated by two LFSR stages (stages 8 and 9) due to the positive displacement (+2) of R_3 with respect to R_2. Neither of the two F/Fs labelled 4 can be deleted since both are used in the normal mode of operation of the circuit. □

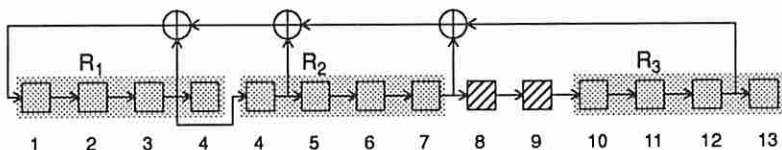


Figure 4.7: A TPG design for Example 4.2

It should be noted that if there exists a $\Delta_i < 0$ and $|\Delta_i| > r_{i-1}$, the number of LFSR stages shared by register R_i and R_{i-1} is less than $|\Delta_i|$. Also note that although $R_{1,1}$ is assigned a label 1, the smallest label in the labelling process may be less than 1. This can be illustrated by the following example.

Example 4.3 Consider a uni-cone kernel represented by a bipartite graph as shown in Figure 4.8(a). Suppose that both R_1 and R_2 are 4 bits wide. The difference in sequential lengths of the paths from R_1 and R_2 to O results in a displacement -5 of R_2 with respect to R_1. Since the width of R_1 is only 4, R_1 and R_2 can share at most 4 LFSR stages in the TPG. The TPG design generated by Procedure *UC_TPG* for a kernel with this bipartite graph representation is shown in Figure 4.8(b), where the smallest label is 0 instead of 1. As can be seen from this TPG design, R_1 and R_2 in fact share only 3 LFSR stages (stages 1, 2 and 3). □

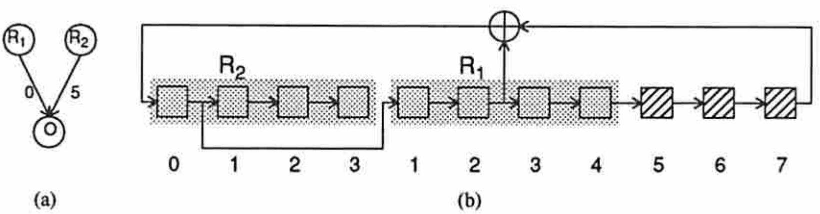


Figure 4.8: (a) the bipartite graph representation of an example kernel; (b) TPG design for (a)

4.4 TPG Design for Multi-cone Kernels

In the previous section, given a pair of consecutive registers R_{i-1} and R_i, the displacement of R_i with respect to R_{i-1} in a TPG is simply the difference in their associated sequential lengths. This is not true for multi-cone kernels since a pair of registers may both drive two output cones and the differences in the associated sequential lengths from these registers to the cones are unequal. This can be illustrated by the example below.

Example 4.4 Consider a balanced BISTable kernel with 2 output cones as shown in Figure 4.9(a). We assume that both R_1 and R_2 are 4 bits wide. The first cone, denoted by Ω_1, is associated with output port O_1 and depends on R_1 and R_2. The sequential lengths of the paths from R_1 and R_2 to O_1 are 2 and 0, respectively. The second cone, denoted by Ω_2, is associated with port O_2 and also depends on R_1 and R_2. However, the sequential lengths of the paths from R_1 and R_2 to O_2 are 1 and 0, respectively. To test this kernel with a TPG consisting of R_1 and R_2, the differences in sequential lengths of paths from R_1 and R_2 to O_1 and O_2 should be considered to determine the displacement of R_2 with respect to R_1 in the TPG. Consider O_1 first. Since the difference in sequential lengths of the paths from R_1 to O_1 and from R_2 to O_1 is +2, a displacement +2 of R_2 with respect to R_1 in the TPG is sufficient. Next we consider O_2. A displacement +1 of R_2 with respect to R_1 in the TPG is sufficient due to the sequential length difference between the paths from R_1 to O_2 and from R_2 to O_2. Therefore, a displacement +2 of R_2 with respect to R_1 in the TPG is sufficient to compensate for the sequential length imbalance. This displacement implies that R_1 and R_2 are separated by 2

LFSR stages in the TPG as shown in Figure 4.9(b).

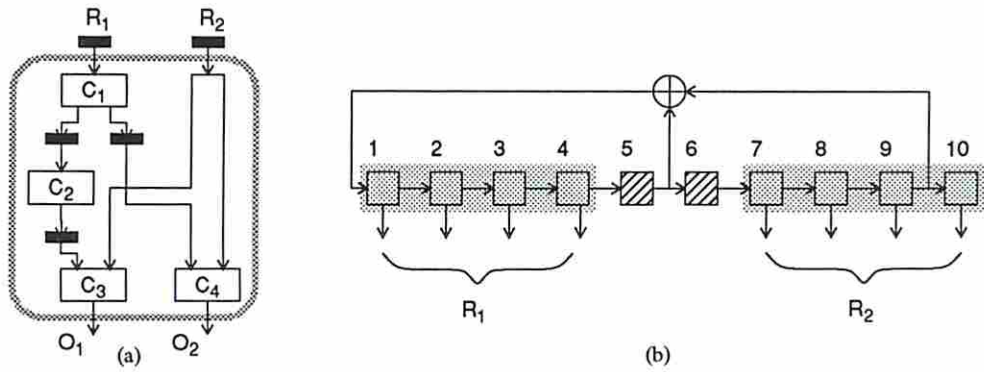


Figure 4.9: (a) an example multi-cone kernel; (b) TPG design for (a)

Once the displacement between registers are determined, the size of the required LFSR can then be determined as described below. To simplify the following analysis, we assume that C_1 and C_2 are vacuous cells. It should be noted that this assumption is only for illustration and the result of the analysis is valid without the assumption. Suppose that at time t, the pattern in the 10 F/Fs shown in Figure 4.9(b) is $P_1(t)P_2(t)\ldots P_{10}(t)$. At time $t+1$, the pattern present at the inputs of C_4 is $P_1(t)\ldots P_4(t)P_7(t+1)\ldots P_{10}(t+1)$, which is equivalent to $P_1(t)\ldots P_4(t)P_6(t)\ldots P_9(t)$. To functionally exhaustively test C_4 (or cone Ω_2), an LFSR of at least 9 stages is required. Similarly at time $t+2$, the pattern present at the inputs of C_3 is $P_1(t)\ldots P_4(t)P_7(t+2)\ldots P_{10}(t+2)$, which is equivalent to $P_1(t)\ldots P_4(t)P_5(t)\ldots P_8(t)$ and thus an LFSR of at least 8 stages is required to functionally exhaustively test cone Ω_1. Therefore, although the maximal cone size (width of the inputs on which a cone depends) of the circuit is 8, an LFSR of degree 9 is required (see Figure 4.9(b)). □

Next we extend the procedure presented in the previous section to deal with multi-cone kernels. In the labelling process (steps 1 and 2), the sequential length information for every pair of input registers (not necessarily consecutive) has to be considered for each cone that depends on both registers (steps 2(a) and 2(b)). For example, suppose $d_{j,x} - d_{i,x} = \delta_{i,j}(x)$ where R_i, R_j ($j < i$) are a pair of registers on which O_x depends, and $d_{i,x}$ is the sequential length from R_i to output port

O_x. There can be more than one cone that depends on both R_i and R_j. Let $\Delta_{i,j} = \max_x \delta_{i,j}(x)$ for all x, where cone Ω_x depends on both R_i and R_j. Clearly the displacement of R_i with respect to R_j must be at least $\Delta_{i,j}$. When R_i and R_j are consecutive registers (i.e. $j = i-1$), $\Delta_{i,j} > 0$ implies a separation of $\Delta_{i,j}$ LFSR stages between R_i and R_j in the TPG; and $\Delta_{i,j} < 0$ implies a sharing of $|\Delta_{i,j}|$ LFSR stages between R_i and R_j. When R_i and R_j are not consecutive registers, The sufficient displacement of R_i with respect to R_{i-1} is simply $u_j + \Delta_{i,j}$ - u_{i-1} and the separation or sharing between R_i and R_{i-1} can then be determined.

Theorem 4.3 The above labelling process guarantees that every register cell is assigned with the smallest label.

Proof Since cells of the same register are assigned with contiguous labels, we will consider only the last cell of each register in this proof. The proof is by induction. We claim that if the last cell of R_{i-1}, namely $R_{i-1,r_{i-1}}$, is assigned with a smallest allowable label, this will lead to assigning the last cell of R_i, namely R_{i,r_i}, with a smallest label.

When $i = 2$ (induction base), the claim is clearly true since R_{1,r_1} is labelled r_1 and R_{2,r_2} is simply labelled $r_1 + d_1 - d_2$. Assume the above claim is false for $i > 2$. Let u_{i-1} be the smallest allowable label for $R_{i-1,r_{i-1}}$. Then assigning u'_{i-1} to $R_{i-1,r_{i-1}}$ leads to the assignment of R_{i,r_i} with u_{i1}, while assigning u_{i-1} to $R_{i-1,r_{i-1}}$ leads to the assignment of R_{i,r_i} with u_{i2}, where $u'_{i-1} > u_{i-1}$ and $u_{i1} < u_{i2}$. However, it can be observed from the labelling process presented above that if there exists no output cone that depends on both R_{i-1} and R_i, R_{i,r_i} will be assigned with the same label (i.e. $u_{i1} = u_{i2}$) independent of the label of $R_{i-1,r_{i-1}}$; and if there exists an output cone that depends on both R_{i-1} and R_i then $u_{i1} > u_{i2}$. Therefore, the assumption cannot be true and this proves our claim. Notice that registers are labelled incrementally, namely register R_i is labelled only if registers R_1, R_2, ..., R_{i-1} have been labelled. This claim is true for all i, where $1 < i \leq n$, and thus the theorem is true. □

Once register cells are labelled, the sufficient number of LFSR stages to fully test every cone can be determined (step 3). Consider a cone Ω_y that depends on p input registers R_{x_1}, R_{x_2}, ..., R_{x_p} as shown in Figure 4.10, where R_{x_k} proceeds

$R_{x_{k+1}}$ in the TPG for $k = 1$ to $p-1$. Let l_{x_k} and u_{x_k} be the labels assigned to the first and the last cells, respectively, of R_{x_k}.

Theorem 4.4 $(u_{x_p} - l_{x_1} + 1 + d_{x_p,y} - d_{x_1,y})$ LFSR stages are sufficient to functionally exhaustively test the logic in cone Ω_x.

Proof Every pair of input ports of C that depend on register R_{x_i} and R_{x_j}, where $j < i$, are connected to LFSR stages having a span of $(u_{x_i} - l_{x_j} + 1)$ F/Fs, as can be seen from Figure 4.10. This number is called the *physical span* of these two input ports. Due to the difference in sequential length, the same ports depend on a span of $(u_{x_i} - l_{x_j} + 1 + d_{x_i,y}$ - $d_{x_j,y})$ LFSR stages as was illustrated in Example 4.4. This number is called the *logical span* of these two input ports. The *logical span* of the output port O_y is defined as the the maximum among the logical spans of all (i, j) pairs. This number determines a sufficient number of LFSR stages to functionally exhaustively test the logic associated with cone Ω_y.

We claim that $(u_{x_p} - l_{x_1} + 1 + d_{x_p,y} - d_{x_1,y}) \geq (u_{x_i} - l_{x_j} + 1 + d_{x_i,y} - d_{x_j,y})$ for any (i, j) pair. This is true by examining the following equation.

$$
\begin{aligned}
& (u_{x_p} - l_{x_1} + 1 + d_{x_p,y} - d_{x_1,y}) - (u_{x_i} - l_{x_j} + 1 + d_{x_i,y} - d_{x_j,y}) \\
= \; & (u_{x_p} - u_{x_i}) - (d_{x_i,y} - d_{x_p,y}) + (l_{x_j} - l_{x_1}) - (d_{x_1,y} - d_{x_j,y}) \\
\geq \; & \Delta_{x_p,x_i} - \delta_{x_p,x_i}(y) + \Delta_{x_j,x_1} - \delta_{x_j,x_1}(y) \\
\geq \; & 0
\end{aligned}
$$

It should be noted that the value $(u_{x_p} - u_{x_i})$ is the displacement of R_{x_p} with respect to R_{x_i} in the TPG design, which is obtained by considering every i that is less than p. Therefore, it is at least as large as Δ_{x_p,x_i}, namely the required displacement of R_{x_p} with respect to R_{x_i} when only a particular i is considered. Similarly the value $(l_{x_j} - l_{x_1})$ is the displacement of R_{x_j} with respect to R_{x_1} and is at least as large as Δ_{x_j,x_1}. Furthermore, $\Delta_{x_p,x_i} \geq \delta_{x_p,x_i}(y)$ and $\Delta_{x_j,x_1} \geq \delta_{x_j,x_1}(y)$ since $\Delta_{x_p,x_i} = \max_y \delta_{x_p,x_i}(y)$ and $\Delta_{x_j,x_1} = \max_y \delta_{x_j,x_1}(y)$. Therefore, $(u_{x_p} - l_{x_1} + 1 + d_{x_p,y} - d_{x_1,y})$ LFSR stages are sufficient to functionally exhaustively test cone Ω_y. □

Once the sufficient number of LFSR stages to functionally exhaustively test each cone has been determined, the sufficient number of LFSR stages to functionally exhaustively test the entire circuit can then be obtained by taking the maximum

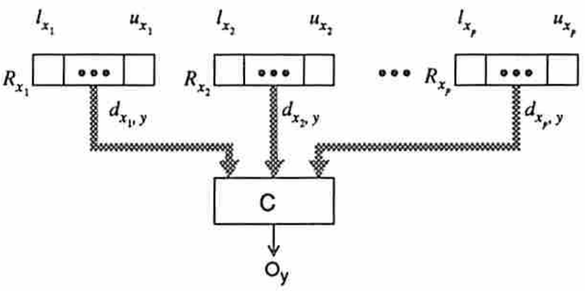


Figure 4.10: An output cone that depends on R_{x_1}, R_{x_2}, ..., R_{x_p}

among the sufficient number of LFSR stages for individual cones. The procedure for generating a TPG for a multi-cone kernel, known as *MC_TPG*, is presented next. Again l_i and u_i, $1 \leq i \leq n$, denote the labels assigned to the first and the last cells, respectively, of register R_i.

Procedure *MC_TPG*

/* INPUT: Input registers R_1, R_2, ..., R_n and output cones Ω_1, Ω_2, ..., Ω_m. Each Ω_x contains a set of registers on which Ω_x depends and the associated sequential lengths. */

/* OUTPUT: a TPG design. */

1. Let $l_1 = 1$ and $u_1 = r_1$. Label $R_{1,j}$ as j for $j = 1$ to r_1.
2. For $i = 2$ to n do
   (a) for $j = 1$ to $i - 1$ do
      i. for each cone Ω_x if Ω_x depends on both R_i and R_j, let $\delta_{i,j}(x) = d_{j,x} - d_{i,x}$.
      ii. let $\Delta_{i,j}$ be the maximum among $\delta_{i,j}(x)$ for all x.
      iii. let $\Delta_i(j)$ be $\Delta_{i,j} + u_j - u_{i-1}$.
   (b) let Δ_i be the maximum among $\Delta_i(j)$ for $j = 1$ to $i - 1$.
   (c) let $l_i = u_{i-1} + 1 + \Delta_i$, and $u_i = l_i + r_i - 1$.
   (d) label $R_{i,j}$ as $(l_i + j - 1)$ for $j = 1$ to r_i
3. Let $M = 0$. For each cone Ω_x in the circuit do
   (a) determine the sufficient number of LFSR stages to functionally exhaustively test Ω_x, say M', as follows. Let R_{x_1} and R_{x_p} be the first

and last registers on which Ω_x depends. Let l_{x_1} and u_{x_p} be the labels of the first and the last cells of R_{x_1} and R_{x_p}, respectively. Then $M' = u_{x_p} - l_{x_1} + 1 + d_{x_p,x} - d_{x_1,x}$.

(b) if $(M' > M)$ then $M = M'$

4. For $i = 1$ to n, convert R_i to a shift register and mark R_i as `unprocessed`.

5. Let R_i be the register having the smallest l value. Mark R_i as `processed`.

6. While (`unprocessed` registers exist) do

   (a) let R_j be the `unprocessed` register having the smallest l value.

   - if $(l_j = u_i + 1)$, connect the output of R_{i,r_i} to the input of $R_{j,1}$.
   - else if $(l_j \leq u_i)$, let the register cell in R_i that is labelled $(l_j - 1)$ fanout to $R_{j,1}$.
   - else create a shift register SR consisting of $(l_j - u_i - 1)$ D F/Fs. Label the cells of SR as $(u_i + 1), \ldots, (l_j - 1)$. Connect the output of R_{i,r_i} to the input of SR and the output of SR to the input of $R_{j,1}$.

   (b) mark R_j as `processed` and let i be j.

7. If $(u_i < M)$, create a shift register SR consists of $(M - u_i)$ D F/Fs. Label the cells of SR as $(u_i + 1), \ldots, M$. Connect the output of R_{i,r_i} to the input of SR.

8. Connect register cells labelled $1, \ldots, M$ as a maximal length LFSR. If two or more F/Fs have the same label, only connect the last F/F.

Procedure *MC_TPG* is a polynomial time algorithm with a complexity of $O(mn^2)$, where m is the number of cones and n is the number of input registers.

Example 4.5 Consider the circuit shown in Figure 4.11(a). The registers are labelled as shown in Figure 4.11(b) where a displacement of R_2 with respect to R_1 is +2 due to the sequential length difference in Ω_1. Based on this register labelling, the physical span of Ω_2 is 10 (from stage 1 to stage 10). Due to the sequential length difference (i.e. +1), the logical span of Ω_2 is 11 (from stage 1 to stage 11). Therefore, an 11-stage LFSR as shown in Figure 4.11(b) is sufficient to functionally exhaustively test the circuit. The TPG shown in Figure 4.11(b) can be obtained by using Procedure *MC_TPG*. □

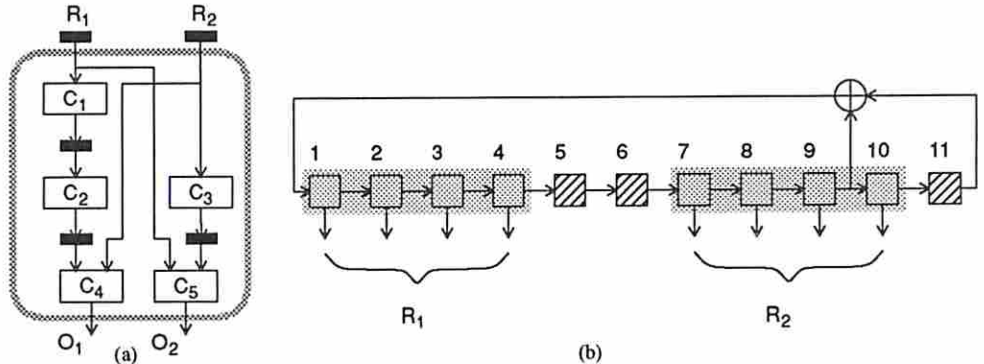


Figure 4.11: (a) a multi-cone kernel for Example 4.5; (b) TPG design for (a)

It should be noted that if the 2 cones in Figure 4.11(a) are tested separately (in 2 test sessions), the test time is approximately $2 \times 2^8 = 2^9$, which is much smaller than the test time 2^{11} using the TPG described above. In such a case, a *reconfigurable TPG*[38] that can test different cones at different test sessions by changing the configurations of the LFSRs might be desired to reduce the test time. For the kernel in Figure 4.11(a), a reconfigurable TPG as shown in Figure 4.12 can be used. When the control line Ω_1/Ω_2 is 0, the TPG is configured to test cone Ω_1 and when the control line Ω_1/Ω_2 is 1, the TPG is configured to test cone Ω_2. Although a reconfigurable TPG may reduce the test time for a multi-cone kernel, the area overhead and performance degradation of such design are usually high. Therefore, a designer can make trade-offs between test time and area overhead when a reconfigurable TPG is more time efficient.

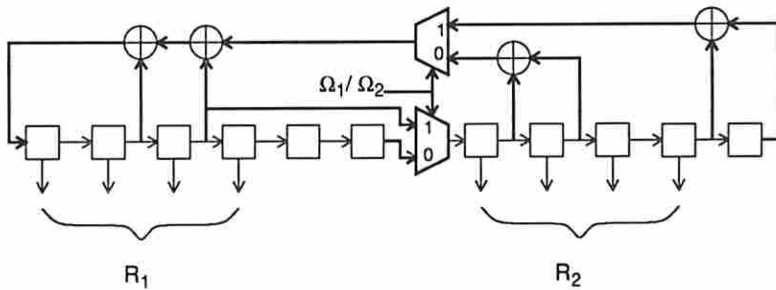


Figure 4.12: A reconfigurable TPG design for kernel in Figure 4.11(a)

4.5 Functionally Pseudo-Exhaustive Testing

Example 4.6 Consider the circuit with 3 cones as shown in Figure 4.13(a). According to Procedure *MC_TPG*, two LFSR stages are required between R_1 and R_2 and one LFSR stage is required between R_2 and R_3. Based on the register labelling shown in Figure 4.13(b), the logical span of Ω_1 is 8 (from stage 1 stage 8); the logical span of Ω_2 is 16 (from stage 1 to stage 16); and the logical span of Ω_3 is 8 (from stage 7 to stage 14). Therefore, a maximal length LFSR of degree 16 is sufficient in the TPG design as shown in Figure 4.13(b). The test time for the circuit using this TPG is approximately 2^{16}.

If the input registers are allowed to be permuted, significant reduction in the LFSR size and test time can be achieved. For example, suppose the input registers are ordered as R_1, R_3 and R_2. A TPG design that employs an LFSR of degree 8 can be obtained by using Procedure *MC_TPG* as shown in Figure 4.13(c). It should be noted that although the input width of the circuit is 12, the test time to functionally exhaustively test the entire circuit using this TPG design is approximately 2^8. This is due to the fact that each cone only depends on a subset of the inputs. □

The concept of pseudo-exhaustive testing has been widely employed in testing combinational circuits, where exhaustive patterns are applied to individual output cones of a combinational circuit. This technique ensures the detection of all irredundant stuck-at faults in the circuit and the associated test time is usually less than that of exhaustive testing. As can be observed from Example 4.6, functionally exhaustive patterns can be applied to individual output cones of a balanced BISTable kernel and the test time of this technique may be less than that of applying exhaustive patterns to the entire kernel. This test strategy is called *functionally pseudo-exhaustive testing*. Functionally pseudo-exhaustive testing ensures the detection of all stuck-at faults in the circuit that would interfere with normal operation.

Research problems in pseudo-exhaustive testing, such as finding the minimal test signal set and test sets[39, 40] and TPG design for pseudo-exhaustive

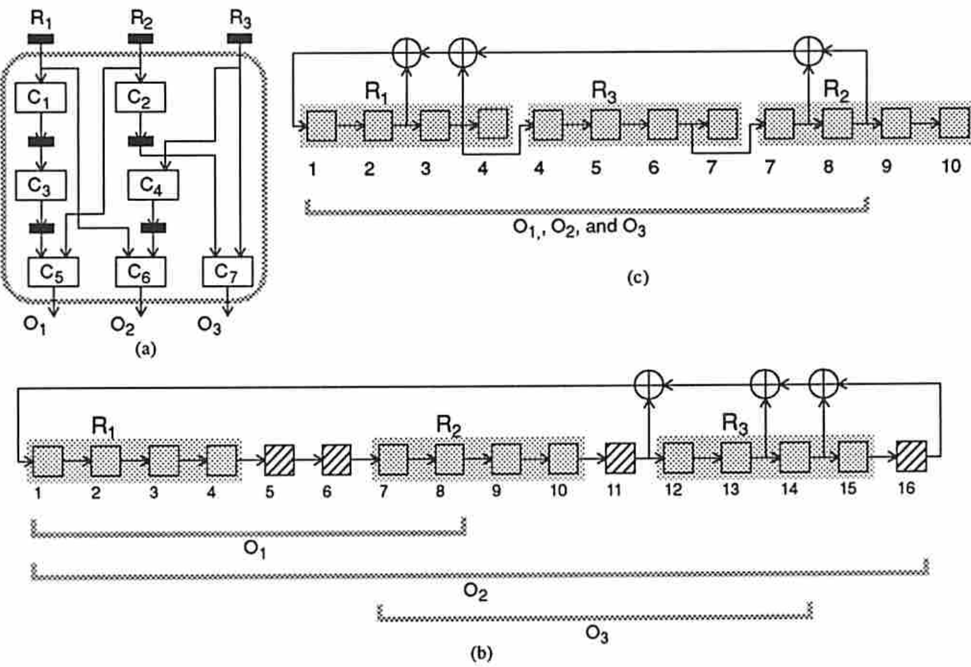


Figure 4.13: (a) a three cones kernel; (b) a TPG design for (a); (c) an alternative TPG design for (a)

testing[41, 39, 38] have been extensively studied. It has been shown that the problem of generating an optimal pseudo-exhaustive test set is NP-complete[39] and the problem of minimal test time TPG design is also hard. The problem of TPG design for functionally pseudo-exhaustive testing differs from this previous work in two aspects. Namely register level instead of gate level components are considered, and the concepts of time shift and sequential length difference must be taken into account. We can easily extend the procedure in finding minimal test signals presented in [39] to deal with register level signals efficiently as illustrated below.

Example 4.7 Consider the the circuit shown in Figure 4.13(a). The cone dependency[39] is shown in the dependency matrix D.

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

$D_{ij} = 1$ if cone Ω_i depends on R_j. Clearly 3 test signals, each of which being a group of 4 wires, are required which implies an LFSR of 12 stages is needed in the TPG design. Therefore, the test time using this approach will be approximately 2^{12} which is much greater than 2^8 obtained in Example 4.6 when employing Procedure *MC_TPG* and the register permutation. □

As was seen in the above example, the result obtained by the extended minimal test signal procedure is not optimal in terms of the LFSR size and test time. This is due to the lack of the ability to handle sequential length information in this procedure. Therefore, this simple extension of an existing procedure that finds the minimal number of test signals[39] does not always generate a minimal test time TPG design for a multi-cone kernel.

As illustrated in Example 4.6, the test time for a multi-cone kernel can vary according to the ordering of the input registers. Under certain register orderings, test time to detect all detectable stuck-at faults may be less than that of applying a functionally exhaustive test set and thus functionally pseudo-exhaustive testing can be applied. It is often desirable to reduce the test time for a kernel by finding a best register ordering. This can be achieved by executing Procedure *MC_TPG* once for

each input register ordering. In practice, the number of input registers of a multi-cone kernel is usually small, say less than 5. In addition, Procedure *MC_TPG* is a polynomial time algorithm. Therefore, it is feasible to employ the above approach to reduce the test time for a multi-cone kernel. It should be noted that the test time for a multi-cone kernel is lower bounded by 2^w, where w is the maximal cone size of the kernel. Therefore, if this lower bound is achieved by a particular register ordering, a minimal test time TPG has been obtained and the test time reduction process can be terminated.

4.6 Performance Evaluation

As discussed in the previous sections, TPGs using the conventional maximal length LFSR may not provide a functionally exhaustive test set to a balanced BISTable kernel. This may lead to inadequate pattern coverage, and thus low fault coverage. Consider the example circuit shown in Figure 4.14(a) that was also considered in Chapter 3. The circuit is not 1-step functionally testable due to the unbalanced path between `exep` and `cps`. Two different BIBS solutions, S_1 and S_2, are shown in Figure 4.14(b) and (c), respectively.

In S_1 registers R_3 and R_5 (in addition to the PI/PO registers) are converted to BILBO registers. Thus the original is partitioned into two kernels, $K_{1,1}$ and $K_{1,2}$, as shown by the shaded blocks in Figure 4.14(b). $K_{1,1}$ can be tested by simply concatenating R_1, R_2 and configuring them as a maximal length LFSR of degree 28. The bipartite graph representation of $K_{1,2}$ and its input registers (i.e. R_3 and R_5) is shown in Figure 4.15(a), where the sequential length of the path from R_3 to O, which denotes the output port that drives R_{10}, is 2 and the sequential length of the path from R_4 to O is 1. Therefore the proposed TPG design may be employed to test $K_{1,2}$. One possible configuration of such a TPG is shown in Figure 4.16(a). In S_2 registers R_3 and R_4 (in addition to the PI/PO registers) are converted to BILBO registers. The original is partitioned into two kernels, $K_{2,1}$ and $K_{2,2}$, as shown by the shaded blocks in Figure 4.14(c). $K_{2,2}$ can be tested by simply configuring R_3 as a maximal length LFSR of degree 24. The bipartite graph representation of $K_{2,1}$ and its input registers (i.e. R_1, R_2 and R_4) is shown

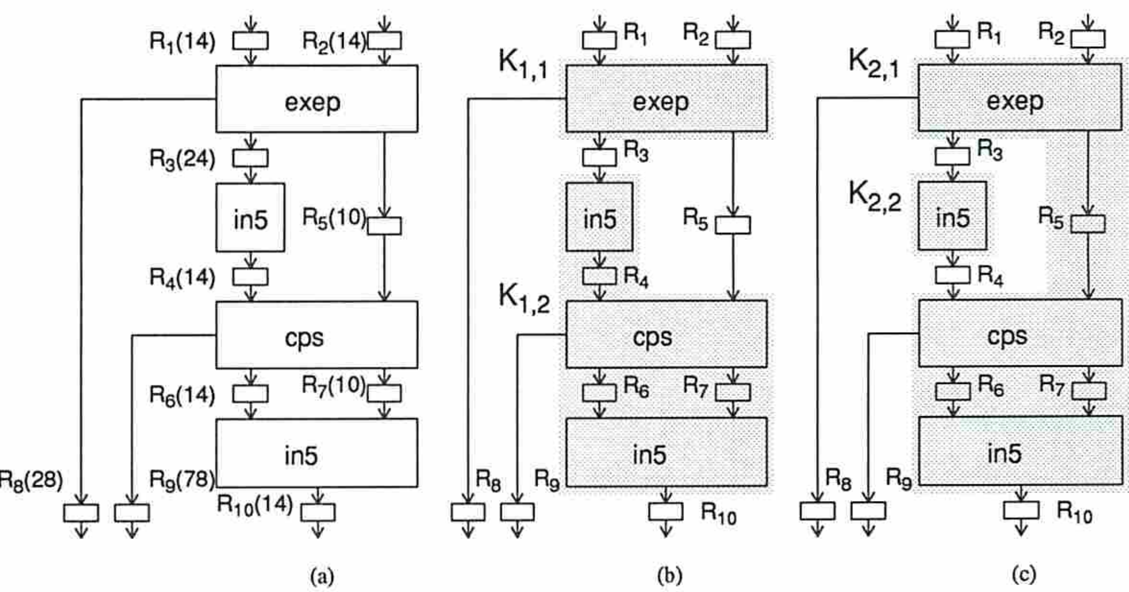


Figure 4.14: (a) an example circuit; (b) S_1; (c) S_2

in Figure 4.15(b), where the sequential lengths of the path from R_1, R_2, and R_4 to O are 2, 2, and 1, respectively. The proposed TPG design can also be used to test $K_{2,1}$ and one possible configuration of such a TPG is shown in Figure 4.16(b).

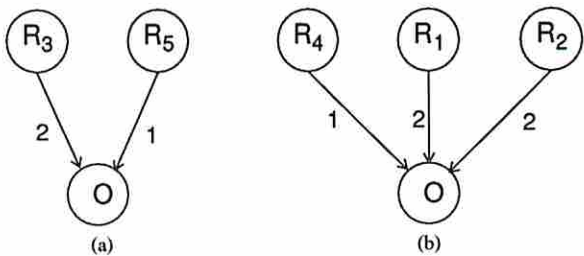


Figure 4.15: Bipartite graph representations for (a) $K_{1,2}$; and (b) $K_{2,1}$

We perform experiments to compare the fault coverage obtained using the proposed TPGs as shown in Figures 4.16(a) and (b) with the fault coverage obtained using simple maximal length LFSR for both $K_{1,2}$ and $K_{2,1}$. Note that the same primitive polynomial is employed to configure the proposed TPG and the simple LFSR for both kernels. In both cases our TPGs require one more D F/F than the

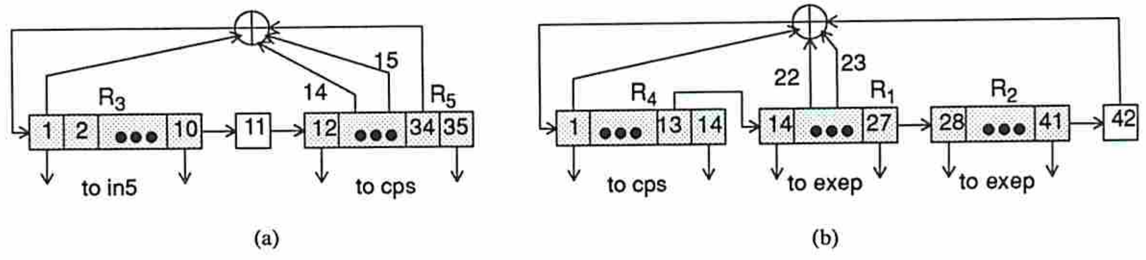


Figure 4.16: TPG for (a) $K_{1,2}$; (b) $K_{2,1}$

simple LFSRs. Table 4.1 shows the results of testing $K_{1,2}$ and $K_{2,1}$ using these two different TPGs and a variety of test lengths. For each test length five different sequences of patterns are applied to the kernels by using different initial states in the TPGs. FC_1 and FC_2 for each kernel are computed as the average (over five experiments) fault coverage using the proposed TPG and simple maximal length LFSR, respectively. Fault coverage is calculated based on the detectable faults in the kernels. That is, we ignore redundant faults. As can be observed, the proposed TPG achieves better fault coverage than the simple LFSR for all lengths of test sequences for both kernels. In fact, the proposed TPG guarantees to achieve 100% fault coverage when the length of test sequences is 2^n, where n is the kernel input width (Theorem 4.1). On the other hand, the complete fault coverage may not be achieved by simple LFSRs.

4.7 Remark

As was seen in previous sections, additional D F/Fs may be required in a TPG design for a balanced BISTable kernel K using Procedure *UC_TPG* or *MC_TPG*. When K is a uni-cone kernel, the additional D F/Fs only increase the TPG hardware but not the test time (Theorem 4.2). However, this is not always true when K is a multi-cone kernel. In this section we will study a few other alternatives in designing TPG for the BIBS TDM.

One simple solution is to design TPGs that have a HOLD mode so that parts of the test patterns can be delayed by the required clock cycles, say Δ, to balance the difference in sequential lengths. However, the test time for the kernel under

K	$K_{1,2}$		$K_{2,1}$	
($\log_2$ test length)	FC_1(%)	FC_2(%)	FC_1(%)	FC_2(%)
5	44.5	44.0	50.4	42.7
6	48.7	47.0	54.0	48.8
7	51.4	50.4	57.1	52.3
8	57.2	53.3	59.8	55.6
9	61.1	58.2	63.9	58.5
10	67.6	66.6	65.2	63.2
11	72.8	70.8	70.0	68.2
12	78.1	73.1	76.7	72.2
13	79.6	77.7	82.6	75.9
14	82.4	79.8	87.5	79.6
15	85.6	80.6	90.2	82.9
16	87.0	82.7	93.4	86.8
17	89.9	85.3	96.1	88.4
18	93.0	86.9	98.1	92.3
19	96.2	89.9	99.2	96.2
20	98.4	92.6	99.4	96.4

Table 4.1: Results of testing $K_{1,2}$ and $K_{2,1}$

test increases by a factor of $(\Delta + 1)$ since each pattern has to be generated and then held for Δ clock cycles. In addition, the control complexity and the TPG hardware increase and the kernel cannot be tested at speed.

Next we consider a counter-based TPG design scheme. Consider a balanced BISTable kernel K represented by the bipartite graph shown in Figure 4.17(a). Let the widths of R_1 and R_2 be n_1 and n_2, respectively.

Theorem 4.5 If R_1 and R_2 are concatenated and configured as a $(n_1 + n_2)$-bit counter, then this counter can functionally exhaustively test K.

Proof Let a *cycle* of the counter be defined as the counting sequence $\underbrace{00\cdots00}_{(n_1+n_2)}$, $\underbrace{00\cdots01}_{(n_1+n_2)}$, ..., $\underbrace{11\cdots11}_{(n_1+n_2)}$. The length of a cycle is clearly $2^{(n_1+n_2)}$. A cycle of the counter can be divided into 2^{n_1} sub-cycles where the length of each sub-cycle is 2^{n_2} as illustrated in Figure 4.17(b). Each sub-cycle is a counting sequence where the first n_1 bits are fixed and the last n_2 bits count from $\underbrace{00\cdots00}_{n_2}$ to $\underbrace{11\cdots11}_{n_2}$.

For simplicity, assume every combinational logic cell in K is transparent, namely consists of wires only. Note that this assumption is only for illustration and the theorem is true without the assumption. Without loss of generality, assume that $d_{1,1} - d_{2,1} = 1$. When the counter generates a cycle, the patterns observed at O_1 are illustrated in Figure 4.17(c). Clearly every pattern observed at O_1 is distinct during one cycle. Thus these patterns cover all possible $2^{(n_1+n_2)}$ combinations and O_1 is tested functionally exhaustively. O_2 can easily be shown to be tested functionally exhaustively by the same counter using similar arguments. Consequently, K can be functionally exhaustively tested by the $(n_1 + n_2)$-bit counter. Note that the counting sequence generated by the counter need not be in ascending order. In fact, any counter-based TPG design that has a cycle length of $2^{(n_1+n_2)}$ and each cycle can be divided into 2^{n_1} sub-cycles is a feasible solution. □

Corollary 4.2 An n-bit counter can functionally exhaustively test any multi-cone balanced BISTable kernel having an input width of n.

Proof Similar to the proof for Theorem 4.5 and omitted. □

As can be seen from the above theorems, a counter-based TPG design can functionally exhaustively test a balanced BISTable kernel. In addition, no extra D

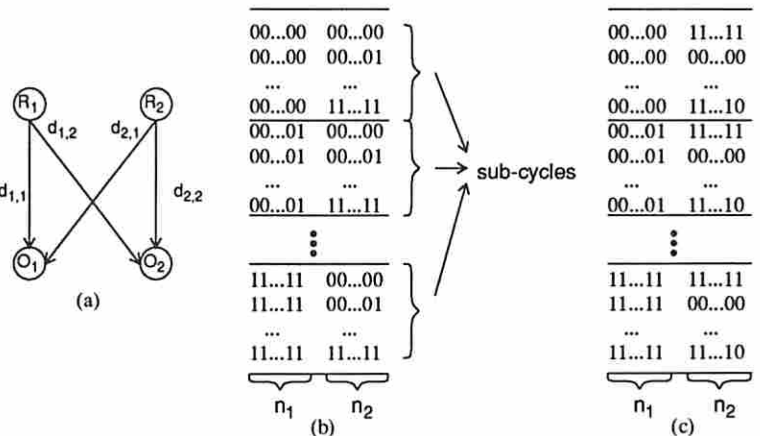
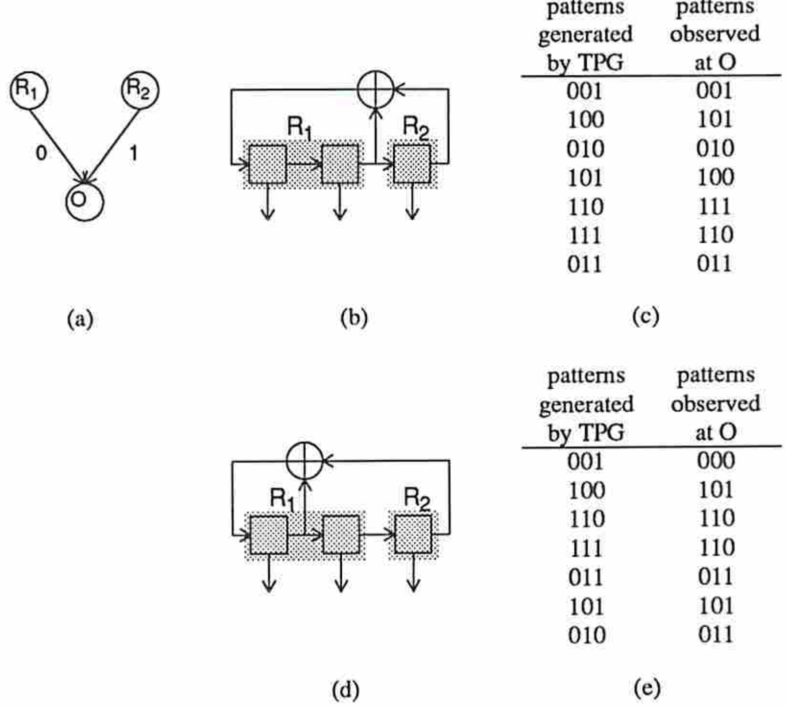


Figure 4.17: (a) a bipartite graph representation of a balanced BISTable kernel; (b) a counting sequence and the sub-cycles generated by a counter; (c) patterns observed at O_1 in (a) when $d_{1,1} - d_{2,1} = 1$

F/Fs are required in the TPG and thus no increase in test time. That is, the test time to functionally exhaustively test a kernel having n inputs using a counter is 2^n. However, the hardware overhead to convert a normal register to a counter is excessively expensive. Furthermore, when a kernel under test can be fully tested using pseudo-random patterns, a counter-based TPG is not appropriate due to its lack of randomness. Thus it may not be practical to employ counters as TPGs for balanced BISTable kernels. Next we consider the TPG design for balanced BISTable kernels using pure LFSRs (i.e. no additional D F/Fs).

Consider the bipartite graph representation of a balanced BISTable kernel shown in Figure 4.18(a), where R_1 is a 2-bit register and R_2 is a 1-bit register. Clearly a TPG using a 3-stage LFSR is sufficient to test the kernel according to Procedure *UC_TPG*. There are two primitive polynomials of degree 3, namely $x^3 + x^2 + 1$ and $x^3 + x + 1$. Based on Procedure *UC_TPG*, one extra D F/F is required in a TPG design for this kernel. This TPG can be configured using either of the above two primitive polynomials. Next consider the TPG shown in Figure 4.18(b) that is configured as a pure LFSR (as opposed to LFSR/SR employed by Procedure *UC_TPG*) using the polynomial $x^3 + x^2 + 1$. A maximal length sequence generated by the TPG and the patterns observed at O assuming

the combinational logic cells are transparent are shown in Figure 4.18(c). It can be observed that the kernel is tested functionally exhaustively by this TPG, which contains only 3 D F/Fs (as opposed to 4 D F/Fs required by Procedure *UC_TPG*). On the other hand, if the TPG is configured using the polynomial $x^3 + x + 1$ as shown in Figure 4.18(d), the TPG does not functionally exhaustively test the same kernel as can be observed from Figure 4.18(e).



patterns generated by TPG	patterns observed at O
001	001
100	101
010	010
101	100
110	111
111	110
011	011

patterns generated by TPG	patterns observed at O
001	000
100	101
110	110
111	110
011	011
101	101
010	011

Figure 4.18: (a) a bipartite graph representation of a balanced BISTable kernel; (b) a TPG employing a pure LFSR; (c) patterns generated by TPG in (b) and patterns observed at O; (d) an alternative TPG; (c) patterns generated by TPG in (d) and patterns observed at O

If R_2 is also a 2-bit register, it can be shown that there does not exist a TPG design that can functionally exhaustively test a kernel represented by the bipartite graph shown in Figure 4.18(a) using a pure 4-stage LFSR. Therefore, additional D F/Fs are required in the TPG design for this kernel. It can be observed from the above discussion that the TPG design for a balanced BISTable kernel using minimal number of F/Fs depends on the degree of polynomials and the polynomial

employed in configuring the LFSR. The polynomial selected to configure the LFSR in a TPG may greatly affect its effectiveness as illustrated in the above example. Therefore, the problem of designing a minimal test time and hardware TPG for a balanced BISTable kernel seems to involve a complicated process similar to the procedures discussed in [42, 38]. Unlike this complicated process, the TPG design procedures presented in this chapter are not sensitive to the polynomial selection. In addition, these procedures can be executed in polynomial time. The design of minimal test time and hardware TPGs remains an open problem.

4.8 Summary

In this chapter we have presented a TPG design scheme for the BIBS TDM that can be used in functionally exhaustive testing and functionally pseudo-exhaustive testing. The TPG design scheme employs a combination of LFSRs and SRs to balance the unequal sequential lengths in a kernel. A new test strategy called functionally pseudo-exhaustive testing was introduced in this chapter. This test strategy guarantees comprehensive fault coverage and requires less test time then that of functionally exhaustive testing.

A procedure for designing a minimal test time TPG for a uni-cone kernel is presented. For a multi-cone kernel, the number of LFSR stages and the test time of a TPG for the kernel often depend on the ordering of its input registers. For most practical circuits, it is feasible to permute the input registers of a kernel and execute Procedure *MC_TPG* for each permutation to reduce the test time. The two TPG design procedures presented in this chapter are both polynomial time algorithms, thus TPG designs for balanced BISTable kernels can be efficiently generated. We have studied the problem of TPG design using counters and pure LFSRs. It remains an open problem to generate minimal test time and hardware TPGs for balanced BISTable kernels.

Chapter 5

Kernel Identification

5.1 Introduction

In the previous chapters we have demonstrated the need to partition a CUC to identify kernels. Each kernel is a test primitive, that is, test patterns are generated and output responses are compressed outside of the kernel. In this dissertation we assume that kernels are disjoint. In addition, each kernel must be tested as one entity in the sense that once the kernel is under test, it is tested to its completion by applying all the required test patterns.

Two types of kernels are allowed in the BITS system, namely combinational kernels and balanced BISTable kernels. The CLARION system[16] partitions and clusters a circuit into combinational cells, fanout cells and registers. Some combinational cells have functions such as bus and MUX. These cells are usually constructed using regular structures and can be easily tested by applying a small set of functional test patterns. Due to this property, combinational cells such as MUXes and busses are often not considered as kernels during a normal test mode. Instead, additional functional test patterns are applied to these cells after the normal test mode to ensure coverage of faults in these cells. The CLARION system identifies combinational cells in a circuit that are not MUXes or busses as combinational kernels. When the BIBS TDM is desired, balanced BISTable kernels in the circuit can be identified using procedures to be presented in this chapter.

In addition to the fact that MUXes and busses are easily testable, they provide I-modes and I-paths[4] that allow different kernels to share test hardware. Sharing

of test hardware often leads to less area overhead. In this chapter we will first consider the problem of using I-paths in designing BISTable circuits to reduce their area overhead. When the number of combinational kernels is large, the amount of test hardware required to test these kernels may still be excessive even with the presence of I-paths. In such a case, the BIBS TDM that employs balanced BISTable kernels often leads to less test hardware overhead. We will present procedures to identify balanced BISTable kernels.

5.2 BISTable Circuit Design with I-paths

In this section we first define I-paths and I-modes. The procedure to identify I-paths in a circuit is then presented. When a kernel is not BISTable due to the absence of available test hardware, the CUC needs to be modified so that the required test hardware for the kernel is present. We will present a procedure to provide such test hardware by creating I-paths.

5.2.1 I-path and I-mode Definition

Let C be a combinational cell and P_i, P_o and P_c be an input port, an output port, and a control port, respectively, of C.

Definition 5.1 *A 4-tuple $IM_C = (P_i, P_o, P_c, v)$ is said to be an* **I-mode** *of C if an arbitrary pattern at P_i can be transferred to P_o unaltered by applying a control value v at P_c.*

Consider a MUX as shown in Figure 5.1, where D_1, D_2, D_3 and D_4 are input ports, O is an output port, and C is a control port. There exist four I-modes (D_1, O, C, `00`), (D_2, O, C, `01`), (D_3, O, C, `10`), and (D_4, O, C, `11`) in this MUX. When a cell has more than one I-mode and each I-mode can be activated by applying a unique control value at its control port, the cell is said to be a *switch*. Clearly MUXes and busses are switches. A register that only operates a LOAD function is an example circuit cell having only one I-mode.

Definition 5.2 *A data path is said to be an* **I-path** *if an arbitrary pattern can be transferred from the source of the path to its destination unaltered.*

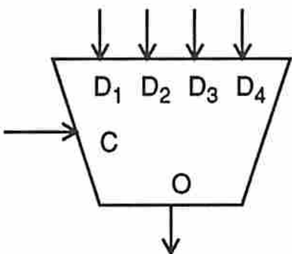


Figure 5.1: A MUX with four I-modes

Clearly each wire edge in a circuit graph represents an I-path. Since there is no logic associated with this I-path, it is called a *trivial I-path*. A non-trivial I-path is a data path that can be represented as $(C_0, (C_1, IM_{C_1}), \ldots, (C_{n-1}, IM_{C_{n-1}}), C_n)$, where an output port of C_i directly feeds an input port of C_{i+1} for $1 \leq i < n$. C_0 and C_n are the source and destination, respectively, of the I-path. Each C_i, $0 < i < n$, is a cell in the circuit having an I-mode IM_{C_i}. A trivial I-path can simply be represented as (C_0, C_1). An I-path P is said to be a *driving path* for a kernel K if the source of P is a register R and the destination of P is K. R is said to be the *driver* of the driving path and the path is said to drives K. Similarly, an I-path P is said to be a *receiving path* for a kernel K if the source of P is K and the destination of P is a register R. R is said to be the *receiver* of the receiving path and the path is said to receive from K.

5.2.2 I-path Identification

When two driving (receiving) paths have a common driver (receiver) and they drive (receive from) two different kernels, the driver (receiver) can be shared by these two kernels. To exploit the potential of sharing test hardware among kernels, driving paths and receiving paths for each kernel should be identified. The identification of these I-paths can be achieved by traversing a CUC from the input and output ports of every kernel in the circuit. The procedure for identifying driving paths and receiving paths, known as *find_I_path*, is presented below.

Procedure *find_I_path*

/* INPUT: a CUC with n kernels, K_1, K_2, ..., K_n. */

/* OUTPUT: all driving paths and receiving paths in the CUC. */

1. Let I_d denote the set of driving paths and I_r denote the set of receiving paths in the circuit. For $i = 1$ to n do steps 2 and 3.

2. For every input port D of K_i, let c be K_i and p be D. $path = (c)$.

   (a) if port p of c is directly fed by a PI, backtrack. Otherwise let s be the cell that directly feeds c through port p.

   (b) if $path$ contains s, backtrack.

   (c) if s is a register, $path = (s)||path$ and add $path$ to I_d.

   (d) let $IM(s)$ be the set of I-modes of s whose output port directly feeds c. If $IM(s)$ is empty, backtrack. Otherwise do the following steps.

      i. select an I-mode, say m, from $IM(s)$ and $IM(s) = IM(s) - \{m\}$.

      ii. let D' be the input port of m, $path = ((s,m))||path$, $p = D'$ and $c = s$. Goto step 2(a).

3. For every output port D of K_i, let c be K_i and p be D. $path = (K_i)$.

   (a) let S be the set of cells that are directly fed by c through port p.

   (b) if S is empty, backtrack. Otherwise select a cell, say s, from S and $S = S - \{s\}$. If $path$ contains s then goto step 3(b),

   (c) if s is a register, $path = path||(s)$ and add $path$ to I_r.

   (d) let $IM(s)$ be the set of I-modes of s whose input port is directly fed by c. If $IM(s)$ is empty, backtrack. Otherwise do the following steps.

      i. select an I-mode, say m, from $IM(s)$ and $IM(s) = IM(s) - \{m\}$.

      ii. let D' be the input port of m, $path = path||((s,m))$, $p = D'$ and $c = s$. Goto step 3(a).

The operation '||' denotes a *list catenation*, namely if $L_1 = (s_1, \ldots, s_m)$ and $L_2 = (t_1, \ldots, t_n)$, then $L_1||L_2 = (s_1, \ldots, s_m, t_1, \ldots, t_n)$. When Procedure *find_I_path* backtracks, its execution is directed to the state of the previous selection and $path$ is restored to the value at that state. Procedure *find_I_path* performs a depth first traversal of the circuit from the input and output ports of each kernels. The complexity of the procedure is $O(km)$, where k is the total number of input and output ports of the kernels in the circuit and m is the total number of cells in the circuit.

5.2.3 I-path Creation

When an input port of a kernel is fed directly by a PI or an output port of another kernel, there is no test hardware available for the input port. Similarly, when an output port of a kernel directly feeds a PO or an input port of another kernel, there is also no test hardware available for the output port. Under these circumstances, the CUC needs to be modified to provide the required test hardware. In the case where a port is fed by a PI or feeds a PO, the BISTable version of the circuit is able to provide the required test hardware by means of boundary scan registers. Currently the BITS system assumes that every PI is driven by a boundary scan register and every PO drives a boundary scan register, thus the required test hardware is provided.

On the other hand, when a kernel directly feeds another kernel, either a transparent register has to be added between these two kernels or additional I-paths are created to provide the test hardware. Consider a portion of a CUC as shown in Figure 5.2(a). There is no available test hardware for the output port P of kernel K_1. In Figure 5.2(b) a transparent register R' is added in the circuit to provide a test register for port P. This register is used only during the test mode and can only be used as a test register for K_1 or K_2. This may not be efficient in terms of test hardware utilization. In Figure 5.2(c) an alternative approach is employed where an existing register (R) is used as a test register for P by creating an additional I-path. This additional I-path is created by adding an extra input port to MUX M_2 and a set of wires from the output port of K_1 to this extra MUX input. Compared with the previous approach where a transparent register is added, this method usually has less area overhead and less extra circuit delay.

The current BITS system employs the latter approach in favor of its low area overhead and circuit delay. The procedure for adding a receiving path, known as *add_r_path*, is presented below.

Procedure *add_r_path*

/* INPUT: a port P of a kernel that has no available test hardware. */

/* OUTPUT: a receiving path for P. */

1. Select a register, say R, from the set of registers in the circuit.

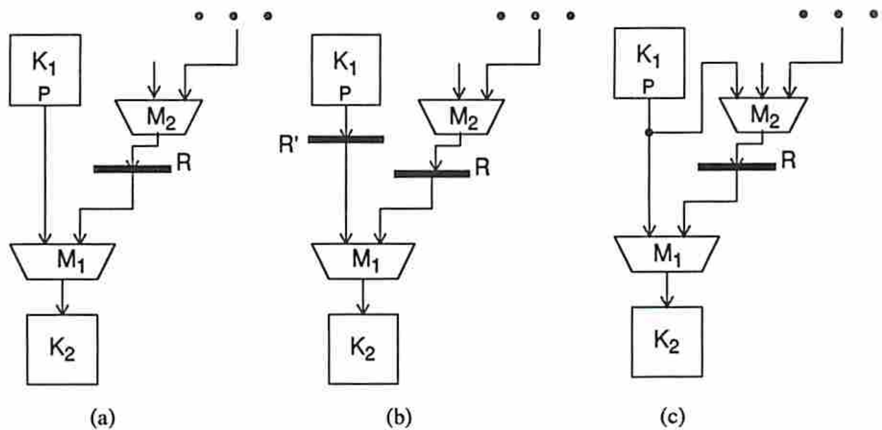


Figure 5.2: (a) A circuit with no available test hardware for port P; (b) adding a transparent register; (c) adding an I-path

2. If there exists an I-path from a MUX or bus to R, create an extra input port on the MUX or bus. Otherwise add a 2-to-1 MUX and connect one input port of the 2-to-1 MUX to the cell that drives R and connect the output port of the MUX to R.

3. Connect the unconnected input port of the MUX or bus in step 2 to port P by adding a set of wires to the circuit.

The procedure for adding a driving path is similar to Procedure *add_r_path* and is omitted. In general, more than one I-path can be created for the same port by selecting a set of registers and adding the required hardware. The selection of registers is based on their sharing potentials. A *sharing potential measure* is associated with each register, say R_i, denoted by $SP(R_i)$. $SP(R_i)$ is calculated as the number of driving paths and receiving paths that contain R_i. In the current BITS system, registers in a CUC are grouped so that registers in the same group have the same values of sharing potential measure. A register is then arbitrarily selected from each group. The additional I-paths are created using the selected registers and the I-path creation procedure presented above.

5.3 BISTable Circuit Design Using Balanced BISTable Kernels

Today's VLSI circuits are often complex and contain many combinational cells. As we demonstrated in Chapter 3, designing BISTable circuits using the BIBS TDM can lead to significantly less area overhead and performance degradation than other BIST TDM, and still achieve a comprehensive fault coverage. In this section the problem of designing BISTable circuit employing the BIBS TDM is referred to as the *BIBS design problem*. We will first define a class of circuits called *Series-Parallel Structure(SPS)*. A polynomial time procedure to make such a circuit BIBS testable with minimal area overhead is then presented. A procedure that employs a branch and bound technique and the procedure for SPSs is developed to generate BIBS designs for general (non-SPS) circuits. When functionally exhaustive or functionally pseudo-exhaustive testing is employed, the test length of a kernel is lower bounded by the maximal cone size of the kernel. When the maximal cone size is large, test length for the kernel may be excessive. Therefore, an extended procedure for the BIBS design is developed to constrain the maximal cone size of a kernel. Finally we will consider the problem of BIBS design when switches and I-paths are present.

5.3.1 Series-Parallel Structure(SPS)

Given a circuit graph G, let $\tilde{G}$ denote the *undirected* version of G and is called an *undirected circuit graph*. For each edge $e = (u, v)$ in $\tilde{G}$, u and v are said to be the *end vertices* of e. Given a pair of edges e_1, e_2 in $\tilde{G}$, e_1 and e_2 are *S-reducible*[43] if e_1 and e_2 share exactly one end vertex, say v, and $deg(v) = 2$. Let $e_1 = (x, v)$ and $e_2 = (v, y)$, an *S-reduction* can be applied to e_1 and e_2 that removes v from $\tilde{G}$ and replaces e_1 and e_2 with a new edge (x, y) (see Figure 5.3(a)). If e_1 and e_2 in $\tilde{G}$ have the same two end vertices x and y, then e_1 and e_2 are *P-reducible*[43]. A *P-reduction* can be operated on e_1 and e_2 that replaces e_1 and e_2 with a single edge e_3 having the same end vertices x and y (see Figure 5.3(b)). An edge e in $\tilde{G}$ is said to be *C-reducible* if the two end vertices of e are the same. A *C-reduction* can be applied to e that removes it from the graph (see Figure 5.3(c)). Finally, an

edge e in $\tilde{G}$ is said to be *I/O-reducible* if one of the end vertices of e is a PI or PO. An *I/O-reduction* can be operated on e that removes it from $\tilde{G}$. S-reductions, P-reductions, C-reductions and I/O reductions can be applied iteratively to an undirected circuit graph.

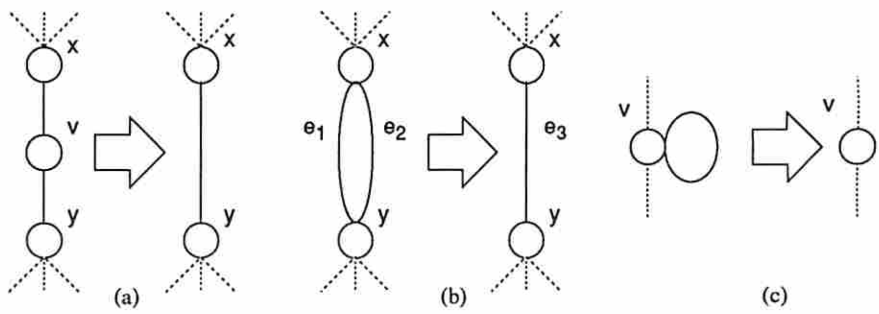


Figure 5.3: (a) S-reduction; (b) P-reduction; (c) C-reduction

A *maximal reduction* of an undirected circuit graph is the graph obtained after iteratively applying the above reductions until no further reduction can be made. A maximal reduction can be obtained by using a procedure similar to that in [43]. Note that the maximal reduction of a circuit graph is unique independent of the order of applying reductions. A circuit is called a *series-parallel structure(SPS)* or is said to be *connected in series-parallel* if the maximal reduction of its undirected circuit graph is a tree. The circuit shown in Figure 3.12(a) is a SPS since its maximal reduction is a tree as illustrated in Figures 5.4(a)-(f). The circuit shown in Figures 3.13(a) is not a SPS. In general, a circuit may not be a SPS itself, but may contain series-parallel sub-circuits.

5.3.2 BIBS Design for SPSs

To employ the BIBS TDM, a CUC is partitioned into a set of balanced BISTable structures, each being a kernel. In addition, for each kernel in the CUC there must be at least one register that drives every input port of the kernel and at least one register that receives from every output port of the kernel. These registers are converted to BILBO registers in the BISTable circuit. It is usually desirable to minimize the number of BILBO registers in a BISTable circuit since this often

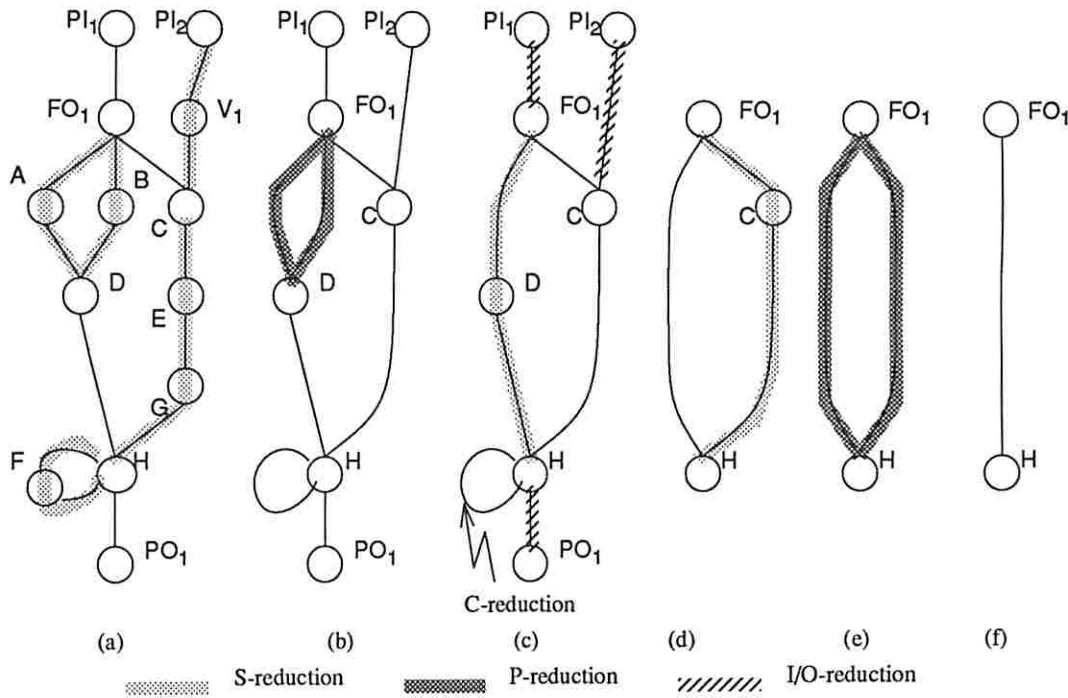


Figure 5.4: (a)-(f) maximal reduction of the circuit graph in Figure 3.3(b)

leads to the minimal area overhead and performance degradation. In this chapter we assume that an LFSR with L_1 stages has less area overhead than an LFSR with L_2 stages if $L_1 < L_2$. Therefore the area overhead of a BISTable circuit is proportional to the total number of LFSR stages in the BILBO registers employed in the circuit. A BIBS testable design for a CUC having the minimal area overhead is referred to as an *optimal BIBS solution*.

Recall that a path in G is said to form a cycle if it contains at least one register edge and starts and ends at the same vertex; a subgraph of G is called an URFS if it contains two vertices where the paths between them have unequal number of register edges; and a BILBO edge represents a register converted to a BILBO register in a BISTable circuit. A BILBO edge (u, v) is denoted as a *cut* in the circuit graph since the cells represented by u and v cannot be in the same kernel according to the third requirement for a balanced BISTable kernel. The circuit graph of an optimal BIBS solution is called an *optimal partition* since it contains a set of partitions where each is surrounded by BILBO edges (cuts). The BILBO

edges in an optimal partition are said to be associated with the optimal partition. A vertex v in a subgraph X of G is called an *input vertex* of X if v is directly fed by another vertex u in G that does not belong to X. Similarly, a vertex is called an *output vertex* of X if it directly feeds another vertex in G not belonging to X. A subgraph X is called a *simple cycle* if the edges and vertices in X form a cycle, and the input and output vertices of X are the same (see Figure 5.5). A *self-loop* is a simple cycle with only one register edge.

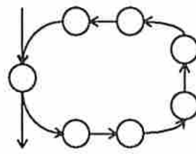


Figure 5.5: A simple cycle

According to the definition of a balanced BISTable structure (Definition 3.1), the circuit graph of such a structure must be free of cycles and URFSs. If a circuit graph contains URFSs or cycles, BILBO edges are required to balance the graph.

Theorem 5.1 There are exactly two BILBO edges in each simple cycle in any optimal partition.

Proof It has been shown that at least two BILBO edges are required in every cycle (Theorem 3.2). Since a simple cycle does not share edges with the remaining circuit graph, if there exists a third BILBO edge in the simple cycle, it does not benefit in balancing the circuit graph but only increases area overhead and thus can not lead to an optimal partition. The same is true if additional edges are converted to BILBO edges. Therefore, there are exactly two BILBO edges in each simple cycle in an optimal partition. □

When a simple cycle is a self-loop, the register in the cycle has to be converted to a CBILBO register. This is the only situation where CBILBO registers are allowed to identify BIBS kernels in this thesis. Note that there can be more than two BILBO edges in a non-simple cycle. Consider the circuit graph shown in Figure 5.6(a), where the subgraph X that contains vertices $\{a, b, c, d, e\}$ and edges $\{(a, b), (b, c), (c, d), (d, e), (e, a)\}$ is a non-simple cycle. The edges (a, b) and (b, c)

are BILBO edges due the cycle as shown in Figure 5.6(b); and the edges (c,d) and (d,e) are also BILBO edges due the URFS as shown in Figure 5.6(c). Therefore, the above 4 BILBO edges are associated with the optimal partition and they also belong to the non-simple cycle X.

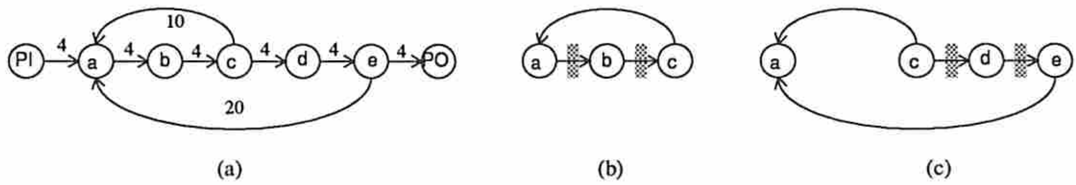


Figure 5.6: (a) an example circuit graph having non-simple cycles; (b) two BILBO edges (a,b) and (b,c); (c) another two BILBO edges (c,d) and (d,e)

Definition 5.3 *A subgraph $X = (V_X, E_X, u, v)$ of G is said to be an* **unbalanced structure (UBS)** *if the paths between u and v in X form cycles and/or URFSs, where $V_X \subseteq V$, $E_X \subseteq E$, and u and v are input and output vertices, respectively, of X. u is called the* **PI** *of X and v is called the* **PO** *of X.*

Based on this definition an UBS has exactly one PI and one PO. On the other hand, an UBS may have more than one input vertex or more than one output vertex. For example, the subgraph ($V = \{C_2, C_3, C_4\}$, $E = \{(C_2, C_3), (C_3, C_4), (C_2, C_4)\}$, $u = C_2$, $v = C_4$) in Figure 3.13(b) has two input vertices, C_2 and C_3, and one output vertex C_4. The subgraph is an UBS since the the paths between C_2 and C_4 form an URFS. Only C_2 is called the PI of the subgraph. An UBS Y is said to *contain* another UBS X if X is a subgraph of Y as in Figure 5.7. An UBS is *non-primitive* if it contains at least one other UBS; otherwise it is *primitive*. By definition, a simple cycle is an primitive UBS. In addition, the PI and the PO of a simple cycle are the same. To distinguish between an edge and a path in a circuit graph, where a path may contains a set of edges, we use thin arcs to represent edges and bold dotted arcs to represent paths in the figures in this chapter.

Theorem 5.2 Let X be an UBS with PI u and PO v. Assume that the sequential lengths of all parallel paths from u to v are different; and the sequential lengths of

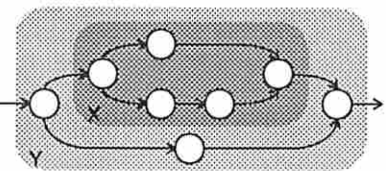


Figure 5.7: A non-primitive UBS Y contains X

all parallel paths from v to u are different. In an optimal partition of X, either (a) there is at least one BILBO edge in every path between u and v; or (b) there are at least two BILBO edges in all but one of the paths between u and v.

Proof Suppose the optimal partition of X is neither (a) nor (b). The former implies that there exists one path, say P_i, between u and v without BILBO edges. Under this assumption the latter implies that there exists another path P_j, where $P_j \neq P_i$, that has less than two BILBO edges. Clearly the subgraph consisting of P_i and P_j is either a cycle or an URFS. However, there is less than two BILBO edges in the subgraph, thus violating the condition described in Theorem3.2 □

Corollary 5.1 If X in Theorem 5.2 is primitive, then in a optimal partition of X either (a) there is one BILBO edge in every path between u and v (see Figure 5.8(a)); or (b) there are two BILBO edges in all but one of the paths between u and v (see Figure 5.8(b)).

Proof If X is primitive then there are exactly two BILBO edges in every cycle and URFS in an optimal partition of X. This is true since if a third BILBO edge is added, it does not benefit in balancing the graph but increases the hardware cost. The corollary follows from this observation. □

Theorem 5.3 In an optimal partition of a circuit graph G, there are at most two BILBO edges in every series path.

Proof If a series path is not contained in an UBS of G, the theorem is obviously true. If a series path is part of a primitive UBS, it follows from Corollary 5.1 that there are at most two BILBO edges in the path. Suppose a series path is part of a non-primitive UBS but not of a primitive UBS, such as the path P_1, P_2 or P_3 in Figure 5.9. In addition to the BILBO edges required to balance the primitive

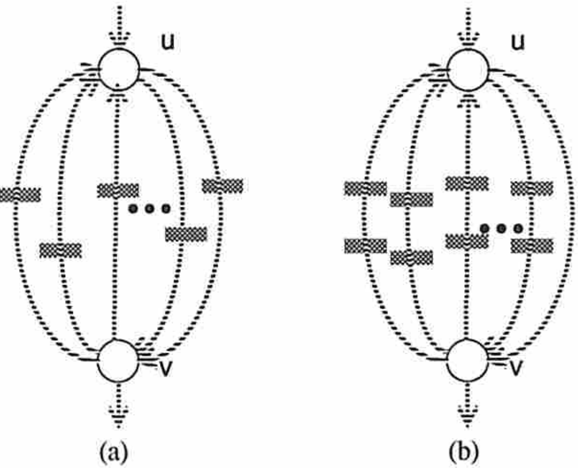


Figure 5.8: (a) One BILBO edge in every path between u and v; (b) two BILBO edges in all but one path between u and v

UBS, at most two BILBO edges in each of the paths are sufficient to balance the non-primitive UBS. Therefore, there are at most two BILBO edges in every series path in an optimal partition of G. □

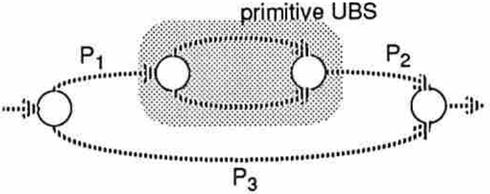


Figure 5.9: Series paths in a non-primitive UBS

Theorem 5.3 implies that only two edges in each series path need to be considered in finding an optimal partition. More specifically, these two edges are the ones with the smallest and second smallest weights. Since wire edges have infinite weights, only register edges will be considered. It can also be observed that if a specific register has been constrained so that it cannot be converted to a BILBO register, we can simply assign an infinite weight to its corresponding edge in the circuit graph. This might happen when a register is in a critical path of a circuit

and it is not allowed to be converted to a BILBO register since that would degrade the circuit performance. Based on this theorem a series path can be simplified by ignoring all the edges in the path except those two that have the smallest and second smallest weights. Next we will first introduce an extended graph model to represent a simplified circuit graph and extend the definitions of S-reduction, P-reduction, C-reduction and I/O-reduction to deal with a (directed) simplified circuit graph. Procedures for simplifying a circuit graph are then presented.

Given a circuit graph $G = (V, E, w)$, a *simplified circuit graph* is a directed graph represented by $\bar{G} = (\bar{V}, \bar{E})$, where $\bar{V} \subseteq V$ and $\bar{e} \in \bar{E} = \bar{V} \times \bar{V}$ is called a *composite edge*. In the original circuit graph, each edge corresponds to a connection in the circuit through a register or wires. On the other hand, more than one register can be represented by a composite edge in a simplified circuit graph. For each composite edge $\bar{e} = (u, v)$, u and v are called the *source* and *sink*, respectively, of $\bar{e}$. Associated with each composite edge $\bar{e}$ are the attributes $L(\bar{e})$, $W_1(\bar{e})$, $W_2(\bar{e})$, $\mathcal{E}_1(\bar{e})$, and $\mathcal{E}_2(\bar{e})$. $L(\bar{e})$ is called the *sequential length* of $\bar{e}$; $\mathcal{E}_1(\bar{e})$ and $\mathcal{E}_2(\bar{e})$ are called the *1st minimal cost edge set* and *2nd minimal cost edge set*, respectively, where each contains a set of edges in the original circuit graph; and $W_1(\bar{e})$ and $W_2(\bar{e})$ are the total weights of the edges in $\mathcal{E}_1(\bar{e})$ and $\mathcal{E}_2(\bar{e})$, respectively. Initially $\bar{V} = V$; $\bar{E}$ contains a set of composite edges where each of them, say $\bar{e}$, corresponds to an edge e in G. $L(\bar{e}) = 1$ if e is a register edge and 0 otherwise; $\mathcal{E}_1(\bar{e}) = \{e\}$; $\mathcal{E}_2(\bar{e}) = \emptyset$; $W_1(\bar{e}) = w(e)$ and $W_2(\bar{e}) = \infty$. As the simplification process (to be described) proceeds, $\bar{G}$ is modified where new composite edges are created by combining existing composite edges.

Definition 5.4 *Two composite edges $\bar{e}_1 = (x, v)$ and $\bar{e}_2 = (v, y)$ are said to be* **S*-reducible** *if $deg(v) = 2$. An* **S*-reduction** *removes v from $\bar{G}$ and replaces $\bar{e}_1$ and $\bar{e}_2$ with a single composite edge $\bar{e} = (x, y)$. $L(\bar{e}) = L(\bar{e}_1) + L(\bar{e}_2)$; $W_1(\bar{e})$ and $W_2(\bar{e})$ are the smallest and second smallest among $W_1(\bar{e}_1)$, $W_2(\bar{e}_1)$, $W_1(\bar{e}_2)$ and $W_2(\bar{e}_2)$; and $\mathcal{E}_1(\bar{e})$ and $\mathcal{E}_2(\bar{e})$ are the edge sets among $\mathcal{E}_1(\bar{e}_1)$, $\mathcal{E}_2(\bar{e}_1)$, $\mathcal{E}_1(\bar{e}_2)$ and $\mathcal{E}_2(\bar{e}_2)$ that have the total weights of $W_1(\bar{e})$ and $W_2(\bar{e})$, respectively. When a tie exists, it is broken arbitrarily.*

Definition 5.5 *Two parallel composite edges $\bar{e}_1$ and $\bar{e}_2$ are said to be* **P*-reducible** *if $L(\bar{e}_1) = L(\bar{e}_2)$. A* **P*-reduction** *replaces $\bar{e}_1$ and $\bar{e}_2$ with a single*

composite edge $\bar{e}$ having the same source and sink as $\bar{e}_1$ (or $\bar{e}_2$). $L(\bar{e}) = L(\bar{e}_1)$; $W_1(\bar{e}) = W_1(\bar{e}_1) + W_1(\bar{e}_2)$; $W_2(\bar{e}) = W_2(\bar{e}_1) + W_2(\bar{e}_2)$; $\mathcal{E}_1(\bar{e}) = \mathcal{E}_1(\bar{e}_1) \cup \mathcal{E}_1(\bar{e}_2)$; and $\mathcal{E}_2(\bar{e}) = \mathcal{E}_2(\bar{e}_1) \cup \mathcal{E}_2(\bar{e}_2)$.

Definition 5.6 *A composite edge $\bar{e}$ is said to be* **C*-reducible** *if $\bar{e}$ forms a simple cycle. A* **C*-reduction** *removes $\bar{e}$ from $\bar{G}$ and associates the edges in $\mathcal{E}_1(\bar{e})$ and $\mathcal{E}_2(\bar{e})$ with the optimal partition of the circuit graph.*

Definition 5.7 *A composite edge $\bar{e} = (u, v)$ is said to be* **I/O*-reducible** *if either u is a PI vertex or v is a PO vertex. An* **I/O*-reduction** *removes $\bar{e}$ and the PI/PO vertex from $\bar{G}$.*

Note that every PI/PO register needs to be configured as a TPG/SA in a BIBS testable circuit, hence the register edges in the original circuit graph that connect PI/POs are converted to BILBO edges. Once this is done and an initial simplified circuit graph is constructed, we can simplify the graph by iteratively applying S*-reductions, P*-reductions, C*-reductions and I/O*-reductions to it until no further reductions can be made. Clearly the simplification process does not affect the optimality of a BIBS solution since all the edges that correspond to the BILBO edges associated with an optimal partition are either converted to BILBO edges after the simplification, or retained in the simplified circuit graph. Procedure *ckt_simplify* is used to generate a simplified circuit graph and is presented below.

Procedure *ckt_simplify*(G)
/* INPUT: A circuit graph $G = (V, E, w)$. */
/* OUTPUT: A simplified circuit graph $\bar{G} = (\bar{V}, \bar{E})$. */

1. Convert all PI/PO register edges in G to BILBO edges.
2. Construct an initial simplified circuit graph $\bar{G}$ from G.
3. While ($\exists\ v \in \bar{V}$ so that $deg(v) = 2$) do *S*_reduce*(v).
4. Let *p_flag* = `false`. For every pair of $u, v \in \bar{V}$ do
    (a) collect all the edges in $\bar{E}$ having u as source and v as sink in *p_list*.
    (b) if ($|p_list| > 1$) then *P*_reduce*(u, v, *p_list*).

5. If (p_flag = `true`) goto step 3.

6. Let c_flag = `false`. While ($\exists$ a C*-reducible edge $\bar{e}$) do $C^*_reduce(\bar{e})$.

7. If (c_flag = `true`) goto step 3.

8. Let I/O_flag = `false`. While ($\exists$ $(u, v) \in \bar{E}$ so that u is a PI or v is a PO) do $I/O^*_reduce(u, v)$.

9. If (I/O_flag = `true`) goto step 3, otherwise return($\bar{G}$).

Procedure $S^*_reduce(v)$

1. Let x and y be the vertices where $(x, v), (v, y) \in \bar{E}$.

2. Apply an S*-reduction on (x, v) and (v, y) to obtain a new composite edge (x, y). $\bar{E} = \bar{E} - \{(x, v), (v, y)\} \cup \{(x, y)\}$ and $\bar{V} = \bar{V} - \{v\}$.

Procedure $P^*_reduce(u,\ v,\ p_list)$

1. Mark every composite edge in p_list as `unprocessed`.

2. While (p_list contains `unprocessed` edges) do

   (a) let $\bar{e}$ be the first `unprocessed` edge in p_list, mark $\bar{e}$ as `processed`, $p_set = \{\bar{e}\}$.

   (b) for every `unprocessed` $\bar{e}' \in p_list$ if $(L(\bar{e}') = L(\bar{e}))$, then $p_set = p_set$ $\cup \{\bar{e}'\}$ and mark $\bar{e}'$ as `processed`.

   (c) if $(|p_set| > 1)$, then apply a P*-reduction on the composite edges in p_set to obtain a new composite edge $e_\bar{new}$. Mark $e_\bar{new}$ as `processed`. $\bar{E} = \bar{E} - p_set \cup \{e_\bar{new}\}$, $deg(u) = deg(u)$ - $|p_set| + 1$, $deg(v) = deg(v)$ - $|p_set| + 1$. Set p_flag to `true`.

Procedure $C^*_reduce(\bar{e})$

1. Let $\bar{e}$ be the composite edge (v, v).

2. Convert edges in $\mathcal{E}_1(\bar{e})$ and $\mathcal{E}_2(\bar{e})$ (if any) to BILBO edges. $\bar{E} = \bar{E} - \{\bar{e}\}$ and $deg(v) = deg(v) - 2$.

3. Set c_flag to `true`.

Procedure $I/O^*_reduce(u,\ v)$

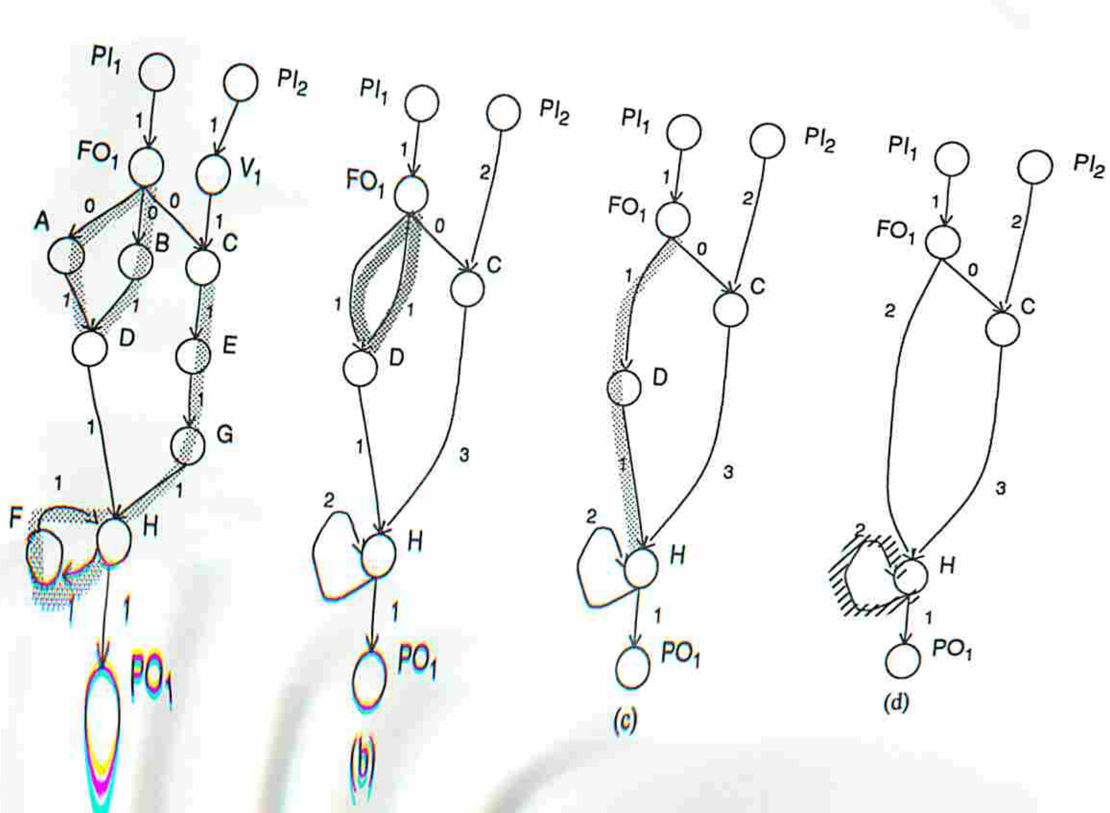
1. If (u is a PI) then $\bar{V} = \bar{V} - \{u\}$ and $deg(v) = deg(v) - 1$; otherwise $\bar{V} = \bar{V} - \{v\}$ and $deg(u) = deg(u) - 1$.
2. $\bar{E} = \bar{E} - \{(u, v)\}$ and set *I/O_flag* to `true`.

Procedure *ckt_simplify* iteratively applies S*-reductions, P*-reductions, C*-reductions and I/O*-reductions to simplify a circuit graph. When no further reductions can be made, $\bar{G}$ is returned as the simplified circuit graph. Since an S*-reduction, C*-reduction or I/O*-reduction can be applied to each vertex at most once and a P*-reduction can be applied to each pair of vertices at most once, the maximal number of reductions applied to a circuit graph is bounded by $O(|V|^2)$. Each reduction can be done in constant time, thus Procedure *ckt_simplify* is a polynomial time ($O(|V|^2)$) algorithm.

Example 5.1 Consider the circuit graph as shown in Figure 3.12(b). An initial simplified circuit graph is shown in Figure 5.10(a) where the sequential length of each composite edge is indicated. An execution trace of Procedure *ckt_simplify* on this graph is shown in Figures 5.10(a)-(g).

S*-reductions are first applied as shown in Figure 5.10(a), followed by P*-reductions as shown in Figure 5.10(b). Once the new composite edge (FO_1, D) is generated, another S*-reduction is applied as shown in Figure 5.10(c). No P*-reduction can be applied at this point, thus a C*-reduction is applied as shown in Figure 5.10(d) and it is followed by I/O*-reductions as shown in Figure 5.10(e). Once the composite edges (PI_1, FO_1), (PI_2, C) and (H, PO_1) are removed from the graph, an S*-reduction can again be applied as shown in Figure 5.10(f). The final simplified circuit graph is shown in Figure 5.10(g) which contains an URFS, thus additional BILBO edges are required to make the circuit BIBS testable. □

If a circuit graph does not contain non-simple cycles and URFSs, a tree structure will be generated by Procedure *ckt_simplify* and an optimal BIBS solution can be easily obtained. In the presence of non-simple cycles and URFSs, a simplified circuit graph is not a tree structure. To make such a circuit BIBS testable, BILBO edges in addition to those converted in Procedure *ckt_simplify* are required. In the remainder of this section, we will restrict our attention to a simplified circuit graph obtained from Procedure *ckt_simplify*.



PI1
PI2
1
1
FO1
V1
0
0
0
1
A
B
C
1
1
1
D
E
1
1
G
1
1
1
F
H
1
1
PO1
PI1
PI2
1
2
FO1
0
C
1
1
D
1
3
2
H
1
PO1
(b)
PI1
PI2
1
2
FO1
0
1
C
D
3
1
2
H
1
PO1
(c)
PI1
PI2
1
2
FO1
0
2
C
3
2
H
1
PO1
(d)

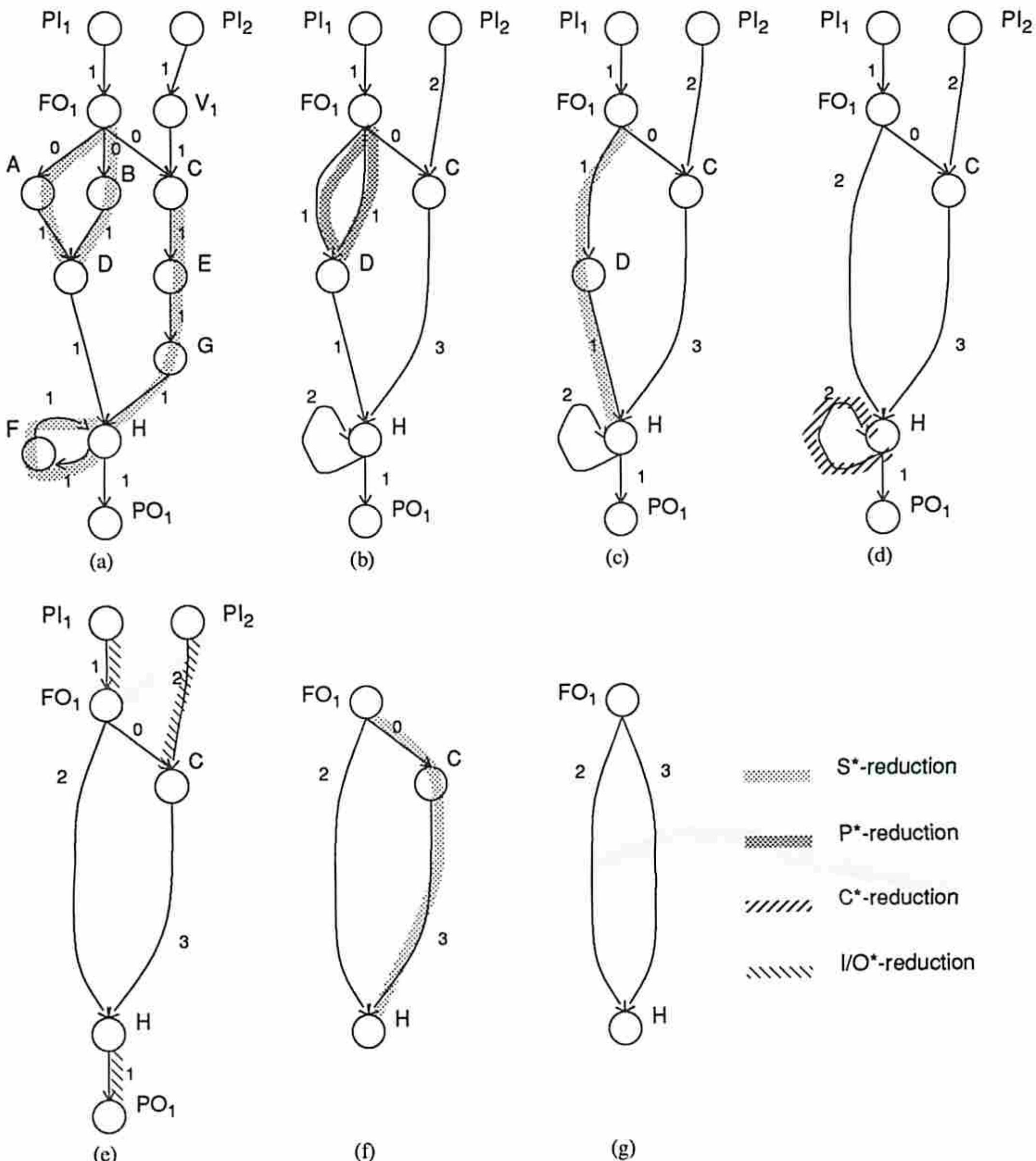


Figure 5.10: (a)-(g) an execution trace of Procedure *ckt_simplify*

Recall that a primitive UBS is an UBS that does not contain any other UBSs. The level of a primitive UBS is defined to be 1. The level of an UBS that only contains primitive UBSs is 2. In general, the level of an UBS that only contains UBSs of level n (or less) is $n+1$. Next we will present a few theorems that lead to an efficient algorithm for finding an optimal partition of circuit graph G. Let Θ denote the set of BILBO edges associated with an optimal partition of G. An optimal partition of a subgraph X is called a *locally optimal partition* of X, and the set of BILBO edges associated with it is denoted by $\Theta(X)$. A vertex v is said to be an *articulation point* in G if there exist vertices x and y in G such that v, x and y are distinct, and every path between x and y in the *undirected* version of G contains v.

Theorem 5.4 For an UBS X with PI u and PO v, if both u and v are *articulation points* in G, then $\Theta(X)$ must be a subset of Θ.

Proof Since u and v are articulation points in G, X cannot be contained in any other UBSs. Therefore, the BILBO edges added to X to balance X has no effects on the subgraph $(G-X)$ (in terms of balance property). Similarly, the BILBO edges added to $(G-X)$ to balance $(G-X)$ has no effects on X. In other words, the BILBO edges in $\Theta(X)$ and the BILBO edges in $\Theta(G-X)$ are mutually exclusive. An optimal partition of G can be obtained by finding locally optimal partitions of X and $(G-X)$ independently and Θ is simply $\Theta(X) \cup \Theta(G-X)$. □

Given a level 1 UBS X with PI u and PO v. Suppose there are m paths between u and v (in either direction) denoted by P_1, P_2, ..., P_m. Each P_i, $1 \leq i \leq m$, is a composite edge e_i in the simplified circuit graph. Therefore, it is necessary and sufficient to either convert edges in every $\mathcal{E}_1(e_i)$ for $i = 1$ to m to BILBO edges; or leave one edge, say e_k, intact and convert edges in both $\mathcal{E}_1(e_i)$ and $\mathcal{E}_2(e_i)$, for $i = 1$ to m and $i \neq k$, to BILBO edges to balance X (Corollary 5.1). Let $S_0(X)$ denote the set of BILBO edges associated with the solution that converts edges in every $\mathcal{E}_1(e_i)$ for $i = 1$ to m to BILBO edges. Let $S_k(X)$ denote the set of BILBO edges associated with the solution that leaves e_k intact and converts edges in both $\mathcal{E}_1(e_i)$ and $\mathcal{E}_2(e_i)$, for $i = 1$ to m and $i \neq k$, to BILBO edges. Clearly $\Theta(X)$ can be obtained from these $m+1$ solutions by selecting the one with the minimal cost

(minimal total weight). Without loss of generality, we assume that $S_1(X)$ has the smallest cost among $S_k(X)$ for $k = 1$ to m.

Theorem 5.5 Let Y be an UBS of level 2 that contains X, then $\Theta(Y)$ must contain either $S_0(X)$ or $S_1(X)$.

Proof Figure 5.11(a) shows UBSs X and Y, where the PI and PO of Y are denoted by u' and v', respectively, and there is an extra path P_{m+2} between u' and v' that does not share edges with P_i, $1 \leq i \leq m$. In general there can be more than one such path. In addition, the paths P_0, P_{m+1} and P_{m+2} in Figure 5.11(a) may contain other level 1 UBSs. However, the theorems, proofs, and procedures presented in this chapter for the simple case can easily be extended to deal with a general graph. The partitions of X that employ BILBO edges in $S_0(X)$ and $S_1(X)$ are shown in Figures 5.11(b) and (c), respectively. Recall that a BILBO edge is denoted by a cut in a circuit graph. In general, a cut on a path in a simplified circuit graph can contain a set of BILBO edges. For example, the cut on P_1 in Figure 5.11(b) contains the edges in $\mathcal{E}_1(e_1)$; similarly the two cuts on P_2 in Figure 5.11(c) contains the edges in $\mathcal{E}_1(e_2)$ and $\mathcal{E}_2(e_2)$.

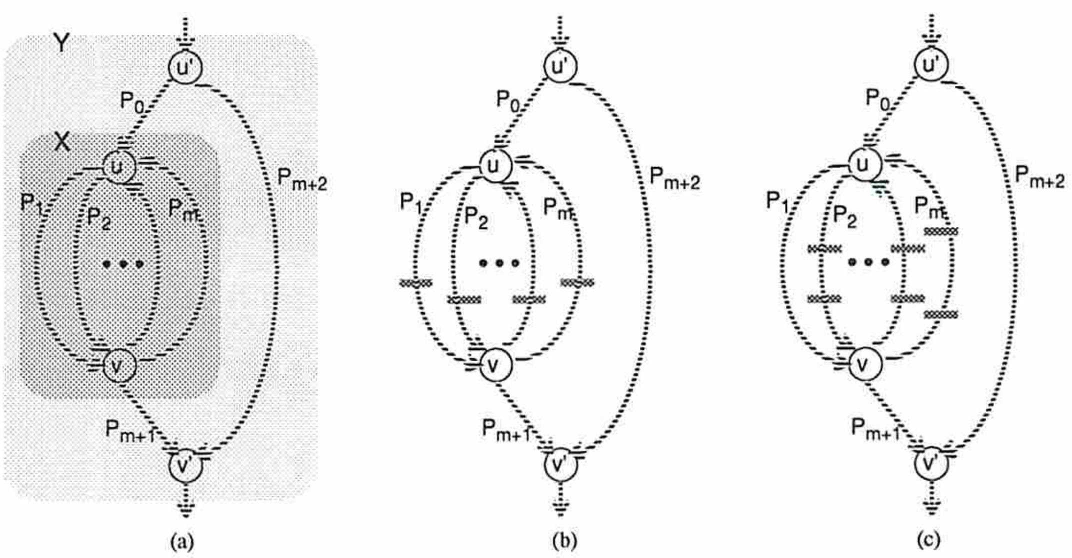


Figure 5.11: (a) UBSs X and Y; (b) partition of X using $S_0(X)$; (c) partition of X using $S_1(X)$

Let $C_{k,1}$ denote the set of BILBO edges when one cut is placed on P_k and $C_{k,2}$

denote the set of BILBO edges when two cuts are placed on P_k. Clearly there must either be at least one cut on every path between u' and v', or be at least two cuts one all but one of the paths between u' and v' to balance Y. Therefore, the set of BILBO edges associated with a partition of Y contains some $S_i(X)$, where $0 \leq i \leq m$. If $\Theta(Y)$ does not contain $S_0(X)$ or $S_1(X)$, then it must contain a $S_k(X)$, where $1 < k \leq m$, and be one of the followings.

1. $S_k(X) \cup C_{0,1} \cup C_{m+2,1}$ (at least one cut on every path between u' and v'.)
2. $S_k(X) \cup C_{m+1,1} \cup C_{m+2,1}$ (at least one cut on every path between u' and v'.)
3. $S_k(X) \cup C_{0,2}$ (at least two cuts on every path between u' and v' except P_{m+2}.)
4. $S_k(X) \cup C_{m+1,2}$ (at least two cuts on every path between u' and v' except P_{m+2}.)
5. $S_k(X) \cup C_{0,1} \cup C_{m+1,1}$ (at least two cuts on every path between u' and v' except P_{m+2}.)
6. $S_k(X) \cup C_{m+2,2}$ (at least two cuts on every path between u' and v' except the path composed of P_0, P_k and P_{m+1}.)

However, if $S_k(X)$ is replaced by $S_1(X)$ in each of the above solutions, it is still a valid partition of Y but with less cost. Therefore, none of the above partitions can be locally optimal, thus contradicts the assumption and $\Theta(Y)$ must contain either $S_0(X)$ or $S_1(X)$. □

Based on Theorem 5.5, $\Theta(Y)$ can be obtained by choosing one of the following solutions that has the minimal cost.

1. $S_0(X) \cup C_{m+2,1}$	6. $S_1(X) \cup C_{0,2}$
2. $S_1(X) \cup C_{0,1} \cup C_{m+2,1}$	7. $S_1(X) \cup C_{1,2}$
3. $S_1(X) \cup C_{m+1,1} \cup C_{m+2,1}$	8. $S_1(X) \cup C_{m+1,2}$
4. $S_0(X) \cup C_{0,1}$	9. $S_1(X) \cup C_{m+2,2}$
5. $S_0(X) \cup C_{m+1,1}$	10. $S_1(X) \cup C_{0,1} \cup C_{m+1,1}$

It should be noted that in solutions 1 to 3 there is at least one cut on every path between u' and v'; and in solutions 4 to 10 there are at least two cuts on all

but one of the paths between u' and v'. In general, for an UBS Y of level n with PI u, PO v, and m paths $P_1, P_2, \ldots, P_m$ between u and v, $\Theta(Y)$ can be obtained from the $m+1$ solutions listed below by choosing the one with the minimal cost.

- $S_0(Y)$: at least one cut on P_i for $1 = 1$ to m.
- $S_k(Y)$, $1 \leq k \leq m$: at least two cuts on P_i for $i = 1$ to m, $i \neq k$.

Corollary 5.2 Let Y be an UBS of level n and Z be an UBS of level $n+1$ that contains Y. Without loss of generality, assume that $S_1(Y)$ has the smallest cost among $S_k(Y)$, $1 \leq k \leq m$. Then $\Theta(Z)$ must contain either $S_0(Y)$ or $S_1(Y)$.
Proof Similar to the proof for Theorem 5.5 and omitted. □

Theorem 5.6 Let X be an UBS of level 1 and Y be an UBS of level 2 that contains X. $S_0(X)$ and $S_1(X)$ are the same as in Theorem 5.5. If the cost of $S_0(X)$ is less than that of $S_1(X)$, then $\Theta(Y)$ contains $S_0(X)$.
Proof It has been shown in Theorem 5.5 that $\Theta(Y)$ contains either $S_0(X)$ or $S_1(X)$. Suppose that $\Theta(Y)$ contains $S_1(X)$, then $\Theta(Y)$ must be one of the followings.

1. $S_1(X) \cup C_{0,1} \cup C_{m+2,1}$ (at least one cut on every path between u' and v'.)
2. $S_1(X) \cup C_{m+1,1} \cup C_{m+2,1}$ (at least one cut on every path between u' and v'.)
3. $S_1(X) \cup C_{0,2}$ (at least two cuts on every path between u' and v' except P_{m+2}.)
4. $S_1(X) \cup C_{m+1,2}$ (at least two cuts on every path between u' and v' except P_{m+2}.)
5. $S_1(X) \cup C_{0,1} \cup C_{m+1,1}$ (at least two cuts on every path between u' and v' except P_{m+2}.)
6. $S_1(X) \cup C_{m+2,2}$ (at least two cuts on every path between u' and v' except the path composed of P_0, P_k and P_{m+1}.)

However, if $S_1(X)$ is replaced by $S_0(X)$ in each of the above solutions, it is still a valid partition of Y but with less cost. Therefore, none of the above partitions can be locally optimal. This contradicts the assumption and thus $\Theta(Y)$ contains $S_0(X)$. □

Corollary 5.3 Let Y be an UBS of level n and Z be an UBS of level $n+1$ that contains Y. $S_0(Y)$ and $S_1(Y)$ are the same as in Corollary 5.2. If the cost of $S_0(Y)$ is less than that of $S_1(Y)$, then $\Theta(Z)$ contains $S_0(Y)$.

Proof Similar to the proof for Theorem 5.6 and omitted. □

A procedure to generate an optimal partition of a circuit graph is developed as described below. UBSs in the circuit graph are first enumerated and levelized. The procedure incrementally partitions UBSs from the lowest level to the highest level. In each iteration only two solutions for the UBS under consideration, say X, need to be produced, namely the solutions that employ $S_0(X)$ and $S_1(X)$. These two solutions are then employed in the next iteration to produce new partitions of the UBS that contains X. According to the theorems presented previously, this procedure guarantees to produce an optimal partition of a circuit graph. The procedure, known as *SPS_balance*, is presented below.

Procedure *SPS_balance*($\bar{G}$)
/* INPUT: A simplified circuit graph $\bar{G}$. */
/* OUTPUT: The set of BILBO edges associated with an optimal partition of $\bar{G}$. */

1. $\Theta = \emptyset$, U = enum_UBS($\bar{G}$). Levelize UBSs in U.
2. If (U not empty) let $i = 1$ and do steps 3-6. Otherwise return (Θ).
3. Select an UBS of level i, say X, from U and find two solutions, $S_0(X)$ and $S_1(X)$, for X as defined in Theorem 5.5.
4. If the PI u and the PO v of X are articulation points in $\bar{G}$, remove X and the UBSs contained in X from U and do
   (a) if ($S_0(X)$ costs less than $S_1(X)$), $\Theta = \Theta \cup S_0(X)$.
   (b) otherwise $\Theta = \Theta \cup S_1(X)$.
   (c) goto step 2.
5. Otherwise let Y denote the UBS of level $i+1$ that contains X. Generate $S_0(Y)$ and $S_1(Y)$ as follows.
   (a) if ($S_0(X)$ costs less than $S_1(X)$) then
      i. $S_0(Y) = S_0(X) \cup C_{m+2,1}$.
      ii. $S_1(Y)$ is one of the following with less cost.

- $S_0(X) \cup C_{0,1}$
- $S_0(X) \cup C_{m+1,1}$

(b) otherwise

i. $S_0(Y)$ is one of the following with the least cost.

- $S_0(X) \cup C_{m+2,1}$
- $S_1(X) \cup C_{0,1} \cup C_{m+2,1}$
- $S_1(X) \cup C_{m+1,1} \cup C_{m+2,1}$

ii. $S_1(Y)$ is one of the following with the least cost.

- $S_0(X) \cup C_{0,1}$
- $S_0(X) \cup C_{m+1,1}$
- $S_1(X) \cup C_{0,2}$
- $S_1(X) \cup C_{1,2}$
- $S_1(X) \cup C_{m+1,2}$
- $S_1(X) \cup C_{m+2,2}$
- $S_1(X) \cup C_{0,1} \cup C_{m+1,1}$

6. $X = Y$, $i = i + 1$ and goto step 4.

Procedure *enum_UBS*$(\bar{G})$

1. Let $U = \emptyset$. Create a copy of $\bar{G}$ called $\bar{G}'$.
2. For every pair of vertices u and v in $\bar{G}'$ do
   (a) put all edges in $\bar{G}'$ that have u as source and v as sink or v as source and u as sink into the set *p_set*.
   (b) if ($|p_set| > 1$) create an UBS and associate every edge in *p_set* with the UBS. Add this UBS to U.
   (c) apply a P*-reduction (ignoring the sequential lengths) on edges in *p_set*, followed by S*-reductions on the new edge generated if applicable.
3. Return(U).

Step 1 can be executed in $O(|\bar{V}|^2)$ since the complexity of Procedure *enum_UBS* is bounded by $O(|\bar{V}|^2)$. The greatest value for the level of an UBS in $\bar{G}$ is bounded by $O(|\bar{E}|)$, thus steps 3-6 are executed at most $O(|\bar{E}|)$ times. Steps 3 and 5 can be executed in $O(n)$, where n is the number of paths between the PI and the PO in the UBS under consideration. Steps 4 and 6 can be executed in constant time. Since n is also bounded by $O(|\bar{E}|)$, the computational complexity of Procedure *SPS_balance*

is bounded by $O(|\bar{E}|^2)$. Once UBSs in a circuit graph are balanced and removed from the graph by the above procedure, the remaining graph only contains trees and reverse trees, where trees represent sub-circuits with fan-outs (see Figure 5.12(a)), and reverse trees represent sub-circuits with fan-ins (see Figure 5.12(b)). These structures are balanced by definition and therefore the corresponding circuit is BIBS testable. Therefore, an optimal BIBS solution for a SPS can be generated in polynomial time using Procedures *ckt_simplify* followed by Procedure *SPS_balance*.

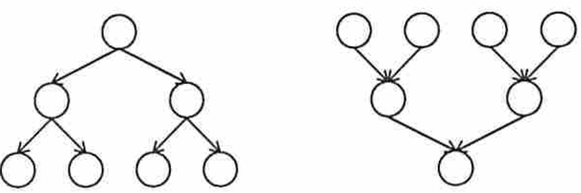


Figure 5.12: (a) A tree; (b) a reverse tree

Example 5.2 Consider an example circuit graph shown in Figure 5.13(a) which is a SPS. By applying Procedure *ckt_simplify*, the simplified circuit graph shown in Figure 5.13(b) is obtained where a set of attributes are associated with each composite edge. For example, $L(e_1) = 1$, $W_1(e_1) = 6$, $W_2(e_1) = \infty$, $\mathcal{E}_1(e_1) = \{(A, C)\}$ and $\mathcal{E}_2(e_1) = \emptyset$; $L(e_6) = 4$, $W_1(e_6) = 6$, $W_2(e_6) = 10$, $\mathcal{E}_1(e_6) = \{(D, F)\}$ and $\mathcal{E}_2(e_6) = \{(L, Q)\}$. It should be noted that during the simplification process, a C*-reduction has been applied to vertex M and the edges (M, N) and (N, M) are converted to BILBO edges; the edges (PI_1, A), (PI_2, A), (PI_3, B), (PI_4, B), (S, PO_1) and (S, PO_2) are PI/PO edges and also converted to BILBO edges.

Since the simplified circuit graph is not a tree structure, additional BILBO edges are required to balance the graph. The shaded subgraphs X, Y and Z in Figure 5.13(b) are the UBSs in the circuit graph with levels 1, 2 and 3, respectively. Procedure *SPS_balance* first finds $S_0(X) = \{(E, G), (E, H)\}$ and $S_1(X) = \{(E, G), (G, K)\}$. Since $S_0(X)$ costs less than $S_1(X)$, only the former needs to be considered in the subsequent process (Theorem 5.6). By using $S_0(X)$, the two solutions for Y are obtained as $S_0(Y) = \{(E, G), (E, H), (D, F)\}$ and $S_1(Y)$

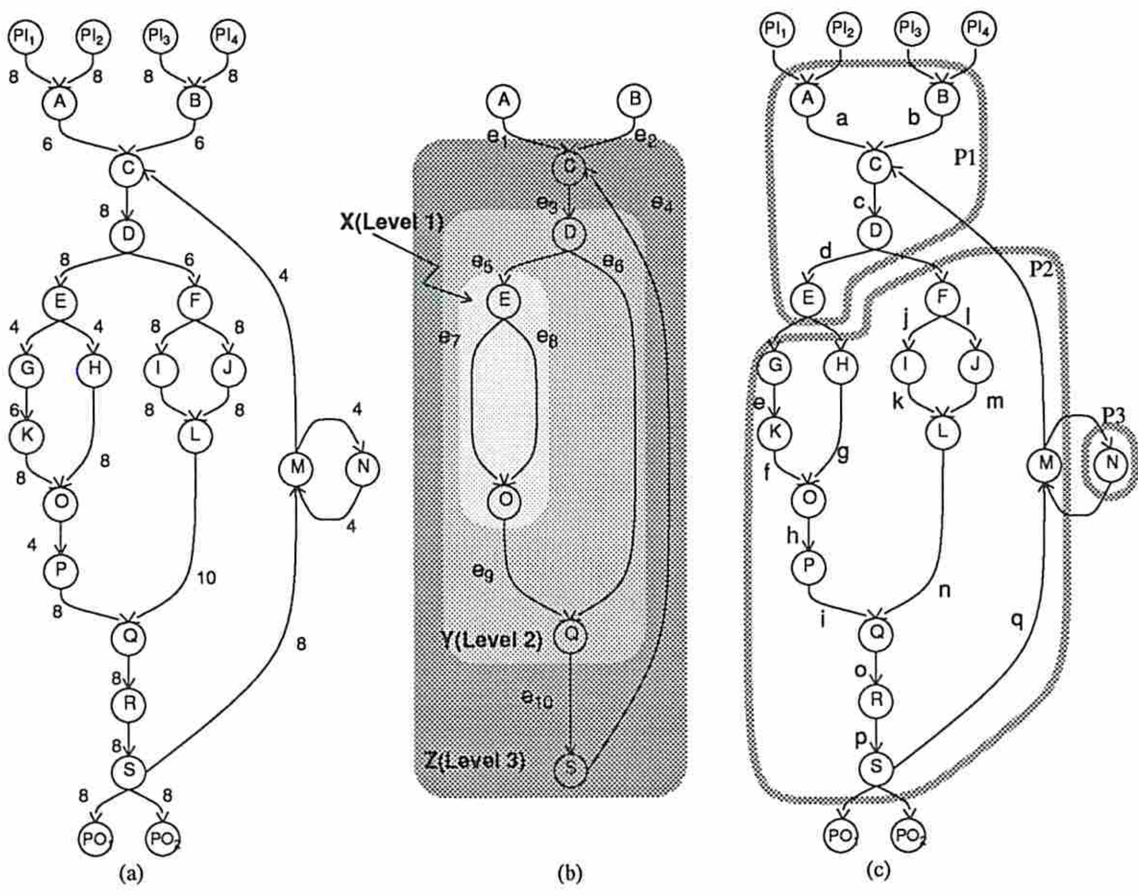


Figure 5.13: (a) An example circuit graph; (b) simplified circuit graph; (c) partitions of the circuit graph

= $\{(E,G), (E,H), (O,P)\}$. Among these two solutions, $S_0(Y)$ leads to an optimal partition that requires the set of BILBO edges $\Theta = \{(E,G), (E,H), (D,F), (M,C)\}$. Figure 5.13(c) shows this optimal partition of the circuit graph where each partition corresponds to a kernel during test. Edges between partitions and between a partition and a PI/PO represent BILBO edges. When pseudo-random testing is employed, each register represented by a BILBO edge is configured as a TPG if it feeds a kernel, and is configured as a SA if it is fed by a kernel. A register is configured as both a TPG and a SA if it feeds a kernel and is fed by another kernel. When functionally exhaustive testing is employed, registers that feed the same kernel are configured as a single TPG by using Procedure *UC_TPG* or *MC_TPG* presented in Chapter 4. For example, a TPG to test the kernel represented by $P2$ can be generated by Procedure *UC_TPG* and is shown in Figure 5.14. A maximal length LFSR of degree 18 is required in the TPG design since the maximal cone size of the kernel is 18. An extra F/F is added between the registers feeding G and H due to the unequal sequential lengths from the TPG to the output cone(s). Similarly, 6 extra F/Fs are added between the registers feeding F and M. The TPG is 25 bits wide due to these extra F/Fs. However, the test time to functionally exhaustively test this kernel is approximately 2^{18}. □

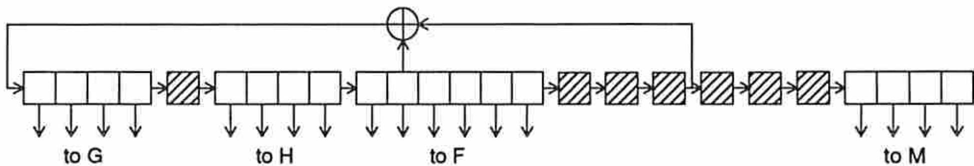


Figure 5.14: TPG design for kernel represented by $P2$

Notice that Procedure *SPS_balance* only generates one BIBS solution that is optimal in terms of area overhead. In general, there may be more than one optimal solution and each of the optimal solutions may lead to a BISTable circuit with different values on attributes such as test application time and performance degradation. In addition, the optimal solution generated may lead to a BISTable circuit that requires a register on a critical path to be converted to a BILBO register. This may not be acceptable if the penalty on circuit performance is significant.

Therefore, a procedure that can generate more than one BIBS solution is desired. As we have mentioned, we can prevent an edge from being used as a BILBO edge in the above procedure by setting an infinite weight on this edge. Using this technique, a family of BIBS solutions can easily be generated by an iterative process as presented below. Let G and $\bar{G}$ denote the circuit graph and simplified circuit graph, respectively.

1. Execute Procedure *ckt_simplify* on G to obtain an initial simplified circuit graph $\bar{G}$. During the execution, whenever a tie occurs in an S*-reduction or P*-reduction, arbitrarily make a choice and record the remaining choices as alternatives.

2. Execute Procedure *SPS_balance* on the current $\bar{G}$ to obtain an optimal BIBS solution.

3. If an alternative exists, modify $\bar{G}$ using this alternative and repeat step 2.

As can be observed from the above process, each of the optimal BIBS solutions obtained employs a unique set of registers, thus a family of BIBS solutions can be generated. Note that alternative solutions can exist for the same simplified circuit graph since ties can also occurs in the execution of Procedure *SPS_balance*. Procedure *SPS_balance* can easily be modified so that alternative solutions are generated.

5.3.3 BIBS Design for non-SPSs

A circuit that is not a SPS complicates the process of finding an optimal BIBS solution. Note that Theorems 5.1, 5.2, 5.3, 5.4 and Corollary 5.1 are still valid for a non-SPS, thus a simplified circuit graph can be obtained by using Procedure *ckt_simplify*. However, Theorems 5.5, 5.6 and Corollaries 5.2, 5.3 are not true in general since an edge can be shared by two different UBSs, where neither of them contains the other. For example, the edge (C_2, C_3) in Figure 3.13(b) is shared by the UBS having vertices C_1, C_2, C_3 and edges (C_1, C_2), (C_2, C_3), (C_1, C_3), and the UBS having vertices C_2, C_3, C_4 and edges (C_2, C_3), (C_3, C_4) and (C_2, C_4). Due to the sharing of edges, a partition of a level n UBS, say X, can not only affect those UBSs that contain X, but also UBSs that share edges with X but do not contain

X. Due to this fact, a local optimal partition may not lead to an optimal partition of the entire circuit graph. Therefore it is not sufficient to only consider $S_0(X)$ and $S_1(X)$ when X is under consideration.

Next we present a branch and bound procedure, known as *non_SPS_balance*, that takes a simplified circuit graph as an input and finds an optimal partition of the graph.

Procedure *non_SPS_balance*($\bar{G}$)
/* INPUT: A simplified circuit graph $\bar{G}$. */
/* OUTPUT: The set of BILBO edges associated with an optimal partition of $\bar{G}$. */

1. Let PS(partial solution) = $\emptyset$. Enumerate and levelize UBSs in $\bar{G}$.
2. *balance*(PS, $\bar{G}$).

Procedure *balance*(PS, Y)

1. While ($\exists$ an UBS X in Y whose PI and PO are articulation points), $PS = PS \cup$ *balance*(PS, X).
2. If (Y is a SPS), $PS = PS \cup$ *SPS_balance*(Y) and return(PS).
3. Let $min_cost = \infty$ and $i = 1$.
4. While (level i UBS exists in Y) do
   (a) select a level i UBS, say X.
   (b) select one possible way of augmenting PS (i.e. adding BILBO edges) to balance X.
   (c) if the cost of PS exceeds min_cost, backtrack to step 4(b) and select a different solution for X.
   (d) remove X from Y and repeat step 4.
5. If (no UBSs exist in Y), do
   (a) if the cost of PS is less than min_cost, let min_cost be the cost of PS.
   (b) backtrack to the previous selection. If no other choices exist, return(PS).

   Otherwise $i = i + 1$ and goto step 4.

In Procedure *non_SPS_balance* the UBSs in a simplified circuit graph are first enumerated and levelized using a procedure similar to *enum_UBS*. Then a recursive procedure (*balance*) is executed. In Procedure *balance* each connected component (i.e. sub-graph whose PI and PO are articulation points) in a simplified circuit graph is recursively balanced (step 1). Step 2 of Procedure *balance* checks if Y is a SPS. If so then $\Theta(Y)$ is a subset of Θ (Theorem 5.4). $\Theta(Y)$ can be efficiently obtained using Procedure *SPS_balance* and included in a partial solution PS. Steps 3-5 incrementally balance the UBSs in Y from the lowest level to the highest level if Y is not a SPS. Whenever a new partial solution is generated, the cost of the partial solution is compared with the minimal cost obtained so far. If the former is greater, the current partial solution cannot lead to an optimal partition and the procedure backtracks (step 4(c)). Once Y is balanced (i.e. no UBSs exist), the current minimal cost is updated. Backtracking is also done in step 5(b) to explore the entire solution space. When the exploration is completed, the current optimal solution is returned.

Note that a non-SPS X can become a SPS if certain edges in X are converted to BILBO edges. This is due to the fact that BILBO edges "cut" a circuit graph by logically removing connections between combinational cells. Consider the simplified circuit graph $G1$ shown in Figure 5.15(a), which is not a SPS since the maximal reduction of the graph is not a tree. Consider the UBS $X1$ as shown in the shaded area in Figure 5.15(b). There exists a partial solution that partitions and balances $X1$ by placing two cuts on the composite edge e_5 (see Figure 5.15(b)). The partitioned graph ($G1'$) is a SPS and thus Procedure *SPS_balance* can be applied to $G1'$ to find $\Theta(G1')$ in polynomial time. Therefore, the solution sub-space that contains the above partial solution can be explored in polynomial time. The composite edge e_5 is said to be *critical* to make $G1$ a SPS. In fact, any one of the composite edges e_2, e_5 and e_6 is critical in making $G1$ a SPS since placing two cuts on any of these edges will make $G1$ a SPS.

Theorem 5.7 Given a circuit graph G that contains an UBS X. Let $S(X)$ denote a set of BILBO edges that partitions and balances X and changes G to be a SPS G'. If $S(X) \subset \Theta$ then $\Theta = S(X) \cup \Theta(G')$.

Proof Suppose there exists a set of BILBO edges $S(G') \not\subseteq \Theta(G')$ such that

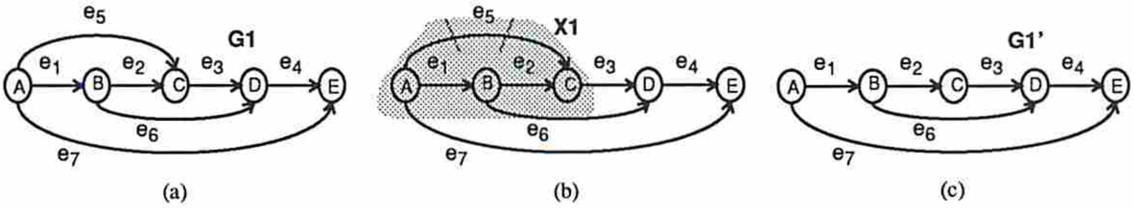


Figure 5.15: (a) a non-SPS $G1$; (b) a partition of $X1$ that modifies a non-SPS ($G1$) to be a SPS ($G1'$) as in (c)

$\Theta = S(X) \cup S(G')$. Clearly $S(G')$ is a set of BILBO edges that balances the subgraph G'. However, the cost of $S(G')$ is greater than that of $\Theta(G')$ since $\Theta(G')$ is associated with the locally optimal partition of G'. Therefore, the cost of $S(X) \cup S(G')$ is greater than that of $S(X) \cup \Theta(G')$. Since the BILBO edges in $S(X) \cup \Theta(G')$ represent a valid partition of G with a cost less than that of $S(X) \cup S(G')$, the assumption that $\Theta = S(X) \cup S(G')$ cannot be true. □

Theorem 5.7 ensures that once a circuit graph is changed to be a SPS by removing the critical edges, using Procedure *SPS_balance* on the remaining graph still preserves the optimality of the solution obtained. Based on this theorem Procedure *non_SPS_balance* can be modified so that once a circuit graph becomes SPS, no further exploration of the remaining solution sub-space is required. Instead, Procedure *SPS_balance* can be used to obtain an optimal partial solution for such a solution sub-space in polynomial time. In addition, the solution obtained by using this more efficient procedure is guaranteed to be optimal. Based on this observation, step 4 of Procedure *balance* is modified as below.

Procedure *balance*(PS, Y)

...

4. If (Y is a SPS), $PS = PS \cup SPS_balance(Y)$ and goto step 5, otherwise if (level i UBS exists in Y) do
   (a) select a level i UBS, say X.
   (b) select one possible way of augmenting PS (i.e. adding BILBO edges) to balance X.
   (c) if the cost of PS exceeds *min_cost*, backtrack to step 4(b) and select a different solution for X.

(d) remove X from Y and repeat step 4.

...

Since a polynomial time procedure instead of an exponential time search process can be employed once a circuit graph becomes a SPS, substantial reduction in the computational complexity may be achieved using this modified procedure. Similar to the problem for a SPS, a family of BIBS solutions for a non-SPS can be generated by an iterative process as described in the previous section.

5.3.4 BIBS Design with Cone Size Constraint

When functionally exhaustive testing or functionally pseudo-exhaustive testing is desired, each kernel must be driven by a single TPG and the test time of a kernel is lower bounded by 2^n, where n is the maximal cone size in the kernel. When n is large, say greater than 22, test time for the kernel may be excessively long. Therefore, the maximal cone size in each kernel must be constrained. In addition, if the paths between a TPG and combinational cells in a kernel have unequal sequential lengths, extra F/Fs are required in the TPG design to provide functionally exhaustive test set as discussed in Chapter 4. These extra F/Fs will increase the cost of the BISTable circuit employing the kernels thus identified.

Again consider the circuit shown in Figure 5.13(a). If the constraint on the maximal cone size is 16, then the partitions $P1$ and $P2$ in Figure 5.13(c) are not valid since the maximal cone sizes of $P1$ and $P2$ are 36 and 18, respectively. A cone size reduction problem similar to the partitioning for pseudo-exhaustive testing problem discussed in [18] is employed to reduce the test application time of a kernel. Following the approach given in [18], given a combinational circuit with a maximal cone size of k and maximal fan-in (of any gate in the circuit) of f, the partitioning problem is to place a minimal number of segmentation cells on interconnections in the circuit so that all the output cones depend on a maximal of r inputs, where $f \leq r < k$. An ILP (integer linear programming) formulation was developed for solving this partitioning problem. For a large circuit, it is not computationally viable to solve the associated ILP problem and thus a heuristic procedure was developed. Our problem is similar to the above problem and the procedure presented in [18] can be extended in the following three aspects to deal

with the kernel cone size constraint problem. First, a balanced sequential circuit instead of a combinational circuit is considered. Secondly, RTL components instead of gate level components are primitive elements and BILBO registers instead of segmentation cells are employed in the partitioning. Thirdly, the circuit obtained after cone size reduction must still be BIBS testable.

Example 5.3 Consider the partitioned circuit graph in Figure 5.13(c) and assume that the cone size constraint is 16. Both $P1$ and $P2$ do not satisfy this constraint and a reduction is required on each of them. Figure 5.16 shows the partitioned graph after cone size reduction. Six partitions (kernels) instead of three partitions are generated. Each partition has a maximal cone size no greater than 16. The cost of this new BIBS solution increases due to the additional BILBO edges on a, b and h. □

Note that the partitioned graph obtained by the extended procedure may not be optimal. This is partly due to the heuristic nature of the cone size reduction procedure, and partly due to the two-step process employed where an initial partitioned graph is obtained in the first step which in turn is further partitioned in the second step to constrain the maximal cone size of each kernel. As mentioned above, the cost of a BISTable circuit may be affected by the extra F/Fs required in the TPG design for kernels in a circuit. Based on our experience in designing TPG (using the `magic` layout tool), a D-type F/F only requires approximately 1/3 the area of a normal LFSR cell. In addition, the number of extra D F/Fs required in TPGs for a circuit is usually limited. Therefore, the extra cost due to D F/Fs in most practical circuits is usually insignificant compared with the cost of LFSR cells in the BILBO registers. However, if under a tight area overhead constraint, the extra cost of D F/Fs should also be considered in generating an optimal BIBS solution.

5.3.5 BIBS Design with Switches and I-paths

As we have discussed above, when switches and I-paths are present in a CUC, test hardware can often be shared among different kernels. For example, consider the data path circuit shown in Figure 5.17(a). By using the I-paths from R_1 to C_1 and

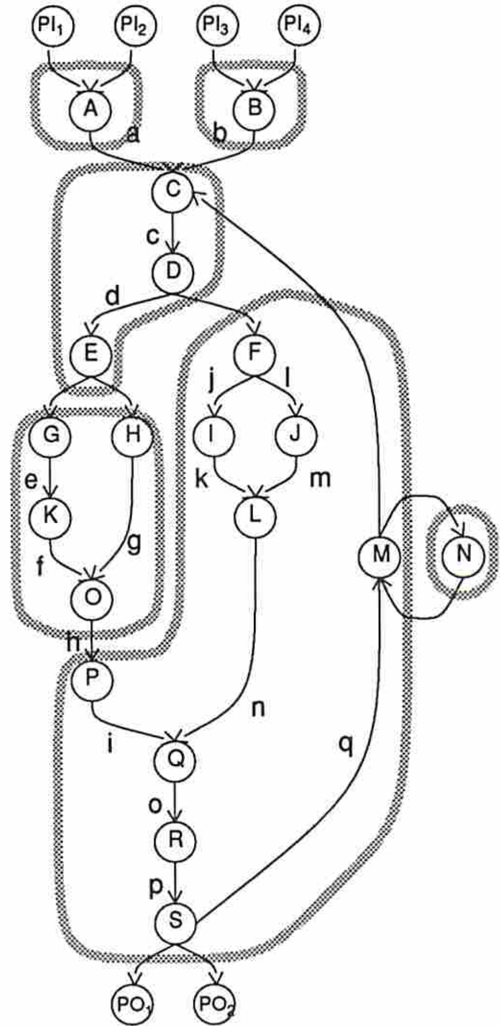


Figure 5.16: Partitioned graph after cone size reduction

from R_1 to C_4, C_1 and C_4 can share the same TPG(R_1). Similarly, C_3, C_5 and C_6 can share the same SA(R_{10}).

The sharing of test hardware usually leads to less area overhead, thus it is desirable to utilize switches and I-paths when possible. To achieve this, the BIBS design procedures presented in the previous sections must be extended to exploit the existence of switches. However, the graph model presented in Chapter 3 does not distinguish between regular logic vertices and vertices that represent switches. This lack of knowledge about switches may lead to a higher area overhead BISTable circuit.

Example 5.4 Again consider the circuit shown in Figure 5.17(a). The circuit graph for this circuit is illustrated in Figure 5.17(b). Clearly this is an unbalanced SPS. Suppose every register in the circuit is 8 bits wide. By using Procedure *SPS_balance*, BILBO edges are placed in the graph as shown in Figure 5.17(c). Two partitions (kernels) are generated with seven BILBO registers. Each of the kernels generated has a maximal cone size of 24, thus the test time to functionally exhaustive test the kernel is 2^{24}. This test time is likely to exceed a designer's constraint. If this is the case, more BILBO registers are needed to reduce the maximal cone size, thus leading to a BISTable circuit with even greater area overhead.

On the other hand, if the vertices BUS, M_1 and M_2 are recognized as switches, only two BILBO registers, namely the PI register (R_1) and PO register (R_{10}), are required to be converted to BILBO registers in a BISTable circuit as illustrated in Figure 5.17(d). In addition, each kernel has a maximal cone size of 8, making the test time much less than the case in Figure 5.17(c). It should be noted, however, that M_1, M_2 and BUS may not be fully tested after applying comprehensive test patterns to kernels in the circuit. Functional patterns may need to be applied to these switches to ensure adequate fault coverage. Although there are more partitions in the design shown in Figure 5.17(d) compared to that in Figure 5.17(c), the test time has been reduced due to the smaller kernel input widths. □

To utilize switches in a BIBS design, we extend the graph model so that two types of vertices exist in a circuit graph. Normal vertices (denoted by circles in Figure 5.17(d)) represent arbitrary combinational cells, fanout cells, vacuous cells, and PI/PO cells. Switch vertices (denoted by squares in Figure 5.17(d))

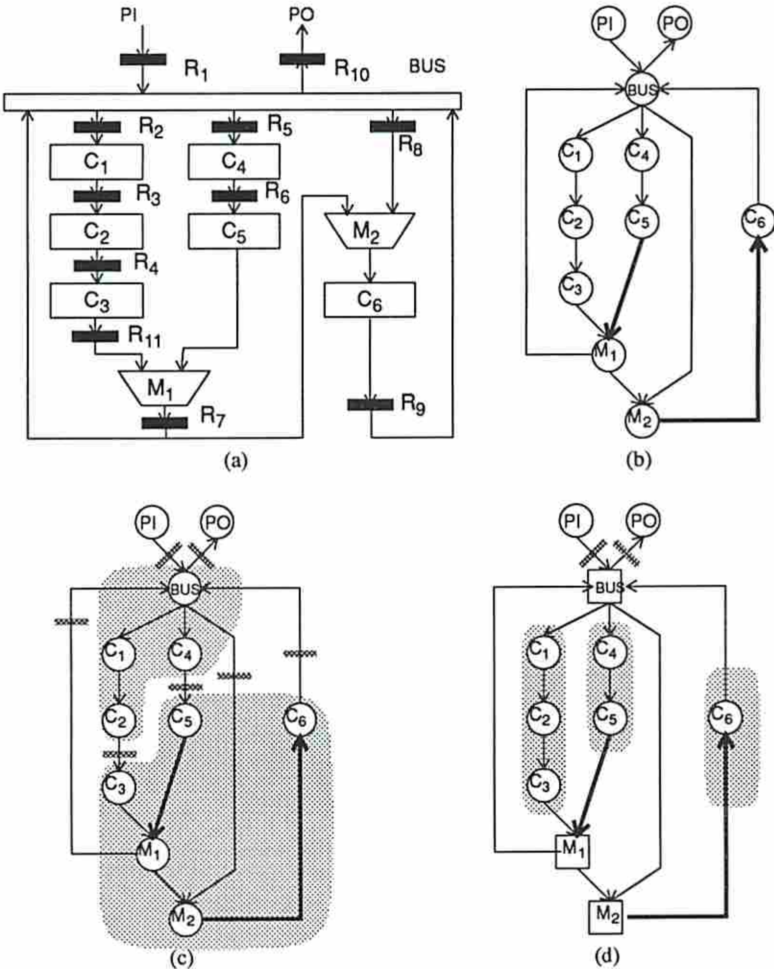


Figure 5.17: (a) An example data path circuit with switches; (b) circuit graph for (a); (c) partitioned graph without using switches; (d) partitioned graph using switches

represent MUXes and busses. Clearly switch vertices should not be included in a partition if its switching capability is to be utilized. When a switch is driven by a register either directly or via an I-path, all the cells that are fed by this switch can share the register as their TPG; similarly when a switch drives a register either directly or via an I-path, all the cells that feed this switch can share the register as their SA. Therefore, a circuit graph is first partitioned into connected components where each component is surrounded by PI/PO vertices and/or switch vertices. Procedures presented in the previous sections can then be applied to each of these components if it is not balanced. Once balanced kernels are identified, driving paths and receiving paths for the ports of these kernels can also be identified by using the procedure presented in Section 5.2.2. These driving and receiving paths are employed in the kernel embedding process to be described in the next chapter.

5.4 Summary

In this chapter we have described the kernel identification process in the BITS system. Combinational kernels are first identified by the CLARION system, and balanced BISTable kernels can then be identified when the BIBS TDM is desired. Switches such as MUXes and busses in a CUC can also be identified. A switch is a cell that has a set of I-modes to transfer patterns from its input ports to output ports unaltered. Switches in a circuit lead to the presence of I-paths that allow for the sharing of test hardware among different kernels. We have presented a procedure to find driving paths and receiving paths in a CUC. Driving and receiving paths are employed in the kernel embedding process described in the next chapter, where embeddings for each kernel are enumerated. When there is no test hardware available for a port of a kernel, the circuit needs to be modified to provide the required test hardware. A procedure that creates I-paths for those kernels was also presented.

We also considered the problem of identifying balanced BISTable kernels when they are allowed. A polynomial time procedure to partition a CUC into balanced BISTable kernels using minimal test hardware has been presented when the CUC is connected in series-parallel. For an arbitrary circuit, a branch and bound procedure

has also been presented to identify balanced BISTable kernels. We also discussed the problem when the maximal cone size of each kernel should be constrained. When switches and I-paths are present in a CUC, the procedures were extended so that these I-paths can be employed during test. Currently switches are excluded from kernels in a circuit. It has been discussed in [35] that in some cases it is advantageous to include certain switches in a kernel for partial scan design. The BIBS TDM may be extended to selectively include switches in BIBS kernels that lead to a superior design. In the BIBS TDM CBILBO registers are only used in self-loops. The constraint may be relaxed when the requirement on the test application time is tight since CBILBO registers can operate as both TPG and SA simultaneously, thus leading to fewer test sessions. These remain open problems and need to be studied further.

Chapter 6

Kernel Embedding Enumeration

6.1 Introduction

Once kernels in a CUC have been identified, each of them needs to be embedded into its associated TDM. When the EXTBILBO, CBILBO or BIBS TDM is employed, the embedding process simply allocates a test register for each input and output port of a kernel. The test registers that are allocated for input ports are configured as TPGs and the test registers that are allocated for output ports are configured as SAs. In addition, a driving (receiving) path must exist between a test register and an input (output) port. When two I-paths contain common components, contentions may occur when the common components are simultaneously used by both I-paths. This is referred to as an I-path conflict. I-path conflicts need to be detected and resolved by the kernel embedding process to avoid contentions in using circuit components. When the COMBILBO TDM is employed, a test register that is configured as a SA for a kernel K_1 may be used to provide test patterns for another kernel K_2. Under this circumstance, kernels K_1 and K_2 are tested simultaneously and a *macro embedding* that contains both K_1 and K_2 is constructed. When test hardware is shared among embeddings for different kernels, the embeddings may not be executed at the same time. On the other hand, embeddings that do not employ common test hardware can be executed simultaneously. Embeddings that can be executed simultaneously are said to be *compatible*, otherwise they are *incompatible*. Compatibility between embeddings needs to be determined to schedule the executions of embeddings.

In this chapter we deal with various aspects of the kernel embedding enumeration process in the BITS system. Conflicts between I-paths are classified into three categories. Each category of conflicts imposes different restrictions on the usage of I-paths. To detect the conflicts between existing driving and receiving paths in a CUC, a set of rules have been developed and will be presented. With the I-path conflict information, a procedure to enumerate the embeddings for each kernel will be presented. Finally we will consider the problem of analyzing the compatibility between embeddings.

6.2 I-path Conflicts

6.2.1 Classification of I-path Conflicts

When two I-paths share components such as registers, MUXes or busses, contentions may occur when the common components are simultaneously used by both I-paths. An I-path conflict imposes certain restriction on the usage of the paths. Conflicts between I-paths can be classified into three categories based on the restrictions they impose as defined below.

Definition 6.1 *A conflict between two I-paths is said to be a* **forbidden conflict(FC)** *if these two paths cannot co-exist in a BISTable circuit.*

Forbidden conflicts are the most restrictive conflicts that disallow I-paths to be employed in the same circuit.

Definition 6.2 *A conflict between two I-paths is said to be a* **hard conflict(HC)** *if these two paths cannot be activated simultaneously.*

Clearly I-paths having HCs cannot be employed in the same embedding for a kernel K.

Definition 6.3 *A conflict between two I-paths is said to be a* **soft conflict(SC)** *if these two paths can be activated simultaneously, but the common components between the paths cannot be used by both I-paths in the same clock cycle.*

The differences between these three conflict categories can be illustrated by the following example.

Example 6.1 Consider the circuit shown in Figure 5.17(a) and the following I-paths.

$$I_1 : (R_1, BUS, R_8, M_2, C_6)$$
$$I_2 : (C_6, R_9, BUS, R_8)$$
$$I_3 : (C_3, R_{11}, M_1, R_7, BUS, R_8)$$
$$I_4 : (C_5, M_1, R_7, BUS, R_2)$$

Note the the I-modes of the components in the above I-paths are omitted for simplicity.

There is a FC between I_1 and I_2 since R_8 is a receiver of I_2 that compresses output responses from C_6, and is also contained in I_1 to transfer test patterns from R_1 to C_6. Clearly I_1 and I_2 must be activated at the same time if they are both employed in a BISTable circuit since they are driving path and receiving path, respectively, for the same kernel. When I_1 and I_2 are activated at the same time, the compressed responses in R_8 are destroyed by the test patterns that are transferred through the I-mode of R_8. Therefore, I_1 and I_2 cannot co-exist in a BISTable circuit.

There is a HC between I_1 and I_3 since R_8 is a receiver of I_3 that compresses output responses from C_3, and is also contained in I_1 to transfer test patterns from R_1 to C_6. Clearly I_1 and I_3 should not be activated at the same time to prevent the compressed responses in R_8 from being destroyed. There is a SC between I_3 and I_4 since they share common components M_1, R_7 and BUS. When I_3 and I_4 are activated at the same time, output responses from C_3 may need to be held in R_{11} for one or more clock cycles to avoid contentions with output responses from C_5. □

6.2.2 Conflict Detection

According to the three categories of I-path conflicts, we have developed a set of rules that can be applied to each pair of driving or receiving paths to detect their conflicts. These rules are summarized below.

Rule 1 Consider two driving paths.

1. If both paths drive the same port, there is a FC.

2. If both paths have the same driver,
   (a) when they drive ports of the same kernel and contain the same set of registers (see Figure 6.1(a)),
      i. if the kernel is tested functionally exhaustively or functionally pseudo-exhaustively, there is no conflict.
      ii. if the kernel is tested by any other test strategy, there is a FC.
   (b) when they drive ports of the same kernel and contain different sets of registers (see Figure 6.1(b)),
      i. if the kernel is tested pseudo-randomly, there is a SC.
      ii. if the kernel is tested using other test strategies, there is a FC.
   (c) when they drive ports of different kernels, there is no conflicts (see Figure 6.1(c)).
3. If one driving path contains the driver of the other driving path,
   (a) when they drive ports of the same kernel, there is a FC (see Figure 6.1(d)).
   (b) when they drive ports of different kernels, there is a HC (see Figure 6.1(e)).
4. If the paths contain common components other than the above,
   (a) when one path contains a normal register (i.e. not a driver) that is not contained in the other path, there is a SC (see Figure 6.1(f)).
   (b) otherwise
      i. if they drive ports of the same kernel, there is a FC (see Figure 6.1(g)).
      ii. if they drive ports of different kernels, there is a HC (see Figure 6.1(h)).

Rule 2 Consider two receiving paths.

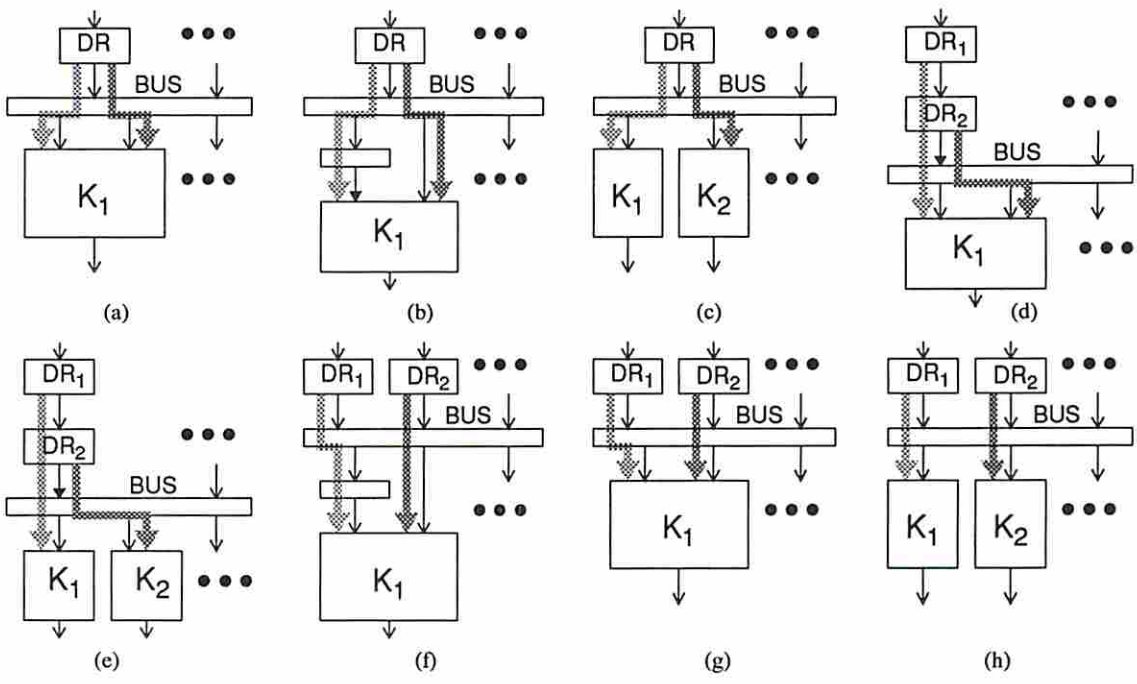


Figure 6.1: (a) Rule 1.2(a); (b) Rule 1.2(b); (c) Rule 1.2(c); (d) Rule 1.3(a); (e) Rule 1.3(b); (f) Rule 1.4(a); (g) Rule 1.4(b).i; (h) Rule 1.4(b).ii

1. If both paths receive from the same port, there is a FC.
2. If both paths have the same receiver,
   (a) when they contain different sets of registers, there is a SC (see Figure 6.2(a)).
   (b) when they contain the same set of registers, there is also a SC (see Figure 6.2(b)).
3. If one receiving path contains the receiver of the other receiving path,
   (a) when they receive from ports of the same kernel, there is a FC (see Figure 6.2(c)).
   (b) when they receive from ports of different kernels, there is a HC (see Figure 6.2(d)).
4. If the paths contain common components other than the above,
   (a) when one path contains a normal register (i.e. not a receiver) that is not contained in the other path, there is a SC (see Figure 6.2(e)).
   (b) otherwise there is also a SC (see Figure 6.2(f)).

Rule 3 Consider a driving path and a receiving path.

1. If the driver of the driving path and the receiver of the receiving path are the same register,
   (a) when the driving path drives and the receiving path receives from ports of the same kernel (see Figure 6.3(a)),
      i. if the BILBO or EXTBILBO TDM is employed, there is a FC.
      ii. if the CBILBO or COMBILBO TDM is employed, there is no conflict.
   (b) when the driving path drives and the receiving path receives from ports of different kernels (see Figure 6.3(b)),

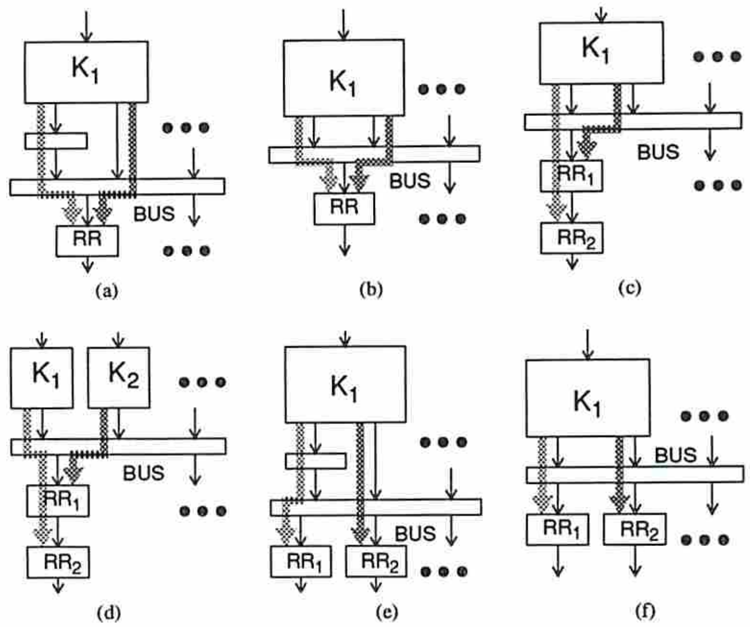


Figure 6.2: (a) Rule 2.2(a); (b) Rule 2.2(b); (c) Rule 2.3(a); (d) Rule 2.3(b); (e) Rule 2.4(a); (f) Rule 2.4(b)

i. if the BILBO or EXTBILBO TDM is employed, there is a HC.

ii. if the CBILBO or COMBILBO TDM is employed, there is no conflict.

2. If the driving path contains the receiver of the receiving path; or the receiving path contains the driver of the driving path,

   (a) when the driving path drives and the receiving path receives from ports of the same kernel, there is a FC (see Figure 6.3(c)).

   (b) when the driving path drives and the receiving path receives from ports of different kernels, there is a HC (see Figure 6.3(d)).

3. If the paths contain common components other than the above,

   (a) when one path contains a normal register (i.e. not a driver or a receiver) that is not contained in the other path, there is a SC (see Figure 6.3(e)).

(b) otherwise

   i. if the driving path drives and the receiving path receives from ports of the same kernel, there is a FC (see Figure 6.3(f)).

   ii. if the driving path drives and the receiving path receives from ports of different kernels, there is a SC (see Figure 6.3(g)).

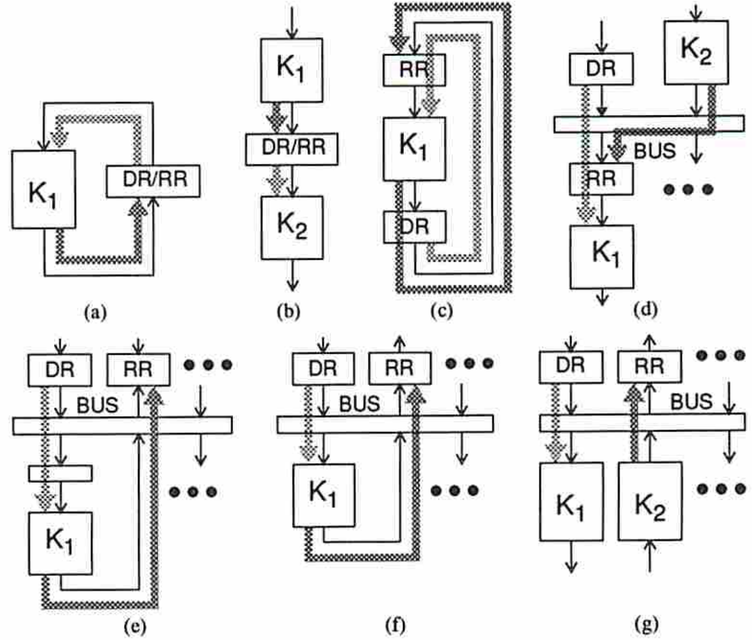


Figure 6.3: (a) Rule 3.1(a); (b) Rule 3.1(b); (c) Rule 3.2(a); (d) Rule 3.2(b); (e) Rule 3.3(a); (f) Rule 3.3(b).i; (g) Rule 3.3(b).ii

These rules are divided into three categories. In the first category a pair of driving paths are considered; in the second category a pair of receiving paths are considered; and in the third category a driving path and a receiving path are considered. Each category of rules is then similarly divided into sub-rules based on the configuration of I-paths. For example, Rules 1.1 and 2.1 deal with I-paths that drive and receive from, respectively, the same port. Rules 1.2, 2.2 and 3.1 deal with various configurations where the driver(s) or receiver(s) are the same component. Rules 1.3, 2.3 and 3.2 deal with various configurations where the driver or receiver

of one I-path is contained in the other I-path. Rules 1.4, 2.4 and 3.3 deal with other test hardware sharing configurations. Note that some of the circuit configurations will not occur in a practical design. For example, the configuration in Figure 6.3(f) forms a combinational loop that is not allowed.

Rules 1.1 and 2.1 are trivial since there is no need to have two I-paths drive or receive from the same port. When the conditions in Rule 1.2(a) are satisfied, the ports that are driven by the two driving paths always receive the same patterns. Unless functionally exhaustive or functionally pseudo-exhaustive testing is employed, there is a FC since only a small portion of all possible patterns can be applied to the kernel and thus the fault coverage of this kernel will be significantly degraded. When functionally exhaustive or functionally pseudo-exhaustive testing is employed, this is not a conflict since all patterns in the normal function can be applied to the kernel. When the conditions in Rule 1.2(b) are satisfied, the types of conflict also depend on the test strategy employed. When pseudo-random testing is employed, test patterns can be held in a register not common to both driving paths, thus the same driver can be shared by the two driving paths. This leads to a SC. Other test strategies require distinct drivers for each input port of a kernel, thus leading to a FC. The conditions in Rule 1.2(c) allow patterns to be broadcasted to different kernels and thus no conflicts occur. Rules 1.3(a) and (b) are due to reasons similar to those in Example 6.1. Rule 1.4(a) leads to a SC since contentions may occur in the common component as can be seen from Figure 6.1(f). The configurations in Rule 1.4(b) lead to contentions that cannot be resolved by adding HOLD mode since there is no available register. Therefore there is a FC if the two paths are driving the same kernel, and a HC if they are driving different kernels.

Note that a SA cannot compress output responses from different sources at the same time. However, if the values in two sequences of output responses arrive at the receiver alternatively, such as the condition in Rule 2.2(a), a combined signature can be obtained in this receiver. This leads to a SC. The condition in Rule 2.2(b) precludes the sharing of the SA at the same time, thus there is a SC. Rules 2.3(a) and (b) are due to reasons similar to those in Example 6.1. Rule 2.4(a) is similar to Rule 1.4(a). The configuration in Rule 2.4(b) also precludes the sharing

of the SA at the same time as in Rule 2.2(b), thus there is a SC. In the BILBO and EXTBILBO TDMs, a register can operate as either a TPG or a SA, but not both, at any given time. On the other hand, a CBILBO register can operate as both TPG and SA simultaneously. In the COMBILBO TDM, the outputs from a register that operates as a SA are employed as test patterns for the kernel fed by this register. Rules 3.1(a) and (b) follow from the above facts. Rules 3.2(a) and (b) have been illustrated in Example 6.1. Rule 3.3(a) is similar to Rule 1.4(a). The configuration in Rule 3.3(b).i forms a combinational loop that is not allowed, thus it is a FC. Rule 3.3(b)ii is similar to Rule 2.4(b) and leads to a SC.

6.3 Embedding Enumeration

To embed a kernel, say K, into a TDM, a driving path that drives K is associated with each input port of K and a receiving path that receives from K is associated with each output port of K. In addition, SCs (if any) that exist between these driving and receiving paths have to be resolvable. Note that not every SC is resolvable. Some SCs can be resolved by holding patterns in a register belonging to a conflicting path for one or more clock cycles. When there is no conflict between the driving paths and receiving paths associated with a kernel, a new test pattern can be applied to the kernel every clock cycle. If SCs exist and are resolved by holding patterns in a register, it is not possible to apply a test pattern every clock cycle. A *latency* for executing an embedding is defined as the number of clock cycles between two consecutive applications of test patterns to the kernel in the embedding. When no SCs exist, a latency of 1 can be employed to execute an embedding, otherwise a latency greater than 1 is required.

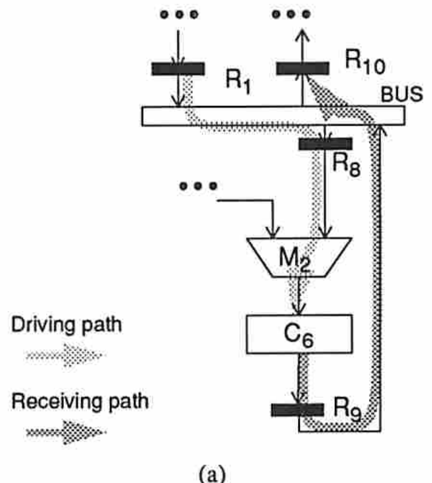
The current BITS system only allows a constant latency to be employed in executing each embedding. This is due to the fact that controlling the execution of an embedding using a variable latency is complicated. The *test length* of an embedding is the number of clock cycles to fully execute the embedding, which is simply the product of the latency for the embedding and the number of test patterns required to test the kernel in the embedding. We ignore the time of shifting in the initial seeds to TPGs and shifting out the final signatures from SAs

of an embedding. To minimize the test length of an embedding, it is desirable to employ a minimal latency for each embedding. The problem of determining the minimal latency for executing an embedding is similar to the pipeline optimization problem[44] and has been discussed in [45, 46]. The procedure presented in [45] is employed in the BITS system to determine the minimal latency.

Example 6.2 Consider the example circuit shown in Figure 5.17(a). Suppose C_6 is a kernel with a sequential depth of D. $D = 0$ when C_6 is a combinational kernel. An embedding for C_6 is constructed using the driving path and receiving path as shown in Figure 6.4(a). Figure 6.4(b) shows an execution of the embedding. Notice that there exists a SC between the driving path and receiving path since both paths contain the component BUS. The usage of the test hardware involved in an execution of the embedding can be summarized in a *reservation table*[44] shown in Figure 6.4(c).

According to the procedure presented in [45], the minimal latency of this embedding is 2. To resolve the SC, test patterns must be applied every other clock cycle. In addition, patterns have to be held in R_9 for one clock cycle to avoid contentions in the BUS when D is even. For this case, an execution of the embedding and its reservation table are shown in Figures 6.4(d) and (e), respectively. Note that to fully test a kernel K, an embedding for K is executed T times, where T is the number of test patterns required by K. □

It should be noted that only SCs are resolvable, hence I-paths with HCs and/or FCs cannot be employed in the same embedding. As can be observed from the above example, registers in the driving and receiving paths associated with an embedding should be able to operate certain functions. For example, R_1 must have the functions TPG and SHIFT (for initialization); R_9 must have the functions LOAD and HOLD (when D is even). If a register does not already possess the required functions, extra hardware is added to modify the register in a BISTable circuit. To specify the required functions for registers in an embedding, a set of register-function pairs, called *r-f pairs*, is also associated with an embedding. A r-f pair is a 2-tuple (R_i, F_i) where R_i is a register in the circuit and F_i is a bit



(a)

Clock cycle	Action
1	R_1(TPG)
2	BUS(R_1), R_8(LOAD)
3	M_2(R_8), C_6
3+D	R_9(LOAD)
4+D	BUS(R_9), R_{10}(SA)

(b)

Component	Clock cycle				
	1	2	3	3+D	4+D
R_1	X				
BUS		X			X
R_8		X			
M_2			X		
R_9				X	
R_{10}					X

(c)

Clock cycle	Action
1	R_1(TPG)
2	BUS(R_1), R_8(LOAD)
3	M_2(R_8), C_6
3+D	R_9(LOAD)
4+D	R_9(HOLD)
5+D	BUS(R_9), R_{10}(SA)

(d)

Component	Clock cycle					
	1	2	3	3+D	4+D	5+D
R_1	X					
BUS		X				X
R_8		X				
M_2			X			
R_9				X	X	
R_{10}						X

(e)

Figure 6.4: (a) An embedding for C_6; (b) an execution of the embedding in (a); (c) reservation table for (b); (d) modified execution of the embedding to resolve SC; (e) reservation table for (d)

vector [T, S, C, L, H, F]. Each bit in F_i has a value of 0 or 1 based on the functions required for R_i as illustrated below.

$$
\begin{cases}
\text{T} = 1 \text{ if } R_i \text{ is required to operate as a TPG, 0 otherwise} \\
\text{S} = 1 \text{ if } R_i \text{ is required to operate as a SA, 0 otherwise} \\
\text{C} = 1 \text{ if } R_i \text{ is required to operate as a CBILBO register, 0 otherwise} \\
\text{L} = 1 \text{ if } R_i \text{ is required to perform a LOAD function, 0 otherwise} \\
\text{H} = 1 \text{ if } R_i \text{ is required to perform a HOLD function, 0 otherwise} \\
\text{F} = 1 \text{ if } R_i \text{ is required to perform a SHIFT function, 0 otherwise}
\end{cases}
$$

Once conflicts in an embedding have been resolved and the required register functions determined, an embedding can simply be represented as a structure that contains a kernel, a driving (receiving) path for each input (output) port of the kernel, a latency, a test length, and a set of r-f pairs. For example, the embedding

e in Example 6.2 can be represented as below, where D is assumed to be even and the number of required test patterns for C_6 is T.

{Embedding: e

Kernel:	C_6	
Driving path:	$(R_1, BUS, R_8, M_2, C_6)$	
Receiving path:	(C_6, R_9, BUS, R_{10})	
Latency:	2	
Test length:	$2T$	
r-f pairs:	$(R_1, (1,0,0,0,0,1))$	
	$(R_8, (0,0,0,1,0,0))$	
	$(R_9, (0,0,0,1,1,0))$	
	$(R_{10}, (0,1,0,0,0,1))$	
Execution:	$\Phi 1$:	$R_1(TPG)$
	$\Phi 2$:	$BUS(R_1), R_8(LOAD)$
	$\Phi 3$:	$M_2(R_8), C_6$
	$\Phi(3+D)$:	$R_9(LOAD)$
	$\Phi(4+D)$:	$R_9(HOLD)$
	$\Phi(5+D)$:	$BUS(R_9), R_{10}(SA)$

}

The Procedure to enumerate embeddings for each kernel, known as KEE, is presented below.

Procedure KEE
/* INPUT: A kernel K, a test length T for K, a set of driving paths for each input port of K, and a set of receiving paths for each output port of K. */
/* OUTPUT: All possible embeddings for K, denoted by E_K. */

1. $E_K = \emptyset$. Let IP_K be the number of input ports of K and OP_K be the number of output ports of K. Let $\mathcal{I}_{d,j}$ and $\mathcal{I}_{r,k}$ be the sets of driving paths and receiving paths for the jth input port and k^{th} output port, respectively, of K.

2. $I^* = \mathcal{I}_{d,1} \times \ldots \times \mathcal{I}_{d,IP_K} \times \mathcal{I}_{r,1} \times \ldots \times \mathcal{I}_{r,OP_K}$.

3. For each I-path combination $\mathcal{I} \in I^*$, if I-paths in $\mathcal{I}$ do not have FCs or HCs, *create_emb*(E_K, K, $\mathcal{I}$, T).

4. Return E_K.

Procedure *create_emb*(E_K, K, $\mathcal{I}$, T)

1. Create an embedding e having kernel K and driving and receiving paths in $\mathcal{I}$.
2. if (*combine_I_paths*(e, $\mathcal{I}$) = `true`) do
   (a) Determine and fill in the r-f pairs of e by examining X.
   (b) $E_K = E_K \cup \{e\}$.

Procedure *combine_I_paths*(e, $\mathcal{I}$)[45]

1. Let X denote the execution of e. Initialize X and let $t = 0$.
2. Let $P_1, \ldots, P_n$ be the driving paths in $\mathcal{I}$, where each P_i $(1 \leq i \leq n)$ is divided into steps so that components in the same step can all be executed in one clock cycle. Note that each step contains exactly one register. Let s_i be the first step in P_i for $1 \leq i \leq n$.
3. Repeat until s_i is empty for all i.
   (a) $t = t + 1$.
   (b) for $i = 1$ to n do
      i. if (s_i can be scheduled in step Φt of X), schedule components in s_i to Φt and let s_i be the next step in P_i.
      ii. Otherwise schedule a HOLD operation for the register in s_i to Φt.
4. Let $P_1, \ldots, P_m$ be the receiving paths in $\mathcal{I}$, where each P_i $(1 \leq i \leq m)$ is also divided into steps. Let s_i be the first step in P_i for $1 \leq i \leq m$.
5. Repeat until s_i is empty for all i.
   (a) for $i = 1$ to m do
      i. if (s_i can be scheduled in step Φt of X), schedule components in s_i to Φt and let s_i be the next step in P_i.
      ii. Otherwise if the register in s_1 operates as a TPG or SA, return `false`; else schedule a HOLD operation for the register to Φt.
   (b) $t = t + 1$.
6. return `true`

Note that Procedure *combine_I_path* is a simplified version of that presented in [45]. Readers should refer to the above article for the relevant theory and details of the procedure. Procedure *KEE* is executed once for every kernel to produce all possible embeddings for the kernel. These embeddings are employed in the design space exploration process to be described in Chapter 8. When the COMBILBO TDM is employed, a register that operates as a SA can also be used as a TPG. Note that although it is used as a TPG, the register is configured as a multiple input signature register(MISR). Due to this reason, unlike a CBILBO register, there is no additional area overhead when a SA is employed as a TPG. A SA register that is also employed as a TPG may be driven by and drive the same kernel as shown in Figure 6.5(a), or be driven by and drive distinct kernels as shown in Figure 6.5(b). In the former case, an embedding for the kernel that both drives and is driven by a SA register can easily be constructed as described above. In the latter case, an embedding for each of K_1 and K_2 as in Figure 6.5(b) can be constructed. However, the execution of these two embeddings are constrained in the sense that they must be executed simultaneously. Due to this restriction in executing these embeddings, the individual embeddings can be merged into a *macro embedding* as shown in Figure 6.5(c). The test length of a macro embedding is the maximum among the test lengths of individual embeddings. Note that the paths from TPGs to kernels, and from kernels to SAs must be I-paths.

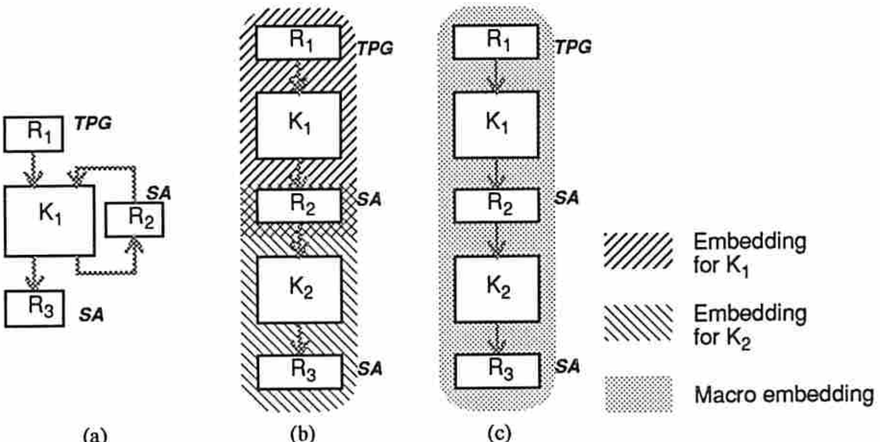


Figure 6.5: Examples of employing COMBILBO TDM

For a complex circuit, there can be more than one way to merge embeddings into macro embeddings. It remains an open problem to generate the appropriate macro embeddings when the COMBILBO TDM is employed.

6.4 Embedding Compatibility Analysis

When the I-paths between two embeddings do not conflict, these two embeddings can be executed simultaneously. Executing embeddings simultaneous leads to a shorter test application time than that of executing embeddings in sequence. In this section we will first classify the compatibility between embeddings into three categories based on their restrictions on executing the embeddings. The process of compatibility analysis is then presented.

6.4.1 Classification of Compatibility

Given a pair of embeddings e_1 and e_2. Let t_1 and t_2 be the test lengths to execute e_1 and e_2, respectively.

Definition 6.4 *e_1 and e_2 are said to be* **fully compatible** *if I-paths associated with e_1 do not conflict with I-paths associated with e_2.*

If e_1 and e_2 are fully compatible, the test application time to execute e_1 and e_2 simultaneously is simply $\max(t_1, t_2)$.

Definition 6.5 *e_1 and e_2 are said to be* **partially compatible** *if there only exist SCs between I-paths associated with e_1 and I-paths associated with e_2. In addition, the test application time to execute e_1 and e_2 simultaneously must be less than $t_1 + t_2$.*

If e_1 and e_2 contain the same kernel, or they are neither fully compatible nor partially compatible, then e_1 and e_2 are said to be **incompatible**.

6.4.2 Compatibility Analysis

It is straightforward to determine whether two embeddings are incompatible or fully compatible by examining the I-paths associated with the embeddings. When

SCs exist between I-paths associated with two embeddings e_1 and e_2, e_1 and e_2 are partially compatible if the test application time to execute e_1 and e_2 simultaneously is less than that of executing e_1 and e_2 in sequence. This can be determined by analyzing a concurrent execution of e_1 and e_2. This process is illustrated below.

Example 6.3 Consider an example circuit shown in Figure 6.6, where K_1 and K_2 are assumed to be combinational kernels. Suppose the number of test patterns required by K_1 and K_2 are $2T$ and T, respectively, where T is a reasonably large integer.

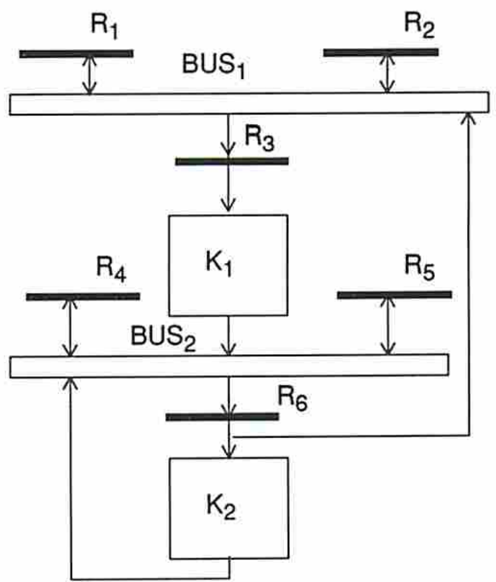
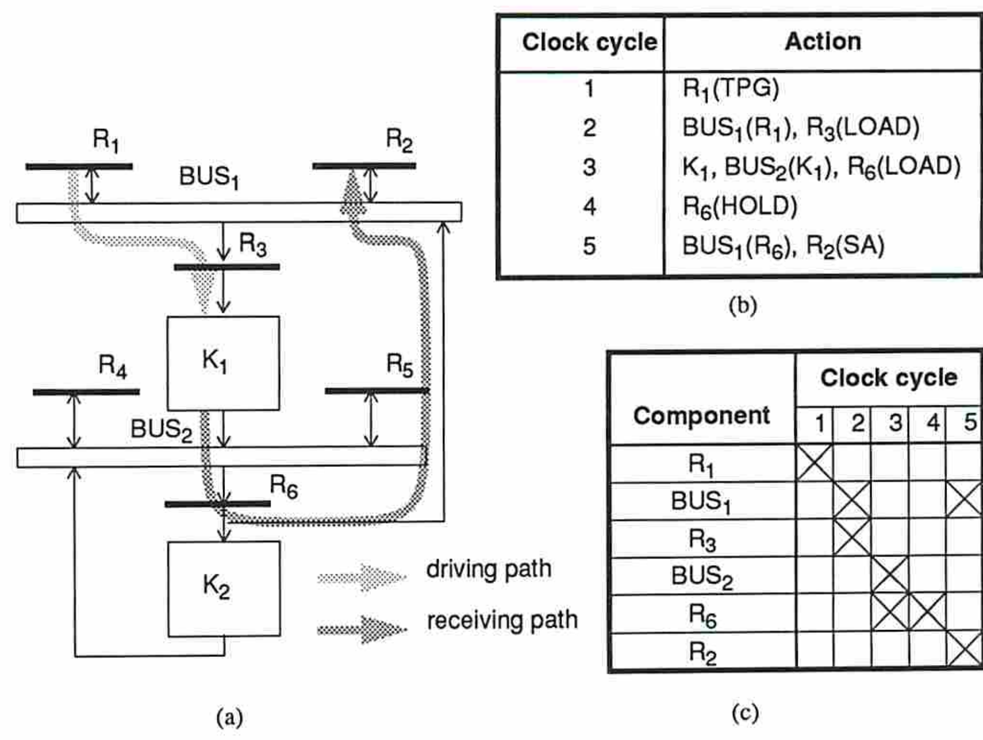


Figure 6.6: An example circuit

Consider an embedding, denoted by e_1, for K_1 as shown in Figure 6.7(a). A minimal latency of 2 to execute e_1 can be obtained by a process similar to that described in Example 6.2. An execution of e_1 and its reservation table using this minimal latency are illustrated in Figures 6.7(b) and (c), respectively. Clearly the test length of e_1 is $4T$. Next consider an embedding, denoted by e_2, for K_2 as shown in Figure 6.8(a). An execution of e_2 and its reservation table are illustrated in Figures 6.8(b) and (c), respectively, where a minimal latency of 2 is employed in executing e_2. This leads to a test length of $2T$ for e_2. The test application time to execute embeddings e_1 and e_2 in sequence is $6T$.



Clock cycle	Action
1	R_1(TPG)
2	$BUS_1(R_1)$, R_3(LOAD)
3	K_1, $BUS_2(K_1)$, R_6(LOAD)
4	R_6(HOLD)
5	$BUS_1(R_6)$, R_2(SA)

(b)

Component	Clock cycle				
	1	2	3	4	5
R_1	X				
BUS_1		X			X
R_3		X			
BUS_2			X		
R_6			X	X	
R_2					X

(c)

Figure 6.7: (a) an embedding for K_1; (b) an execution of embedding in (a); (c) reservation table for (b)

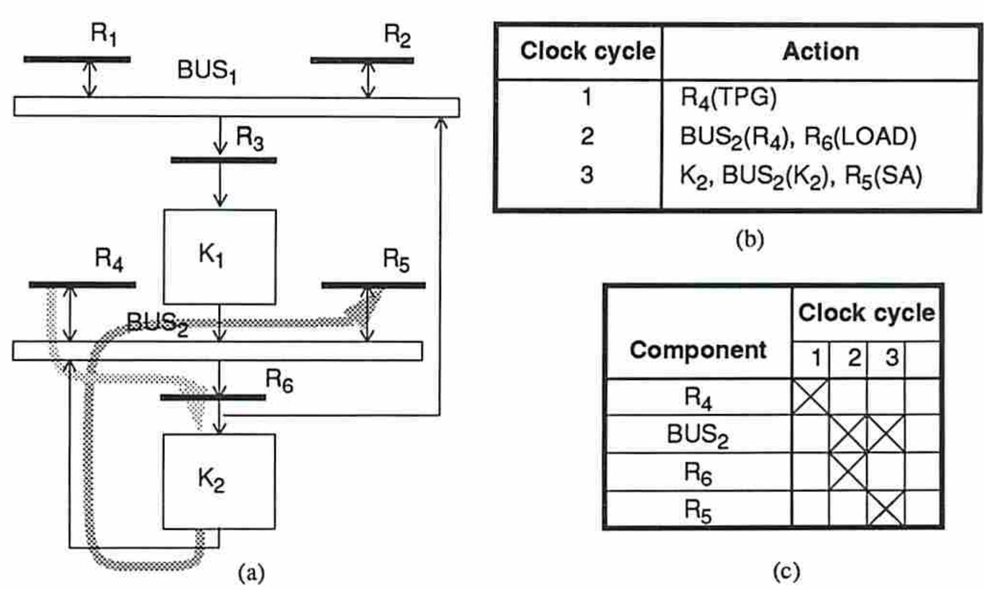


Figure 6.8: (a) an embedding for K_2; (b) an execution of embedding in (a); (c) reservation table for (b)

Clearly SCs exist between the receiving path of e_1 and the driving and receiving paths of e_2 due to the common component BUS_2. Therefore, e_1 and e_2 are not fully compatible. Consider a concurrent execution of e_1 and e_2 as shown in Figure 6.9(a). The reservation table for this execution is illustrated in Figure 6.9(b), where BUS_2 is used three times in each execution. A minimal latency of 3 can be obtained for this concurrent execution. Notice that after T concurrent executions of e_1 and e_2, kernel K_2 is fully tested. Therefore the remaining T test patterns for K_1 can be applied every other clock cycle. This results in a total test application time of $3T + 2T = 5T$. This is less than that of executing e_1 and e_2 in sequence, hence they are partially compatible. □

As can be observed, determining whether two embeddings are partially compatible is a complicated process. In addition, the test controller for a BISTable circuit having partially compatible embeddings is more complicated since a variable latency may be required in testing a kernel. As we mentioned before, the current BITS system does not support a variable latency. Therefore, the system only

Clock cycle	Action
1	R_1(TPG)
2	$BUS_1(R_1)$, R_3(LOAD)
3	K_1, $BUS_2(K_1)$, R_6(LOAD), R_4(TPG)
4	$BUS_1(R_6)$, R_2(SA), $BUS_2(R_4)$, R_6(LOAD)
5	K_2, $BUS_2(K_2)$, R_5(SA)

(a)

Component	Clock cycle				
	1	2	3	4	5
R_1	X				
BUS_1		X		X	
R_3		X			
BUS_2			X	X	X
R_6			X	X	
R_4			X		
R_2				X	
R_5					X

(b)

Figure 6.9: (a) a concurrent execution of embeddings for K_1 and K_2; (b) reservation table for (a)

exploits the concurrency between fully compatible embeddings but not partially compatible embeddings.

6.5 Summary

In this chapter we have presented the kernel embedding enumeration process. The I-path concept has been employed in constructing embeddings. To avoid contentions in components shared by different I-paths, I-path conflicts are detected and resolved (if possible) when conflicting I-paths are employed an embedding. I-path conflicts are classified into three categories based on the restrictions of their usage in a BISTable circuit. A set of I-path conflict rules has been presented. Each rule can be applied to a pair of driving and/or receiving paths to detect the existence of conflicts. The representation for an embedding has been described. For each embedding a minimal latency for its execution can be determined by a procedure similar to the pipeline optimization procedure. The kernel embedding enumeration procedure has been presented. Embeddings produced by this procedure are employed by the design space exploration process to generate a family of BISTable circuits. When conflicts exist between I-paths associated with two embeddings, the embeddings cannot be executed simultaneously. Compatibility

between embeddings has been defined and the process of compatibility analysis has been presented. Compatibility information is required in scheduling the executions of embeddings that will be described in the next chapter.

Chapter 7

Test Scheduling

7.1 Introduction

Once every kernel has been embedded into a TDM, a test schedule that specifies the order of executing the embeddings and determines the test application time of a BISTable circuit is required. The test scheduling problem has been shown to be NP-hard[21]. It is a more general problem than the scheduling problem encountered in high-level synthesis[47] where, in addition to a compatibility relationship, a precedence relationship is also considered.

In this chapter we first define the test scheduling problem. A BISTable circuit can be scheduled using different test scheduling strategies. The test scheduling strategies are discussed and the strategy employed in the BITS system is described. The differences between incremental test scheduling and non-incremental test scheduling are discussed. The advantages of using an incremental test scheduler in the BITS system is also described. The incremental test scheduling procedure employed in the BITS system is then presented. A theoretical lower bound for the test time of a BISTable circuit is developed. This lower bound can easily be calculated by a procedure presented, and is an indicator of the quality of our test scheduler. A series of experiments is performed using the test scheduler to evaluate the performance of out test scheduler. Comparisons are made with the technique presented in [25] that is superior to other existing approaches.

7.2 Test Scheduling Problem

7.2.1 Problem Statement

A BISTable circuit often contains a set of kernels. Each kernel, say K_i, is embedded into a TDM, and the process of executing an embedding for K_i so that K_i is fully tested is referred to as a *test*, t_i. Associated with a test are a test length to fully test the embedded kernel and a set of test hardware that is employed in the embedding. Two tests are said to be *compatible* if they can run concurrently; otherwise they are *incompatible*. Compatibility is a reflective and symmetric relation, but is not transitive. Given a test for each kernel in a BISTable circuit, a *test compatibility graph(TCG)* can be constructed in which a vertex appears for each test and an edge exists between a pair of vertices representing two tests t_i and t_j if t_i and t_j are compatible. Consider the circuit shown in Figure 7.1. The embeddings for the kernels are shown in the figure by thick arrows in different shades. Let t_i be the test for kernel K_i by using the embedding as shown in Figure 7.1, where $1 \leq i \leq 6$. The TCG for the tests in Figure 7.1 is shown in Figure 7.2.

The test scheduling problem is to find a schedule to run all tests of a BISTable circuit once and the total test application time is minimal. When the test lengths of tests in a BISTable circuit are all the same, the test scheduling problem can be formulated as a *clique covering* problem[9, 21]. The TCG for tests in a circuit is first constructed. Cliques of the TCG can then be enumerated. A cover of vertices in the TCG using a minimal number of cliques determines a minimal time test schedule. The minimal covering problem is NP-complete[23], thus it is not always computationally feasible to employ the clique covering formulation in finding a minimal time test schedule. In addition, test lengths of tests in a BISTable circuit are usually not all the same. This further complicates the test scheduling problem. The unequal test length scheduling problem is considered in this dissertation.

7.2.2 Classification of Test Scheduling Strategies

In unequal test length scheduling problem, let T_i denote the test length of t_i. Consider two compatible tests t_1 and t_2 where $T_1 > T_2$. If t_1 and t_2 are initiated at

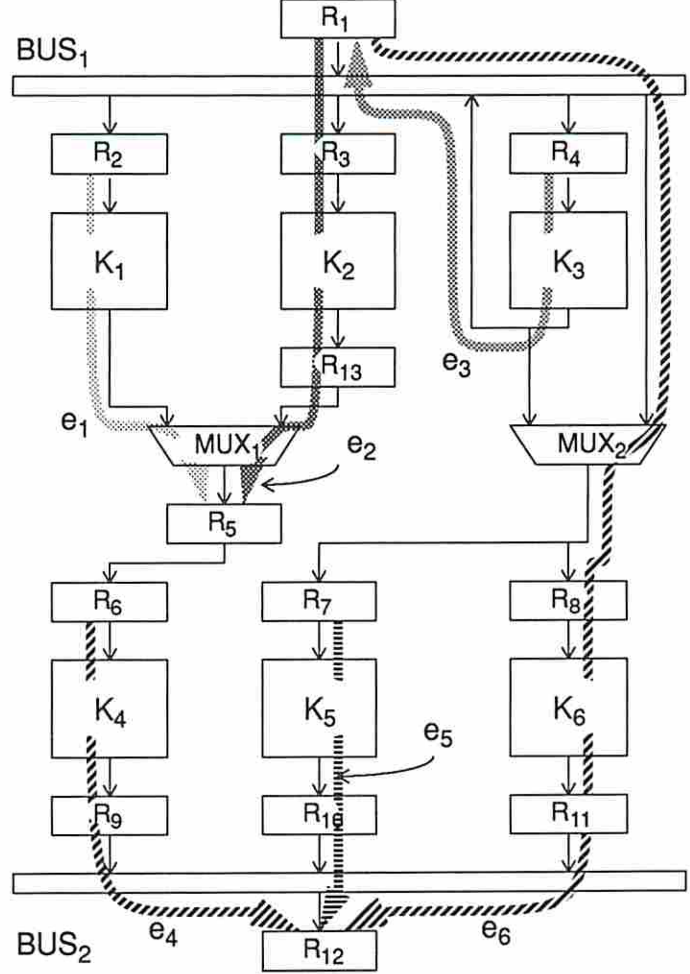


Figure 7.1: An example circuit

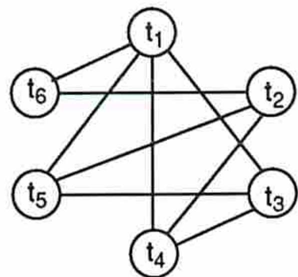


Figure 7.2: A TCG for the tests in Figure 7.1

the same time, t_2 will finish before t_1. There are three test scheduling strategies to deal with this situation. (1) If a test controller cannot shift out a signature of a test when other tests have not completed, the SA associated with the embedding in t_2 holds the signature in the SA for $(T_1 - T_2)$ clock cycles. The final signatures of t_1 and t_2 are then shifted out for comparison after t_1 finishes. This test scheduling strategy is called *non-partitioned testing*[21]. In practice, the test time of t_2 can be extended to be T_1 by applying $(T_1 - T_2)$ arbitrary patterns after T_2 clock cycles to simplify the test controller since by doing this there will be no need to hold the signature of t_2. (2) If a test controller can shift out a signature of a test independent of other tests, the signature of t_2 can be shifted out after T_2 clock cycles and another test that is compatible with t_1 can be initiated at the same time. This test scheduling strategy is called *partitioned testing with run to completion*[21]. (3) If the states of TPGs and SAs can be stored and restored at any time, t_1 can be interrupted upon the completion of t_2 and the states in the TPG and SA associated with the embedding in t_1 stored. The signature of t_2 is shifted out for comparison and a set of tests that may or may not contain t_1 can be initiated at the same time. When an interrupted test is resumed, the states of the TPG and SA associated with the embeddings in this test are restored before the test is resumed. This test scheduling strategy is called *partitioned testing*[21].

In general, partitioned testing results in the shortest test application time and non-partitioned testing results in the longest test application time as discussed in [21]. However, the test controller design for partitioned testing is much more complicated than the design for the other two strategies. The test controller synthesizer in the BITS system allows for the partitioned testing with run to completion. Therefore, the test scheduler to be presented is based on this test scheduling strategy.

7.2.3 Incremental Test Scheduling vs. Non-incremental Test Scheduling

A test scheduler is said to be *incremental* if it can generate a partial test schedule given a partially specified circuit. In addition, as more test hardware is allocated to make the circuit fully specified (and thus BISTable), the test scheduler incrementally produces a complete test schedule. Most of the existing test schedulers[21, 25, 22, 26, 24] are non-incremental since a fully specified circuit is required.

Incremental test scheduling is important in the BITS system since it may greatly reduce the search space of the design space exploration process. The search space of the design space exploration process can be depicted as a tree as shown in Figure 7.3. The root node represents the original (non-BISTable) circuit and the leaf nodes represent feasible BISTable circuits. Each feasible BISTable circuit is simply a combination of embeddings so that every kernel in the circuit is embedded. The depth of the tree is the number of kernels in the circuit. The branches from the root node represent possible embeddings for the first kernel considered. Similarly, the branches from a node at level i represent possible embeddings for the ith kernel considered. If the number of embeddings for the ith kernel considered is E_i , then the total number of feasible BISTable circuits is $D = \Pi_{i=1}^{n} E_i$, where n is the number of kernels in the circuit. If a non-incremental test scheduler is employed in the design space exploration process, a test schedule cannot be produced until a fully specified circuit (i.e. a feasible BISTable circuit) is generated. Therefore, the test scheduler has to be executed once for each of the D circuits to fully explore the design space.

Suppose each solid node in Figure 7.3 represents a partially specified circuit that cannot lead to a representative and acceptable BISTable circuit. Then all the nodes descending from it as shown in the shaded area can be pruned. A BISTable circuit is *representative* if there does not exist other BISTable circuit that is superior to it in terms of every BIST parameter. A BISTable circuit is *acceptable* if its value of every BIST parameter satisfies the design constraint. This pruning process allows us to explore a smaller subspace without missing any representative BISTable circuit.

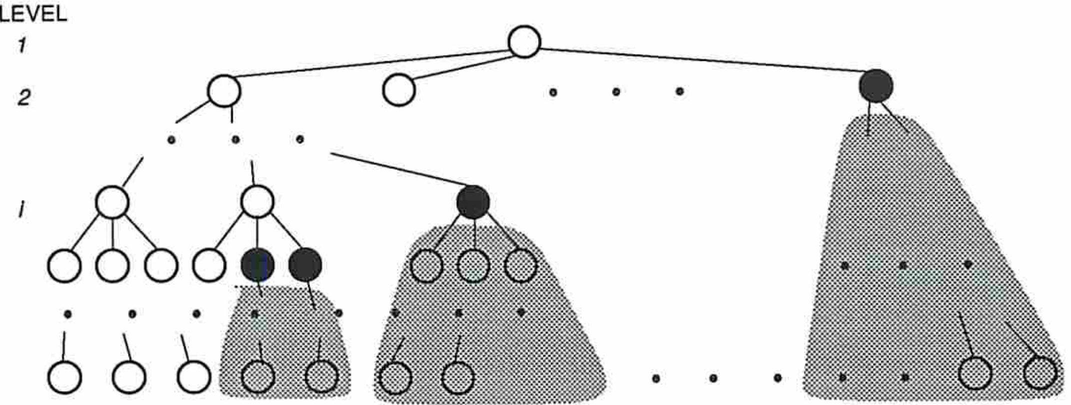


Figure 7.3: An example search tree

7.3 Test Scheduling Procedure

In this section we will first describe the graph model employed in the test scheduler. A polynomial time procedure to generate a test schedule for a circuit either fully specified or partially specified is then presented. This test scheduler calculates a lower bound for the test time of a partially specified circuit as the scheduling process proceeds. This bound is used to reject those partially specified circuits that cannot lead to a representative and acceptable BISTable circuit. A theoretical lower bound for the test time of a fully specified circuit is developed and a procedure to calculate this bound is presented.

7.3.1 Graph Model for a Test Schedule

As mentioned in the previous sections, compatible tests can run concurrently. This results in a shorter test application time than that of running the tests in sequence. The concurrency between tests in a test schedule is represented by means of a set of *test concurrency trees(TCTs)*. Each TCT is a levelized tree (not necessarily binary) where the root node has the greatest level. A set of TCTs that represents a test schedule is called a *test concurrency forest(TCF)*. Tests are uniquely assigned to nodes in TCTs of a TCF. The test length of a test t_i, is denoted by T_i. A node at level l can only contain tests with test lengths between 2^l and $2^{l+1} - 1$. A test

assigned to a node v must be compatible with every test assigned to the nodes descending from v, as well as other (if any) tests that are also assigned to v. An example TCT is shown in Figure 7.4(a). In this TCT test t_1 assigned to node n_1 is compatible with tests t_2, t_3, t_4, t_5 and t_6. Test t_6 assigned to node n_5 is compatible with tests t_1, t_2 and t_4. It may or may not be compatible with tests t_3 and t_5.

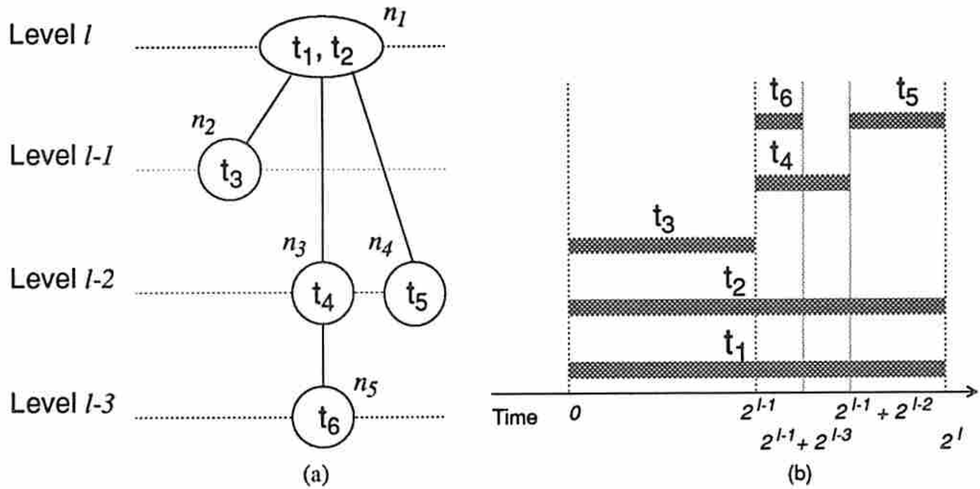


Figure 7.4: (a) an example TCT; (b) its corresponding test schedule

Given a TCF, there is a unique test schedule that can be generated by traversing every TCT in the TCF. A procedure, known as *TCF2TS*, that constructs a unique test schedule from a given TCF is presented below.

Procedure *TCF2TS*
/* INPUT: A TCF, F, consists of a set of TCTs. */
/* OUTPUT: A test schedule corresponds to F. */

1. $start_time = 0$. /* *start_time* is a global variable */

2. For each TCT in F, let v be the root node of the TCT. *traverse(v)*.

Procedure *traverse(v)*

1. For each test t_i assigned to v, schedule t_i from *start_time* for T_i clock cycles.

2. $end_time = start_time + \max_i T_i$. /* *end_time* is a local variable */

3. For each child node, say u, of v from left to right, $traverse(u)$.

4. $start_time = end_time$.

Example 7.1 Consider a TCF F that consists of only one TCT as shown in Figure 7.4(a). Assume the test lengths of the tests in this TCT are: $T_1 = T_2 = 2^l$, $T_3 = 2^{l-1}$, $T_4 = T_5 = 2^{l-2}$ and $T_6 = 2^{l-3}$. Given this TCF, Procedure *TCF2TS* produces the test schedule shown in Figure 7.4(b). Tests t_1, t_2 and t_3 are initiated at time 0. After 2^{l-1} clock cycles t_3 is finished and tests t_4 and t_6 are initiated, while t_1 and t_2 continue without interruption. t_6 and t_4 finish at time $2^{l-1} + 2^{l-3}$ and $2^{l-1} + 2^{l-2}$, respectively, and t_5 is initiated at time $2^{l-1} + 2^{l-2}$. Finally at time 2^l, tests t_1, t_2 and t_5 are complete. The total test application time for these tests is thus 2^l clock cycles, which is much better than $3^l + 2^{l-3}$, the test application time when tests are initiated in sequence. □

As was mentioned earlier that if t_1 and t_2 run concurrently and t_2 finishes before t_1, t_2 can be extended by applying arbitrary patterns to simplify the test controller. Consider the test schedule shown in Figure 7.4(b). Note that t_6 runs concurrently with t_1, t_2 and t_4, and it finishes earlier than the other tests. Since there are no other tests being initiated from time $(2^{l-1} + 2^{l-3})$ to $(2^{l-1} + 2^{l-2})$, t_6 can be extended so that it finishes at time $2^{l-1} + 2^{l-2}$ without affecting the total test time. By doing this the test controller can be simplified since the signatures of t_4 and t_6 can be shifted out simultaneously instead of separately. In addition, the fault coverage of K_6 may be increased since more test patterns are applied.

7.3.2 Scheduling Procedure

As was seen in the previous section, a unique test schedule can be generated once the concurrency of tests in a BISTable circuit is represented by a TCF. Therefore, the test scheduling problem reduces to the construction of a TCF that requires the minimal total test application time for a set of tests. A TCF is constructed incrementally where tests are processed sequentially based on their test lengths. Each node, say v, in a TCT is associated with a set of tests $TS(v)$, a set of child-nodes $C(v)$, a test length $TL(v) = \max_{t_i \in TS(v)} T_i$, and a slack time $ST(v) =$

$TL(v) - \sum_{u \in C(v)} TL(u)$. The level of v is denoted by $\ell(v)$. $TL(v)$ indicates the test time required to complete every test assigned to v or a node descending from v. $ST(v)$ is the time not yet occupied by child-nodes of v, and intuitively indicates the capacity of v to accommodate other child-nodes. For example, consider the TCT in Figure 7.4(a). Node n_1 is associated with $TS(n_1) = \{t_1, t_2\}$, $C(n_1) = \{n_2, n_3, n_4\}$, $TL(n_1) = 2^l$ and $ST(n_1) = 2^l - 2^{l-1} - 2^{l-2} - 2^{l-2} = 0$. A node v can *accommodate* a test t_i if t_i is compatible with every test assigned to v, and t_i is compatible with tests in v's parent nodes. Given a test t_i to be processed, a score function is used to evaluate every node in the current TCF that can accommodate t_i. The function is defined as

$$SF(v, t_i) = \begin{cases} \max(ST(v), T_i) & \text{if } \ell(v) = l \\ ST(v) - T_i & \text{if } \ell(v) > l \end{cases}$$

where $2^l \leq T_i < 2^{l+1}$.

$SF(v, t_i)$ is a measure of the expected slack time associated with v if t_i is accommodated in v. t_i can be accommodated in v by assigning t_i to v when $\ell(v) = l$, or by assigning t_i to a child node of v when $\ell(v) > l$. It should be noted that in the former case, the test length e of v, $TL(v)$, is equal to $ST(v)$ since there are no child nodes associated with v due to the order tests are processed. Once a score is calculated for every node in the current TCF that can accommodate t_i, the actual node that is used to accommodate t_i is determined by maximizing the score function. After such a node, denoted by u, has been selected, t_i is appended to $TS(u)$ and both $TL(u)$ and $ST(u)$ are set to be $\max(TL(u), T_i)$ when $\ell(u) = l$. When $\ell(u) > l$ a new node v' at level l is created and $TS(v')$, $C(v')$, $TL(v')$ and $ST(v')$ are set to be $\{t_i\}$, $\emptyset$, T_i and T_i, respectively. In addition, node v' is appended to $C(u)$ and $ST(u)$ (but not $TL(u)$) is set to be $ST(u) - T_i$. If there does not exist a node in the current TCF that can accommodate t_i, then a new TCT is created with a root node v', where $TS(v')$, $C(v')$, $TL(v')$ and $ST(v')$ are set to be $\{t_i\}$, $\emptyset$, T_i and T_i, respectively, and $\ell(v') = l$. The procedure that constructs a TCF from a set of tests and their TCG, known as *genTCF*, is presented next.

Procedure *genTCF*
/* INPUT: A list of tests, T, sorted in descending order of test length. */
/* OUTPUT: A TCF for T. */

1. $F = \emptyset$. Repeat step 2 until T becomes empty. Return F. /* F denotes a TCF */

2. Let t_i be the first test in T, *build_TCT*(t_i), remove t_i from T.

Procedure *build_TCT(t_i)*

1. Let l be an integer so that $2^l \leq T_i < 2^{l+1}$. $max_time = -1$, $max_emb = 0$, $\#_of_emb = 0$ and $node = \texttt{nil}$.

2. For each root node $u \in F$, $node$ = *search_node(u, $node$, #_of_emb)*.

3. If ($node = \texttt{nil}$) then create a new root node u' at level l with $TS(u') = \{t_i\}$, $C(u') = \emptyset$ and $TL(u') = ST(u') = T_i$.

4. Otherwise do

   (a) if ($\ell(node) = l$) then $TS(node) = TS(node) \bigcup \{t_i\}$ and $TL(node) = ST(node) = \max(TL(node), T_i)$.

   (b) otherwise create a new node v at level l with $TS(v) = \{t_i\}$, $C(v) = \emptyset$, $TL(v) = ST(v) = T_i$. Set $C(node) = C(node) \bigcup \{v\}$ and $ST(node) = ST(node) - T_i$.

Procedure *search_node(v, node, #_of_emb)*

1. If t_i is compatible with every test in $TS(v)$ then goto step 2, else return *node*.

2. $\#_of_emb = \#_of_emb + |TS(v)|$.

   (a) if ($SF(v, t_i) > max_time$) then $node = v$, $max_time = SF(v, t_i)$ and $max_emb = \#_of_emb$.

   (b) otherwise if (($SF(v, t_i) = max_time$) **and** ($\#_of_emb > max_emb$)) then $node = v$ and $max_emb = \#_of_emb$.

3. For every node v' in $C(v)$, $node$ = *search_node(v', node, #_of_emb)*, return *node*.

Notice that $|TS(v)|$ represents the number of tests assigned to node v. Procedure *build_TCT* is executed once for every test, thus once for every kernel. For each test, Procedure *search_node* searches for the most appropriate node in the current TCF to accommodate the test under consideration based on the score function in a depth-first fashion. In the case when two nodes have the same score, the number of

tests contained in the nodes and their parent nodes are considered as the selection criterion to break the tie. When there does not exist a node that can accommodate the test under consideration, Procedure *search_node* returns a value of `nil` and a new node is created. Procedure *genTCF* is a heuristic procedure since in each iteration of *build_TCT*, the node selected to accommodate a test is obtained using the heuristic score function SF.

Example 7.2 Consider the circuit and the embeddings for the kernels shown in Figure 7.1. Suppose the test lengths of the tests are $T_1 = 255$, $T_2 = T_3 = 63$, $T_4 = 31$, $T_5 = 127$ and $T_6 = 15$. Tests are considered in the order t_1, t_5, t_2, t_3, t_4 and t_6. The process of constructing a TCF for these tests using Procedure *genTCF* is shown in Figure 7.5, where the value of ST next to each node in the figure denotes the slack time associated with the node. In step (a) a root node n_1 containing t_1 is created. In step (b) a child node of n_1 containing t_5 is created, and the slack time associated with n_1 is changed to 128. A new root node n_3 containing t_2 is created in step (c) since t_2 is not compatible with t_1. In step (d) t_3, which is compatible with both t_1 and t_5 but not with t_2 is considered. Therefore only n_1 and n_2 can accommodate t_3. $SF(n_1, t_3) = 128 - 63 = 65$ and $SF(n_2, t_3) = 127 - 63 = 64$, Therefore n_1 is selected to accommodate t_3. Since $2^5 \leq T_3 = 63 < 2^6$, a new node n_4 at level 5 that is a child node of n_1 is created and t_3 is assigned to n_4. The slack time of n_1 is changed to 65 accordingly. Similarly n_5 and n_6 are created and associated with n_1 and n_3, respectively, as their child nodes in steps (e) and (f). Once the final TCF as shown in Figure 7.5(f) is constructed, the corresponding test schedule can be generated by Procedure *TCF2TS* and is shown in Figure 7.6(a). The test application time of this BISTable circuit is 318 clock cycles, which is the same as in [25] and better than that obtained in [22], which is 349 clock cycles. □

Note that the test schedule shown in Figure 7.6(a) contains two *test sessions*, which follows from the fact that there are two TCTs in the TCF shown in Figure 7.5(f). The first test session tests kernel K_1, K_3, K_4 and K_5; the second test session tests K_2 and K_6. Again, some tests can be extended by applying arbitrary patterns to simplify the test controller under the constraint that the test time

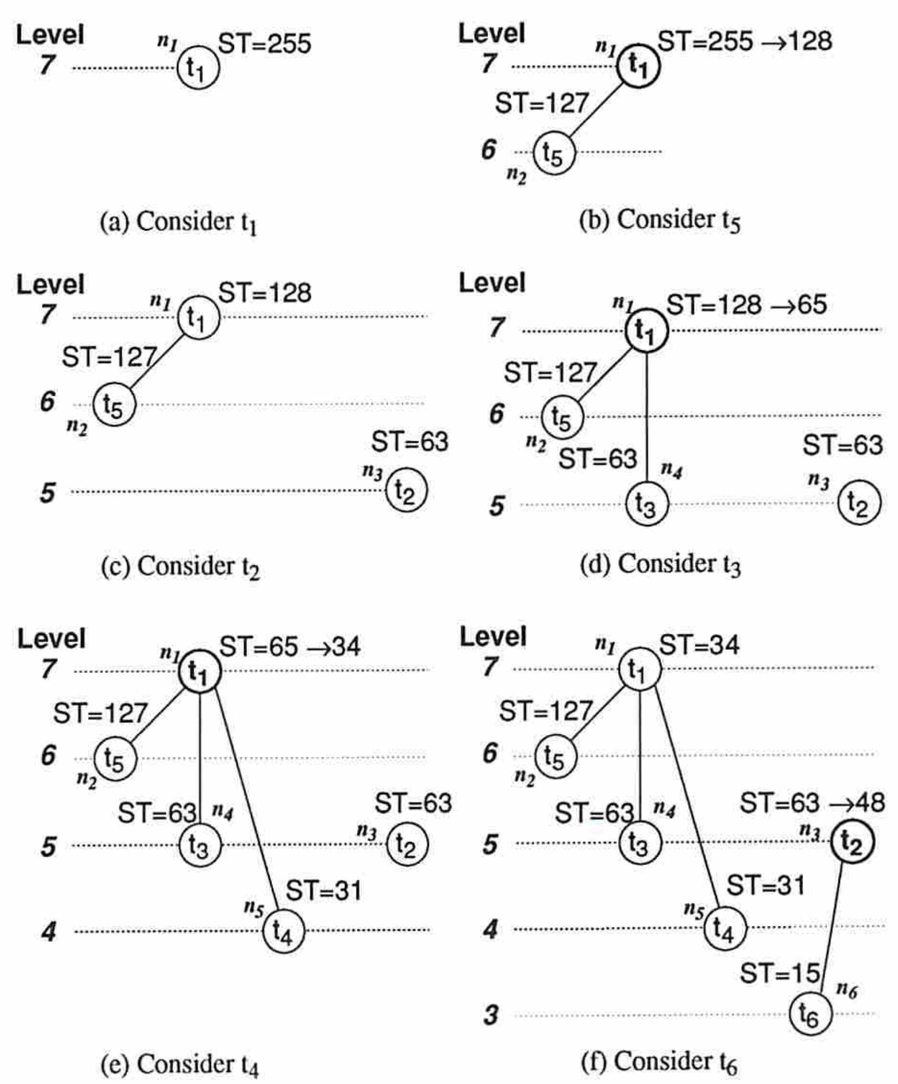


Figure 7.5: The process of Procedure *genTCF* (steps (a) through (f))

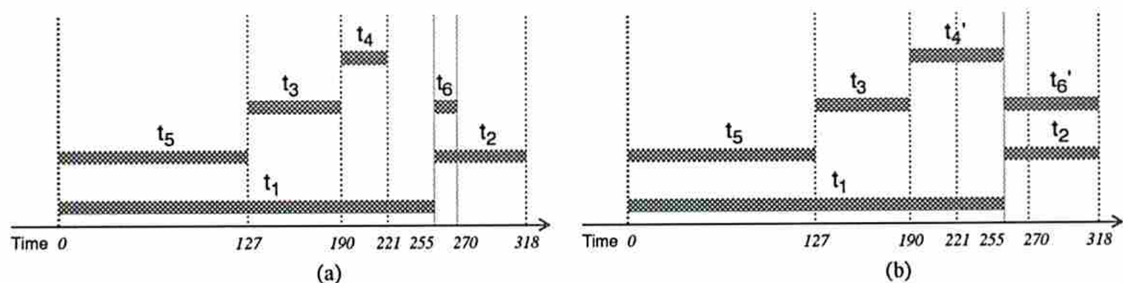


Figure 7.6: (a) test schedule for the BISTable circuit in Figure 7.1; (b) a modified test schedule from (a) to simplify test controller

remains the same. A modified test scheduler is shown in Figure 7.6(b) where t_4 and t_6 in Figure 7.6(a) are extended to be t'_4 and t'_6. The procedure for extending tests in a test scheduler to simplify a test controller (and increase fault coverage) without increasing the total test time will be presented in the next chapter.

Lemma 7.1 When Procedure *build_TCT* is called for the ith time, its complexity is $O(i)$.

Proof It is obvious that the number of nodes in the current TCF after Procedure *build_TCT* has been executed $(i-1)$ times is at most $(i-1)$. Procedure *search_node* visits each node in the current TCF at most once, where a compatibility test and a calculation of the score function can be performed in constant time. The remainder of Procedure *build_TCT* is done in constant time, hence the complexity is $O(i)$. □

Theorem 7.1 The complexity of finding a test schedule for n tests using the procedures presented in this section is $O(n^2)$.

Proof The complexity of Procedure *genTCF* is $O(\sum_{i=1}^{n} i) = O(n^2)$. Once a TCF is constructed, a unique test schedule can be generated by Procedure *TCF2TS* in $O(n)$ time since it is a tree traversal algorithm and the number of nodes in a TCF is bounded by n. Therefore, the complexity of finding a test schedule for n tests is $O(n^2)$. □

Note that in each step shown in Figure 7.5, a partial schedule for the tests already considered can be obtained. A test time for this partial schedule can then

be calculated by simply adding the test length of every root node. Therefore, a partially specified circuit having an unacceptable test time can be rejected immediately. This can be illustrated by the example below.

Example 7.3 Suppose the maximal allowable test time for the circuit shown in Figure 7.1 is 318 clock cycles. Assume that an alternative embedding for K_1 is e_1', where R_1 instead of R_2 operates as a TPG. The embeddings for the other kernels remain the same. Let t_i' be the test for K_1 using embedding e_1'. t_1' is compatible with t_4, t_5 and t_6 but not with t_2 and t_3. The process of constructing a TCF for these tests is shown in Figure 7.7. In step (d) after t_3 has been assigned to a node, the test time of this partial schedule is $255 + 63 + 63 = 381$ clock cycles, which exceeds the maximal allowable test time and thus this BISTable circuit can be rejected without further processing. Notice that a partially specified circuit containing embeddings e_1', e_5, e_2 and e_3 can lead to many different feasible BISTable circuits by selecting one embedding for each of the remaining kernels, namely K_4 and K_6. These feasible BISTable circuits will not be generated in the design space exploration process due to the early rejection of this partially specified circuit. □

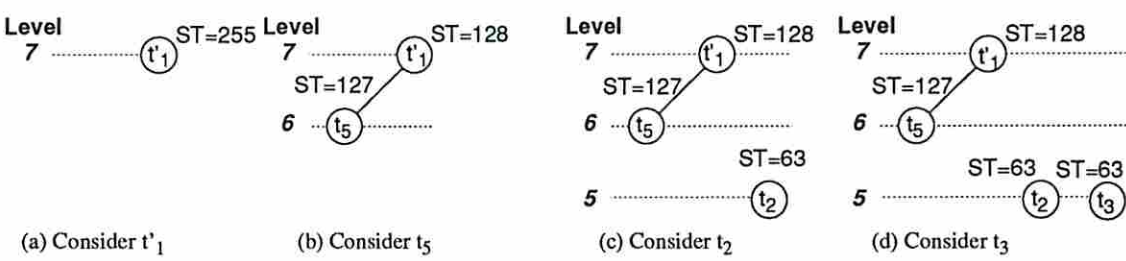


Figure 7.7: The process of Procedure *genTCF* for Example 7.3

7.3.3 Test Time Lower Bound Estimation

Given a set of tests and their TCG, a *test incompatibility graph(TIG)* can be constructed in which a vertex appears for each test and an edge exists between a pair of vertices representing two tests t_i and t_j if t_i and t_j are incompatible. A *clique* in a graph G is a maximally connected subgraph of G.

Lemma 7.2 Let C_j be a clique in a TIG containing a set of vertices representing tests $t_1, t_2, \ldots, t_n$. Let T_i denote the test length of t_i. Then the test time required to execute the tests in the TIG is lower bounded by $T_{C_j} = \sum_{1 \le i \le n} T_i$.

Proof Clearly tests in C_j are pair-wise incompatible, thus none of them can run concurrently with any other test in C_j. Therefore tests in C_j have to be initiated in sequence and the test execution time for the tests in C_j, denoted by T_{C_j}, is $\sum_{1 \le i \le n} T_i$. This is obviously a lower bound for the test time of the tests in the TIG. □

Theorem 7.2 Let $CS = \{C_1, C_2, \ldots, C_m\}$ be the set of distinct cliques in a TIG. Then the test time of tests in the TCG is lower bounded by $\max_{1 \le j \le m} T_{C_j}$.

Proof Follows from Lemma 7.2. □

The procedure to calculate a lower bound for the test time of a BISTable circuit, known as *TT_lower_bound*, is presented below.

Procedure *TT_lower_bound*
/* INPUT: A BISTable circuit C_b. */
/* OUTPUT: A lower bound for the test time of C_b. */

1. Construct a TIG G for tests in C_b.
2. Enumerate cliques in G. Let CS denote the set of these cliques.
3. For every $C_j \in CS$, $T_{C_j} = \sum_{t_i \in C_j} T_i$.
4. $lb = \max_{C_j \in CS} T_{C_j}$, return($lb$).

The lower bound calculated by Procedure *TT_lower_bound* may not be the greatest lower bound for the test time of a BISTable circuit. However, it can still be used to evaluate the quality of the test schedule generated by Procedure *schedule*.

Example 7.4 The TIG for the tests in Example 7.2 is shown in Figure 7.8. Cliques in the TIG are enumerated and shown with their associated test time below.

$$\begin{array}{ll} C_1 = \{t_1, t_2\} & T_{C_1} = 255 + 63 = 318 \\ C_2 = \{t_2, t_3\} & T_{C_2} = 63 + 63 = 126 \\ C_3 = \{t_3, t_6\} & T_{C_3} = 63 + 15 = 78 \\ C_4 = \{t_4, t_5, t_6\} & T_{C_4} = 31 + 127 + 15 = 173 \end{array}$$

Therefore, the lower bound calculated by Procedure *TT_lower_bound* is 318 clock cycles. It can be observed that the test schedule generated by the procedures presented in this section is optimal. □

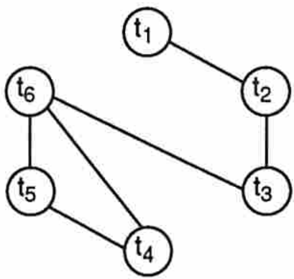


Figure 7.8: The TIG for tests in Example 7.2

Procedure *TT_lower_bound* can be extended to calculate a lower bound for the test time of a CUC. Recall that a set of embeddings can be constructed for each kernel in a CUC by the kernel embedding process presented in the previous chapter. Two kernels, K_1 and K_2, are said to be *incompatible* if every embedding for K_1 is incompatible with every embedding for K_2. Once this kernel compatibility analysis has been performed, a *kernel incompatibility graph(KIG)* similar to a TIG can be constructed. Procedure *TT_lower_bound* can then be executed on a KIG to determine a lower bound for the test time of a CUC. This bound can be used to help a designer specify a reasonable test time constraint.

7.4 Performance Evaluation

Among the test scheduling schemes presented[21, 22, 25, 26, 24], the approach in [25] seems to produce the best results in most cases. To evaluate the performance of the test scheduler presented in this thesis, the procedure presented in [25] has been implemented and experiments have been conducted to compare with our test scheduler. The experiments employ randomly generated graphs to represent TCGs. As in [25], each test length is restricted to be $2^k - 1$ for some integer value of k, where k is also randomly generated. The procedure in [25] also employs a TCF to represent the concurrency between tests, however, it is quite different from the

procedure presented here.

A series of experiments has been conducted to compare these two schemes. The results from each experiment are based on 100 random graphs. Two parameters are specified in each experiment, namely the number of tests (vertices) n and a parameter S_{MAX} that determines the maximal test length of the tests in the experiment. The test length of each test is $2^k - 1$, where k is randomly selected between 1 and S_{MAX}. The results are evaluated with respect to three metrics, namely, (1) the time t required to execute a schedule, (2) the execution time r for the procedures, and (3) the number of nodes m in the TCF generated by the procedures. These metrics are indicators of performance, time complexity, and space complexity, respectively, for the test schedulers. The results are summarized in Table 7.1. The quantities t_1, r_1 and m_1 are calculated by taking the average of the t's, r's and m's, respectively, obtained by running our scheduler on 100 graphs for a given pair of parameters. Similarly the quantities t_2, r_2 and m_2 are obtained by running the scheduler described in [25]. t_{lb} is the average lower bound test time estimated by Procedure *TT_lower_bound* on the same graphs. Procedure *TT_lower_bound* is an exponential time algorithm since it enumerates cliques in a graph. Therefore it is not computationally feasible to apply it to large graphs. Our program cannot handle the case when the number of tests exceeds 50.

The comparative results between the two scheduling techniques and the comparisons between the test time required by our test scheduler and the estimated lower bound are shown in Table 7.2. The column under t_1/t_2 compares the quality of the test schedules generated by the two schemes. The column under r_1/r_2 compares the computational complexity of the two procedures. And the column under m_1/m_2 compares the space complexity of the two procedures. The time complexity of the scheduler presented in [25] is $O(ne)$, where n is the number of tests and e is the number of edges in the TCG. Compared with our test scheduler, the scheduler in [25] is more complicated since e is usually greater than n. In general, the scheme presented in this thesis requires much less CPU time and memory than the more complicated scheme in [25]. In addition, as S_{MAX} increases, in most cases the efficiency of our scheme over that in [25] also increases. In most cases we achieve a slightly better performance in terms of the test schedules generated. The scheme

Parameters		Averaged results						
n	S^*_{MAX}	t_1	r_1	m_1	t_2	r_2	m_2	t_{lb}
10	1	4	.0007	4	4	.0020	4	4
10	5	50	.0008	8	50	.0020	12	49
10	10	970	.0008	9	974	.0025	23	967
20	1	6	.0025	6	6	.0077	6	5
20	5	78	.0028	16	79	.0060	24	74
20	10	1614	.0033	18	1627	.0072	49	1594
20	20	986262	.0030	19	987103	.0092	101	981133
30	1	9	.0052	9	8	.0185	8	6
30	5	103	.0057	24	105	.0127	34	94
30	10	2119	.0060	27	2146	.0150	70	2057
30	20	1350456	.0063	29	1358229	.0175	153	1332539
40	1	11	.0095	11	10	.0333	10	7
40	5	121	.0095	31	122	.0210	44	106
40	10	2463	.0107	36	2503	.0238	93	2332
40	20	1579648	.0103	38	1599495	.0288	211	1548568
50	1	13	.0135	13	12	.0560	12	8
50	5	141	.0145	38	142	.0313	54	118
50	10	2886	.0148	45	2916	.0363	115	2647
50	20	1809779	.0153	48	1827550	.0413	264	1754156
100	1	21	.0392	21	19	.2103	19	-
100	5	218	.0378	68	219	.1000	97	-
100	10	4451	.0398	87	4499	.0987	217	-
100	20	2840975	.0423	95	2871117	.1083	521	-

n: number of tests
$2^{S_{MAX}} - 1$: maximal test length
$t_1(t_2)$: test application time required by our scheme(scheme in [25])
t_{lb}: estimated lower bound test time
$r_1(r_2)$: CPU time (in seconds) consumed by our scheme(scheme in [25])
$m_1(m_2)$: number of nodes in a TCF generated by our scheme(scheme in [25])
* - the actual number of test lengths can be multiplied by a reasonable factor say 1000. The test application time is multiplied by the same factor.

Table 7.1: Summary of the experimental results

presented in [25] slightly outperforms ours for cases of equal test lengths. This may be due to the fact that in [25] they utilize a TCG to find the maximal compatible test set before a scheduler is executed. The last column t_1/t_{lb} compares the test time of schedules generated by our scheduler with the estimated lower bound test time. It can be observed that the test time generated by our scheme is very close to the estimated lower bound, and it seems to get closer to the lower bound as S_{MAX} increases. This implies that we tend to get "near optimal" results and not much improvement can be made.

Although the improvement of our scheme over that presented in [25] is not significant in terms of the test time, it is believed that this new test scheduler is superior due to the following reasons. First, the improvement of our scheme in terms of time and memory efficiency is substantial. Hence larger problems can be handled much more efficiently. Furthermore, if we are working in an environment where memory is the bottleneck as is the case with most CAD tools, the saving of memory in this scheme is very important. Secondly, our scheduling scheme does not require a fully specified circuit as opposed to all the other approaches. The advantage is that a lower bound on test time can be calculated for any partially specified circuit, hence non-representative circuits are rejected early in the exploration process.

7.5 Summary

We considered the test scheduling problem in this chapter. Partitioned testing with run to completion strategy is employed in the test scheduler. This test scheduling strategy strikes a good balance between test application time and test controller complexity. The advantage of an incremental test scheduler has been demonstrated in the context of a BISTable circuit design process, such as our BITS system.

We have presented an incremental test scheduler that has been shown to generate "near optimal" test schedules. The computational complexity of this test scheduler is $O(n^2)$, where n is the number of kernels in a CUC. Therefore, it is more efficient than most of the existing test scheduling procedures. In addition,

Parameters		Comparisons			
n	S^*_{MAX}	t_1/t_2	r_1/r_2	m_1/m_2	t_1/t_{lb}
10	1	1.000	.333	1.000	1.000
10	5	.992	.417	.698	1.020
10	10	.996	.333	.399	1.003
20	1	1.000	.326	1.000	1.200
20	5	.995	.472	.668	1.043
20	10	.992	.465	.378	1.012
20	20	.999	.327	.191	1.007
30	1	1.035	.279	1.035	1.500
30	5	.985	.447	.675	1.096
30	10	.987	.400	.391	1.030
30	20	.994	.362	.190	1.014
40	1	1.046	.285	1.046	1.571
40	5	.990	.452	.693	1.141
40	10	.984	.448	.389	1.056
40	20	.988	.358	.183	1.020
50	1	1.051	.241	1.051	1.625
50	5	.992	.463	.692	1.194
50	10	.990	.408	.390	1.090
50	20	.990	.371	.182	1.031
100	1	1.091	.186	1.091	-
100	5	.998	.378	.708	-
100	10	.989	.404	.405	-
100	20	.990	.390	.182	-

Table 7.2: Comparisons between the two scheduling schemes and the test time lower bound

the performance of our scheduler seems to be superior to a state of the art scheduling scheme based on experimental results. Theoretical lower bounds for the test time of a BISTable circuit, as well as for the test time of a CUC, have been developed. Procedures to calculate such bounds have also been presented. The process of using the incremental test scheduler to reject a non-representative circuit has been illustrated. This will be considered in further details in the next chapter.

Chapter 8

Design Space Exploration

8.1 Introduction

Given a CUC and a selected TDM, there may be several embeddings for each kernel in the circuit. This leads to many different BISTable versions for the original circuit. Each BISTable circuit has a certain value in terms of each BIST parameter such as area overhead, test application time, and performance degradation. In general not all BISTable circuits need to be generated since some of them may not satisfy the design constraints. In addition, one BISTable circuit may have inferior values in terms of all BIST parameters than those of another BISTable circuit. In such a case the former circuit is said to be *non-representative*. It is desired not to generate non-representative BISTable circuits in exploring a design space since this reduces the design space and speeds up the exploration process. Most existing BISTable circuit design systems[4, 5, 27, 7] employ an adaptive approach in exploring a design space. That is, an initial BISTable circuit is produced based on certain criteria, and then modifications are made to the initial circuit until an acceptable solution is found. This lack of a global view of a design space may not lead to the generation of the 'best' BISTable circuit with respect to the design constraints. In addition, it may be desirable to provide a designer with a family of BISTable circuits, where each of them has different value in terms of every BIST parameter. The designer can then make trade-offs between BIST parameters and select the best BISTable circuit based on the design constraints. It is often difficult and inefficient for a designer to make such trade-offs using an adaptive system

as described above. Due to these reasons, a BISTable circuit design system that systematically and efficiently explores a design space is desired.

In this chapter we first illustrate a typical design space and define a representative BISTable circuit. A procedure that systematically explores a design space and generates a family of representative BISTable circuits including a minimal area overhead design, a minimal test time design, and a minimal performance degradation design is presented. When a CUC is complex, its design space may be large and time consuming to fully explored. Techniques to reduce the size of a design space are presented. Once a representative BISTable circuit is generated it may contain small SA registers, say less than 8 bits. As the width of a SA register decreases, the probability that a faulty signature equals a fault-free signature (i.e. *aliasing probability*) increases. In this case the BITS system will merge small SAs into larger ones to reduce the aliasing probability. For each representative BISTable circuit, a test schedule is generated by the test scheduler presented in the previous chapter. Each test schedule needs to be translated into a *test plan* so that a test controller can be synthesized. The process of test plan generation is also presented.

8.2 Design Space and Representative BISTable circuits

From the previous chapters, it was observed that several embeddings may exist for each kernel. A *feasible BISTable circuit* is defined as a combination of embeddings so that every kernel is associated with an embedding, and the conflicts between embeddings are properly resolved by means of test scheduling. For a complex CUC, there can be many feasible BISTable circuits. Based on our experience the design space of a CUC can be represented as shown in Figure 8.1(a), assuming test time and area overhead are the only BIST parameters considered.

In a design space as shown in Figure 8.1(a), the point where test time is minimal is called a *minimal test time BISTable circuit*; the point where area overhead is minimal is called a *minimal area overhead BISTable circuit*. Between these two points are intermediate BISTable circuits. A BISTable circuit is non-representative if both its test time and area overhead are inferior to those of another BISTable

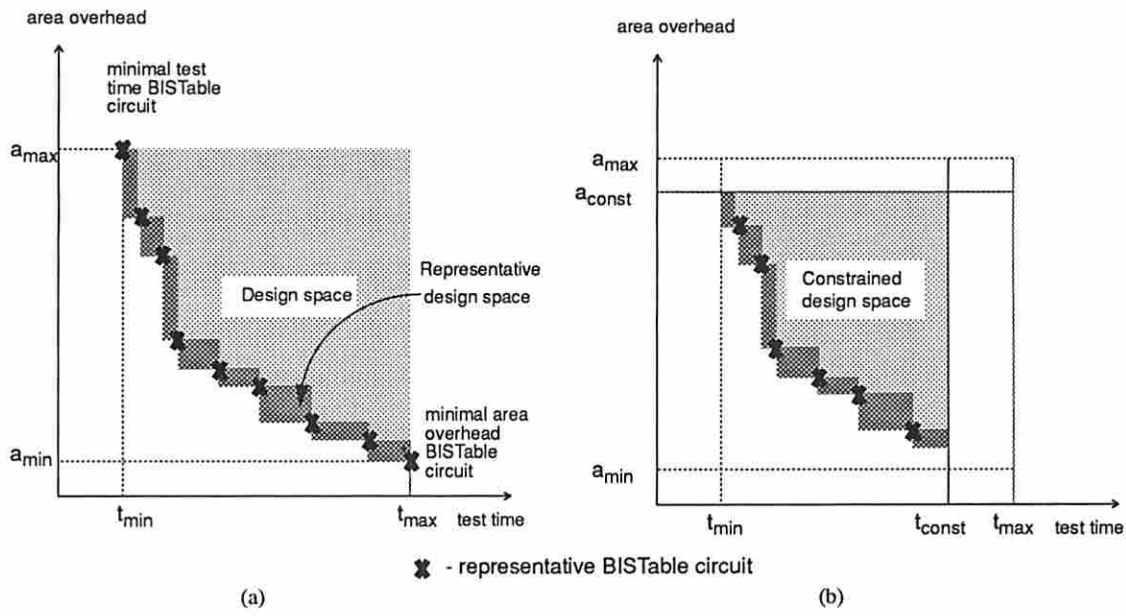


Figure 8.1: (a) a typical design space; (b) a design space with design constraints

circuit; otherwise it is representative. The subset of the design space shaded darker in Figure 8.1(a) is called *representative design space*. Only BISTable circuits in the representative design space need to be generated by the design space explorer.

When a designer specifies constraints on test time and area overhead, denoted by t_{const} and a_{const} respectively, the design space may be constrained if t_{max} is greater than t_{const} or a_{max} is greater than a_{const}. A constrained design space is illustrated in Figure 8.1(b).

8.3 Design Space Exploration Procedure

The design space exploration procedure is a branch and bound procedure that implicitly enumerates feasible BISTable circuits. The test scheduler presented in the previous chapter is integrated with the branch and bound procedure. Therefore a lower bound for the test time of every partially specified circuit can easily be calculated. Once a partially specified circuit is identified as unpromising (i.e. cannot lead to a representative BISTable circuit), the design space explorer backtracks and thus no feasible BISTable circuits that follow this partially specified circuit will be

generated.

The design space explorer incrementally specifies a BISTable circuit by associating an embedding with each kernel. For each kernel under consideration, the embeddings for this kernel are considered in sequence. Initially kernels are sorted in descending order based on the number of required test patterns. The branch and bound search process is carried out in a depth first fashion. As we have illustrated in the previous chapter, the search space of this process is a tree where every non-leaf node represents a partially specified circuit and every leaf node represents a feasible BISTable circuit. Area overhead of a partially specified circuit is estimated based on the required test hardware, such as TPGs and SAs, by examining the r-f pairs of the embeddings associated with the circuit. Test time of a partially specified circuit is calculated by the integrated test scheduler as discussed in the previous chapter. Performance degradation of a partially specified circuit is estimated by examining the critical paths in the circuit. Each register that is modified in the partially specified circuit introduces a certain amount of delay. If the register is in a critical path, it may affect circuit performance. Therefore the extra delays on every critical path is first calculated, and then the performance degradation can be determined by taking the worst case scenario. Other BIST parameters such as extra I/O pins can also be estimated for each partially specified circuit if desired. A partially specified circuit is identified as *unpromising* if there exists a representative BISTable circuit already generated that has superior values in terms of all BIST parameters to those of the partially specified circuit. From our experience, for most circuits only a small portion of the feasible BISTable circuits are generated by the design space explorer. This makes the BITS system efficient and practical. The procedure to explore a design space, known as *DSE*, is presented below.

Procedure *DSE*
/* INPUT: A CUC and a set of embeddings for each kernel in the circuit. */
/* OUTPUT: A family of representative BISTable circuits. */

1. Let $KS = \{K_1, K_2, \ldots, K_n\}$ be a set of kernels in the CUC. The number of required test patterns for K_i is greater than or equal to the number of required test patterns for K_{i+1}, for $k = 1$ to $n - 1$. Let $t = \infty$.

2. For $i = 1$ to n do

(a) let E_i be the set of embeddings for K_i.

(b) select an embedding from E_i, say e, that has not been employed. If no such embedding exists, goto step 2.

(c) if the test length of e is less than t do

i. schedule the test for e.

ii. let t be the test length of e.

(d) otherwise do

i. let TS be the set of tests already scheduled. $TS = TS \cup \{\text{test for } e\}$.

ii. sort TS based on their test lengths in descending order.

iii. schedule tests in TS.

(e) estimate values for all BIST parameters of this partially specified circuit. If the circuit is unpromising, backtrack.

3. Record the fully specified circuit as a representative BISTable circuit. Backtrack.

Recall that in the test scheduler presented in the previous chapter, tests are scheduled in a descending order based on their test lengths. When the latency of an embedding is greater than 1, the test length of the embedding is greater than the number of required test patterns for the embedded kernel. In such a case tests are not ordered properly and the test scheduler cannot properly handle this set of tests. To ensure that tests are scheduled in a correct order, a variable t is used in Procedure *DSE* to keep track of the current shortest test length of the tests already scheduled. Initially t is set to a large number, denoted by ∞. Procedure *DSE* first selects kernel K_1, i.e. the kernel requires the most number of test patterns. An embedding for K_1, say e, is then selected. If the test length of e is not greater than t, then the tests are still in a proper order and the test for e can be scheduled. t is updated accordingly. On the other hand, if the test length of e is greater than t, tests are not properly ordered. In this case the partial schedule and the current TCF for the tests already scheduled are discarded. After that, these tests are sorted in descending order based on their test lengths and a new partial schedule is computed using this correct order. The above process is repeated for every kernel until each kernel is associated with an embedding, or the partially specified circuit is unpromising. In the former case, a representative BISTable circuit is generated and DSE backtracks to the search state where the previous selection is made so

that the design space can be fully explored. In the latter case, DSE also backtracks to reject the unpromising partially specified circuit.

Notice that in the worst case when tests for embeddings are scheduled in increasing test lengths, the complexity of the test scheduler increases to $O(n^3)$ since the scheduling process has to start over again for each test. However, based on our experience the latency of most embeddings is 1. Therefore, rescheduling rarely occurs.

When only a minimal area overhead design, a minimal test time design, or a minimal performance degradation design is desired, Procedure *DSE* is tailored to more efficiently generate such a design by keeping a bound on the respective attribute and pruning all circuits that exceed the bound. For example, when only a minimal area overhead design is desired, a variable min_a is employed to represent the current minimal area overhead. Step 2(e) in Procedure *DSE* will determine a partially specified circuit as unpromising if it has an area overhead greater than min_a. In addition, step 3 in Procedure *DSE* only records one design, i.e. a minimal area overhead design, instead of a family of BISTable designs. A fully specified circuit will become a new minimal area overhead design if its area overhead is smaller than min_a, or if its area overhead equals min_a and it is superior to the current minimal area overhead design in terms of all other attributes. A minimal test time design and a minimal performance degradation design can be efficiently generated by modifying Procedure *DSE* in a similar fashion.

8.4 Complexity Issues

As we discussed in the previous chapter, the total number of feasible BISTable circuits in a CUC is $\Pi_{i=1}^{n}|E_i|$, where E_i denotes the set of embeddings for kernel K_i and n is the number of kernels in the circuit. In the worst case, Procedure *DSE* has to examine each feasible BISTable circuit when no pruning of partially specified circuit is possible. The bounding techniques in Procedure *DSE* usually significantly reduce the search space by pruning unpromising partially specified circuits. However, for a complex circuit its search space after pruning may still be excessively

large. Therefore we develop two techniques, namely using only dominating I-paths and selectively exploration of a design space, to further reduce a search space.

8.4.1 Using Dominating I-paths

A BISTable circuit is said to be *better* than another BISTable circuit if the former has superior values to those of the latter in terms of all BIST parameters.

Definition 8.1 *A driving (receiving) path is said to* **dominate** *another driving (receiving) path if given any BISTable circuit that employs the former path, replacing it with the latter path does not lead to a better BISTable circuit.*

In most data path circuits having many MUXes and/or busses, the number of driving and receiving paths may be large. It is often true that some driving (receiving) paths dominate other driving (receiving) paths. Since driving or receiving paths that are dominated by other paths never lead to representative BISTable circuits, they need not be considered in the design space exploration process. Two classes of I-path domination have been identified, namely equivalent I-paths and subsuming I-paths.

Consider a portion of a circuit as shown in Figure 8.2. There are four driving paths I_1, I_2, I_3 and I_4 for the input port of K. I_1, I_2, I_3 and I_4 are said to be *equivalent* since they contain the same components except the drivers, and the drivers are not employed in any other I-paths. Equivalent I-paths can be formally defined as follows.

Definition 8.2 *Given two driving (receiving) paths, say I_1 and I_2, that contain the same components except the drivers (receivers), say R_1 and R_2. If R_1 is only contained in I_1 and R_2 is only contained in I_2, then I_1 and I_2 are said to be* **equivalent**.

Theorem 8.1 If I_1 and I_2 are equivalent, then I_1 dominates I_2 and I_2 dominates I_1.

Proof Without loss of generality, we assume that both I_1 and I_2 are driving paths. If I_1 and I_2 are equivalent, let R_1 and R_2 be the drivers of I_1 and I_2, respectively. The remaining components in I_1 and I_2 are identical. In addition, either both

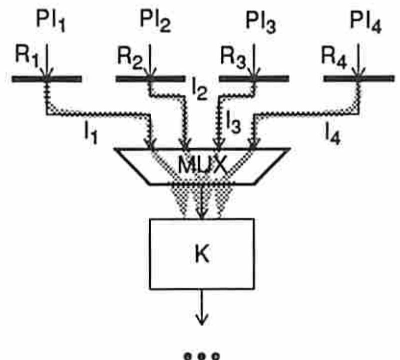


Figure 8.2: Equivalent I-paths

R_1 and R_2 are not contained in any other I-paths; or for each I-path (I_1 and I_2 excluded) that contains R_1, it also contains R_2. In either case, given a BISTable circuit that employs I_1, replacing I_1 with I_2 leads to a BISTable circuit having the same value as the original one in terms of every BIST parameter. Therefore I_1 dominates I_2. Similarly we can prove that I_2 dominates I_1. □

If two I-paths are equivalent, clearly only one of them needs to be considered and the choice can be made arbitrarily. Therefore, given a set of equivalent I-paths, one of them is selected as a dominating I-path and the remainder are said to be dominated by the selected path. Equivalent I-paths can be found in many data path circuits.

Definition 8.3 *A driving (receiving) path I_1 is said to* **subsume** *another driving (receiving) path I_2 if every component in I_1 is also contained in I_2.*

An example subsuming I-path can be found in the circuit shown in Figure 7.1. The receiving path (K_3, $MUX_2(K_3)$, R_8) subsumes the receiving path (K_3, $BUS_1(K_3)$, $MUX_2(BUS_1)$, R_8) since every component in the former is contained in the latter.

Theorem 8.2 If I_1 subsumes I_2, then I_1 dominates I_2.
Proof Clearly if a component C_i in I_1 is also contained in some other I-path, say I_3, and C_i leads to a conflict between I_1 and I_3, then the same conflict exists between I_2 and I_3 since C_i is also contained in I_2. Given any BISTable circuit

that employs I_1, if I_1 is replaced with I_2, conflicts in the original BISTable circuit remain in the new BISTable circuit. In addition, the area overhead of the new BISTable circuit cannot be less than that of the original BISTable circuit since all the test hardware added to the original circuit is also required in the new circuit. Consequently, the value for every BIST parameter in the new BISTable circuit cannot be better than that of the original one, thus I_1 dominates I_2. □

The current BITS system automatically identifies dominating I-paths once driving and receiving paths in a CUC have been constructed. Only these paths are employed in the kernel embedding and design space exploration processes.

8.4.2 Selective Exploration

As pointed out earlier, the worst case complexity of the search space is $CSS = \Pi_{i=1}^{n}|E_i|$, where n is the number of kernels and E_i is the set of embeddings for K_i. For a complex circuit, it may be desirable to restrict each $|E_i|$ in order to reduce the complexity of a search space.

If $|E_i| > 1$ it may be possible to determine *a priori* that an embedding in E_i never leads to a representative BISTable circuit. Thus the embedding can be ignored by Procedure *DSE*. For example, consider the circuit shown in Figure 8.3. There are two possible embeddings, $e_{2,1}$ and $e_{2,2}$, for K_2 as shown in the figure. Both embeddings are compatible with the embedding for K_1, namely $e_{1,1}$. In this circuit $e_{2,1}$ is clearly better than $e_{2,2}$ since it allows R_1 to be shared between $e_{1,1}$ and $e_{2,1}$, thus leading to less area overhead. In addition, the sharing does not cause incompatibility. Therefore Procedure *DSE* only needs to consider $e_{2,1}$ when kernel K_2 is under consideration. $e_{2,1}$ is said to be a *representative embedding* for K_2.

In general it may not always be possible to guarantee that one embedding is better than another embedding since the relationship between test hardware sharing and compatibility is more complicated than shown in the above example. Therefore heuristic approaches are employed that choose a set of representative embeddings for each kernel to be employed in Procedure *DSE*. Suppose the embeddings for every kernel in a CUC have been enumerated. Given an arbitrary embedding e, two measures are defined as the criteria in choosing embeddings.

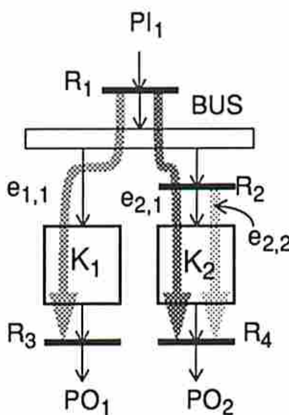


Figure 8.3: An example circuit to illustrate selective exploration

Definition 8.4 *The number of embeddings that share test hardware with e, denoted by $DS(e)$, is called the* **degree of sharing** *for embedding e. The number of embeddings that are compatible with e, denoted by $DC(e)$, is called the* **degree of compatibility** *for embedding e.*

Intuitively, greater values of $DS(e)$ tend to reduce area overhead and performance degradation of a BISTable circuit. Also, greater values of $DC(e)$ tend to reduce test time of a BISTable circuit. Therefore, given two embeddings $e_{i,1}$ and $e_{i,2}$ for the same kernel K_i, if $DS(e_{i,1}) \geq DS(e_{i,2})$ and $DC(e_{i,1}) \geq DC(e_{i,2})$ then employing $e_{i,1}$ tends to produce a better BISTable circuit than employing $e_{i,2}$. By this assumption an embedding whose values for both the degree of sharing and the degree of compatibility are worse than those of another embedding for the same kernel is ignored by Procedure *DSE*. Once the above reduction on the embeddings for each kernel has been done, if there exists an $|E_i|$ that is still too large (greater than a pre-defined value l), further reduction on E_i is required. In this case the remaining embeddings in E_i are divided into l (or less) groups $E_{i,1}$, $E_{i,2}$, ..., $E_{i,l}$. An embedding e is in group $E_{i,j}$ if

$$DS_{min} + (j-1) \times \Delta \leq DS(e) \leq DS_{max} + j \times \Delta,$$

where $DS_{min}(DS_{max})$ is $\min_{e \in E_i} DS(e)$ ($\max_{e \in E_i} DS(e)$) and $\Delta = \lceil (DS_{max} - DS_{min}) / l \rceil$. The reduction on E_i is done by choosing one embedding (if any)

from each group that has the greatest degree of compatibility, and ignoring the remaining embeddings in the same group. Ties are broken arbitrarily if two or more choices exist. This heuristic is to ensure that the embeddings considered have a variety of values in terms of the degree of sharing and the degree of compatibility.

The two heuristics developed in this section have been implemented in the BITS system. In the design space exploration process, the number of feasible BISTable circuits (i.e. CSS) is first calculated. If this number exceeds a certain threshold value (currently set as $100,000$), heuristics are invoked to reduce the complexity of the search space.

8.5 SA Merging to Reduce Aliasing Probability

It has been shown that the aliasing probability of an n-bit SA is approximately $1/2^n$, independent of the primitive polynomial employed in configuring the SA and the model used to characterize the error patterns[2]. In practice, the width of a SA is increased if its aliasing probability is not acceptable. In this section we consider the problem of reducing the aliasing probability of a BISTable circuit by combining two or more SAs into a larger SA or adding F/Fs (if necessary) to existing SAs. We assume that the individual SA for each output port of every kernel has been determined and the polynomial to configure the SA selected. The maximal allowable aliasing probability is assumed to be $1/2^L$.

Given a BISTable circuit, initially a SA, say R, is associated with each output port, say P, of a kernel. In addition, the width of R is equal to the width of P. An unacceptable aliasing probability occurs when the width of R is less than L. Consider a SA R in a BISTable circuit having a width less than L.

Definition 8.5 *A register R' is said to be* **mergable** *with R if combining R and R' into a larger SA does not invalidate the BISTable circuit.*

Register merging may invalidate a BISTable circuit if its test schedule becomes infeasible due to the introduced contentions in test hardware. This can be illustrated by the next example.

Example 8.1 Consider the BISTable circuit shown in Figure 7.1 and a test schedule for the circuit as shown in Figure 7.6. Suppose the EXTBILBO TDM is employed and R_1 is an SA register having an insufficient width. R_4 is not mergable with R_1 since R_4 drives and R_1 receives from the same kernel (i.e. K_3). If R_1 and R_4 are merged into one register, this register would simultaneously operate as a TPG and a SA, which is not allowed. R_5 and R_{12} are also not mergable with R_1 due to the same reason. R_2 is not mergable with R_1 although R_2 drives and R_1 receives from different kernels. This is because the test for K_3 (i.e. t_3) runs concurrently with the test for K_1 (i.e. t_1) as can be seen from Figure 7.6. If R_1 and R_2 are merged into one register, it would also simultaneously operate as a TPG and a SA. On the other hand, R_6 and R_7 are mergable with R_1 since no contention in the test hardware may occur. □

Let S_i denote the set of registers that operate as SAs in test session TS_i and have a width less than L. In general, a register R' is not mergable with a SA register R in S_i if R' is part of a driving path (including its driver) or part of a receiving path (excluding its receiver) activated in TS_i. Note that R and R' may be employed in an embedding for the same kernel or in embeddings for different kernels. Based on the above observation, given a register R in S_i, a set of mergable registers with R in TS_i, denoted by $MS_i(R)$, can be identified by examining the driving and receiving paths activated in TS_i. Note that it may be necessary to merge a SA R with more than one register in $MS_i(R)$ to obtain a sufficient width for the merged SA. Suppose that R is to be merged with two other registers R_1 and R_2. Clearly both R_1 and R_2 are mergable with R in TS_i. This implies that neither R_1 nor R_2 are part of a driving path or a receiving path (receivers excluded) activated in TS_i. Therefore the BISTable circuit will not be invalidated by this merging since the merged SA does not introduce conflicts. This is true for general cases, i.e. merging R with n registers. Consequently, merging a SA R with any number of registers in $MS_i(R)$ will not invalidate a BISTable circuit. Let MS be the set of registers that are mergable with at least one register in S_i for all TS_i. Clearly $MS = \bigcup_{\forall i \text{ and } R_j \in S_i} MS_i(R_j)$. Registers in MS that already operate as SAs or TPGs are preferable than other normal registers in MS since the former require less extra area overhead than the latter for the register merging. Therefore, a cost

can be associated with each register in MS to denote the extra hardware required if it is merged with a SA. Based on the above discussion, the SA register merging problem is formulated as a covering problem described below. If R_k is mergable with a SA register, say R_j, in test session TS_i, and the width of R_k is denoted by $|R_k|$, then combining R_j and R_k increases the width of the SA by $|R_k|$. A merging matrix M_i having $|S_i|$ rows and $|MS|$ columns can be constructed for each test session TS_i. $M_i[j,k] = |R_k|$ if $R_k \in MS_i(R_j)$ and $R_j \in S_i$. Otherwise $M_i[j,k] = 0$. For each register in MS, say R_k, a cost c_k denotes the extra hardware required to configure R_k as an LFSR and be merged with a SA register. c_k usually depends on the width and the functions of R_k. A binary variable x_k denotes whether R_k is selected in the merging process. Namely, if $x_k = 1$ then R_k is merged with at least one SA register and 0 otherwise. L_j denotes the required width for SA register R_j. In practice L_j can be a constant, say 32. An ILP formulation for the covering problem is shown below.

$$\begin{cases} \text{minimize} & \sum_{R_k \in MS} c_k \cdot x_k \\ \text{subject to} & \sum_{R_k \in MS} M_i[j,k] \cdot x_k + |R_j| \geq L_j \quad \forall R_j \in S_i \text{ and } \forall TS_i \end{cases}$$

The objective function is to minimize the total extra hardware required for the SA merging. The constraints ensure that after the merging each SA register has a sufficient width. Notice that there may be no mergable registers exist for a SA, say R_j, or combining the mergable registers with R_j does not lead to a sufficient width, In which case additional registers that are only used in the test mode have to be added to register R_j so that it can be configured as an LFSR of degree L_j during test mode.

Example 8.2 Assume L_j is 16 for every SA register. Consider a BISTable circuit having two test sessions. Assume R_1, R_2 and R_3 are the only SA registers having insufficient widths. Suppose the widths of R_1, R_2 and R_3 are 6, 8, and 12, respectively. In test session TS_1 R_1 and R_2 operate as SAs and in test session TS_2 R_2 and R_3 operate as SAs. Let $MS_1(R_1) = \{R_2, R_4, R_5\}$, $MS_1(R_2) = \{R_1, R_4\}$, $MS_2(R_2) = \{R_1, R_3, R_6\}$, and $MS_2(R_3) = \{R_2, R_6, R_7\}$. The widths of registers R_4, ..., R_7 are assumed to be 16 and none of them already operates as a TPG or

a SA.

Clearly $S_1 = \{R_1, R_2\}$, $S_2 = \{R_2, R_3\}$ and $MS = \{R_1, R_2, R_3, R_4, R_5, R_6, R_7\}$. The merging matrices is shown below.

$M_1 =$

	R_1	R_2	R_3	R_4	R_5	R_6	R_7
R_1	0	8	0	16	16	0	0
R_2	6	0	0	16	0	0	0

$M_2 =$

	R_1	R_2	R_3	R_4	R_5	R_6	R_7
R_2	6	0	12	0	0	16	0
R_3	0	8	0	0	0	16	16

Suppose the cost vector is $\{0, 0, 0, 1, 1, 1, 1\}$, i.e. $c_1 = c_2 = c_3 = 0$ and $c_4 = \ldots = c_7 = 1$. Then the ILP formulation is

$$\begin{cases} \text{minimize} & x_4 + x_5 + x_6 + x_7 \\ \text{subject to} & 8 \cdot x_2 + 16 \cdot x_4 + 16 \cdot x_5 + 6 \geq 16 \\ & 6 \cdot x_1 + 16 \cdot x_4 + 8 \geq 16 \\ & 6 \cdot x_1 + 12 \cdot x_3 + 16 \cdot x_6 + 8 \geq 16 \\ & 8 \cdot x_2 + 16 \cdot x_6 + 16 \cdot x_7 + 12 \geq 16 \end{cases}$$

One minimal cost solution is $x_2 = 1$, $x_3 = 1$, and $x_4 = 1$. This leads to combining R_1, R_2, and R_4 as one SA in TS_1 and combining R_2 and R_3 as another SA in TS_2. □

Once a BISTable circuit is obtained an ILP formulation for solving the SA merging problem can be determined as shown above. BITS uses an ILP solver called `lp-solve` to find an optimal solution. Currently *test only* registers are added when no feasible solutions exist in the ILP problem. How to formulate the ILP so that test only registers are included in the optimization process remains an open problem.

8.6 Test Plan Generation

Once a BISTable circuit has been generated and its test schedule determined, a test plan that specifies how to initialize test hardware, perform tests, and observe

the final signature for each test session is required. A test plan is executed by a test controller that generates and applies the required control signals to the BISTable circuit during test. In this section the specification of a test plan is first discussed and then the process of translating a test schedule to a test plan is presented.

8.6.1 Test Plan Specification

A test plan consists of four sections, namely a chain definition section, a register group definition section, a control line definition section, and a test segment definition section[48]. In the chain definition section, boundary scan registers (if any) form one chain and the TPG and SA registers in a BISTable circuit form the second chain. In the register group definition section, each group of registers that are concatenated as a TPG or a SA is defined. When a register operates as a TPG or a SA by itself, it also forms a register group that contains a single register. In some cases a register may appear in two or more groups since it is concatenated with different set of registers to test different kernels. The control line definition section simply contains a list of control lines that are considered to be PIs of the BISTable circuit. The test segment definition section contains a list of test segments. In a test segment one or more kernels can be tested concurrently for the same period of time. Since signatures are shifted out at the end of each test segment, to reduce the test controller complexity the number of test segments should be minimized under the constraint that the total test application time remains the same. This can often be achieved by extending a test by applying arbitrary patterns after it finishes and before new tests are initiated as discussed in the previous chapter. For example, consider the test schedule shown in Figure 7.6(a). The total number of test segments is 6 as illustrated below. The first test session contains t_1, t_5, t_3 and t_4 and is divided into four test segments $TS_{1,1}$, $TS_{1,2}$, $TS_{1,3}$ and $TS_{1,4}$. In $TS_{1,1}$ K_1 and K_5 are tested concurrently for 127 clock cycles. In $TS_{1,2}$ K_1 and K_3 are tested concurrently for 63 clock cycles. In $TS_{1,3}$ K_1 and K_4 are tested concurrently for 31 clock cycles. Finally in $TS_{1,4}$ K_1 alone is tested for 34 clock cycles. Similarly, the second test session is divided into two test segments. When tests t_4 and t_6 are extended to be t'_4 and t'_6 as shown in Figure 7.6(b), the number of test segments is reduced to be 4. Thus leads to two less shift operations.

8.6.2 From Test Schedule to Test Plan

As can be observed from the previous section, test sessions in a test schedule often need to be divided into test segments in specifying a test plan. Some tests may be extended to reduce the number of test segments. The division of test sessions and the extension of tests can be achieved by traversing the TCF that represents the test schedule using Procedure *TCF2TSG*, which is presented below.

Procedure *TCF2TSG*
/* INPUT: A TCF, F, consists of a set of TCTs. */
/* OUTPUT: A list of test segments. */

1. $TSG = \emptyset$.
2. For each TCT in F, let v be the root node of the TCT. Create a test segment $SG = \emptyset$. $genTSG(v, TL(v), SG)$.
3. return TSG.

Procedure *genTSG(v, t, SG)*

1. Create a new test segment SG' that contains the same tests as in SG.
2. Add tests in v to SG.
3. If (v is a leaf node) do
   (a) set the test length of SG to be t.
   (b) append SG to TSG. $LSG = SG$
4. Otherwise do
   (a) for each child node u of v do
      i. $LSG = genTSG(u, TL(u), SG)$
      ii. $t = t - TL(u)$.
   (b) if ($t > 0$), let the test time of LSG be t' and extend the test length of LSG to be $(t + t')$.
5. $SG = SG'$. Return(LSG).

During the execution of Procedure *genTSG*, LSG represents the last test segment considered. In step 4(b) if t is greater than 0, then there are no tests initiated

after LSG for a period of t clock cycles. Therefore applying t more patterns to tests in LSG leads to one less test segment, and thus a simpler test controller as was discussed in Chapter 7.

Example 8.3 Consider the TCF shown in Figure 7.4(a). An execution trace of Procedure *TCF2TSG* on this TCF is shown in Table 8.1. The letters in boldface illustrate the inputs to each recursive call of Procedure *genTSG*. Procedure *TCF2TSG* traverses a TCF in a depth first fashion. That is, nodes in the TCF are visited in the order of n_1, n_2, n_3, n_5 and n_4. As can be observed from the table, three test segments are generated. They are $SG_1 = \{t_1, t_2, t_3\}$, $SG_2 = \{t_1, t_2, t_4, t_6\}$, and $SG_4 = \{t_1, t_2, t_5\}$. The test lengths for SG_1, SG_2, and SG_3 are 2^{l-1}, 2^{l-2}, and 2^{l-2}, respectively. Note that the test length of t_6 is extended from 2^{l-3} to 2^{l-2}. □

8.7 Summary

We considered the design space exploration process in this chapter. An example design space and a representative design space have been illustrated. A branch and bound procedure has been presented to generate a family of representative BISTable circuits, including a minimal test time design, a minimal area overhead design, and a minimal performance degradation design. The incremental test scheduler presented in the previous chapter was integrated in the branch and bound procedure so that unpromising partially specified circuits can be rejected early in the exploration process. When only a minimal area overhead design, a minimal test time design, or a minimal performance degradation design is desired, the design space exploring procedure can execute more efficiently by using a tighter bound on the respective attribute. When a CUC is complex, the complexity of the search space to fully explore its design space may be intractable. In such a case, heuristics are invoked to reduce the complexity. We presented two heuristics in restricting the number of feasible BISTable circuits to be explored. When a BISTable circuit contains SA registers having small widths, the aliasing probability may be too large. In this case SA registers need to be merged with other registers to increase their widths. This has been formulated as a covering problem and presented in this

1st recursion level	2nd recursion level	3rd recursion level
$v = n_1,\ t = 2^{l},\ SG = \varnothing$ $SG' = \varnothing$ $SG = \{t_1, t_2\}$		
genTSG $LSG = \{t_1, t_2, t_3\}$	$v = n_2,\ t = 2^{l-1},\ SG = \{t_1, t_2\}$ $SG' = \{t_1, t_2\}$ $SG = \{t_1, t_2, t_3\}$ set test length of SG as 2^{l-1} $TSG = (\{t_1, t_2, t_3\})$ $LSG = \{t_1, t_2, t_3\}$ $SG = \{t_1, t_2\}$	
$t = 2^{l} - 2^{l-1} = 2^{l-1}$	$v = n_3,\ t = 2^{l-2},\ SG = \{t_1, t_2\}$ $SG' = \{t_1, t_2\}$ $SG = \{t_1, t_2, t_4\}$	
genTSG $LSG = \{t_1, t_2, t_4, t_6\}$	*genTSG* $LSG = \{t_1, t_2, t_4, t_6\}$	$v = n_5,\ t = 2^{l-3},\ SG = \{t_1, t_2, t_4\}$ $SG' = \{t_1, t_2, t_4\}$ $SG = \{t_1, t_2, t_4, t_6\}$ set test length of SG as 2^{l-3} $TSG = (\{t_1, t_2, t_3\},$ $\{t_1, t_2, t_4, t_6\})$ $LSG = \{t_1, t_2, t_4, t_6\}$ $SG = \{t_1, t_2, t_4\}$
	$t = 2^{l-2} - 2^{l-3} = 2^{l-3}$, extend test length of $\{t_1, t_2, t_4, t_6\}$ to be $2^{l-3} + 2^{l-3} = 2^{l-2}$ $LSG = \{t_1, t_2, t_4, t_6\}$ $SG = \{t_1, t_2\}$	
$t = 2^{l-1} - 2^{l-2} = 2^{l-2}$ *genTSG* $LSG = \{t_1, t_2, t_5\}$	$v = n_4,\ t = 2^{l-2},\ SG = \{t_1, t_2\}$ $SG' = \{t_1, t_2\}$ $SG = \{t_1, t_2, t_5\}$ set test length of SG as 2^{l-2} $TSG = (\{t_1, t_2, t_3\},$ $\{t_1, t_2, t_4, t_6\},$ $\{t_1, t_2, t_4\},$ $\{t_1, t_2, t_5\})$ $LSG = \{t_1, t_2, t_5\}$ $SG = \{t_1, t_2\}$	
$t = 2^{l-2} - 2^{l-2} = 0$, $SG = \varnothing$		

Table 8.1: An execution trace of Procedure *TCF2TSG*

chapter. Finally we considered the process of automatically producing a test plan for a BISTable circuit generated by the design space explorer. In a test plan, a test session may need to be divided into test segments. In addition, some tests may be extended to reduce the number of test segments so that the test controller can be simplified. The synthesis of the test controller needs information about these test segments. A procedure to generate the required test segments was presented.

Chapter 9

Selection Problem

9.1 Introduction

In this chapter we consider a generic selection problem where given a set of objects, be it a family of BISTable circuits, a variety of automobiles, or a set of TDMs, a user selects the best object based on his/her requirements and preferences on the attributes associated with the objects. When the number of objects under selection is large, it is often difficult and painstaking for a user to manually select the best object. For such situations, a system that intelligently guides a user in exploring the selection space is desired. A knowledge base that contains the criteria to evaluate objects is required. These selection criteria are put into the knowledge base by a domain expert. Since different users often have different preferences, it is desirable to be able to customize the selection criteria for each individual user so that the knowledge base reflects his/her preferences. However, to properly customize a knowledge base requires thorough knowledge of the selection domain. Since a typical user (non-expert) is often not able to customize a knowledge base and the default knowledge base created by a domain expert is employed. Unfortunately, the default knowledge base does not always accurately reflect the user's preferences. It is desirable to have a machine learning capability in the selection system so that as the selection process proceeds, the knowledge base is automatically modified to reflect a user's preferences. To make an intelligent choice, it is important to specify a set of reasonable initial requirements. When the requirements over-constrain a selection problem, most of the objects under selection tend to be bad compared

with the requirements. Therefore, a selection system should also be able to help a user in specifying initial requirements.

A generic selection problem is first defined in this chapter. An expert selection system to help a user make an intelligent choice is developed. An overview of the selection system is illustrated. We develop a probability model to represent the relationship between attributes associated with objects under selection. The probability model is employed and a procedure developed to help a user specify appropriate initial requirements. The knowledge base that is employed in evaluating objects is then described. The process of customizing a knowledge base is also presented. A machine learning mechanism that dynamically modifies a knowledge base to reflect a user's preferences is then presented. The knowledge base modification problem has been formulated as a linear programming problem that can be solved efficiently using the simplex algorithm[49]. Finally an example execution run to select a BISTable circuit is presented.

9.2 Problem Statement

A selection problem is defined as follows.

Definition 9.1 *Given a set of objects under consideration, $O = \{O_1, O_2, \ldots, O_N\}$. There are M attributes, $\{a_1, a_2, \ldots, a_M\}$ associated with each object. An object, say O_i, consists of a set of attribute values, $\{v_1(i), v_2(i), \ldots, v_M(i)\}$, where $v_j(i)$ is the value of attribute a_j associated with object O_i. Given a requirements r_i for each attribute a_i specified by a user, the* **selection problem** *is to find an object in O that* **satisfies** *all the requirements and is* **better** *than the other objects in O.*

It should be noted that requirements in a selection problem may be changed by a user during the selection process if no objects satisfy the initial requirements. Therefore, the selection problem is more complicated than a constraint satisfaction problem[50] whose constraints are fixed during its search process.

Example 9.1 Consider a simple selection problem of shopping for a new TV set. A person may initially require that the TV set has a big screen, stereo surround sound, life-time warranty, and priced at under \$100. However, no TV sets on the

market satisfy these requirements. If the person has to purchase a TV set, he or she has to modify the requirements so that a least one TV set on the market matches or exceeds the modified requirements. □

As can be seen from the above example, requirements may over-constrain a selection problem, i.e. no object satisfies the requirements. Once requirements are specified, objects under consideration are compared with the requirements, as well as compared with each other. An object O_i is said to *satisfy* the requirements if its attribute values match or exceed the required values. O_i is said to be *better* than a different object O_j if under an evaluation system by either a user or a selection system, O_i is more preferable than O_j. The evaluation system uses a score function that will be discussed in the following sections.

9.3 Selection System Overview

A *Self-Adaptive Expert Selection System(SAESS)* has been implemented to help a user solve a variety of selection problems. An overview of SAESS is illustrated in Figure 9.1.

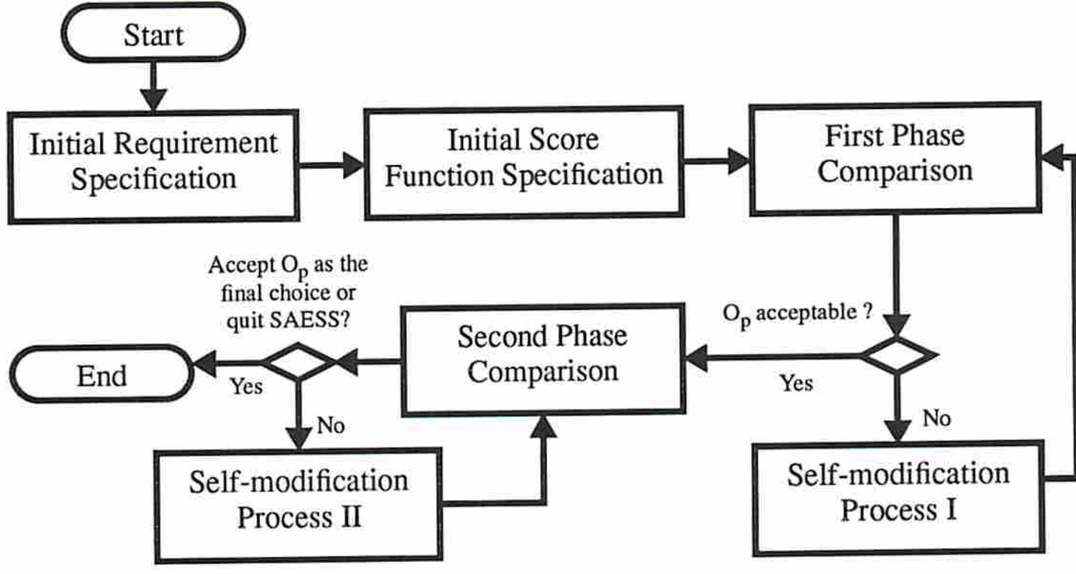


Figure 9.1: An overview of SAESS

In the initial requirement specification process, objects under consideration are first analyzed and the relationships between attributes associated with objects is determined. The information about attribute relationships is employed by SAESS to help a user specify appropriate initial requirements. The initial score function specification process allows a user to customize the knowledge base for the current selection problem. A user may choose not to customize the knowledge base if he or she is not a domain expert. For this case a default knowledge base that contains a pre-defined score function is employed. Once the requirements and an initial score function are obtained, SAESS calculates a score for each object in O and sorts them in descending order. The object having the largest score, say O_p, is recommended by the system as the best selection. If O_p is not acceptable, this implies that either there is no satisfactory objects in O, or the initial score function does not accurately reflect the user's preferences. A self modification process is invoked at this point where SAESS learns information from the user and automatically modifies the score function based on the obtained information. On the other hand, if O_p is acceptable then SAESS goes through a second phase comparison where O_p is compared with the n top ranked objects in O. Currently the value for n is set as 5. If some of these n objects are better than O_p, this also implies that the current score function does not accurately reflect the user's preferences. Hence another self modification process is invoked to modify the score function. The second phase comparison process is repeated until a final choice is made, or the user decides to quit the selection process if no satisfactory object is found.

9.4 Initial Requirement Specification

As was shown in Example 9.1, unrealistic requirements often over-constrain a selection problem that leads to the result that most or all of the objects under consideration are bad compared with the requirements. In this case the object having the largest score is merely the best among a group of " bad" objects. Such an object is not acceptable. To avoid this problem, correlations between attributes based on the attribute values of objects under consideration are studied. A probabilistic

approach is employed and a procedure based on the correlations between attributes is presented that guides a user to specify appropriate initial requirements.

9.4.1 A Probability Model

9.4.1.1 Preliminaries

Let N be the number of objects under consideration and M be the number of attributes associated with each object. For each attribute a_j, an interval $I_j = [min_j, max_j]$ represents the range of possible values for a_j among these N objects. I_j is partitioned into D_j subintervals $I_{j,1}, I_{j,2}, \ldots, I_{j,D_j}$ so that attribute values that belong to the same subinterval have similar values in terms of a_j. The details for partitioning of subintervals will be discussed in the next section. Given an arbitrary object O_i, let $v_j(i)$ denote the value of a_j for O_i. Clearly $v_j(i)$ must belong to a subinterval of I_j. Let X_j denote the second index of this subinterval, where $1 \leq X_j \leq D_j$.

Example 9.2 Let the range of a_j be $I_j = [0, 100]$. Suppose I_j is partitioned into four subintervals as follows.

$$I_{j,1} = [0, 30)$$
$$I_{j,2} = [30, 60)$$
$$I_{j,3} = [60, 70)$$
$$I_{j,4} = [70, 100]$$

If $a_j = 65$, then $X_j = 3$. □

Let the set of objects under consideration be a *sample space*, denoted by Ω. Any set of arbitrarily chosen objects is called an *event*, denoted by ω. A *discrete random variable* X_j can be defined as $X_j : \Omega \rightarrow \{1, 2, \ldots, D_j\}$ and $\mathbf{P}(X_j = x_j)$ is simply the probability of $X_j(\omega) = x_j$, where $\omega \in \Omega$[51]. In other words, given an arbitrary object O_i, the probability that $v_j(i)$ belongs to subinterval I_{j,x_j} is $\mathbf{P}(X_j = x_j)$.

Let $S(X_j = x_j)$ be the set of objects where the value of a_j belongs to subinterval I_{j,x_j}. Then a discrete random variable X_j can be defined for a_j and $\mathbf{P}(X_j = x_j)$ can be calculated as

$$\mathbf{P}(X_j = x_j) = \frac{|S(X_j = x_j)|}{N}.$$

9.4.1.2 Correlation between random variables

Consider two attributes a_j and a_k. The discrete random variables X_j and X_k corresponding to a_j and a_k, respectively, are defined as

$$X_j : \Omega \rightarrow \{1, 2, \ldots, D_j\}$$
$$X_k : \Omega \rightarrow \{1, 2, \ldots, D_k\}.$$

The probabilities $\mathbf{P}(X_j = x_j)$ and $\mathbf{P}(X_k = x_k)$ can be calculated independently as

$$\mathbf{P}(X_j = x_j) = \frac{|S(X_j = x_j)|}{N} \quad \text{and}$$
$$\mathbf{P}(X_k = x_k) = \frac{|S(X_k = x_k)|}{N}.$$

where $1 \leq x_j \leq D_j$ and $1 \leq x_k \leq D_k$.

A *joint mass function* of X_j and X_k[51] is defined as

$$\begin{aligned} f_{X_j, X_k}(x_j, x_k) &= \mathbf{P}(X_j = x_j \text{ and } X_k = x_k) \\ &= \frac{|S(X_j = x_j) \cap S(X_k = x_k)|}{N}. \end{aligned}$$

The *covariance* of X_j and X_k[51] is defined as

$$\begin{aligned} & cov(X_j, X_k) \\ = \; & \mathbf{E}(X_j, X_k) - \mathbf{E}(X_j)\mathbf{E}(X_k) \\ = \; & \sum_{x_j, x_k} x_j x_k f_{X_j, X_k}(x_j, x_k) - \sum_{x_j} x_j \mathbf{P}(X_j = x_j) \sum_{x_k} x_k \mathbf{P}(X_k = x_k). \end{aligned}$$

And finally the *correlation* of X_j and X_k[51] is

$$\rho(X_j, X_k) = \frac{cov(X_j, X_k)}{(Var(X_j) \cdot Var(X_k))^{1/2}}$$

where $Var(X_j) = \mathbf{E}(X_j^2) - (\mathbf{E}(X_j))^2$ is the *variance* of X_j, and $Var(X_k) = \mathbf{E}(X_k^2) - (\mathbf{E}(X_k))^2$ is the *variance* of X_k.

The correlation between two random variables has the interesting property that $0 \leq |\rho(X_j, X_k)| \leq 1$. When $|\rho(X_j, X_k)| = 1$, there exist real numbers a, b so that $X_j = aX_k + b$. When $\rho(X_j, X_k) = 0$, X_j and X_k are said to be *uncorrelated*.

As $|\rho(X_j, X_k)|$ becomes closer to 1, the dependency between X_j and X_k becomes stronger.

9.4.1.3 Correlation between attributes

Clearly we can define a discrete random variable for each attribute as described in the previous sections. The correlation between every pair of discrete random variables X_j and X_k can also be calculated as illustrated above.

Definition 9.2 *A pair of attributes a_j and a_k are said to be* **strongly correlated** *if and only if $|\rho(X_j, X_k)| \geq \gamma$, where γ is a pre-defined threshold value.*

The implication of two attributes being strongly correlated is that the decision on the requirement for one attribute will greatly affect the decision on the requirement for the other attribute. Therefore for each attribute a_j, a *correlated attribute set* C_j can be defined as the attributes other than a_j that are strongly correlated with a_j. The decision on the requirement for an arbitrary attribute in C_j affects the decisions on requirement for a_j. Conversely, the decision on the requirement for a_j also affects the decision on the requirement for any attribute in C_j. Given the set of objects under consideration, the correlated attribute set for each attribute can be identified *a priori*. This information can be employed to guide a user in specifying appropriate initial requirements.

Example 9.3 Consider the automobile selection problem presented in [52]. There are 49 automobiles under consideration and 10 attributes are associated with each automobile as show in Table 9.1. The number of subintervals in each attribute is also shown in Table 9.1. Table 9.2 lists the correlation of each pair of attributes. The threshold value(γ) is 0.7 in this example. Boldface numbers in Table 9.2 indicate the strongly correlated attribute pairs.

From the strongly correlated attribute pairs as shown in Table 9.2, the correlated attribute set for each attribute can easily be derived. The correlated attribute sets are enumerated below.

C_{Price} = {*MPG(city), MPG(highway), HP, Fuel Cap., Quality*}
$C_{MPG(city)}$ = {*Price, MPG(highway), HP, Fuel Cap.*}
$C_{MPG(highway)}$ = {*Price, MPG(city), Fuel Cap.*}

Attribute	Price	MPG (city)	MPG (highway)	HP	Seating Cap.	Cargo Cap.	Fuel Cap.	Resale Value	Reliability	Quality
D_j	7	4	4	5	3	5	3	4	3	4

Table 9.1: Number of subintervals in each attribute

$|\rho(X_{j1}, X_{j2})|$

	a_k								
a_j	MPG (city)	MPG (highway)	HP	Seating Cap.	Cargo Cap.	Fuel Cap.	Resale Value	Reliability	Quality
Price	**0.80**	**0.77**	**0.78**	0.22	0.13	**0.81**	0.65	0.29	**0.86**
MPG(city)		**0.87**	**0.73**	0.10	0.36	**0.82**	0.49	0.07	0.67
MPG(highway)			0.63	0.21	0.35	**0.74**	0.52	0.11	0.59
HP				0.15	0.08	**0.77**	0.54	0.27	**0.71**
Seating Cap.					0.23	0.36	0.40	0.18	0.18
Cargo Cap.						0.16	0.02	0.12	0.07
Fuel Cap.							0.65	0.26	**0.75**
Resale Value								0.46	0.60
Reliability									0.35

Table 9.2: Absolute values of correlations

$$C_{HP} = \{Price,\ MPG(city),\ Fuel\ Cap.,\ Quality\}$$
$$C_{Fuel\ Cap.} = \{Price,\ MPG(city),\ MPG(highway),\ HP,\ Quality\}$$
$$C_{Quality} = \{Price,\ HP,\ Fuel\ Cap.\}$$
$$C_{Seating\ Cap.} = C_{Cargo\ Cap.} = C_{Resale\ Value} = C_{Reliability} = \emptyset$$

□

9.4.1.4 Conditional Probabilities and Likelihood Subinterval Set

Suppose a_j and a_k are a pair of strongly correlated attributes. Assume that the requirement for a_j, r_j, has been specified and belongs to subinterval $I_{j,s}$. The problem of predicting the subintervals of I_k to which attribute values of objects that satisfy r_j belong is considered in this section. A *conditional probability* of X_k given $X_j = s$ is defined as

$$\begin{aligned} \mathbf{P}(X_k = x_k | X_j = s) &= \text{Probability of } X_k = x_k, \text{ given } X_j = s \\ &= \frac{|S(X_j = s) \cap S(X_k = x_k)|}{|S(X_j = s)|} \end{aligned}$$

The *conditional expectation* of X_k, given $X_j = s$, can thus be calculated as

$$\mathbf{E}(X_k|X_j = s) = \sum_{x_k} x_k \cdot \mathbf{P}(X_k = x_k|X_j = s)$$

Note that $\mathbf{E}(X_k|X_j = s)$ is a random variable[51]. Intuitively, $\mathbf{E}(X_k|X_j = s)$ suggests the expected subinterval to which a_k most likely belongs. Unfortunately, a conditional expectation can assume any real value in $[1, D_k]$, thus it may not be a *discrete* random variable. To predict the most appropriate subintervals to which r_k belong, the following definition is introduced.

Definition 9.3 *A* **likelihood subinterval set** *of a_k given $X_j = s$, denoted by $\mathbf{L}(X_k|X_j = s)$, is a subset of $\{1, 2, \ldots, D_k\}$ so that the attribute value in terms of a_k of an object that satisfies r_j belongs to a subinterval $I_{k,t}$, where $t \in \mathbf{L}(X_k|X_j = s)$.*

Clearly $\{1, 2, \ldots, D_k\}$ is the maximal $L(X_k|X_j = s)$. In general, $L(X_k|X_j = s) \subset \{1, 2, \ldots, D_j\}$ if a_j and a_k are strongly correlated. In practice, the objective of predicting the appropriate subinterval(s) for r_k, given r_j, is to find a subset of $\{1, 2, \ldots, D_K\}$ that covers most of the objects that satisfy r_j. Let $\mathbf{L}_m(X_k|X_k = s)$ denote such a subset. A straightforward approach is to first calculate $\mathbf{E}(X_k|X_j = s)$, and then let $\mathbf{L}_m(X_k|X_j = s)$ be $\{\lfloor \mathbf{E}(X_k|X_j = s) \rfloor, \lceil \mathbf{E}(X_k|X_j = s) \rceil\}$.

Example 9.4 Consider the same selection problem as in Example 9.3. It can be observed from Table 9.2 that attributes $MPG(city)$ and HP are strongly correlated. Let a_1 denote $MPG(city)$ and a_2 denote HP. Suppose the requirement for a_1 has been specified and belongs to subinterval $I_{1,4}$, thus $X_1 = 4$. The conditional probabilities for the automobiles under consideration and the conditional expectation of X_2 are summarized below.

$$\begin{aligned}
&\mathbf{P}(X_2 = 1|X_1 = 4) = 0.56 \\
&\mathbf{P}(X_2 = 2|X_1 = 4) = 0.44 \\
&\mathbf{P}(X_2 = 3|X_1 = 4) = 0.00 \\
&\mathbf{P}(X_2 = 4|X_1 = 4) = 0.00 \\
&\mathbf{P}(X_2 = 5|X_1 = 4) = 0.00 \\
&\mathbf{E}(X_2|X_1 = 4) = 1 \times 0.56 + 2 \times 0.44 = 1.44
\end{aligned}$$

Clearly $\mathbf{L}_m(X_2|X_1 = 4)$ is $\{1, 2\}$. This means that if we specify a requirement for attribute HP that belongs to either $I_{2,1}$ or $I_{2,2}$, it is likely that there exist certain objects satisfying both requirements. This can easily be validated by checking the conditional probabilities shown above since $\mathbf{P}(X_2 = 1|X_1 = 4) + \mathbf{P}(X_2 = 2|X_1 = 4) = 0.56 + 0.44 = 1$. In other words, every object whose attribute value of $MPG(city)$ belongs to subinterval $I_{1,4}$ has an attribute value of HP belonging to either $I_{2,1}$ or $I_{2,2}$. □

Unfortunately, this simple approach does not always lead to a good prediction as can be illustrated by the example below.

Example 9.5 Consider the same selection problem as in Example 9.4. Suppose r_1 has been specified and $X_1 = 1$. Then the conditional probabilities for the automobiles under consideration and the conditional expectation of X_2 are summarized below.

$$\mathbf{P}(X_2 = 1|X_1 = 1) = 0.08$$
$$\mathbf{P}(X_2 = 2|X_1 = 1) = 0.00$$
$$\mathbf{P}(X_2 = 3|X_1 = 1) = 0.16$$
$$\mathbf{P}(X_2 = 4|X_1 = 1) = 0.38$$
$$\mathbf{P}(X_2 = 5|X_1 = 1) = 0.38$$
$$\mathbf{E}(X_2|X_1 = 4) = 1 \times 0.08 + 3 \times 0.16 + 4 \times 0.38 + 5 \times 0.38 = 3.98$$

Therefore $\mathbf{L}_m(X_2|X_1 = 1)$ is $\{3, 4\}$. However, by examining the conditional probabilities it can be seen that among the objects whose attribute values of $MPG(city)$ belong to $X_{1,1}$, 46% of them have attribute value of HP not belonging to either $I_{2,3}$ or $I_{2,4}$. □

An alternative approach to generate $\mathbf{L}_m(X_k|X_j = s)$ is to employ the information provided by conditional probabilities. Note that if the distributions of X_j and X_k are uniform and X_j and X_k are uncorrelated, then the distribution of $\mathbf{P}(X_k|X_j = s)$ is also uniform (i.e. $\mathbf{P}(X_k = x_k|X_j = s) \approx 1/D_k$, $1 \leq x_k \leq D_k$). Due to the correlation of X_j and X_k, the distribution of conditional probabilities will be biased. Consequently, some of the conditional probabilities

are greater than $1/D_k$. Based on this observation, $\mathbf{L}_m(X_k|X_j = s)$ can be calculated as $\{t \in \{1, 2, \ldots, D_k\} | \mathbf{P}(X_k = t|X_j = s) > 1/D_k\}$. Using this approach, $\mathbf{L}_m(X_2|X_1 = 1)$ in the above example is $\{4, 5\}$ that covers 76% of the objects that satisfy r_1. This approach will be employed in the procedure described in the next section for specifying initial requirements.

Conditional probabilities of more than two random variables (i.e. probabilities of one variable, given the values of all the other variables) can be calculated in a similar manner as in the case of two random variables. Therefore the above approach can easily be extended to predict the appropriate subintervals for a requirement, given the requirements for two or more strongly correlated attributes.

9.4.2 Procedure for Initial Requirement Specification

As pointed out earlier, attributes in a selection problem are often of unequal importance. Therefore, a relative weight can be assigned to each attribute by a user to represent its importance. Let attributes be sorted by their relative weights in descending order and denoted as $\{a_1, a_2, \ldots, a_M\}$.

In some cases an attribute is not strongly correlated to any other attribute, thus the requirement for this attribute can be specified independently of other attributes. On the other hand, if an attribute, say a_j, is strongly correlated to a set of attributes A, the requirement specified for a_j is related to the specification of requirements for attributes in A. Therefore in specifying the requirement for a_j, attributes in A whose requirements have been assigned should be taken into account. The procedure to help a user specify initial requirements, known as *IRS*, is presented below. We assume that the correlated attribute set for an attribute a_j, denoted by C_j, has been obtained.

Procedure *IRS*
/* INPUT: A set of attributes, $\{a_1, a_2, \ldots, a_M\}$, sorted by relative weights in descending order. */
/* OUTPUT: A requirement for each attribute. */

1. For $j = 1$ to M do the following.
2. If $(C_j \neq \emptyset)$ do
   (a) let $C'_j = \{k \in C_j \mid \text{requirement for } a_k \text{ has been specified }\}$.

(b) if ($C'_j = \emptyset$) then acquire the requirement for a_j from user.

(c) otherwise do

i. let I_{k,s_k} be the subinterval to which r_k belongs, for $k \in C'_j$.

ii. calculate the conditional probabilities $\mathbf{P}(X_j = x_j | \bigcap_{k \in C'_j} X_k = s_k)$ for $1 \leq x_j \leq D_j$.

iii. generate $\mathbf{L}_m(X_j | \bigcap_{k \in C'_j} X_k = s_k)$, abbreviated as L_j, as described in the previous section.

iv. determine an interval of attribute values $I_j = \bigcup_{k \in L_j} I_{j,k}$.

v. display I_j to user as the recommended range of the requirement for a_j.

vi. acquire the requirement for a_j from user.

3. Otherwise acquire the requirement for a_j from user.

Clearly Procedure *IRS* is a polynomial time procedure. By using this procedure and following the recommendation, a user can specify a set of reasonable initial requirements.

9.5 Evaluation of Objects and the Score Function

Each object under consideration consists of a set of attribute values for the attributes associated with the object, where the attributes may be of different units. To make an objective comparison between two objects or between an object and the requirements, a score function that converts incommensurable attribute values to a common measure, called a *score*, can be employed. We adopt a similar score function as in PLA-ESS[29]. Namely, a score function is a weighted combination of a set of *penalty-credit functions(PCF)* (to be described). Once a score function is defined, a score can be calculated for each object under consideration. Objects can then be compared by their scores. The problem of customizing PCFs will also be described in this section.

9.5.1 Score Function

A PCF is associated with each attribute that converts an attribute value to a unitless measure called a *PC value*. A PCF associated with attribute a_j, denoted by PCF_j, is characterized by a set of $(D_j + 1)$ break points $\{P_{j,0}, P_{j,1}, \ldots, P_{j,D_j}\}$. Each break point $P_{j,k}$ is an x-y pair $(x_{j,k}, y_{j,k})$, where $0 \leq k \leq D_j$. Between two consecutive break points, say $P_{j,k-1}$ and $P_{j,k}$, is a linear function

$$y = a \times (x - x_{j,k-1}) + b$$

where $a = (y_{j,k} - y_{j,k-1})/(x_{j,k} - x_{j,k-1})$ and $b = y_{j,k-1}$, for all (x, y) between these two break points. For convenience, an *x-displacement* of break point $P_{j,k}$ with respect to break point $P_{j,k-1}$, denoted by $\delta x_{j,k}$, is defined as $x_{j,k} - x_{j,k-1}$. Similarly a *y-displacement* of break point $P_{j,k}$ with respect to break point $P_{j,k-1}$, denoted by $\delta y_{j,k}$, is defined as $y_{j,k} - y_{j,k-1}$.

An example PCF is shown in Figure 9.2 where D_j is assumed to be 4. The x-axis represents attribute values in terms of a_j and the y-axis represents PC values. As can be observed, a PCF is a collection of linear functions that approximates a more general non-linear function. Recall that the attribute value of object O_i in terms of a_j is denoted as $v_j(i)$. A *normalized* PC value of $v_j(i)$ is defined as the difference between the PC value of $v_j(i)$ and the PC value of r_j. Usually if $v_j(i)$ is better than r_j then its normalized PC value is positive (a credit); if $v_j(i)$ is worse than r_j then its normalized PC value is negative (a penalty). The bounds for attribute values can be determined by examining the attribute values of objects under consideration. The bounds for attribute values uniquely determine the interval I_j employed in the previous section. In addition, the $(D_j + 1)$ break points partition I_j into D_j subintervals. The PCF customization process essentially determines these break points. This process will be considered in the next section.

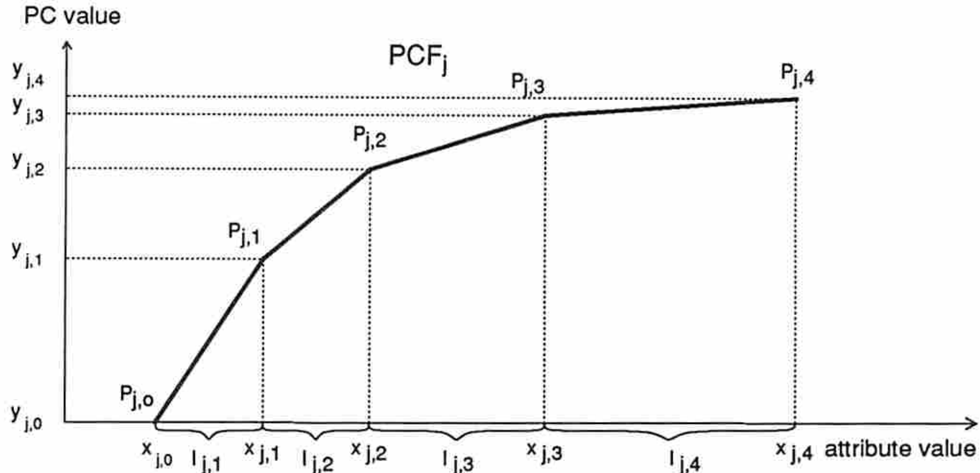


Figure 9.2: An example PCF

Let I_{j,X_j} be the subinterval to which r_j belongs and $I_{j,X_j(i)}$ be the subinterval to which $v_j(i)$ belongs. The normalized PC value of $v_j(i)$, denoted by $PC_j(i)$, is

$$
\begin{aligned}
PC_j(i) = \; & \Big(\sum_{k=1}^{X_j(i)-1} \delta y_{j,k} + (v_j(i) - x_{j,X_j(i)-1}) \times \frac{\delta y_{j,X_j(i)}}{\delta x_{j,X_j(i)}}\Big) \\
& -\Big(\sum_{k=1}^{X_j-1} \delta y_{j,k} + (r_j - x_{j,X_j-1}) \times \frac{\delta y_{j,X_j}}{\delta x_{j,X_j}}\Big).
\end{aligned}
$$

The formula in the first parenthesis calculates the PC value of $v_j(i)$, and the formula in the second parenthesis calculates the PC value of r_j. This can be illustrated by the following example.

Example 9.6 Consider the PCF shown in Figure 9.2. Suppose the requirement for a_j and an attribute value of object O_i are given as shown in Figure 9.3. Clearly $v_j(i)$ belongs to the subinterval $I_{j,3}$ and r_j belongs to the subinterval $I_{j,2}$. The PC value of $v_j(i)$ is

$$\delta y_{j,1} + \delta y_{j,2} + (v_j(i) - x_{j,2}) \times \frac{\delta y_{j,3}}{\delta x_{j,3}}$$

and the PC value of r_j is

$$\delta y_{j,1} + (r_j - x_{j,1}) \times \frac{\delta y_{j,2}}{\delta x_{j,2}}$$

as shown in the first and second parentheses, respectively, of the formula in Figure 9.3. Thus $PC_j(i)$ is the difference between these two values. □

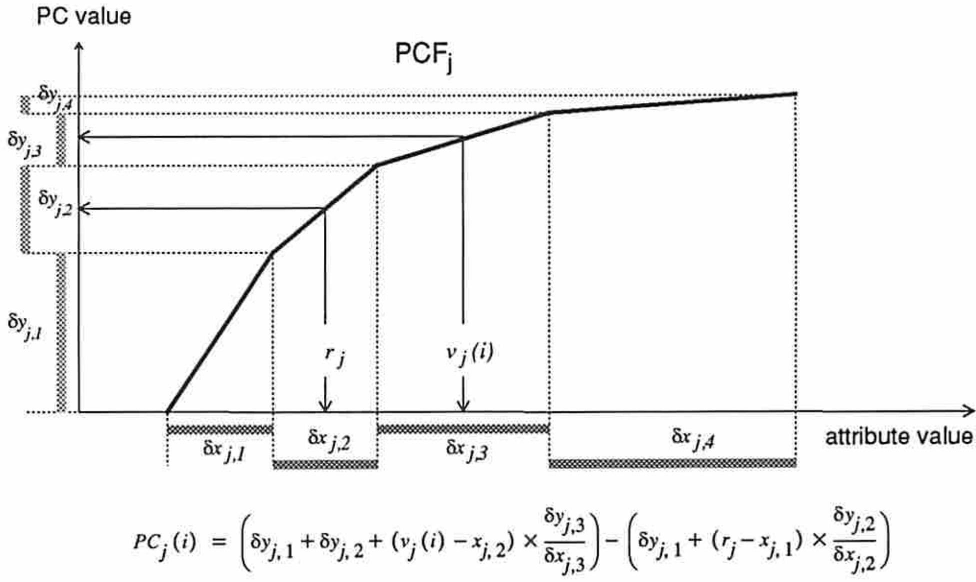


Figure 9.3: Illustration of calculating a normalized PC value

The formula for a normalized PC value can be simplified by introducing the following vectors and vector operations. Let the vector $\delta\mathbf{y}_j$ be $(\delta y_{j,1}, \delta y_{j,2}, \ldots, \delta y_{j,D_j})$. $\delta\mathbf{y}_j$ is called the *parameter* vector for attribute a_j. Let the vector $\mathbf{c}_j$ be

$$(\underbrace{1,\ldots,1}_{X_j(i)-1}, \frac{v_j(i) - x_{j,X_j(i)-1}}{x_{j,X_j(i)}}, \underbrace{0,\ldots,0}_{D_j-X_j(i)}) - (\underbrace{1,\ldots,1}_{X_j-1}, \frac{r_j - x_{j,X_j-1}}{x_{j,X_j}}, \underbrace{0,\ldots,0}_{D_j-X_j}).$$

$\mathbf{c}_j$ is called the *coefficient vector* for $v_j(i)$. Note that both vectors contain D_j elements. It can be observed that $PC_j(i)$ is the vector inner product $\mathbf{c}_j(i) \bullet \delta\mathbf{y}_j$.

As mentioned earlier, a score function is a weighted combination of PC functions, namely the PC function for each attribute is weighted by a number associated with the attribute. This number is called a *relative weight* associated with attribute a_j, denoted by w_j, which can be specified by a user to represent the relative importance of a_j. An attribute having a greater relative weight has more impact on

the score than an attribute having a less relative weight. Once the relative weights associated with attributes are obtained, the score function is

$$
\begin{aligned}
S(i) &= \sum_{j=1}^{M} w_j \times PC_j(i) \\
&= \sum_{j=1}^{M} w_j \times \mathbf{c}_j(i) \bullet \delta \mathbf{y}_j \\
&= \sum_{j=1}^{M} \mathbf{c}_j(i) \bullet w_j \delta \mathbf{y}_j \\
&= \sum_{j=1}^{M} \mathbf{c}_j(i) \bullet \mathbf{u}_j
\end{aligned}
$$

where $\mathbf{u}_j = w_j \delta \mathbf{y}_j$ is called the *weighted parameter* vector for a_j.

Given a score function thus defined, object O_p is said to be *better* than object O_q if $S(p) > S(q)$. In addition, an object O_i is said to be *unacceptable* if and only if there exists an attribute a_j for which $PC_j(i)$ is negative. Note that the requirements can be modified by a user during a selection process. Thus an unacceptable object may become acceptable once requirements are modified. One objective of the selection problem is to both identify an object O_i and a requirement vector $\mathbf{r} = (r_1, r_2, \ldots, r_M)$ so that O_i is acceptable and better than all the other objects (i.e. $S(i) \geq S(j)\ \forall j \neq i$).

9.5.2 Customizing PCFs

Two parameters determine a PCF, namely the number of subintervals and the set of break points. Recall that a PCF is a collection of linear functions as illustrated in Figure 9.2 that approximates a non-linear function. As the number of subintervals increases, a PCF can more accurately approximates this function. On the other hand, as the number of subintervals in each PCF increases, so does the complexity of the score function and the amount of computation. Therefore a balance between accuracy and computational complexity should be obtained in determining the number of subintervals.

Note that the number of subintervals for each attribute need not be the same. For example, consider the automobile selection problem presented in [52]. The

attribute *price* usually has a wide range from under \$10,000 to over \$100,000. The attribute *seating cap.* has a limited range, say from 2 persons to 6 persons. Consequently, there should be more subintervals for attribute *price* than for attribute *seating cap.*. Currently SAESS helps a user make the decisions on the number of subintervals as shown in the flowchart in Figure 9.4. A *unit*, μ_j, is acquired from a user for each attribute. μ_j represents the smallest range of attribute values that the user considers as significant. For example, in the automobile selection problem a user may specify a unit of \$1,000 for the attribute *price* and a unit of one person for the attribute *seating cap.*. In this case, two cars having the prices \$11,500 and \$12,000 are considered as of the same price.

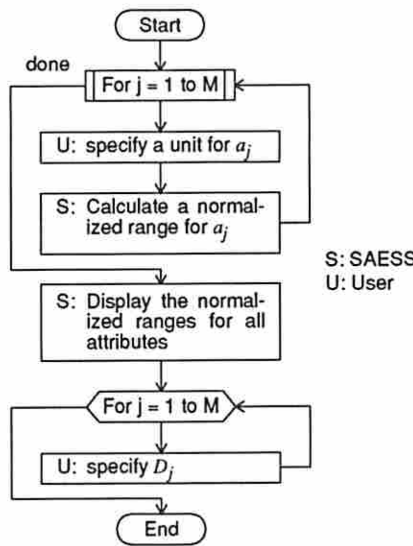


Figure 9.4: Process of determining number of subintervals

Once a unit for each attribute is given, say μ_j, a *normalized range*, Υ_j, for the attribute can be calculated as follows.

$$\Upsilon_j = \frac{max_j - min_j}{\mu_j}$$

where max_j and min_j are the maximal and minimal attribute values, respectively, in terms of a_j among the objects under consideration. The normalized range for each attribute roughly indicates the variations of attribute values. An attribute

having a greater normalized range should usually have more subintervals than an attribute having a smaller normalized range. The normalized ranges for all attributes are then displayed to a user as a guideline in determining the number of subintervals for each attribute.

Once D_j, the number of subintervals for attribute a_j, has been determined, the set of (D_j+1) break points can be specified. Clearly $x_{j,0} = min_j$ and $x_{j,D_j} = max_j$. A straightforward approach in specifying the intermediate break points along the x-axis is to assign them evenly between min_j and max_j. That is,

$$x_{j,k} = x_{j,0} + \Delta x_j \times k$$

where $1 \leq k < D_j$ and $\Delta x_j = (max_j - min_j)/D_j$. This approach may be reasonable when the distribution of attribute values is uniform. However, in general the distribution of attribute values is not uniform, thus a more complex approach to assign the intermediate break points along the x-axis needs to be employed. One such approach is to divide I_j (i.e. $[x_{j,0}, x_{j,D_j}]$) into D_j subintervals so that each subinterval contains approximately the same number of objects. Clearly when the distribution of attribute values is uniform, this approach leads to similar break points as those obtained by using the simple approach. When the distribution is not uniform, regions of attribute values that are dense (having more objects) tend to have more break points than regions of attribute values that are sparse. Finally, for each break point that has been assigned a value on the x-axis, the user specifies its PC value on the y-axis.

As was mentioned earlier, a default knowledge base that contains the break points and their associated PC-values specified by a domain expert is provided to a user. Thus he/she can skip the complex process of customizing the knowledge base.

9.6 First Phase Comparison and Self Modification Process I

By using the score function defined in the previous section we can form a total ordering relation on the objects under consideration. The score function is an evaluation of each object with respect to the requirements. When a score function appropriately reflects a user's preferences, the object with the largest score should be the best choice. Therefore, the first phase comparison orders objects based on this assumption. The object having the largest score, say O_p, is recommended to the user as the best choice. However, a score function may not properly reflect a user's preferences. In which case O_p may not be acceptable, or the best object may not have the largest score. Therefore, parameters in the score function need to be modified based on the user's preferences. When O_p is not acceptable, the first self modification process is employed as described below.

For SAESS to automatically modify its parameters, a user needs to provide certain information. First, since O_p is not acceptable, there must be certain attribute values of O_p that are not satisfactory. SAESS acquires the attributes $a_j \in \{1, 2, \ldots, M\}$ of which $v_j(p)$ is not acceptable. The index set of attributes, namely $\{1, 2, \ldots, M\}$, is thus partitioned into two subsets J and $\overline{J}$ so that if $j \in J$ then $v_j(p)$ is not acceptable and if $j \in \overline{J}$ then $v_j(p)$ is acceptable. Secondly, it is useful if the user can identify objects that are more preferable or less preferable to O_p. Let I and $\bar{I}$ be two disjoint subsets of the index set of objects, namely $\{1, 2, \ldots, N\}$, so that object O_q is more preferable to O_p if $q \in I$ and less preferable to O_p if $q \in \bar{I}$.

If the user is able to provide the above information, then the score function modification process can be formulated as follows. Define $\mathbf{\Delta u}_j$ for $j = 1$ to M

as the modification vectors, where $\mathbf{\Delta u}_j = (\Delta u_{j,1}, \Delta u_{j,2}, \ldots, \Delta u_{j,D_j})$. A modified score function can thus be derived as

$$\begin{aligned} S'(i) &= \sum_{j=1}^{M} \mathbf{c}_j(i) \bullet (\mathbf{u}_j + \mathbf{\Delta u}_j) \\ &= \sum_{j=1}^{M} (\mathbf{c}_j(i) \bullet \mathbf{u}_j + \mathbf{c}_j(i) \bullet \mathbf{\Delta u}_j) \\ &= S(i) + \sum_{j=1}^{M} \mathbf{c}_j(i) \bullet \mathbf{\Delta u}_j. \end{aligned}$$

Since O_q is more preferable to O_p if $q \in I$, the modified score function should reflect this fact. This leads to the constraint that $S'(q) > S'(p)$ for $q \in I$. Similarly, O_q is less preferable to O_p if $q \in \bar{I}$. This leads to the constraint that $S'(q) < S'(p)$ for $q \in \bar{I}$.

The first constraint can be expressed as

$$\begin{aligned} & S'(q) > S'(p) && \forall q \in I \\ \Rightarrow\ & S(q) + \sum_{j=1}^{M} \mathbf{c}_j(q) \bullet \mathbf{\Delta u}_j > S(p) + \sum_{j=1}^{M} \mathbf{c}_j(p) \bullet \mathbf{\Delta u}_j && \forall q \in I \\ \Rightarrow\ & \sum_{j=1}^{M} (\mathbf{c}_j(q) - \mathbf{c}_j(p)) \bullet \mathbf{\Delta u}_j > S(p) - S(q) && \forall q \in I. \end{aligned}$$

Similarly, the second constraint can be expressed as

$$\begin{aligned} & S'q < S'p && \forall q \in \bar{I} \\ \Rightarrow\ & \sum_{j=1}^{M} (\mathbf{c}_j(q) - \mathbf{c}_j(p)) \bullet \mathbf{\Delta u}_j < S(p) - S(q) && \forall q \in \bar{I}. \end{aligned}$$

The objective of the score function modification process is to gradually modify a score function so that the above two constraints are both satisfied. Therefore, an objective function can be derived as

$$\text{minimize } \Delta = \sum_{j=1}^{M} \sum_{k=1}^{D_j} |\Delta u_{j,k}|.$$

This problem can thus be formulated as a mathematical programming problem below.

$$
\begin{cases}
\text{minimize} & \Delta = \sum_{j=1}^{M} \sum_{k=1}^{D_j} |\Delta u_{j,k}| \\
\text{subject to} & \sum_{j=1}^{M} (\mathbf{c}_j(q) - \mathbf{c}_j(p)) \bullet \mathbf{\Delta u}_j > S(p) - S(q) \qquad \forall q \in I \\
& \sum_{j=1}^{M} (\mathbf{c}_j(q) - \mathbf{c}_j(p)) \bullet \mathbf{\Delta u}_j < S(p) - S(q) \qquad \forall q \in \bar{I}
\end{cases}
$$

There does not exists an efficient algorithm to solve a general mathematical programming problem. However, notice that in this formulation the constraints are linear and the objective function is quadratic. If the sign for each $\Delta u_{j,k}$ (namely greater or less than 0) can be determined *a priori*, the objective function can be forced to be a linear function. In such a case a linear programming(LP) formulation can be derived and an efficient algorithm exists to solve a LP problem. Heuristics are employed to determine the sign of each $\Delta u_{j,k}$ as described below. First, since $v_j(p)$ is not acceptable for $j \in J$, if O_q is more preferable to O_p then it is more likely that $v_j(q) \geq v_j(p)$. On the other hand, it is more likely that $v_j(q) \leq v_j(p)$ for $j \in \bar{J}$. Restricting $\delta u_{j,k}$ for $j \in J$ to be non-negative and $\delta u_{j,k}$ for $j \in \bar{J}$ to be non-positive tends to satisfy the constraints that $S'(q) > S'(p)$ for $q \in I$ and $S'(q) < S'(p)$ for $q \in \bar{I}$. This can be illustrated by the effects of modifying a weighted PCF as shown in Figure 9.5. In Figure 9.5 the original and modified weighted PCFs for a_j are shown. We assume that $v_j(p)$ is not acceptable and the attribute value of O_q, where $q \in I$, in terms of a_j is superior to $v_j(p)$. According to the above heuristics, each $\Delta u_{j,k}$ $(0 \leq k \leq D_j)$ is forced to be non-negative. As can be seen from the figure, this leads to an increase in the weighted PC value of O_q with respect to the weighted PC value of O_p in terms of a_j. Therefore, the modified score of O_q may become greater than that of O_p.

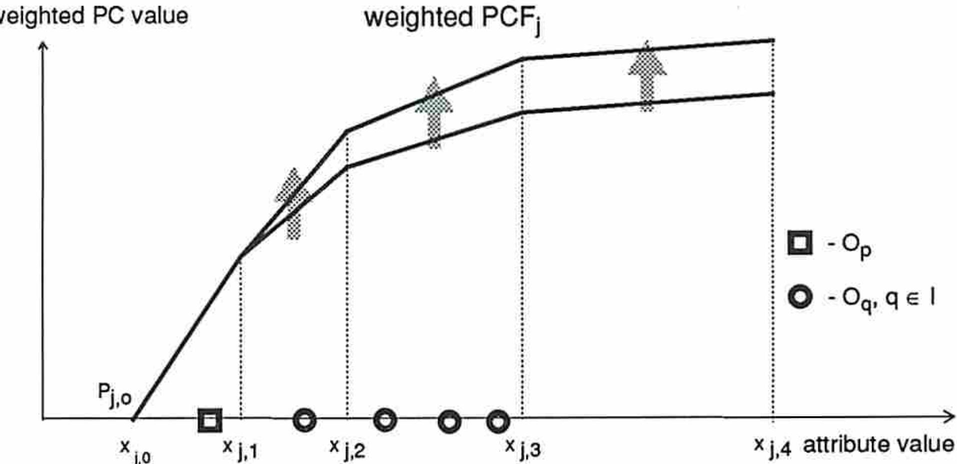


Figure 9.5: Effects of score function modification

By applying the above restrictions a LP formulation for the score function modification problem can be derived as shown below.

$$\begin{cases} \text{minimize} & \Delta = \sum_{j\in J}\sum_{k=1}^{D_j} \Delta u_{j,k} - \sum_{j\in \bar{J}}\sum_{k=1}^{D_j} \Delta u_{j,k} & \\ \text{subject to} & \sum_{j=1}^{M}(\mathbf{c}_j(q) - \mathbf{c}_j(p)) \bullet \mathbf{\Delta u}_j > S(p) - S(q) & \forall q \in I \\ & \sum_{j=1}^{M}(\mathbf{c}_j(q) - \mathbf{c}_j(p)) \bullet \mathbf{\Delta u}_j < S(p) - S(q) & \forall q \in \bar{I} \\ & \Delta u_{j,k} \geq 0 & \forall j \in J, 1 \leq k \leq D_j \\ & \Delta u_{j,k} \leq 0 & \forall j \in \bar{J}, 1 \leq k \leq D_j \end{cases}$$

A revised Simplex algorithm[49] is then employed to solve the LP formulation in $O(n^2)$ time, where n is the number of parameters (i.e. Δu terms) in the formulation.

To help a user identify the index sets I and $\bar{I}$, SAESS employs a user-friendly approach where objects other than O_p are compared with O_p side-by-side in terms of every attribute using a bar-chart diagram. An example bar-chart diagram is shown in Figure 9.6. Note that the attributes in the bar-chart diagram are ordered based on their relative weights. For each object O_q, the user can then determine whether it is more preferable to O_p by observing the bar-chart diagram. Note that

for a complex selection problem, the number of objects under consideration may be extremely large. This makes comparing every object with O_p tedious and time consuming. Therefore, only the n top ranked objects, based on the score function, are employed in the above comparison process. Currently n is set as 5. From these n comparisons, SAESS can normally extract the desired information, namely index sets I, $\bar{I}$, J, and $\bar{J}$, for the score function modification process. If after n comparisons there are still no preferable objects, the next n ranked objects are compared against O_p. This process is repeated until preferable objects are found or the user decides to quit the process.

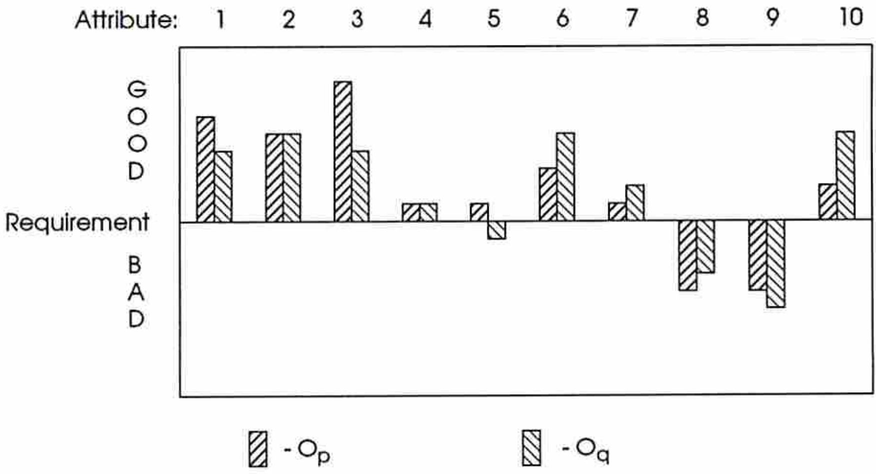


Figure 9.6: An example bar-chart diagram

The flowchart for the first phase comparison and self modification process I is illustrated in Figure 9.7. The letter 'S' denotes a system operation; the letter 'U' denotes a user response.

9.7 Second Phase Comparison and Self Modification Process II

Once a user determines that the object having the largest score, say O_p, is acceptable, the second phase comparison is executed. Since O_p is acceptable, the current requirements are changed to match the attribute values of O_p. In the second phase

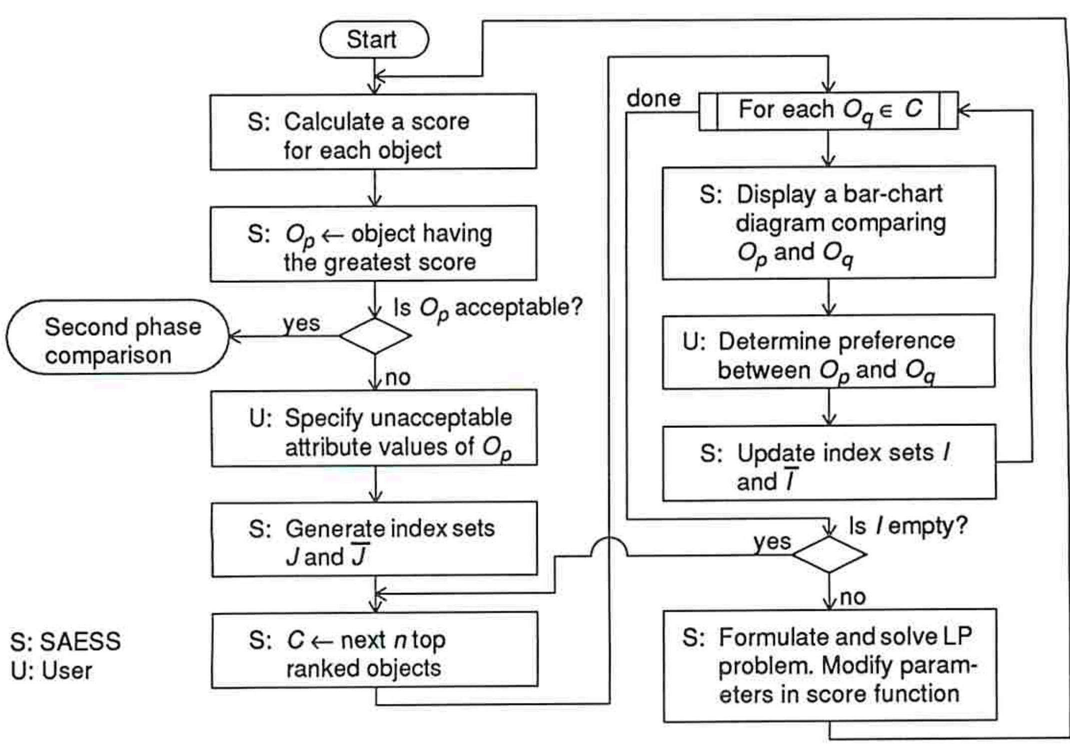


Figure 9.7: Flowchart for the first phase comparison and self modification process I

comparison a user has a chance to confirm the final selection or compare O_p with other objects. As mentioned earlier, when a score function does not accurately reflect a user's preferences, the best object may not have the largest score. Therefore, the second phase comparison allows the user to compare O_p with the remaining objects. A bar-chart comparison scheme similar to that described in the previous section is also employed in this process. The object with the second largest score, say O_q, is first compared with O_p. If it is more preferable than O_p, then a second self modification process is executed to modify the score function. The attribute values of O_p that are inferior to those of O_q are specified by the user and represented by the index set J. The index set I simply contains O_q. A LP formulation similar to that presented in the previous section is derived and solved. The score function is then modified. A new score is computed using the modified score function and the current requirements are changed to match the attribute values of the new object having the largest score. On the other hand, if O_q is less preferable than O_p, the user may repeat the above comparison process using the object having the next largest score or quit SAESS with the selection of O_p or without selection. The flowchart for the second phase comparison and self modification process II is illustrated in Figure 9.8.

9.8 A Case Study

In this section we present an execution trace of a run dealing with the selection of a BISTable circuit. In this example there are 50 BISTable circuits under consideration and 6 attributes associated with each BISTable circuit. The attributes are area overhead, test time, performance impact, number of test sessions, extra I/O pins, and fault coverage. These BISTable circuits are shown in Table 9.3. As can be seen, it is difficult to select the best design by inspecting each of the designs shown in the table. We will illustrate how a design can be selected by SAESS. In the execution trace, each of the user responses is preceded by a '*'. Comments are added to indicate various execution steps in SAESS.

```
Script started on Tue Feb 8 14:21:47 1994
cauchy.usc.edu{1}>*saess
```

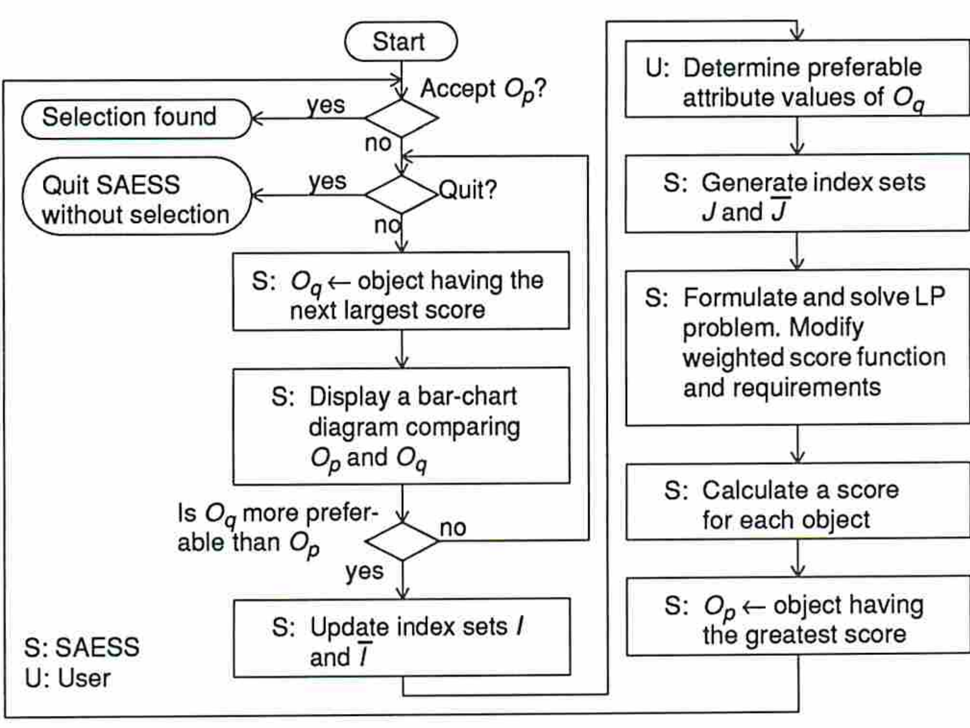


Figure 9.8: Flowchart for the second phase comparison and self modification process II

```
  *************************************************
  *  WELCOME TO SAESS - AN EXPERT SELECTION SYSTEM  *
  *************************************************

Please enter knowledge base name: *bits_ckt

  ***   BISTable circuits selection adviser   ***
There are 50 BISTable circuits under consideration. 6
attributes are associated with each BISTable circuit.
```

<initial weights and requirements specification>

```
For your application, some attributes may be more impor-
tant than the others.  It will be helpful if the initial
relative weights are provided. The following are default
weights. You can accept or modify them.

        attribute name       weight
     --------------------------------
      ( 1) Area overhead        10.0
      ( 2) Fault coverage       10.0
      ( 3) Performance impact    8.0
      ( 4) Test time             7.0
      ( 5) Test sessions         5.0
      ( 6) Extra I/O             3.0
     --------------------------------
    Enter the first weight to change (0 for none): *0

To select the best BISTable circuit, a set of initial
requirements  is needed.  The followings are  default
requirements. You can accept or modify them if desired.

        attribute name        requirement
     ------------------------------------------
      ( 1) Area overhead      <= 8.0    %
      ( 2) Fault coverage     >= 99.9   %
      ( 3) Performance impact <= 6.0    %
      ( 4) Test time          <= 100000 cycles
      ( 5) Test sessions      <= 4      sessions
      ( 6) Extra I/O          <= 2      pins
     ------------------------------------------
        Do you want to accept them? (Y/N) *n
```

```
  Enter requirement for Area overhead: *5
  Enter requirement for Fault coverage: *100

******************************************
Performance impact is strongly correlated to
    Area overhead
Based on the statistics of the available
BISTable circuits, the suggested range of
requirement for Performance impact is:
    from 0.0% to 5.0%
******************************************
  Enter requirement for Performance impact: *3

******************************************
Test time is strongly correlated to
    Area overhead and Performance impact
Based on the statistics of the available
BISTable circuits, the suggested range of
requirement for Test time is:
    from 80000 cycles to 300000 cycles
******************************************
  Enter requirement for Test time: *120000


******************************************
Test session is strongly correlated to
    Area overhead and Performance impact
Based on the statistics of the available
BISTable circuits, the suggested range of
requirement for Test time is:
    from 2 sessions to 4 sessions
******************************************
  Enter requirement for Test sessions: *3
  Enter requirement for Extra I/O: *2

The modified requirements are:
       attribute name         requirement
   -----------------------------------------
    ( 1) Area overhead      <= 5.0    %
    ( 2) Fault coverage     >= 100.0  %
    ( 3) Performance impact <= 3.0    %
    ( 4) Test time          <= 120000 cycles
```

```
( 5) Test sessions      <= 3      sessions
( 6) Extra I/O          <= 2      pins
-------------------------------------------
Are you satisfied with them? (Y/N) *y
```

<initial score function specification>

```
Another important factor in selecting a good BIST-
able circuit is how you grade different  attribute
values of  BISTable circuits compared with the re-
quirements. Usually if the value of  some BISTable
circuit is better than the requirement,  its grade
will be greater than the grade of  requirement and
vice versa.  SAESS provides  you a set of  default
grades, you can simply accept them or regrade them
if desired.
  Please enter your choice:
   (1) Use default grades.
   (2) Customize grades.
  Enter choice: *1
```

<calculate a score for each BISTable circuit. ckt19 has the largest score and the candidate set contains ckt18, ckt21, ckt20, ckt15>
<enter first phase comparison>

```
  Here is the BISTable circuit with the highest score:

  BISTable circuit name: ckt19   score: 0.0573
```

attribute name	attribute value		requirement		contribution
Area overhead	5.7	%	5.0	%	-0.0080
Fault coverage	100.0	%	100.0	%	0.0000
Performance impact	0.0	%	3.0	%	0.0335
Test time	98304	cycles	120000	cycles	0.0144
Test sessions	2	sessions	3	sessions	0.0174
Extra I/O	2	pins	2	pins	0.0000

```
  Can you accept this BISTable circuit? (Y/N/H)
  Enter H if you want to see the description of ckt19: *n
```

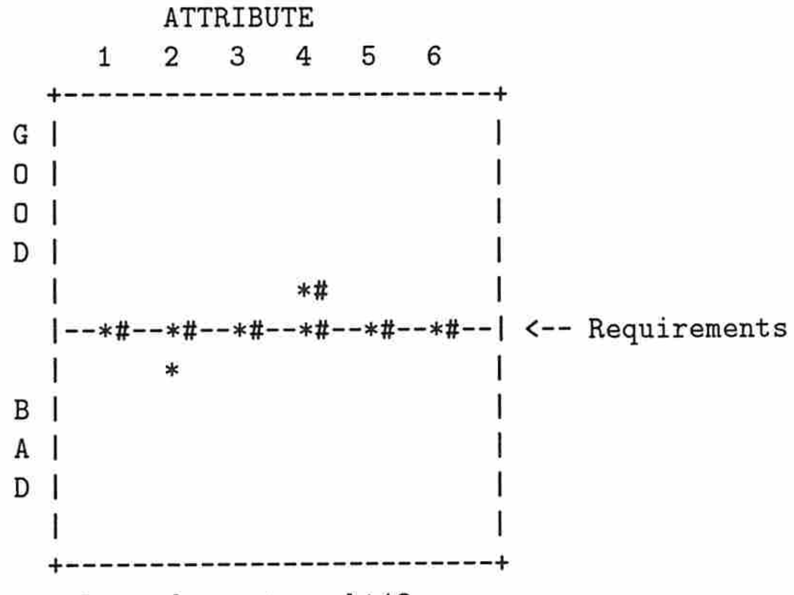
<self modification process, obtain index sets J and $\bar{J}$>

```
  Please specify which attributes of ckt19 are not acceptable:
    ( 1) Area overhead
```

```
    ( 2) Fault coverage
    ( 3) Performance impact
    ( 4) Test time
    ( 5) Test sessions
    ( 6) Extra I/O
  Enter the first unacceptable attribute number: *1
  Enter the next unacceptable attribute number
    (Enter 0 for none): *0
```

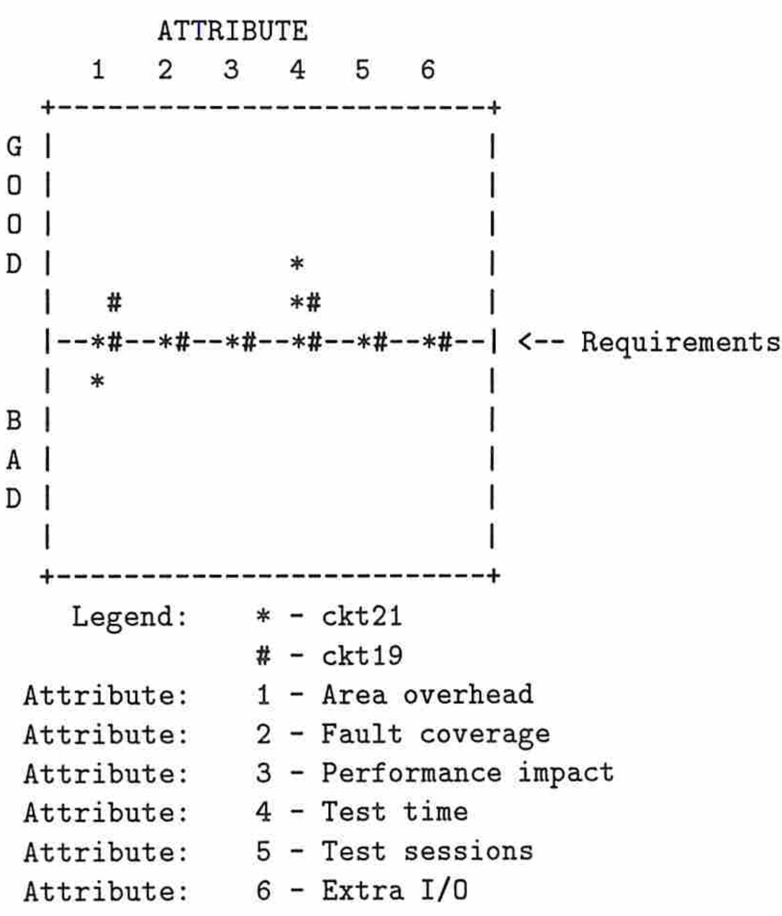
<obtain index sets I and $\bar{I}$>

```
Below some promising BISTable circuits are shown, for
each please specify whether it is better than ckt19.
```



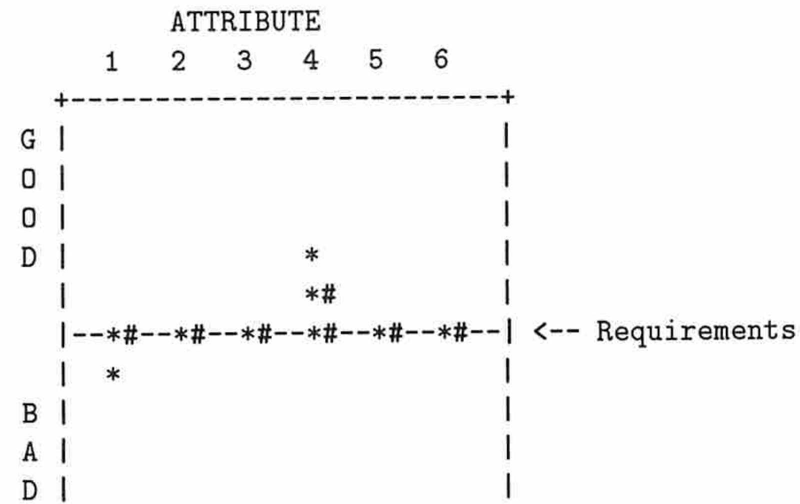

```
      Legend:     * - ckt18
                  # - ckt19
    Attribute:    1 - Area overhead
    Attribute:    2 - Fault coverage
    Attribute:    3 - Performance impact
    Attribute:    4 - Test time
    Attribute:    5 - Test sessions
    Attribute:    6 - Extra I/O

  Is ckt18 better than ckt19? (Y/N/Q/H)
  Enter Q if not sure, enter H for more
  detailed descriptions of ckt18: *n
```

<ckt18 goes into set $\bar{I}$>




```
Is ckt21 better than ckt19? (Y/N/Q/H)
Enter Q if not sure, enter H for more
detailed descriptions of ckt21: *n
```

<ckt21 goes into set $\bar{I}$>



```
    |                          |
    +--------------------------+
      Legend:     * - ckt20
                  # - ckt19
   Attribute:     1 - Area overhead
   Attribute:     2 - Fault coverage
   Attribute:     3 - Performance impact
   Attribute:     4 - Test time
   Attribute:     5 - Test sessions
   Attribute:     6 - Extra I/O

 Is ckt20 better than ckt19? (Y/N/Q/H)
 Enter Q if not sure, enter H for more
 detailed descriptions of ckt20: *n
```

<ckt20 goes into set $\bar{I}$>

```
            ATTRIBUTE
        1   2   3   4   5   6
    +--------------------------+
  G |                          |
  O |                          |
  O |                          |
  D |                          |
    |  *            #          |
    |--*#--*#--*#--*#--*#--*#--| <-- Requirements
    |              *           |
  B |                          |
  A |                          |
  D |                          |
    |                          |
    +--------------------------+
      Legend:     * - ckt15
                  # - ckt19
   Attribute:     1 - Area overhead
   Attribute:     2 - Fault coverage
   Attribute:     3 - Performance impact
   Attribute:     4 - Test time
   Attribute:     5 - Test sessions
   Attribute:     6 - Extra I/O

 Is ckt15 better than ckt19? (Y/N/Q/H)
 Enter Q if not sure, enter H for more
```

```
  detailed descriptions of ckt15: *h

  BISTable circuit name: ckt15    score: 0.0132

attribute name     attribute value  requirement    contribution
---------------------------------------------------------------
Area overhead      4.4    %         5.0    %         0.0067
Fault coverage     100.0  %         100.0  %         0.0000
Performance impact 0.0    %         3.0    %         0.0335
Test time          165888 cycles    120000 cycles    -0.0247
Test sessions      3      sessions  3      sessions  0.0000
Extra I/O          2      pins      2      pins      0.0000
---------------------------------------------------------------
  Is ckt15 better than ckt19? (Y/N/Q/H)
  Enter Q if not sure, enter H for more
  detailed descriptions of ckt15: *y
```

<ckt15 goes into set I>
<formulate and solve a LPP, then modify score function>

```
  ***************************************************
  Based on the analysis, ckt19 is unacceptable due to
  its poor values on the following attribute(s):
     Area overhead
  To favor some other BISTable circuit, more emphases
  have been put on the above attributes.
  ***************************************************

Do you want to keep ckt19 for future consideration? *y
```

<next iteration of first phase comparison>

```
 Here is the BISTable circuit with the highest score:

 BISTable circuit name: ckt15    score: 0.0000
attribute name     attribute value  requirement    contribution
---------------------------------------------------------------
Area overhead      4.4    %         5.0    %         0.0067
Fault coverage     100.0  %         100.0  %         0.0000
Performance impact 0.0    %         3.0    %         0.0167
Test time          165888 cycles    120000 cycles    -0.0212
Test sessions      3      sessions  3      sessions  0.0000
Extra I/O          2      pins      2      pins      0.0000
---------------------------------------------------------------
```

```
  Can you accept this BISTable circuit? (Y/N/H)
  Enter H if you want to see the description of ckt15: *y
```

<change requirements to match values of ckt15>
<enter second phase comparison>

```
          *** LAST CHANCE CONFIRMATION ***
If you accept ckt15 as your selection, the requirements will
be changed to match the values of ckt15.
    Will you accept it as your selection? (Y/N) *n

 Now you have the following options:
  (1) Find different promising BISTable circuit for comparison.
  (2) Quit without solution.
  (3) Accept ckt15 as the selection.
 Enter your choice: *1

             ATTRIBUTE
         1   2   3   4   5   6
      +-------------------------+
    G |                         |
    O |                         |
    O |                         |
    D |                         |
      |   #         *           |
      |--*#--*#--*#--*#--*#--*#--| <-- Requirements
      |              #          |
    B |                         |
    A |                         |
    D |                         |
      |                         |
      +-------------------------+
        Legend:     * - ckt19
                    # - ckt15
     Attribute:     1 - Area overhead
     Attribute:     2 - Fault coverage
     Attribute:     3 - Performance impact
     Attribute:     4 - Test time
     Attribute:     5 - Test sessions
     Attribute:     6 - Extra I/O

  Do you think ckt19 is better than ckt15? (Y/N/H/E)
  Enter E to exit to previous menu.
```

```
Enter H if you want to see the description of ckt19: *n
```

<ckt19 goes into set $\bar{I}$*>*

```
           ATTRIBUTE
       1   2   3   4   5   6
    +---------------------------+
  G |                           |
  O |                           |
  O |  *                        |
  D |  *                        |
    |  *#                       |
    |--*#--*#--*#--*#--*#--*#--| <-- Requirements
    |              *#           |
  B |              *            |
  A |                           |
  D |                           |
    |                           |
    +---------------------------+
     Legend:     * - ckt12
                 # - ckt15
   Attribute:    1 - Area overhead
   Attribute:    2 - Fault coverage
   Attribute:    3 - Performance impact
   Attribute:    4 - Test time
   Attribute:    5 - Test sessions
   Attribute:    6 - Extra I/O
```

```
Do you think ckt12 is better than ckt15? (Y/N/H/E)
Enter E to exit to previous menu.
Enter H if you want to see the description of ckt12: *y
```

<ckt12 goes into set I*>*
<obtain index sets J *and* $\bar{J}$*>*

```
Please specify which attributes of ckt12 are superior to
those of ckt15:
  ( 1) Area overhead
  ( 2) Fault coverage
  ( 3) Performance impact
  ( 4) Test time
  ( 5) Test sessions
  ( 6) Extra I/O
Enter the first attribute number: *1
```

```
  Enter the next attribute number.
    (Enter 0 for none): *0
```

<*formulate and solve a LPP, then modify score function*>

```
  Do you want to keep ckt15 for future consideration? *y

Based on the criteria, ckt12 is the best BISTable circuit.

  BISTable circuit name: ckt12   score: 0.0026
attribute name    attribute value contribution
----------------------------------------------
Area overhead       3.0    %          0.0233
Fault coverage      100.0  %          0.0000
Performance impact 0.0     %          0.0156
Test time          296960 cycles     -0.0362
Test sessions       3      sessions  0.0000
Extra I/O           2      pins      0.0000
----------------------------------------------

          *** LAST CHANCE CONFIRMATION ***
If you accept ckt12 as your selection, the requirements will
be changed to match the values of ckt12.
    Will you accept it as your selection? (Y/N) *y
```

<*change requirements to match values of ckt12*>

```
            *** SOLUTION FOUND ***
                    Bye!
cauchy.usc.edu{2}>*exit
script done on Tue Feb 8 14:24:39 1994
```

9.9 Summary

SAESS provides a user-friendly environment to help a user in making an intelligent selection. We have implemented SAESS and used the system for the selection of BIST designs and automobiles[52]. In this latter example case the large number of attributes to consider and the number of automobiles under consideration make

Circuit	Area overhead	Test time	Performance impact	# of test sessions	Extra I/O pins	Fault coverage
ckt1	7.79%	106,560	5.33%	5	2	100%
ckt2	8.36%	73,792	5.33%	5	2	100%
ckt3	8.92%	73,792	5.33%	5	4	99.95%
ckt4	9.23%	65,600	5.33%	5	4	99.98%
ckt5	8.38%	42,048	5.33%	4	2	100%
ckt6	9.80%	41,024	5.33%	4	4	99.00%
ckt7	10.71%	34,048	9.33%	3	2	100%
ckt8	12.11%	33,856	9.33%	5	2	99.96%
ckt9	12.43%	40,960	5.33%	4	2	100%
ckt10	13.34%	32,832	9.33%	5	4	99.50%
ckt11	14.50%	32,768	9.33%	5	2	99.05%
ckt12	2.95%	296,960	0.00%	3	2	100%
ckt13	3.69%	231,424	0.00%	3	2	99.90%
ckt14	4.04%	294,912	0.00%	2	2	99.94%
ckt15	4.42%	165,888	0.00%	3	2	99.99%
ckt16	4.67%	163,840	0.00%	2	2	99.84%
ckt17	4.99%	100,352	0.00%	3	2	99.30%
ckt18	5.41%	98,304	0.00%	3	2	99.90%
ckt19	5.69%	98,304	0.00%	2	2	100%
ckt20	6.43%	67,584	4.00%	2	2	99.99%
ckt21	6.60%	65,536	4.00%	2	2	100%
ckt22	6.80%	65,336	4.44%	2	2	99.55%
ckt23	3.05%	277,360	0.00%	3	2	99.00%
ckt24	15.00%	32,768	5.00%	4	2	99.55%
ckt25	10.11%	35,034	9.33%	4	4	99.50%
ckt26	7.55%	106,322	3.33%	4	4	100%
ckt27	8.76%	83,792	5.33%	5	2	99.50%
ckt28	9.92%	63,792	3.33%	5	4	99.99%
ckt29	10.23%	55,600	7.33%	4	4	99.97%
ckt30	7.38%	52,048	7.33%	4	2	100%
ckt31	8.80%	91,024	6.33%	4	2	99.50%
ckt32	10.89%	34,000	7.33%	3	2	99.00%
ckt33	13.11%	30,856	10.33%	3	2	99.99%
ckt34	13.43%	30,960	7.33%	4	2	100%
ckt35	13.04%	32,932	8.33%	4	4	99.00%
ckt36	14.56%	32,748	9.00%	4	4	99.55%
ckt37	2.99%	296,960	2.00%	3	2	100%
ckt38	3.89%	230,424	0.00%	3	2	99.90%
ckt39	4.22%	294,612	0.00%	2	2	99.55%
ckt40	4.55%	165,688	0.00%	3	4	99.50%
ckt41	4.87%	162,840	0.00%	2	4	99.84%
ckt42	4.99%	101,352	0.00%	2	2	99.99%
ckt43	5.66%	98,604	0.00%	3	2	99.00%
ckt44	5.89%	98,204	0.00%	2	2	99.50%
ckt45	6.56%	67,584	5.00%	2	2	99.99%
ckt46	6.90%	65,236	4.00%	2	2	99.99%
ckt47	6.99%	65,336	4.00%	2	2	99.55%
ckt48	3.14%	277,360	0.00%	3	2	99.99%
ckt49	15.00%	32,768	6.00%	4	2	99.55%
ckt50	10.85%	35,000	7.33%	4	4	99.99%

Table 9.3: Example BISTable circuits

manual selection difficult and time consuming. By employing SAESS, a user can explore the selection space more easily since the score function directs the user's attention to a small subset of objects under consideration. In addition, the self modification processes automatically modify the score function when it does not accurately reflect the user's preferences. Due to this machine learning capability, a user need not be a domain expert to make a good selection. SAESS is a generic selection system in the sense that it is not domain dependent. Therefore, it can be easily adapted to any selection problem.

As discussed in the previous chapters, there are often many BISTable versions for a CUC. Several attributes such as area overhead, test application time, and performance degradation are associated with each BISTable circuit. In addition, a designer usually has certain requirement for each of these attributes based on the design constraints. An execution trace of using SAESS to select a BISTable circuit has been presented.

Chapter 10

BITS Implementation and Experiments

In this chapter the implementation of the BITS system is described, followed by experimental results obtained from using the BITS system to make several data path circuits BISTable.

10.1 BITS Implementation

BITS is implemented on a Sun SPARC system 300 with 32MB of memory under SunOS Release 4.1.3. The programs are written in C++ language and extensively use Cbase function calls. BITS is a menu driven system with a pop-up window showing the CUC and various manipulations on the circuit. The top level menu and the pop-up window showing an example CUC are illustrated in Figure 10.1. An input circuit to the BITS system is in the Cbase format. Most of the top level menu options have several sub-menu options and are briefly described below.

ToolBox

The ToolBox option provides a list of miscellaneous pre-processing options to make an input circuit suitable for the subsequent BIST design processes. The options in ToolBox are shown below.

```
BITS TOOLBOX MENU OPTIONS:
 1. FANCI (FANout Cell Insertion)
 2. COPI (COntrol Port Insertion)
 3. BOUNSI (BOUNdary Scan register Insertion)
 4. CRIPI (CRItical Path Identification)
```

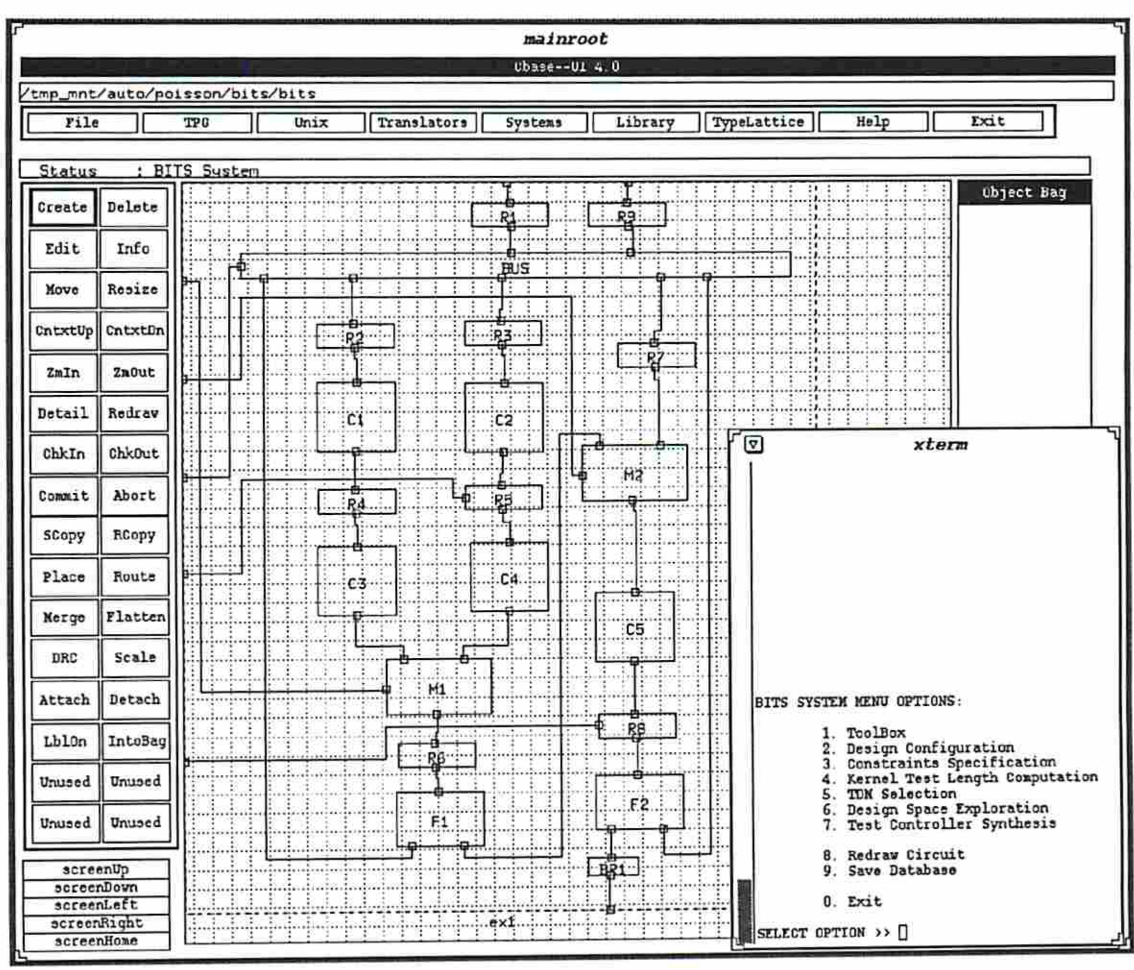


Figure 10.1: The top level menu and pop-up window of BITS

5. CIRGE (CIRcuit Graph Extraction)

The FANCI option replaces implicit fanouts (i.e. a carrier feeding more than one port) in a CUC with explicit fanout cells so that the CUC satisfies the assumed circuit model of BITS. The I-modes associated with these fanout cells are automatically added. For the data path circuits translated from the ADAM high-level synthesis system[15] developed by the USC Design Automation group, the control lines and control ports associated with the circuits are lost. This control information is essential for the test controller since the test controller attempts to utilize the existing control lines. Therefore, the COPI option does a reverse engineering to restore the necessary control lines and control ports.

Currently the test controller in BITS supports the IEEE 1149.1 boundary scan standard, where every PI (PO) of a CUC must directly feed (be fed by) a boundary scan register. These boundary scan registers are not used in the normal operation, but can operate as TPGs or SAs during the testing of a circuit. An extension to the IEEE 1149.1 standard is to convert a functional register to a boundary scan register when it is fed by a PI or feeds a PO, and only add additional boundary scan registers where necessary. The BOUNSI option provides these two boundary scan register insertion operations.

To evaluate the performance impact of a BISTable circuit, critical paths in a circuit need to be specified. Critical path analysis is a complicated process that requires extensive circuit simulation. The CRIPI option does not attempt to analyze the critical paths in a circuit. Instead, it simply provides a handy function where the data paths (if any) between every pair of registers are identified and displayed to the user via the circuit shown in the pop-up window (see Figure 10.2). The depth of every such data path is computed as the number of gates along the path. A user can then easily identify the critical paths if a critical path analysis has been performed.

The CIRGE option extracts a directed graph from a CUC in the format specified by the *dot* drawing tool developed by Koutsofios and North[53] and stored in a file. This circuit graph file can then processed by the *dot* program to generate a layout of the circuit graph in a graphics language such as PostScript. This option enables a user to better visualize the connections between cells in a CUC, which

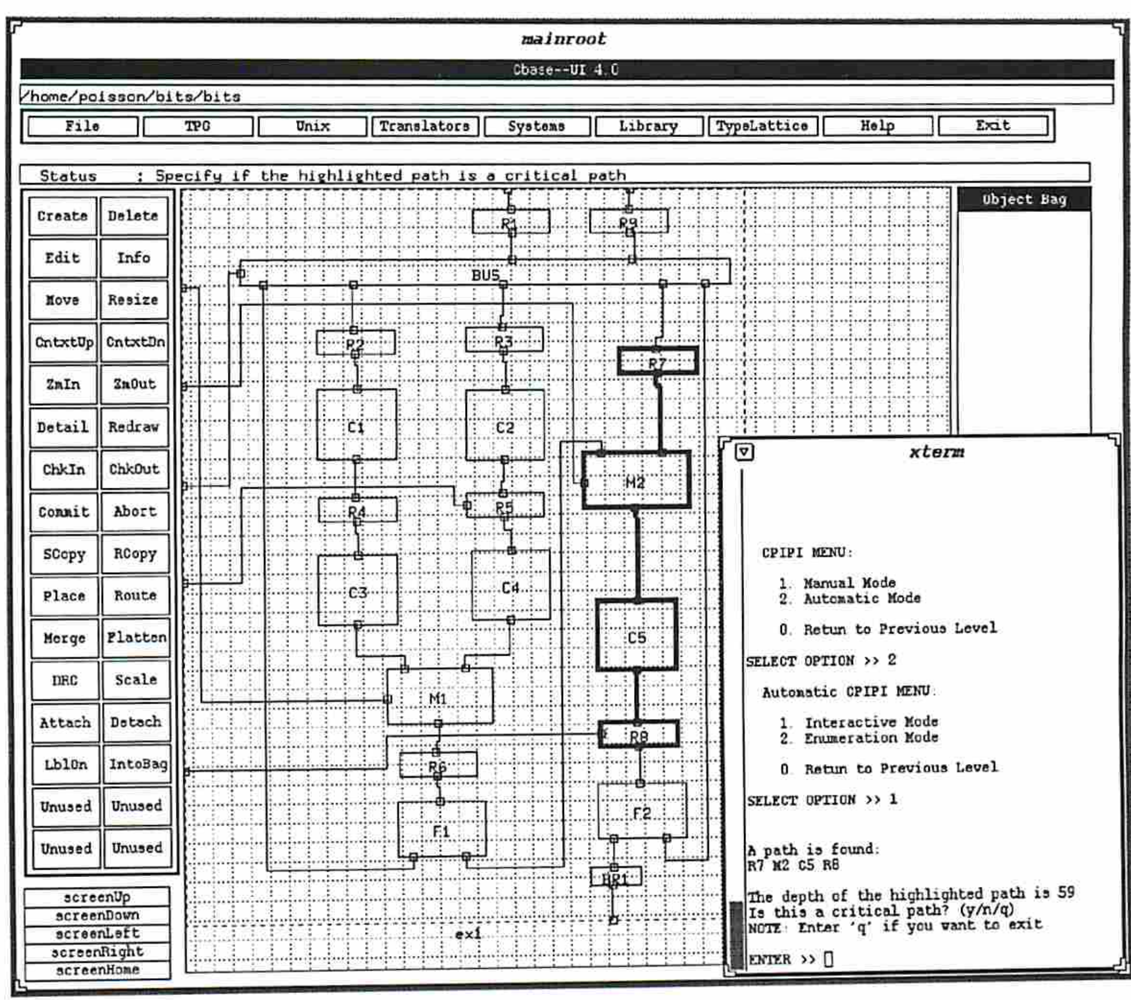


Figure 10.2: Illustration of the CRIPI option in ToolBox

is especially helpful when the CUC is too complex for the placement and routing procedures in the Cbase user interface to handle. The graph generated by the *dot* program for the example circuit in Figure 10.1 is shown in Figure 10.3.

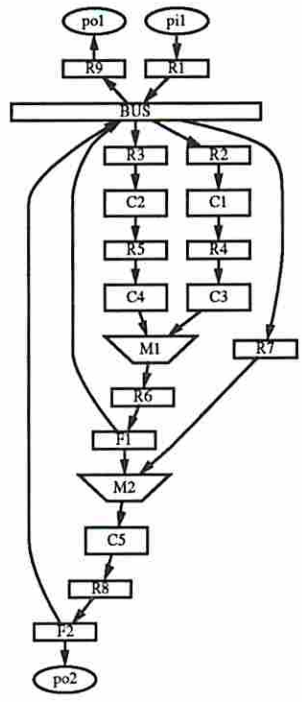


Figure 10.3: Circuit graph generated by the *dot* program

Design Configuration

The Design configuration option supports circuit partitioning and clustering used to identify the kernels under test. Three kinds of circuit configurations within a CUC are identified as shown below.

```
DESIGN CONFIGURATION OPTIONS:
  1. Combinational Kernel Identification
  2. Balanced Sequential Kernel Identification
  3. Kernel Partitioning
```

When only combinational kernels are allowed and there exist several small combinational cells, it may be desirable to cluster small combinational cells into larger

kernels. This can be done by the clustering routine in the CLARION system[16]. When balanced BISTable kernels are allowed, the procedure described in Chapter 5 is employed to identify such kernels. If there exist combinational kernels with too many inputs, the test lengths to fully test the kernels may be excessive. In this case, these kernels need to be partitioned to constrain their test lengths. Currently the partitioning for pseudo-exhaustive testing has been implemented using the procedure presented in [18]. The partitioning for pseudo-random testing remains an open problem.

Constraint Specification

A circuit designer usually has certain design constraints that have to be satisfied by the BISTable circuits generated. These constraints are specified using the constraint specification options shown below.

```
CONSTRAINT SPECIFICATION MENU:
 1. Hardware Overhead
 2. Fault Coverage
 3. Test Time
 4. Fault Type
 5. Register Constraint
```

In BITS the hardware overhead is calculated as follows. When the total number of gates in a CUC is known, the area overhead is calculated as the ratio between the extra gates required in the test hardware and the total gate count in the CUC. When the total gate count is unknown, which may occur when library cells are employed in a design and the gate-level design of the circuit has not been performed, then the area overhead is calculated as the ratio between the extra gates required in the test hardware and the number of gates in the F/Fs. The latter number indicates the test hardware compared with the register hardware. Note that the routing area is not taken into account. Test time is measured in terms of the number of clock cycles required to test a BISTable circuit. The actual test application time can be computed by multiplying this number with the clock rate during the test mode. A user can specify a value for the hardware overhead, the fault coverage, or the test time constraint to restrict the design space search

process. The fault type specification option allows a user to target either unmodeled faults or stuck-at faults. When unmodeled faults are targeted, the test strategies that can be employed are constrained to be either exhaustive or pseudo-exhaustive testing. When a register is in a critical path, the performance of the circuit may be degraded if the register is modified to be a test register. The register constraint option allows a user to direct the BISTable design process by preventing certain registers from being used as test registers.

Kernel Test Length Computation

This option estimates the test length of each kernel in the CUC for each test strategy supported by BITS. The estimation for exhaustive and functionally exhaustive testing is straightforward. The estimation for pseudo-exhaustive testing is done by identifying the largest cone in each kernel using a routine in the procedure presented by Srinivasan *et al.*[18]. Currently the estimation for pseudo-random testing is based on the technique presented by Lempel *et al.*[34]. This technique estimates a lower bound on the test length for a kernel to achieve 100% fault coverage. A more efficient and flexible approach presented by Majumdar and Sastry[54] may be employed to estimate the test length for pseudo-random and random testing in future. The estimation for random testing remains an open problem. Once the test lengths of a kernel, say K, for all test strategies are computed, the estimated test length of K is the minimum of the above values. The test strategy for K is simply the strategy that achieves this minimal test length.

TDM Selection

TDM selection option performs a lower bound estimation on the BIST parameters such as test hardware and test time for the supported TDMs, namely BILBO, CBILBO, and BIBS. This estimation is based on the procedures developed by Njinda[55]. Once the estimated values of test hardware and test time are obtained, a user can then select the TDM to employ in the design space exploration process. Note that once a TDM is selected, it is employed for for every kernel in the circuit.

Design Space Exploration

This option allows a user to explore the BISTable design space using the menu options shown below.

```
DESIGN SPACE EXPLORATION OPTIONS:
 1. Find Minimal Test Time Design
 2. Find Minimal Area Overhead Design
 3. Find Minimal Performance Impact Design
 4. Find a Family of Designs
 5. Preview Designs Generated
```

The user can choose to only generate a minimal area overhead design, a minimal test time design, a minimal performance impact design, or to explore the entire design space and generate a family of designs. Procedure *DSE* presented in Chapter 8 is employed during the design space exploration process. Once BISTable designs are generated, the user can select a design and preview the design using the pop-up window. The design preview process is done by simulating the execution of the test plan associated with the selected design as illustrated below.

Consider the circuit shown in Figure 10.1. The design preview process for a minimal test time BISTable circuit using the BILBO TDM is shown in Figures 10.4 and 10.5. The estimated test lengths of kernels C_1, C_2, C_3, C_4, and C_5 are 32, 256, 32, 32, and 256 clock cycles, respectively. Two test sessions are required where C_2, C_3 and C_5 are tested in the first session (see Figure 10.4) and C_1 and C_4 are tested in the second session (see Figure 10.5). Note that the test for C_3 is extended from 32 clock cycles to 256 clock cycles to reduce the test controller complexity as was discussed in Chapter 7. The test hardware required in a BISTable circuit can also be observed from the design preview option. In this example the test hardware required is: R_4 and R_5 operate as BILBO registers, R_1 and R_8 operate as TPGs, and R_6 and BR_1 operate as SAs.

For each representative BISTable circuit generated, a test plan that specify the test hardware and execution of the circuit is automatically produced as discussed in Chapter 8. In addition, if there exist SA registers that are too small, they are automatically merged as described before. For the minimal test time BISTable circuit shown above, a portion of the test plan generated is shown below.

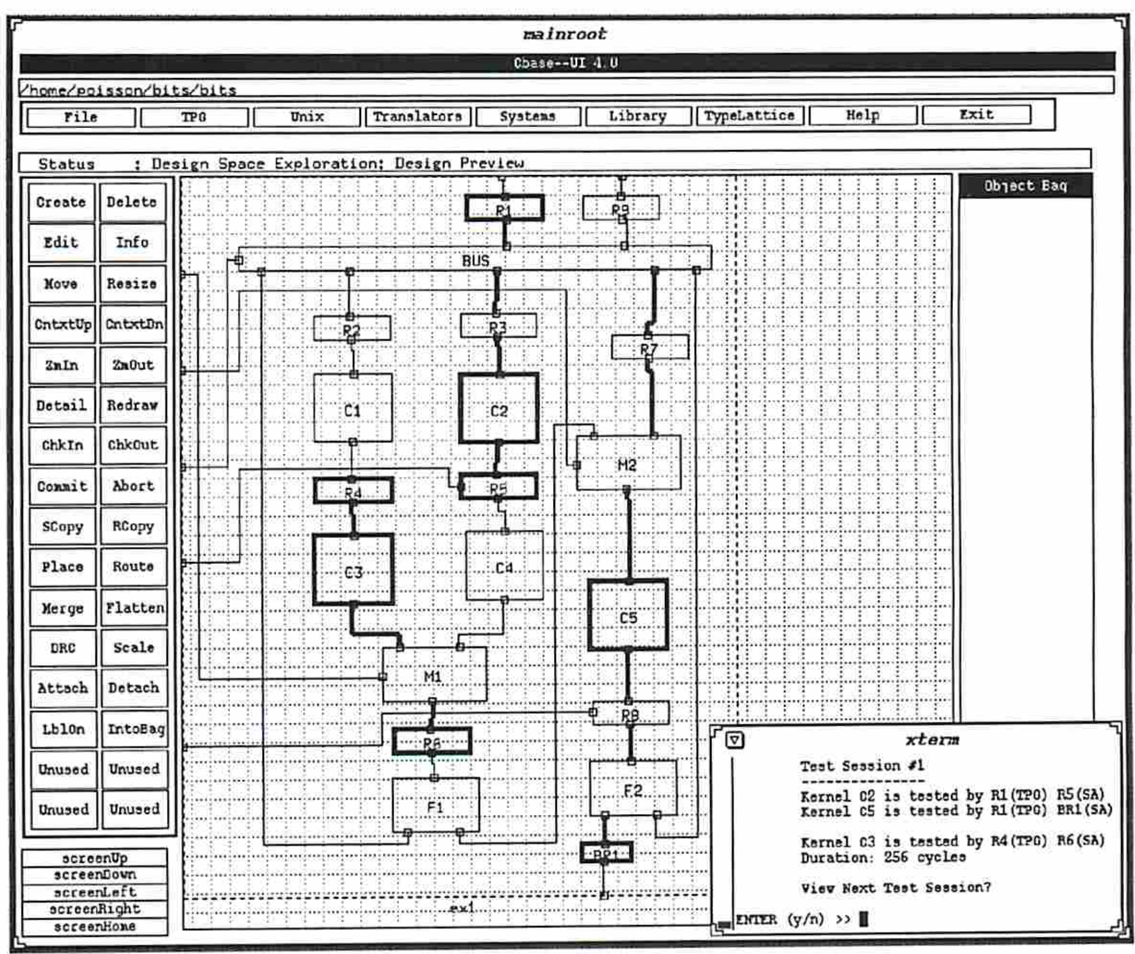


Figure 10.4: BISTable circuit preview - first test session

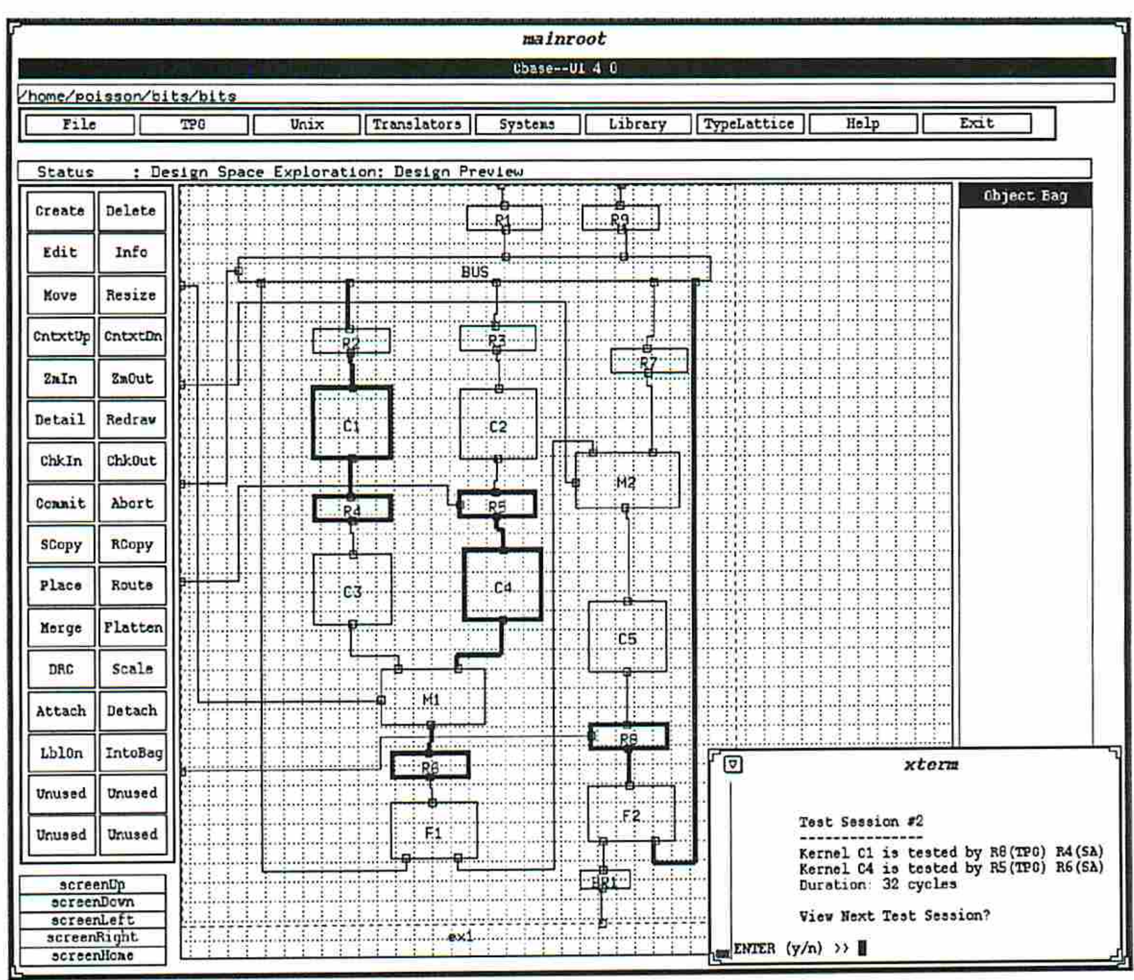


Figure 10.5: BISTable circuit preview - second test session

```
BEGIN CHAIN_DEF
  CHAIN 0;  /* boundary scan chain */
    CHAIN_TYPE = BOUNDARY_SCAN;
    BEGIN_CHAIN
      REG_NAME = R1, LENGTH = 12, INPUTBS;
      REG_NAME = R9, LENGTH = 12, OUTPUTBS;
      REG_NAME = BR1, LENGTH = 8, OUTPUTBS;
    END_CHAIN
  CHAIN 1;
    CAHIN_TYPE = GENERAL;
    BEGIN_CHAIN
      REG_NAME = R1, LENGTH = 12;
      REG_NAME = R9, LENGTH = 12,
      REG_NAME = BR1, LENGTH = 8,
      REG_NAME = R4, LENGTH = 8;
      REG_NAME = R8, LENGTH = 8;
      REG_NAME = R5, LENGTH = 8;
      REG_NAME = R6, LENGTH = 8;
    END_CHAIN
END_CHAIN_DEF

BEGIN_REG_GROUP_DEF
  REG_NAME = RG1, CONSTITUENTS = R1,
      FUNC = PG,
  REG_NAME = RG3, CONSTITUENTS = R4,
      FUNC = PG,
  REG_NAME = RG4, CONSTITUENTS = R8,
      FUNC = PG,
  REG_NAME = RG5, CONSTITUENTS = R5,
      FUNC = PG,
  REG_NAME = RG6, CONSTITUENTS = R5 BR1 R6,
      FUNC = SA,
```

```
  REG_NAME = RG7, CONSTITUENTS = R4 R6,
      FUNC = SA,
END_REG_GROUP_DEF

      <control port definition deleted>

BEGIN_SESSION_DEF
  SESSION 0;
    TDM = BILBO;
      BEGIN_INITIALIZE

      <initialization...>

      BEGIN_APPLICATION
        PHASE = 0,

      <control signal specification deleted>

          REG_VAL = RG1:PG RG3:PG RG6:SA;
        APPLY_CYCLES = 256;
      END_APPLICATION
  SESSION 1;
    TDM = BILBO;
      BEGIN_INITIALIZE

      <initialization...>

      BEGIN_APPLICATION
        PHASE = 0,

      <control signal specification deleted>

          REG_VAL = RG4:PG RG5:PG RG7:SA;
        APPLY_CYCLES = 32;
      END_APPLICATION
END_SESSION_DEF
```

Note that registers R_5, R_6 and BR_1 are combined as one SA (RG_6) during the first test session. In the second test session R_6 and BR_1 are combined as one SA (RG_8), and R_4 and R_1 are combined as another SA (RG_7).

Test Controller Synthesis

A test controller can be synthesized for each BISTable circuit generated by using the test plan associated with the circuit. The test controller synthesis problem has been addressed by Mukherjee *et al.*[56].

10.2 Experimental Results

Several data path circuits that are either manually designed using the Cbase user interface or synthesized by the ADAM system have been made BISTable using the BITS system. The BISTable designs generated for these circuits are presented and discussed in the following sections.

10.2.1 The DCT Circuit

The DCT circuit is synthesized by the ADAM system based on the implementation presented in [57]. This circuit performs a discrete cosine transformation and is employed in a prototype video chip that conforms to the international low bit-rate video coding standard.

The width of the data path in the DCT circuit is 8-bit. The circuit contains 3 adders, 3 subtractors, 8 multipliers, 39 registers, and numerous MUXes. The test length for each adder, subtractor, and multiplier are 2^{12}, 2^{12} and 2^{13} clock cycles, respectively, to achieve 100% fault coverage using pseudo-random testing based on the estimation of the test length computation module in BITS. BITS generates a family of BISTable circuits summarized in Table 10.1. The performance degradation is computed as follows. Let Δ be the allowed delay for all critical paths in the original circuit, which is determined by the circuit designer. In each BISTable version of the circuit the delay of each critical path, say CP_j, is first calculated as $\Delta_j + \delta_j$, where Δ_j is the delay of CP_j in the original circuit and δ_j is the additional delay added to this path due to test hardware. The performance degradation is $(\max_j(\Delta_j + \delta_j) - \Delta)/\Delta$ for all CP_j having $\Delta_j + \delta_j > \Delta$. The last column shows the number of test registers, i.e. BILBO registers, TPGs, SAs, and CBILBO register (in this order), employed in each BISTable circuit. Note that the area overhead of

different types of test registers are different, thus the area overhead of a BISTable circuit may not be proportional to the total number of test registers. For example, although circuits #2 and #3 employ the same number (i.e. 17) of test registers, they have different area overhead. The actual number of feasible BISTable circuits is much greater than the number of representative circuits generated. Due to the pruning process in the design space exploration, the designs that are inferior to any of these representative circuits are not generated. The first 11 circuits are the results obtained when only combinational kernels are allowed, and the last 3 circuits (shaded in Table 10.1) are the results obtained using the BIBS TDM. Circuit #14 is obtained using the BIBS TDM and allowing CBILBO registers. The number of test sessions is also considered as a BIST parameter since it roughly indicates the test controller complexity. Namely, as the number of test sessions increases, so does the number of states in the finite state machine of the test controller, thus leading to a more complex test controller. As can be seen from the table, the CBILBO TDM usually leads to shorter test application time and less number of test sessions, but more area overhead. Also notice that the BIBS TDM leads to a design (#12) having the minimal area overhead.

10.2.2 The AR Filter Circuits

A family of auto-regressive (AR) filter circuits are considered where each of them is synthesized by the ADAM system. These circuits are produced from the same data flow graph with different performance and chip area requirements. The circuit characteristics are summarized in Table 10.2. As is shown in the table, circuit AR2 has a data input latency of 2 and uses the greatest number of functional units (6 adders and 8 multipliers). As the number of functional units decreases, the data input latency increases and the throughput decreases. Again the data paths in these circuits have a uniform width of 8 bits. The estimated test lengths for the adders and multipliers are the same as before.

The representative BISTable circuits generated by the BITS system for each of these circuits are summarized in Tables 10.3, 10.4,10.5, and 10.6. Note that these AR filter circuits extensively employ MUXes, which results in each combinational

Circuit	TDM	Area overhead	Test time	Performance impact	# of test registers*	# of test sessions
#1	BILBO	6.60%	20,480	5.00%	3/8/5/0	5
#2	BILBO	6.77%	16,384	5.00%	4/8/5/0	4
#3	BILBO	6.95%	16,384	5.00%	5/7/5/0	3
#4	BILBO	6.97%	16,384	5.00%	3/9/7/0	2
#5	BILBO	8.06%	12,288	5.00%	3/10/10/0	2
#6	CBILBO	7.21%	16,384	6.67%	3/8/5/1	8
#7	CBILBO	7.86%	16,384	5.00%	0/11/7/2	3
#8	CBILBO	8.05%	16,384	5.00%	0/11/8/2	2
#9	CBILBO	8.95%	12,288	5.00%	0/11/10/2	3
#10	CBILBO	9.13%	8,192	5.00%	0/11/11/2	2
#11	CBILBO	9.32%	8,192	5.00%	0/11/12/2	1
#12	BIBS	6.14%	270,336	5.00%	3/8/6/0	12
#13	BIBS	7.23%	270,336	5.00%	4/9/6/0	10
#14	BIBS with CBILBO	6.77%	262,144	5.00%	2/9/6/1	12

* # of BILBO registers / # of TPGs / # of SAs / # of CBILBO registers

Table 10.1: BISTable designs for the DCT circuit

Circuit	# functional units	# registers	# MUXes	Latency
AR2	14	52	34	2
AR4	12	38	22	4
AR6	5	42	17	6
AR8	4	35	11	8

Table 10.2: Characteristics of the AR filter circuits

Circui t	TDM	Area overhead	Test time	Performance impact	# of test registers	# of test sessions
#1	BILBO	8.39%	20,480	11.67%	8/12/4/0	3
#2	BILBO	8.51%	16,384	11.67%	9/11/4/0	3
#3	BILBO	8.63%	12,288	11.67%	10/10/4/0	2
#4	CBILBO	8.98%	16,384	23.33%	6/12/4/2	7
#5	CBILBO	9.57%	16,384	20.00%	4/12/4/4	7
#6	CBILBO	9.87%	16,384	23.33%	3/12/4/5	2

Table 10.3: BISTable designs for circuit AR2

cell being a connected component by itself in the circuit graph as defined in Chapter 5. Therefore the kernels identified by the BIBS TDM are still combinational logic blocks and thus no additional representative designs can be generated. It is interesting to observe that though these circuits realize the same logic behavior they lead to BISTable circuits having quite different values of the BIST parameters. For example, most BISTable circuits for AR8 have less area overhead, test time, and performance impact than those of the BISTable circuits for AR2. In other words, AR8 tends to have better testability attributes than AR2. On the other hand, the BISTable circuits for AR4 always require less area overhead than the BISTable circuits for AR2. However, the test time requirements for most of the BISTable circuits for AR4 are greater than those of AR2. Therefore once a circuit designer has produced a family of designs that realize a data flow graph, he/she can simply use the BITS system to generate a family of BISTable versions of each of the designs. Trade-offs can be made between the total chip area, test application time, and the final circuit performance before the design is sent out for fabrication. This greatly reduces the design cycle time and the costs associated with the design process.

Circui t	TDM	Area overhead	Test time	Performance impact	# of test registers	# of test sessions
#1	BILBO	5.78%	32,768	13.33%	7/5/1/0	5
#2	BILBO	5.92%	28,672	13.33%	8/4/1/0	4
#3	CBILBO	8.01%	16,384	13.33%	1/4/0/7	2

Table 10.4: BISTable designs for circuit AR4

Circuit	TDM	Area overhead	Test time	Performance impact	# of test registers	# of test sessions
#1	BILBO	5.06%	32,768	10.00%	2/5/1/0	3
#2	BILBO	5.28%	12,288	10.00%	3/4/1/0	2
#3	BILBO	6.93%	16,384	10.00%	0/7/5/0	1
#4	CBILBO	7.31%	16,384	11.67%	0/6/4/2	1

Table 10.5: BISTable designs for circuit AR6

Circuit	TDM	Area overhead	Test time	Performance impact	# of test registers	# of test sessions
#1	BILBO	4.13%	16,384	0.00%	1/3/1/0	3
#2	BILBO	4.61%	12,288	5.00%	0/4/2/0	2
#3	BILBO	4.90%	12,288	0.00%	1/3/2/0	2
#4	BILBO	5.38%	8,192	5.00%	0/4/3/0	2
#5	BILBO	5.96%	8,192	0.00%	2/3/2/0	2
#6	BILBO	6.15%	8,192	0.00%	0/4/4/0	1
#7	CBILBO	5.63%	8,192	0.00%	0/3/2/1	2
#8	CBILBO	6.40%	8,192	0.00%	0/3/3/1	1

Table 10.6: BISTable designs for circuit AR8

Circui t	TDM	Area overhead	Test time	Performance impact	# of test registers	# of test sessions
#1	BILBO	5.48%	832	0.00%	2/1/1/0	4
#2	BILBO	5.54%	320	0.00%	2/1/2/0	3
#3	BILBO	6.02%	288	5.08%	2/1/2/0	3
#4	BILBO	7.21%	288	11.86%	3/1/1/0	2
#5	BILBO	7.27%	288	10.17%	2/2/2/0	2
#6	BILBO	8.94%	288	5.08%	3/2/1/0	2
#7	CBILBO	7.21%	256	5.08%	1/1/2/1	3
#8	BIBS	2.09%	896	0.00%	0/1/2/0	3
#9	BIBS	2.56%	320	5.08%	0/1/2/0	2
#10	BIBS	3.34%	640	0.00%	0/2/2/0	3
#11	BIBS	4.59%	576	0.00%	0/3/2/0	3

Table 10.7: BISTable designs for circuit `ex1`

10.2.3 Miscellaneous Circuits

Two manually designed circuits are also considered to demonstrate various aspects of the BITS system. The first circuit has been shown in Figure 10.1. The widths of R_1 and R_9 are 12 bits and the widths of the remaining registers are 8 bits. The estimated test length for each combinational cell is the same as that described in the previous section. The critical path s of the circuit are specified as the path from $R7$ through $M2$ and $C5$ to $R8$ and the path from $R6$ through $F1$ and $M2$ to $R8$. The delay of these critical paths are also specified. When only combinational kernels are allowed, the representative BISTable circuits generated by BITS are summarized in Table 10.7 as circuits #1-7.

When the BIBS TDM is employed two balanced BISTable kernels, B_1 and B_2, are identified as illustrated in Figure 10.6. Kernel $B1$ consists of C_1, R_4, and C_3 and kernel $B2$ consists of C_2, R_5, and C_4. Using the BIBS TDM four additional BISTable circuits are generated and summarized in Table 10.7 as circuits #8-11.

Next we consider the circuit shown in Figure 10.7. Let the estimated test length

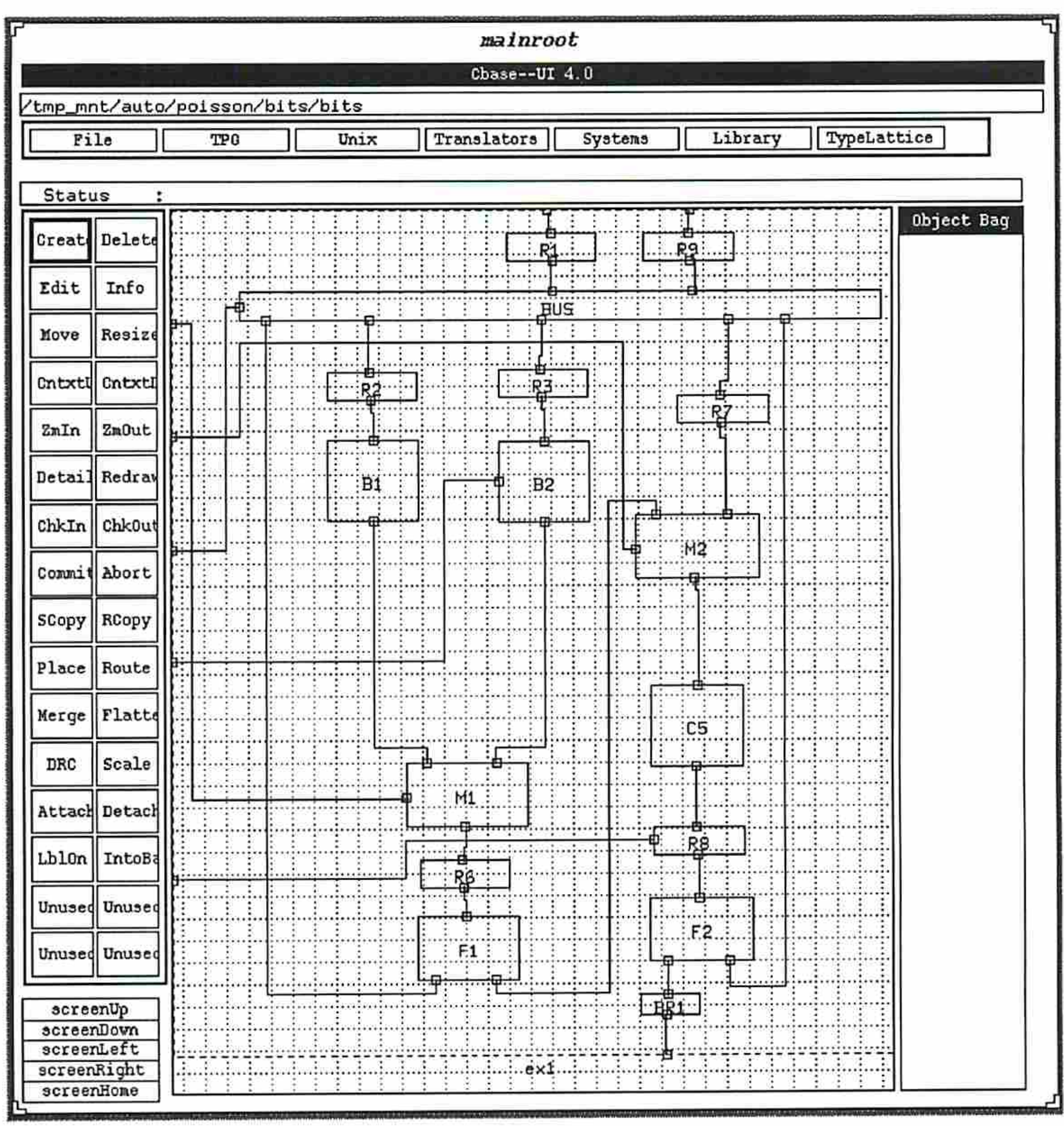


Figure 10.6: Balanced version of `ex1` generated by the BIBS TDM

of combinational cell C_i be denoted by T_i. These estimated test lengths are $T_1 = T_5 = 2^{15}$, $T_2 = T_3 = 2^6$, $T_4 = 2^{13}$, $T_6 = 2^{10}$, and $T_7 = 2^8$. The widths of R_1, R_2, R_{11}, R_{12}, and R_{14} are 12 bits; the width of R_6 is 6 bits; and the widths of the remaining registers are 8 bits. When only combinational kernels are allowed, the representative BISTable circuits generated are summarized in Table 10.8 as circuits #1-16. Note that combinational cells C_1, C_2, C_3, and C_4 are connected in series-parallel and the sequential lengths of the paths between C_1 and C_4 are identical, thus leading to a balanced BISTable kernel B_1 (see Figure 10.8) that consists of C_1, C_2, C_3, C_4, R_5, R_7, R_6, and R_8 when the BIBS TDM is employed. On the other hand, although the combinational cells C_5, C_6, and C_7 are also connected in series-parallel, the sequential lengths of the paths from C_6 to C_5 are unequal. The BIBS TDM converts R_{10} and R_{13} to be BILBO registers and clusters C_6, R_9, and C_7 into another balanced BISTable kernel B_2 (see Figure 10.8). Ten additional BISTable circuits are generated using the BIBS TDM and summarized in Table 10.8 as circuits #17-25.

As can be seen from Tables 10.7 and 10.8, the BISTable circuits generated using the BIBS TDM have significantly less area overhead and often lead to less performance impact than those circuits generated only using combinational kernels. In addition, the increase in test application time for the former circuits is often negligible considering the fast clock rate of testing these circuits. Therefore, the BIBS TDM seems to be a superior BIST methodology for data path circuits when the circuit area and performance constraints are tight.

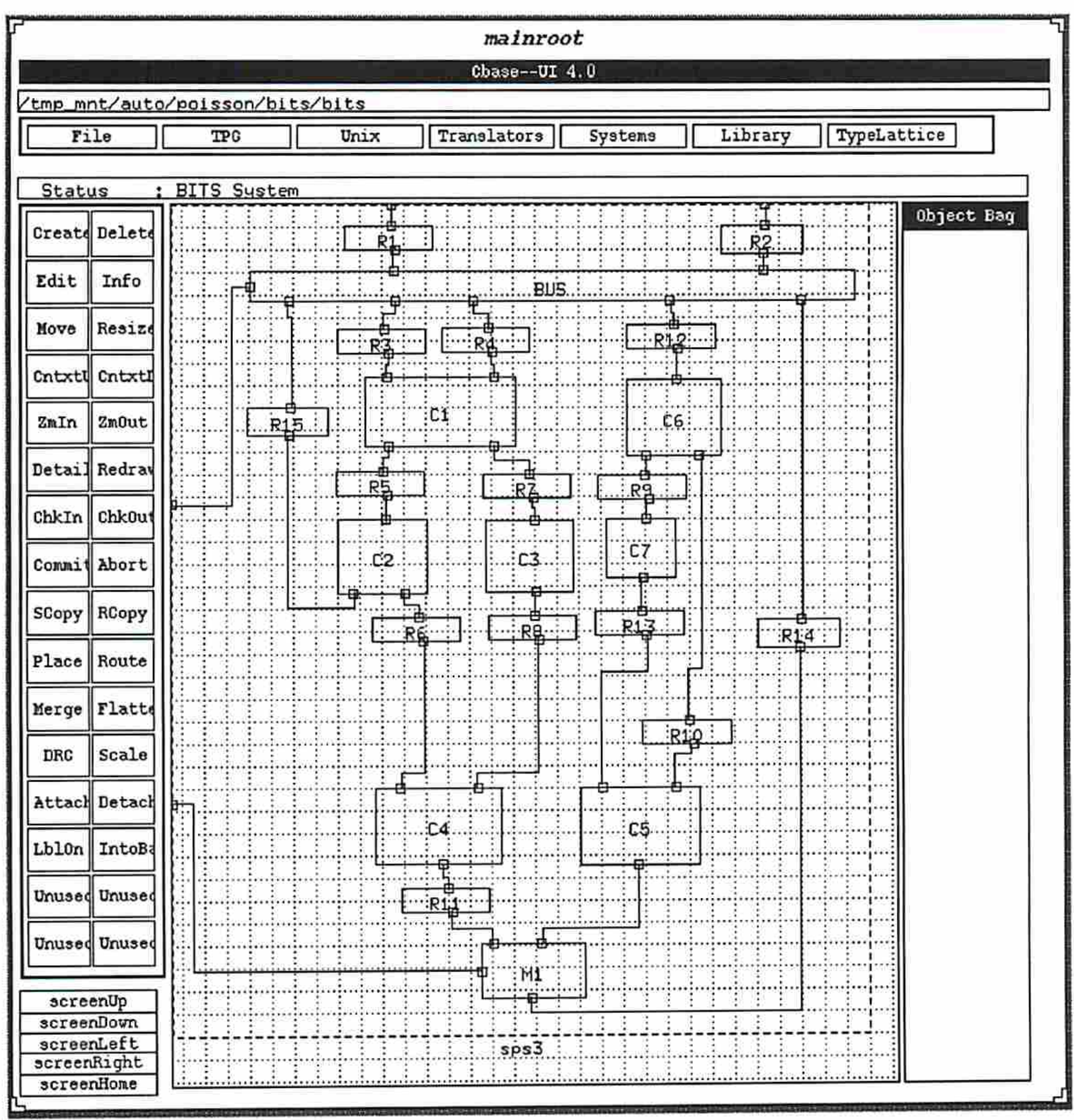


Figure 10.7: Test circuit sps3

Circuit	TDM	Area overhead	Test time	Performance impact	# of test registers	# of test sessions
#1	BILBO	7.79%	106,560	5.33%	7/1/1/0	5
#2	BILBO	8.36%	73,792	5.33%	8/1/0/0	5
#3	BILBO	8.92%	73,792	5.33%	9/0/0/0	5
#4	BILBO	9.23%	65,600	5.33%	7/1/2/0	5
#5	BILBO	8.38%	42,048	5.33%	7/3/1/0	4
#6	BILBO	9.80%	41,024	5.33%	8/1/1/0	4
#7	BILBO	10.71%	34,048	9.33%	7/3/2/0	3
#8	CBILBO	8.50%	106,560	5.33%	6/1/1/1	6
#9	CBILBO	9.06%	73,792	5.33%	7/1/0/1	6
#10	CBILBO	9.94%	65,600	5.33%	6/1/2/1	6
#11	CBILBO	10.67%	42,048	5.33%	5/3/1/2	5
#12	CBILBO	11.90%	41,024	5.33%	5/1/1/3	5
#13	CBILBO	12.11%	33,856	9.33%	5/1/1/3	5
#14	CBILBO	12.43%	40,960	5.33%	4/1/1/4	4
#15	CBILBO	13.34%	32,832	9.33%	5/1/2/3	5
#16	CBILBO	14.50%	32,768	9.33%	3/2/3/4	5
#17	BIBS	2.95%	296,960	0.00%	2/1/1/0	3
#18	BIBS	3.69%	231,424	0.00%	2/2/1/0	3
#19	BIBS	4.04%	294,912	0.00%	2/2/1/0	2
#20	BIBS	4.42%	165,888	0.00%	2/3/1/0	3
#21	BIBS	4.67%	163,840	0.00%	2/1/2/0	2
#22	BIBS	4.99%	100,352	0.00%	5/0/1/0	3
#23	BIBS	5.41%	98,304	0.00%	2/2/2/0	3
#24	BIBS	6.43%	67,584	4.00%	3/2/2/0	2
#25	BIBS	6.60%	65,536	4.00%	2/3/3/0	2

Table 10.8: BISTable designs for circuit sps3

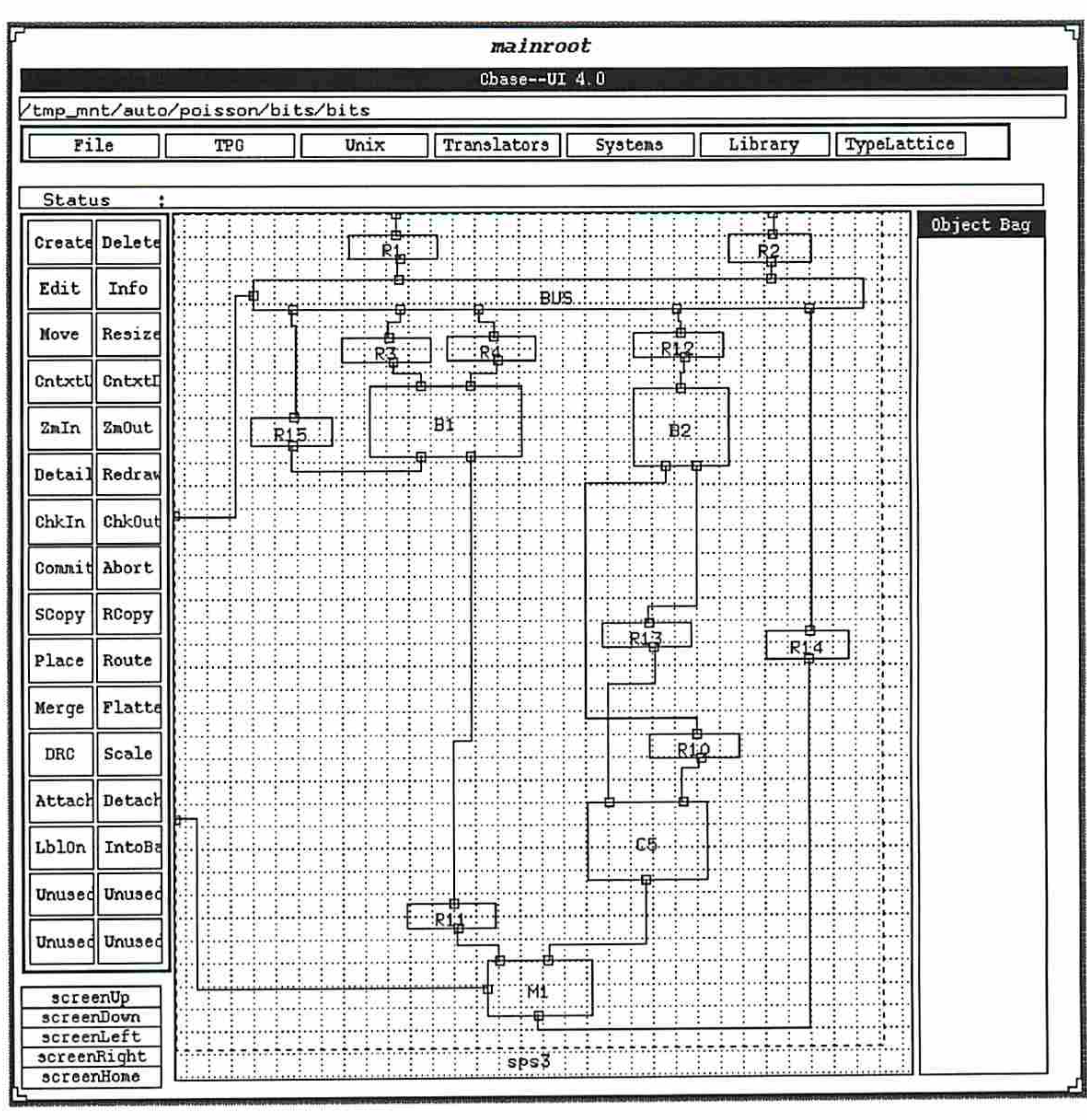


Figure 10.8: Balanced version of sps3 generated by the BIBS TDM

Chapter 11

Conclusion

This thesis has addressed many aspects of the BISTable circuit design problem. An integrated system, BITS, has been implemented that generates a family of BISTable circuits for a CUC. A designer can trade off one BIST parameter for another parameter using the BITS system so that the design constraints are satisfied. The research presented in this thesis can be classified into three major areas: the BIBS TDM and its associated TPG design, test scheduling and the BISTable design space exploration, and the BISTable design selection. Below we summarize the contributions of the research presented in this thesis, and discuss open problems and future research topics related to this thesis for each of these areas.

11.1 BIBS TDM and its Associated TPG Design

Summary of Contributions

- The concepts of k-step functionally detectable fault and k-step functionally testable circuit have been introduced.
- Analyses and experiments have been performed to demonstrate that 1-step functionally testable circuits are often superior to k-step ($k > 1$) functionally testable circuits in terms of fault coverage.

- A novel TDM, called BIBS, has been invented that significantly reduces the test hardware and performance impact associated with a BISTable circuit, and guarantees 1-step functional testability.

- The necessary and sufficient conditions for a circuit to be BIBS testable have been identified.

- Theoretical and experimental results show that the BIBS TDM requires less test hardware than an existing methodology[8] that also guarantees 1-step functional testability. Experiments also show that 100% fault coverage can often be achieved by a small subset of an exhaustive test set when the BIBS TDM is employed.

- A polynomial time procedure has been implemented to generate kernels that are BIBS testable using minimal test hardware when a CUC is connected in series-parallel. A branch and bound procedure has also been developed to generate BIBS testable kernels when the circuit is not connected in series-parallel.

- Polynomial time procedures have been implemented to design TPGs that can generate exhaustive test sets for BIBS testable kernels. This TPG design scheme employs a combination of LFSRs and SRs, and results in relatively low hardware overhead.

- BIBS testable kernels are classified into two categories, namely uni-cone kernels and multi-cone kernels. The test time to guarantee a comprehensive fault coverage for a uni-cone kernel using the TPG generated by the above procedure has been shown to be minimal.

- A new test strategy called functionally pseudo-exhaustive testing is proposed that is applicable to many multi-cone kernels. This test strategy can achieve the same fault coverage as that of 1-step functional testing, and requires significantly less test time. The TPG design process for this test strategy has also been developed.

Open Problems

- From the experiments we observed superior fault coverage for circuits that are 1-step functional testable to circuits that are k-step ($k > 1$) functional testable when random resistant faults exist in the circuits. The difference in fault coverage is negligible when there are no random resistant faults in the circuits. In this case, it may be desirable to allow non-1-step functionally testable kernels since larger kernels can be employed, thus leading to less test hardware. Criteria for identifying acceptable non-1-step functionally testable kernels need to be developed and the estimation of the test lengths needs to be studied further.

- The TPG design for uni-cone kernels guarantees a minimal test time to achieve a comprehensive fault coverage. However, it has been shown that the hardware associated with the TPG design may not be minimal. In addition, the TPG design for multi-cone kernels does not guarantee a minimal test time to achieve a comprehensive fault coverage. More complex procedures presented in [42, 38] may be extended to generate the TPGs having minimal test time and hardware. This problem needs to be studied.

- When a CUC extensively employs switches (such as the AR filter circuits discussed in the previous chapter), the BIBS TDM may not identify complex sequential kernels since switches are not allowed to be parts of the kernels. The BIBS TDM may be extended to selectively include switches in kernels if it leads to a superior design.

- In the BIBS TDM CBILBO registers are only used in self-loops. The constraint may be relaxed when the requirement on the test application time is tight since CBILBO registers can operate as both TPG and SA simultaneously, thus leading to fewer test sessions. In this case the third requirement of the BIBS TDM is not necessary and the problem of identifying BIBS kernels is similar to the balancing problem in the BALLAST partial scan design[35]. The procedures presented in Chapter 5 for identifying BIBS kernels need to be extended to deal with this problem.

11.2 Test Scheduling and BISTable Design Space Exploration

Summary of Contributions

- Conflicts between I-paths have been classified based on the restrictions they impose on a BISTable circuit. Procedures to detect I-path conflicts have been implemented based on a set of rules presented in this thesis.
- A procedure to identify dominating I-paths has been implemented to reduce the design space complexity.
- A procedure to enumerate all possible ways to embed kernels into a TDM has been implemented. The execution of each embedding that resolves the conflicts (if any) of the employed I-paths is also generated by this procedure. A procedure that systematically explores the BISTable design space using such embeddings has been implemented to generate a family of BISTable circuits having a range of values in terms of a list of BIST parameters. A minimal area overhead design, a minimal test time design, and a minimal performance degradation design can also be generated by this procedure.
- An efficient test scheduler has been implemented that employs the partitioned testing with run to completion strategy. The test scheduler is incremental in the sense that a partial test schedule can be generated for a partially specified circuit. Unpromising BISTable circuits can be rejected early in the design process, thus leading to efficient design space exploration. Experiments have been performed and the results demonstrate the superiority of the proposed test scheduler.
- The problem of merging small SA registers into larger LFSRs to reduce the aliasing probability of a BISTable circuit has been studied and formulated as a ILP problem. A procedure to solve this problem has been implemented.
- A procedure to generate a test plan for each BISTable circuit has been implemented. Such a test plan is required by the test controller synthesizer.

Open Problems

- Preliminary research is performed in embedding kernels into COMBILBO TDM. For a complex circuit the embedding process tends to be complicated. The COMBILBO embedding and its associated test length estimation problem need to be studied further.

- A minimal area overhead, minimal test time, or minimal performance degradation design may be more efficiently generated using specialized procedures. Preliminary work has been done in formulating these problems and further research is required to develop such procedures.

- In the test scheduling problem only the partitioned testing with run to completion strategy is considered. Other test scheduling strategies such as non-partitioned testing and partitioned testing can lead to different test application time and test controller complexity for the same BISTable circuit. Studies on the trade-offs when using alternative test scheduling strategies should be performed.

- Other BIST TDMs such as CSTP can also be employed in the BITS system. The embedding process for such TDMs need to be studied and the design space exploration process may need to be extended.

11.3 BISTable Design Selection

Summary of Contributions

- A generic expert selection system, SAESS, has been implemented to help a user make an intelligent choice among objects under consideration. The selection criteria are stored in a domain dependent knowledge base. SAESS is applicable to different selection problems by switching the domain knowledge base. The applicability of SAESS to the BISTable circuit selection problem has been demonstrated.

- SAESS is self-adaptive in the sense that the knowledge base is modified to reflect a user's preferences as the selection process proceeds. The self modification process is formulated as a linear programming problem that is efficiently solved by the revised simplex algorithm.

- To avoid a user from over-constraining a selection problem, a procedure that employs a probability model has been implemented to advise a user in specifying a set of realistic requirements.

Reference List

[1] M.A. Breuer. A Methodology for the Design of Testable VLSI Chips. In *Proc. IEEE Workshop on Test Envirenments*, pages 7–38, 1985.

[2] M. Abramovici, M.A. Breuer, and A.D. Friedman. *Digital Systems Testing and Testable Design*. W.H. Freeman and Company, 1990.

[3] B. Konemann, J. Mucha, and G. Zwiehoff. Built-In Logic Block Observation Technique. In *Proc. Int'l. Test Conf.*, pages 37–41, 1979.

[4] M.S. Abadir and M.A. Breuer. A Knowledge Based System for Designing Testable VLSI Chips. *IEEE Design and Test of Computers*, 3(4):56–68, August 1985.

[5] S. Bhawmik. *An Integrated CAD System for the Design of Testable VLSI Circuits*. PhD thesis, Indian Institute of Technology, Kharagpur, India, 1988.

[6] P.R. Chalasani, S. Bhawmik, A. Acharya, and P. Palchaudhuri. Design of Testable VLSI Circuits with Minimum Area Overhead. *IEEE Trans. on Computer*, C-38(10):1460–1462, October 1989.

[7] K. Kim, J.G. Tront, and D.S. Ha. BIDES: A BIST Design Expert System. *J. of Electronic Testing: Theory and Applications*, 2:165–179, 1991.

[8] A. Krasniewski and A. Albicki. Automatic Design of Exhaustively Self-Testing Chips with BILBO Modules. In *Proc. 1985 Int'l. Test Conf.*, pages 362–371, 1985.

[9] S.P. Lin, C.A. Njinda, and M.A. Breuer. A Systematic Approach for Designing Testable VLSI Circuits. In *Proc. ICCAD*, pages 496–499, Nov. 1991.

[10] M.J. Ohletz, T.W. Williams, and J.P. Mucha. Overhead in Scan and Self-Testing Designs. In *Proc. Int'l. Test Conf.*, pages 460–470, 1987.

[11] L.T. Wang and E.J. McCluskey. Concurrent Built-In Logic Block Observer (CBILBO). In *Int'l. Symp. on Circuits and Systems, vol.3*, pages 1054–1057, 1986.

[12] K. Kim, D.S. Ha, and J.G. Tront. On Using Signature Registers as Pseudorandom Pattern Generators in Built-In Self-Testing. *IEEE Trans. on Computer-Aided Design*, 7(8):919–928, August 1988.

[13] S.P. Lin, S.K. Gupta, and M.A. Breuer. A Low Cost BIST Methodology and a Novel Test Pattern Generator Design. In *Proc. The European Design and Test Conference*, pages 106–112, February 1994.

[14] Rajiv Gupta, W. H. Cheng, Rajesh Gupta, I. Hardonag, and M. A. Breuer. An Object-Oriented VLSI CAD Framework: A Case Study in Rapid Prototyping. *IEEE Computer*, pages 28–37, May 1989.

[15] R. Jain, K. Kucukcakar, M. Mlinar, and A. Parker. Experience with the ADAM Synthesis System. In *Proc. 26th Design Automation Conf.*, pages 56–61, July 1989.

[16] I. Parulkar and M.A. Breuer. Extraction of a High-level Structural Representation from Circuit Descriptions with Application to DFT/BIST. to appear in Proc. 31th Design Automation Conf., 1994.

[17] R. Gupta, R. Srinivasan, and M.A. Breuer. CRETE: Hierarchical Reorganization of Circuits for DFT and BIST. *IEEE Design & Test of Computers*, pages 49–57, September 1991.

[18] R. Srinivasan, S.K. Gupta, and M.A. Breuer. An Efficient Partitioning Strategy for Pseudo-Exhaustive Testing. In *Proc. 30th Design Automation Conf.*, pages 242–248, 1993.

[19] R. Gupta, R. Gupta, and M.A. Breuer. BALLAST: A Methodology for Partial Scan Design. In *Proc. 19th Int'l. Symp. on Fault-Tolerant Computing*, pages 118–125, June 1989.

[20] M.S. Abadir, J. Newman, D. D'Souza, and S. Spencer. Partitioning Hierarchical Designs for Testability. In *Proc. of Intl. test Conf.*, pages 174–183, October 1991.

[21] G. Craig, C. Kime, and K. Saluja. Test Scheduling and Control for VLSI Built-In Self-Test. *IEEE Trans. on Computers*, C-37(9):1099–1109, September 1988.

[22] C.H. Chen. Graph Partitioning for Concurrent Test Scheduling in VLSI Circuit. In *Proc. 28th Design Automation Conf.*, pages 287–290, June 1991.

[23] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[24] C.H. Chen and J.T. Yuen. Concurrent Test Scheduling in Built-in Self-Test Environment. In *Proc. Int'l. Conf. on Computer Design*, pages 256–259, 1992.

[25] W.B. Jone, C.A. Papachristou, and M. Pereira. A Scheme for Overlaying Concurrent Testing of VLSI Circuits. In *Proc. 26th Design Automation Conf.*, pages 531–536, June 1989.

[26] M. Garg, A. Basu, T.C. Wilson, D.K. Banerji, and J.C. Majithia. A New Test Scheduling Algorithm for VLSI Systems. In *Proc. 4th Int'l. Symp. on VLSI System*, pages 148–153, 1991.

[27] M.S. Abadir. Testability Insertion Guidance Expert System (TIGER). In *Proc. Int'l. Conf. on Computer-Aided Design*, pages 562–565, November 1989.

[28] X.-A. Zhu and M.A. Breuer. A Knowledge-Based System for Selecting a Test Methodology for a PLA. In *Proc. 22nd Design Automation Conf*, pages 259–265, June 1985.

[29] X.-A. Zhu. *A Knowledge-Based System for Testable Design Methodology Selection.* PhD thesis, University of Southern California, Dept. of Electrical Engineering, August 1986. CRI Technical Report CRI-86-23.

[30] X.-A. Zhu and M.A. Breuer. A Knowledge-Based System for Selecting Test Methodologies. *IEEE Design and Test of Computers*, pages 41–59, October 1988.

[31] A. Krasniewski and S. Pilarski. Circular Self-Test Path: A Low-Cost BIST Technique for VLSI Circuits. *IEEE Transactions on Computer-Aided Design*, pages 46–55, January 1989.

[32] D. Vir Das, S.C.Seth, and V.D. Agrawal. Estimating the Quality of Manuafactured Digital Sequential Circuits. In *Proc. 1991 Int'l. Test Conf.*, pages 210–217, 1991.

[33] Chih-Ang Chen and Sandeep K. Gupta. BIST Test Pattern Generators for Stuck-Open and Delay Testing. to appear in European Design and Test Conf., 1994.

[34] M. Lempel, S.K. Gupta, and M.A. Breuer. Test Embedding with Discrete Logarithms. to appear in 12th IEEE VLSI Test Symposium, 1994.

[35] R. Gupta. Advanced Serial Scan Design for Testability. Technical Report Tech. Report CRI-91-10, University of Southern California, March 1991.

[36] K. Kucukcakar and A.C. Parker. MABAL: A Software Package for Module And Bus ALlocation. *Int'l. J. of Computer Aided VLSI Design*, 2:419–426, 1990.

[37] L.T. Wang and E.J. McCluskey. Complete Feedback Shift Register Design for Built-In Self-Test. In *Proc. Int'l. Conf. on Computer-Aided Design*, pages 56–59, November 1986.

[38] R. Srinivasan, S.K. Gupta, and M.A. Breuer. Novel Test Pattern Generators for Pseudo-Exhaustive Testing. In *Proc. Int'l. Test Conf.*, pages 1041–1050, 1993.

[39] E.J. McCluskey. Verification Testing - A Pseudoexhaustive Test Technique. *IEEE Trans. on Computers*, C-33(6):541–546, June 1984.

[40] Z. Barzilai, J. Savir, G. Markowsky, and M.G. Smith. The Weighted Syndrome Sums Approach to VLSI Testing. *IEEE Trans. on Computers*, C-30(12):996–1000, December 1981.

[41] D.T. Tang and L.S. Woo. Exhaustive Test Pattern Generation with Constant Weight Vectors. *IEEE Trans. on Computers*, C-32(12):1145–1150, December 1983.

[42] Z. Barzilai, D. Coppersmith, and A. Rosenberg. Exhaustive Bit Pattern Generation in Discontiguous Positions with Applications to VLSI Testing. *IEEE Trans. on Computers*, C-32(2):190–194, February 1983.

[43] K.J. Lee. Switch-Level Test Generation for CMOS Circuits. Ph.D Dissertation, University of Southern California, August 1991.

[44] K.Hwang and F.A.Brigg. *Computer Architecture and Parallel Processing.* McGraw-Hill, New York, 1984.

[45] M.S. Abadir and M. A. Breuer. Test Schedules for VLSI Circuits Having Built-in Test Hardware. *IEEE Trans. on Computers*, C-35(4):361–367, April 1986.

[46] S.P. Lin, C.A. Njinda, and M.A. Breuer. A Systematic Approach for Designing Testable VLSI Circuits. Technical Report Tech. Report CRI-91-18, University of Southern California, July 1991.

[47] M. McFarland, A. Parker, and R. Camposano. Tutorial on High-Level Synthesis. In *Proc. 25th Design Automation Conf.*, pages 330–336, July 1988.

[48] D. Mukherjee. Test Plan Specification for the BITS System. Private communication.

[49] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., New Jersey, 1982.

[50] E. Rich. *Artificial Intelligence.* McGraw-Hill, New York, 1983.

[51] G.R. Grimmett and D.R. Stirzaker. *Probability and Random Processes.* Clarendon Press, Oxford, 1982.

[52] S.P. Lin, M.A. Breuer, and C.A. Njinda. A Self-Adaptive Expert Selection System(SAESS) and its Application to Selection Problems. In *Proc. Third Int'l. Conf. on Software Eng. and Knowledge Eng.*, pages 116–121, June 1991.

[53] E. Koutsofios and S.C. North. *Drawing Graph with* dot - dot *User's Manual.* AT&T Bell Laboratories, Murray Hill, NJ, June 1993.

[54] A. Majumdar and S. Sastry. Test Length Prediction in Random Testing of Combinational Circuits. Private communication.

[55] C.A. Njinda. Estimators for BITS. Private communication.

[56] D.Mukherjee, C.A. Njinda, and M.A. Breuer. Synthesis of Optimal 1-Hot Coded On-Chip Controller for BIST Hardware. In *Proc. ICCAD*, pages 236–239, Nov. 1991.

[57] H. Fujiwara, M.L. Liou, M.T. Sun, K.M. Yang, M. Maruyama, K. Shomura, and K. Ohyama. An All-ASIC Implementation of a Low Bit-Rate Video Codec. *IEEE. Trans. on Circuits and Systems for Video Technology*, 2(2):123–134, June 1992.