Copyright © 1992, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

AUTOMATED TESTING IN AN INTEGRATED SYSTEM DESIGN ENVIRONMENT

by

Kevin Tyrone Kornegay

Memorandum No. UCB/ERL M92/104

10 September 1992

AUTOMATED TESTING IN AN INTEGRATED SYSTEM DESIGN ENVIRONMENT

by

Kevin Tyrone Kornegay

Memorandum No. UCB/ERL M92/104

10 September 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

AUTOMATED TESTING IN AN INTEGRATED SYSTEM DESIGN ENVIRONMENT

by

Kevin Tyrone Kornegay

Memorandum No. UCB/ERL M92/104

10 September 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Automated Testing in an Integrated System Design Environment

by

Ph.D. Kevin T. Kornegay Department of EECS

Abstract

While the performance, density, and complexity of application-specific systems increase at a rapid pace, very few advances are being made in making them more easily testable, diagnosable, and maintainable. Yet in today's VLSI industry, designers are required to produce high quality and more reliable systems. Furthermore, testability, diagnosability, and maintainability are three of the most important factors contributing to system life-cycle costs. Even though testability bus standards, like JTAG Boundary Scan, have been developed to help eliminate these costs, there exists a need for efficient hardware and software tools to support them. Hence, a testability design and hardware support environment for application-specific systems is described which provides a designer with a set of hardware modules and circuitry, that support the Boundary Scan standard and software tools for automatic incorporation of testability hardware, as well as automatic test vector and test program generation. To describe the test features of the various hardware components which make up these systems, a set of high-level languages are provided.

Robert W. Brodersen

Chairman of Committee

To my wife Felicia:

My deepest gratitude goes to my best friend, companion, and wife Felicia for her endless love, support, and patience. She was always there when I needed her. It was her undying faith in me that enabled me to complete my dissertation.

Table of Contents

List of Figures

List of Tables

Acknowledgments

A large number of people have contributed in many different ways to the successful completion of this research. First I would like to thank God for providing me with the spiritual strength I needed to complete this dissertation. My greatest appreciation is of course to my advisor Professor Robert W. Brodersen for his encouragement, guidance, and support during the course of this research. I consider it a privilege to have worked with him. His scholarship and insight was a constant source of invaluable constructive criticisms have greatly enhanced the quality and presentation of this dissertation.

I would also like to thank Professors A. Richard Newton, Charles Stone and Dean David A. Hodges for serving on my dissertation committee. Next I would like to thank Professor Andy Neurether and Chairman Paul Gray for their encouragement, friendship, and sound advise. Of course, I cannot forget my dearly departed father for instilling in me the importance of education and hard work, and my mother for her encouraging words and for making me stay in and study while other kids where outside playing.

My heartfelt thanks go to all the members of *bjgroup*, especially my cubicle mates Anantha Chandrakasan, Jane Sun, and Mani Srivastava. A special thanks to Susan Mellers, Brian Richards, and Phil Schrupp for their invaluable help in this endeavor.

I am also fortunate to be acquainted with people who's friendship I will cherish for the rest of my life they include: Colin Parris, Robin Coger, Greg Uviegahara, Jon Duster, Charles Brooks, Johnathan Reason, and Drs. Chris Gerveshi, Sheila Humphrys, and Jit Kumar. I am also grateful to Tom Boot, Cheryl Craigwell, Peggye Brown, Carole Frank, Kia Cooper, Kevin Zimmerman, and Kirk Thege. Finally, I cannot forget all of my colleagues and friends in Cory Hall who's names I did not mention. This research was jointly sponsored by AT&T Bell Laboratories Cooperative Research Fellowship Program and the Defense Advanced Research Project Agency.

CHAPTER 1

INTRODUCTION

Recent advances in manufacturing and packaging technology have made it possible to design very large, high performance VLSI systems. The process of taking a requirement for a digital system and implementing that system in hardware begins with design and ends with test. In the past, design and test was regarded as two separate steps but today, they are thought of as two closely integrated tasks. So today, designers are faced with this unformidable challenge and they can no longer adopt the old "toss it over the wall" attitude. Some of the barriers they are faced with that compound the testing of these systems are:

- the constant demand for greater integration;
- the widespread adoption of advanced packaging technology like surface mount and multi-chip modules (MCMs) employed on both one-sided and two-sided printed circuit boards (PCBs);
- the smaller distance between pins of surface mount devices;

- the inability to test PCBs via bed of nails access;
- the increasing cost of Automatic Test Equipment (ATE) and associated test fixture;
- the growing gap in speed between the device under test (DUT) and the ATE;
- the increasing consumer demand for high quality, reliability, and maintainability.

Hence, developing a testability design and hardware support environment that helps designers overcome some of these barriers would be of great value. This dissertation addresses issues related to the automation of test in our system design environment. The testability design techniques and dedicated hardware used are intended to reduce the cost of adding test. Furthermore, these techniques are applied to a special class of systems that perform dedicated tasks called application-specific systems.

1.1 What is Design for Testability?

Testing is the process of exercising a system to determine whether it performs its intended functions. If an incorrect response is observed, a second objective of testing is to diagnose why the device behaved incorrectly. Furthermore, in order to meet the stringent demands imposed on today's designers, such as reduced device to market time and reduced cost, testing can no longer be considered as an afterthought, it must now be considered as part of the design process.

Test complexity can be converted into costs associated with the testing process. There are several aspects of this cost, such as the cost of test pattern generation, the cost of fault simulation and generation of fault location information, the cost of test equipment, and the cost related to the testing process itself, namely the time required to isolate and detect a fault. Because these costs can be high and may even exceed design costs, it is important that they be kept within reasonable bounds. One way to accomplish this goal is to make

use of design for testability (DFT) techniques [Williams83]. Testability is a design characteristic that influences various costs associated with testing, while DFT techniques are design efforts specifically employed to ensure that a device is testable. Most DFT techniques require the addition of extra hardware to the design. These design modifications affect such factors such as area, device pin count, and performance.

1.2 Background: DFT Methodologies and Standards

Several well know design for testability techniques are covered in this section. These techniques were developed for chips and printed circuit boards. Since these techniques deal with the total design methodology they are considered structured methods, as opposed to, ad hoc approaches which do not. Most chip-level structured DFT techniques are built upon the concept that if the values in all of the latches can be controlled to any specific value, and if they can be observed with a straightforward operation, then test vector generation can be reduced to that of doing test generation for the combinational circuits between the controlled latches.

Built-in-self-test is the capability of a device (chip, board, or system) to test itself. Building BIST into the design consumes added circuit and slightly increases pin count, but at the same time results in reductions to the costs of testing when compared with an external test using ATE. BIST achieves these savings by reducing the costs of test pattern generation and fault simulation, shortening the test time by running tests at circuit speeds, simplifying the external test equipment, and easily adopting to engineering changes.

To better address problems of board-level and system-level testing, several DFT standards have been developed. The primary objective of these standards is to ensure that all of the components of a board and/or system contain common DFT circuitry that will make test

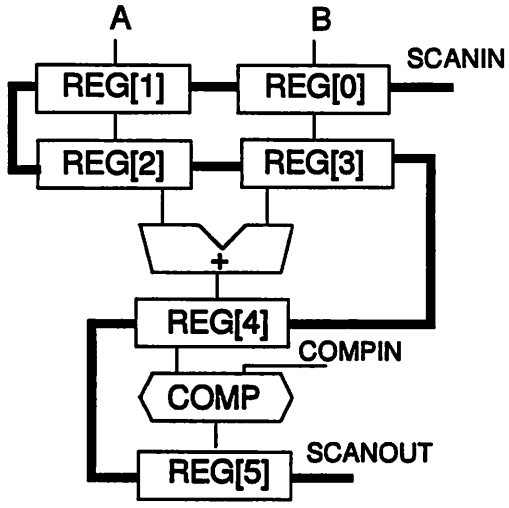


Figure 1-1: Scan path example.

development and testing of the system and its components more effective and less costly.

1.2.1 Scan Path

The scan path [Eichelberger77][Funatsu75][Williams73] methodology is the most widely used method for testing those parts of the circuit that are constructed of clocked D-type flip-flops interconnected by combinational logic. As illustrated in Figure 1-1, it is based on converting the circuit's D flip-flops into a serial scan path chain denoted by the thick black line threading the circuit flip-flops. When the circuit is placed in test mode, the circuit is configured as a shift register and test data can be shifted in on every clock cycle. By returning the circuit to normal mode for one clock cycle, the contents of the register are applied to the combinational circuitry and the results are captured at the register inputs. If the circuit is then placed in again, the results of the proceeding test can be shifted out for examination.

1.2.2 Built-In-Self-Test

Built-In-Self-Test (BIST) [McCluskey85a,b] techniques fall into two categories, off-line (or nonconcurrent) and on-line (or concurrent). Off-line BIST requires a mechanism for supplying test patterns to the device under test and a means for comparing the device's responses to known good response as illustrated in Figure 1-2. Additionally, both mechanism and means must be compact enough to implement. There are many ways to generate stimulus but, the two most widely used ones are called exhaustive and random testing.

Stimulus generation in exhaustive testing, the test length is 2^n tests, where n is the number of inputs to the circuit. Since all possible test patterns are applied, all possible single and multiple stuck faults are detected (excluding redundancies). The tests are generated with any process that cycles exhaustively throughout the circuit input space, such as a binary counter or an n-stage autonomous linear feedback shift register (ALFSR). An ALFSR is a series connection of D-type flip-flops with no external inputs and with all feedback

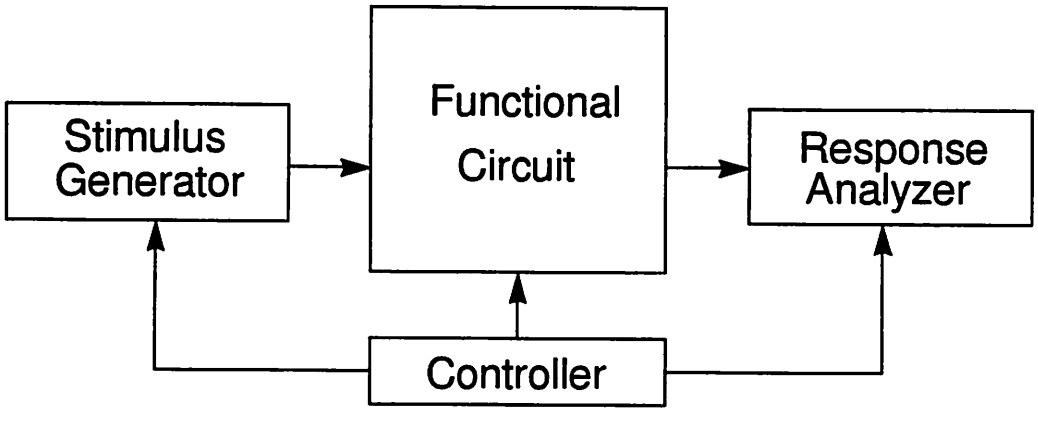


Figure 1-2: General form of an off-line BIST structure.

provided by means of exclusive-or gates. Exhaustive testing for chips with high input pin count requires relatively long test times, but in [Bozorgui80] it is suggested that circuits can be added to partition such structures into subcircuits, each of whose input pin count is low enough to permit exhaustive testing in a reasonable amount of time. Random testing implies the application of a randomly chosen subset of 2^n possible input patterns. A guarantee of the test coverage for the subset can be obtained by running the tests against a fault model. The number of the applied tests or the size of the subset is constrained by the economically allowable test time. While circuit partitioning is not needed, some logic modification may be necessary to ensure adequate coverage from the limited test set. A linear feedback shift register is the typical choice for a random test generator since its output data is approximately random.

Response analysis on-chip storage of a fault dictionary (all test inputs with the correct output response) requires too much memory to be a practical method. The simplest practical method for analyzing the output response is to match the outputs of two identical circuits. The cheapest way to do this is to compress the output responses before comparing them. The compressed response is signature of the device under test, and comparison is made to the precomputed and stored reference signature. The most widely used data compression method is signature analysis which uses a linear feedback shift registers and the signature is the state of the register following the completion of a test [McCluskey85a,b].

1.2.3 Boundary Scan Standard

The Boundary Scan [IEEE90a][Maunder90] standard consists of a dedicated serial test bus which resides on a board, a protocol which controls the I/O pins that connect the chips to the test bus, and control logic that resides on chip to interface the test bus to the DFT

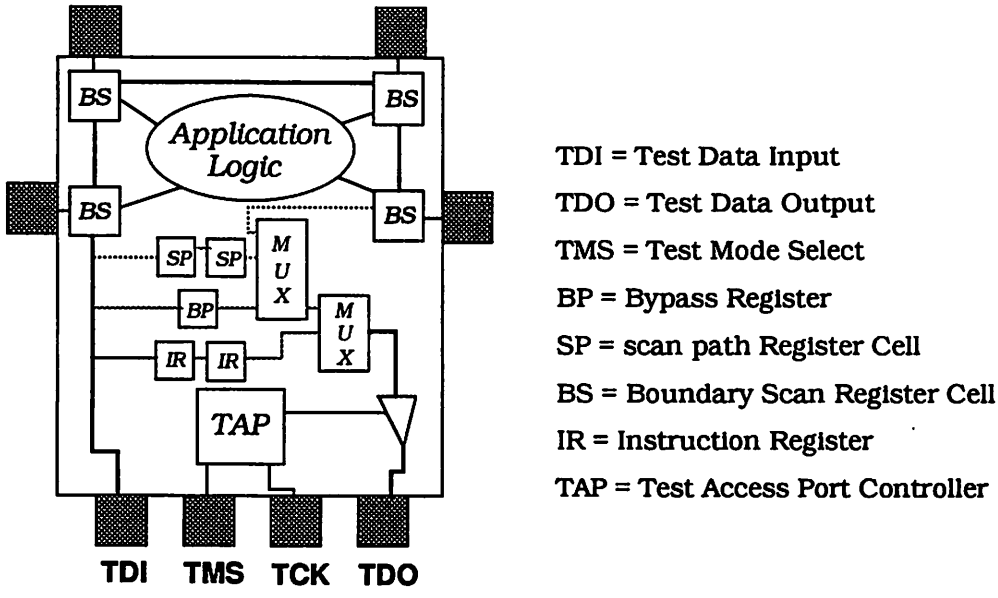


Figure 1-3: Boundary Scan architecture.

circuitry residing on the chip. The primary reasons for Boundary Scan are to allow efficient testing of board interconnect, and to facilitate isolation and testing of chips either through the test bus or with additional circuitry.

With Boundary Scan, chip-level testing can be supported at the board-level by simply connecting Boundary Scan register cells between the chip's application logic and I/O pins as shown in Figure 1-3. There are two major components associated with this standard, namely Boundary Scan register and the test access port (TAP) controller. The application logic represents the normal chip design prior to the inclusion of logic required to support the standard. This circuitry may include Scan Path or BIST hardware. If so, the scan paths are connected via the test bus circuitry to the chip's scanin and scanout pins. The remainder of the test bus circuitry consists of the boundary scan register, a 1-bit bypass register, and a n-bit instruction register. The test bus consists of the test clock (*TCK*), the test mode signal (*TMS*), the test data input (*TDI*) signal, and the test data output (*TDO*)

signal. Test instructions and test data are sent to a chip over the *TDI* line, while test results and status information are sent from the chip over the *TDO* line. Control of the test bus circuitry is carried out by the TAP controller which receives its commands from the *TMS* line.

1.3 Previous Work

In recent years, there has been a great deal of work in the areas of DFT automation systems, hardware controller systems, and custom chip solutions, for example, [Abadir85,89][Agrawal84][Beenker89][Emori90][Fasang85][Fung86][Geewala89][Hallenbeck89][Lien88][Lien89][Lien90][Samad86][Samad89][Swan89][TI90][Yau90]. While it is beyond the scope of this dissertation to examine all of the work, some of the related work will be discussed in the section that follows.

1.3.1 Design Automation Systems for Testability

Some CAD systems, like silicon compilers, automatically incorporate of testability by following stringent guidelines to add special purpose test hardware, while other systems, are more dedicated and use artificial intelligence techniques to guide the designer in selecting DFT solutions. The Test Engineer's Assistant (TEA) [Hallenbeck89] is an example of such a system.

Test Engineer's Assistant

TEA is a CAD environment developed to provide the knowledge base and tools needed by a system designer for incorporating testability features into a board design. TEA helps the designer meet the requirements of fault coverage and ambiguity group size. Fault coverage is defined as the percentage of faults that can be detected out of the total

population of all single stuck-at faults of a device under test with a particular test set. Ambiguity group is defined as the smallest hardware entity in a given level of the system design hierarchy (that is, board, subsystem, and system) to which a fault can be isolated. TEA interfaces to commercially available or prototype, beta-site tools to create an environment in which the designer can perform design capture, functional verification, design for testability, fault simulation, functional verification, and test program generation for a particular automatic test equipment system.

The design methodology used in TEA addresses testability issues at all stages of design (preliminary, detailed, and final) and at each level of the system hierarchy. Hardware and software resources are identified during the preliminary design stage. In contrast to traditional design practices, test resources are also determined at this stage. During detailed design, specific functions of system resources are identified and verified through simulation and checked against system requirements, including testability. Trade-offs are made at this point to ensure that system requirements are met. TEA aids the designer in identifying and implementing test resources and verifying that they will meet system test and diagnosis requirements.

TEA uses the hierarchical test methodology shown in Figure 1-4, that is composed of a set of subsystems communicating through a system bus. Each subsystem is composed of a number of boards communicating through a subsystem bus during normal operation and through a test bus in test mode. Each board interfaces to the test bus through a test interface. The test interface receives test data and control information from the test bus and uses this information to initiate tests and receive results by controlling the chip's local test hardware on the board. The test interface can be a single chip and it can directly interface to standard testability buses, such as the Boundary Scan test bus for communicating test data to and from the board. The test data is generated by a Test

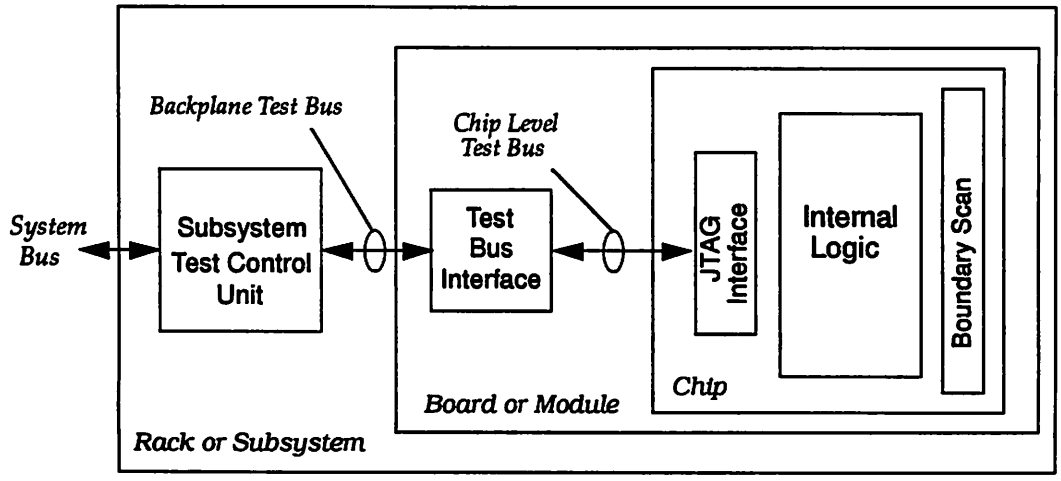


Figure 1-4: Hierarchy of test and maintenance buses.

Control Unit, a subsystem used exclusively to provide test data for every device under test in the system and to analyze the results. This unit can be embedded in the system, or its function can be performed by automatic test equipment.

1.3.2 Test Hardware: Custom Test Controllers

Several custom test controllers have been proposed. These systems were developed to function in a structured testability hierarchy like that shown in Figure 1-4 and require dedicated software to configure them for test and debug operations. The advantages of a hierarchical test methodology are interoperability at each level of subassembly due to standardized test interfaces and reduced overall test and maintenance costs.

Module Test and Maintenance Controller

Lien and Breuer describe the Module Test and Maintenance Controller (MMC) [Lien88][Lien89] system that is capable of controlling the self-test process of a board by

accessing each chip's test structures through a Boundary Scan test bus. It is intended to be part of a hierarchy of test controller that are embedded into a target system's physical hierarchy. In their hierarchy, each testable chip contains an on-chip test and maintenance controller (CMC); each testable board contains a module test and maintenance controller; each testable subsystem contains a subsystem test and maintenance processor; and each system has a system test and maintenance processor. The architecture for an MMC consists of a 16-bit general or special purpose processor, a ROM, a RAM, a test channel, test and maintenance processor with a Boundary Scan interface, and a bus driver/receiver, which supports an expansion bus.

The major functions of the processor are listed below:

- transfer data between memory and test channels
- compare test results with stored good results
- transfer data between memory and expansion units
- execute test and/or diagnostic programs
- transfer data between memory and the subsystem maintenance processor

Once initialized by the processor, the primary function of the test channel is to control the Boundary Scan test bus. Other functions of the test channel are listed below:

- serve as a Boundary Scan master
- transmit instructions to and receive status information from chips
- generate and transmit pseudorandom test data and receive compact results
- transmit deterministic test data to and receive test results from chips
- generate interrupts and also direct interrupts between chips and the processor
- and keep count of the number of tests applied and the number of bits in each test vector or instruction that is transmitted.

The bus driver/receiver is a bidirectional interface to the MMC, and the RAM and ROM are used to store test data and seed vectors required for BIST operations respectively.

1.4 Summary

The work described in the previous section only provided partial solutions to the two part problem of automatically incorporating DFT features into application-specific systems and providing hardware and software to support them. The uniqueness of the work presented in this thesis is that it is a fully integrated solution to the system test problem that deals with both pre-design and post-design test issues in an automated fashion.

In the remaining chapters are described the details of the hardware and software for the testing environment. Chapters 2 presents an overview an overview of the SIERA design system and the test strategy employed in the system. The chip and board level test hardware is discussed in Chapters 3 and 4 respectively. In Chapter 5, a detailed explanation of the software used for designing and controlling the test hardware. Test applications and some actual test sessions are presented in Chapter 6. Finally, Chapter 7 presents conclusions and some suggestions for future work.

CHAPTER 2

THE SIERA DESIGN ENVIRONMENT

Advances in VLSI technology has led to the creation of chips which resulted in a complexity that resulted in a bottleneck in the chip's overall development time. This bottleneck was alleviated by the use of silicon compilers which produce the physical information required to fabricate chips from higher level descriptions of the design, these could be a symbolic layout, a circuit schematic, a behavioral description of a microarchitecture, an instruction set, or an algorithm for signal processing.

The compiler then transforms this high level description into a physical representation required by the fabrication foundry. This transformation occurs in several steps. For example, a compiler might transform a behavioral description of a design into a logic gate level representation. A major advantage of this approach is that designers can work at higher levels of abstraction without having to know specifics about the IC design and process technology. Another and probably most important advantage of this approach is the ability to rapidly produce chips. Further advances have also led to the creation of very

complex systems. Even though these systems contain hundreds of components, tools that support the integration of these components to make up the system are still in a primitive state. Additionally, these same tools do not exhibit usability and rapid prototyping features. One such CAD environment that does exhibit these features is SIERA [Srivastava91][Sun91][Srivastava92]. An overview of SIERA is presented in this chapter. along with a discussion of the test strategy employed by SIERA along with the associated testing environment.

2.1 Overview of SIERA

SIERA is an integrated CAD environment for the design of complex, application-specific systems, where a system is a set of hardware modules that interact with each other and the environment to collectively perform some function. In the context of this system, a module can be a single chip or group of interconnected chips. SIERA's origin stems from the LAGER [Rabaey86][Shung91][Brodersen92] system, a custom chip design environment. Some of the transformation steps used in LAGER are also used in SIERA which include behavioral-to-structural and structural-to-physical. Dedicated tools were developed to tackle the tasks associated with each transformation step. Many problems inherent in system design that do not exist in chip design are addressed by SIERA such as:

- behavioral representation of systems
- simulation of behavioral representation
- structural representation of systems
- simulation of structural representation
- physical representation of systems
- and simulation of physical representation.

SIERA was developed to provide a CAD environment that is capable of synthesizing an architecture, using both hardware and software modules, to implement the a system specified in the form of a process network as described in [Srivastava91][Sun91][Srivastava92]. Figure 2-1 illustrates a simplified high level view of

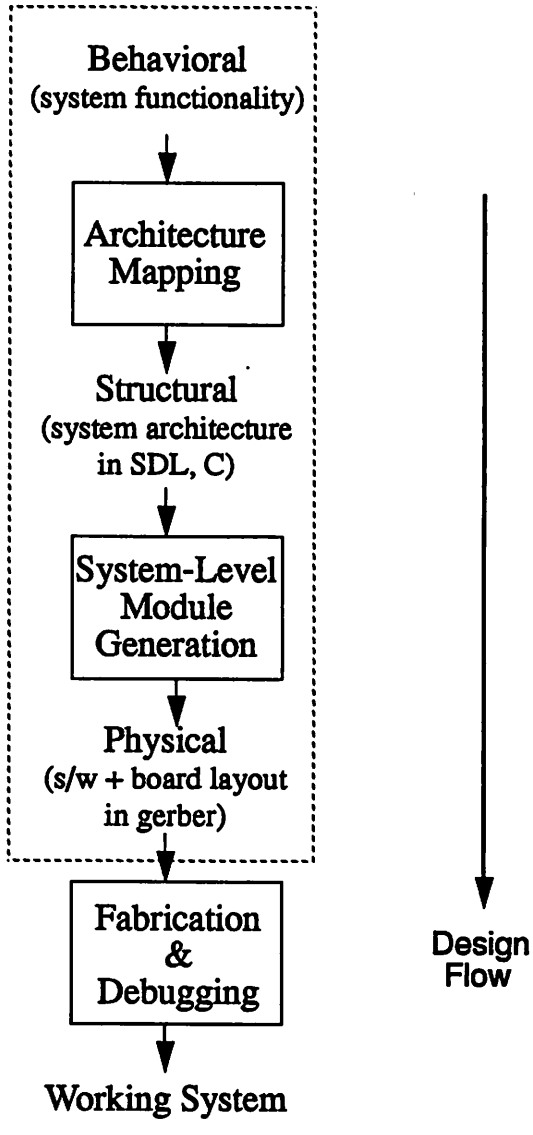


Figure 2-1: High level view of SIERA.

SIERA. First, a behavioral representation of the design is mapped to an appropriate architecture producing a structural representation which is then manually partitioned and mapped onto a four-layer system architecture template as shown in Figure 2-2. The two lowest layers consist of custom boards where each board contains one or more software programmable processing modules based around programmable digital signal processors, and running real-time customizable OS kernels. Each processing module in turn coordinates a number of application-specific slave modules which can be either software programmable or dedicated hardware modules. The custom boards reside in a back-plane bus, typically VME, and are slaves to an off-the-shelf single-board computer which also runs a real-time OS kernel. This constitutes the third layer. The processing modules on the custom boards interact with the master single-board computer through a standard, parameterized software and hardware interface. The VME master single-board computer in turn, communicates with a UNIX workstation, which makes up the final layer, using standard (Ethernet) and software protocols.

The software modules can be mapped to the top three layers which can be mapped to the top three layers. For example, a software module is mapped either as a process on the workstation, a process on the VME master single-board computer, or as a process on the DSP processing module residing on a custom board. Layers 2 and 3 are used for process with increasing real-time requirements while, the workstation is used for non-real-time front-end or interactive processes. The architecture mapping as specified by the user is accomplished by selecting a library module for each block in the block diagram. An implementation of a block is either in the form of program code meant for execution as a software process, or a behavioral or structural representation of a dedicated hardware process.

Designs in SIERA are managed by the Design Manager called DMoct, which automates

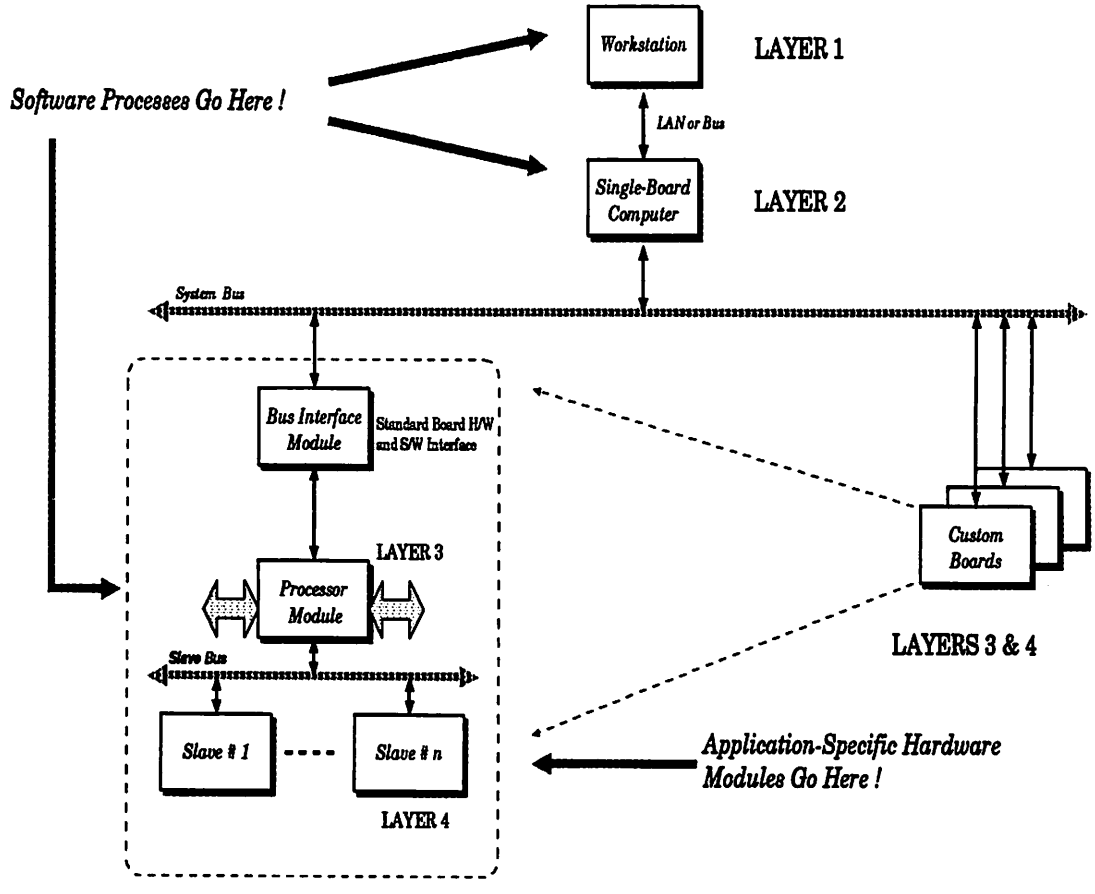


Figure 2-2: Layering topology for architecture template.

the generation of a hierarchical system comprised of parts created by a large variety of tools. It accesses these tools in the proper order to preserve the design hierarchy contained in a textual or schematic representation of the design. The design flow as managed by DMoct is shown in Figure 2-3. The textual representation is written in a structural design language called SDL [Brodersen92][Shung91] which has lisp like constructs, furthermore, design constraints and parameters are also passed along with this representation. After parsing the SDL file, an initial representation of the design called the structure-master view is created and stored in the OCT object-oriented data base [Octools][Harrison86].

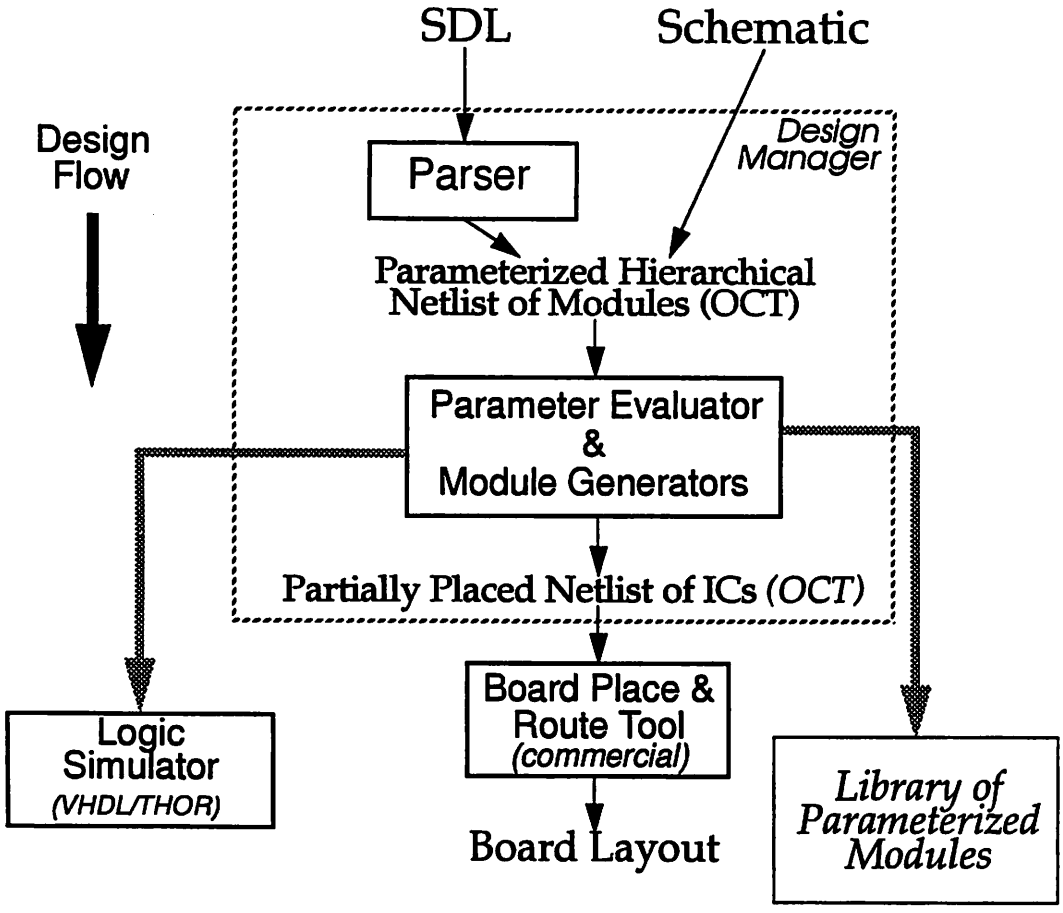


Figure 2-3: Typical PCB design flow managed by DMoct.

OCT was developed specifically for electronic CAD applications and it offers a straightforward mechanism for storing all information pertinent to an evolving design. Following this step, the parameters are evaluated and together with structure-processor tools (which are dedicated tools that only operates on structure-master views), produce a structure-instance view which is an expanded view of the total design as represented in OCT. Finally, layout generation tools operate on structure-instance views to produce output files containing physical geometry and implementation specific information required to fabricate a chip, MCM, or board.

2.1.1 System Design Methodology

Embodied within the SIERA framework is a vertically integrated design methodology that supports the development of application-specific systems at all levels from the high level description to the board implementation and software generation. SIERA uses an "application-driven" approach which is based on examining actual implementations of example systems, developing an initial design methodology, automating and improving the design methodology through experience gained from the example systems. This approach is different from the classical "tool-driven" method which develops a design methodology after developing a set of automated tools. The design methodology used in SIERA consists of two parts, namely Module Generation and Architecture Generation. Module Generation will be discussed here, any information regarding architecture and software generation can be found in [Srivastava92].

Module Generation is the physical implementation of a system from an architecture comprised of hardware modules, where emphasis is placed on generation of multi-chip board level hardware modules. A major feature of this hardware generation strategy is a library consisting of reusable parameterized board-level sub-system modules that can be integrated into a custom board design. These modules can be used for communication, signal processing, data acquisition, or testing applications. Some modules are fixed and while others can be customized for a given application via parameters provided by the designer. For example, parameters can determine the type and size of a memory module. The combination of the sub-system library and the hardware modules provides an environment unmatched by commercially available board design tools. A typical board design may consist of a number of these modules connected together to achieve the desired functionality. Module generators produce sub-system netlists and component placement information which are all processed by DMoct to produce a final board netlist.

The board netlist is then processed by a structure-processor called oct2rinf, which produces an output file that is compatible with a commercial router.

2.1.2 Hardware Module Generation

The primary objective of the hardware module generation environment was to provide an environment that can handle arbitrary hardware architectures implemented as custom boards using custom and commercial ICs, as well as, the capability to explore alternative implementations. Module generation was founded on the basis of pre-existing ASIC generators, like those in LAGER, that automatically produce circuitry required to implement dedicated chip level macro-functions and then, tie them together to achieve the desired functionality. These same concepts also proved to be extremely useful at the board level, where one or more custom and/or commercial ICs are grouped together to implement a complex function. This environment also uses SDL to describe the designs, OCT to store the design information, and DMoct to manage the design flow. Additionally, chip level module generators and behavioral tools are also available to the board designers. As a result, a complex board design can be represented as a netlist of high-level modules that are maintained in a library consisting of parameterized reusable modules (adders, multipliers, etc.) or as a behavioral description (FSM controllers, decoders). The final step toward completing a board design is the layout, for this, several layout generation tools were developed. This section is only intended to present the reader with an overview of the hardware module generation environment, a more complete discussion is in [Shung89].

2.2 Test Strategy used in SIERA

Traditional approaches to system test often employ a three level strategy. First, an

engineer with very little or no test experience runs a system self test to quickly determine whether the system is operating correctly. If a failure occurs, the engineer runs additional tests on each of the boards that make up the system, to locate the faulty board and then replaces it. The faulty board is then returned to the manufacturer, where their test engineer, who is experienced, uses sophisticated ATE to locate the faulty chip and replaces it.

The traditional approach presents many problems, three of the most important ones are listed below:

1. the additional cost of shipping the boards out for repair;
2. the time required to develop a good functional test vectors;
3. difficulty of duplication of the problem.

The difficulty of the duplication problem refers to the situation where one level of test indicates a failure and that failure is transparent at the next level. This in part is due to intermittent faults and/or differences in test procedures used at various levels of testing. The causes of these problems, which have been identified [Breuer85], are environmental dependency, which means that a failure is caused by environmental conditions like temperature and vibration, false alarms caused by design errors or transient faults, incompatibility of tests, which is caused by the use of inconsistent testability techniques at different levels of the system hierarchy, and faults in the test hardware.

The approach described above is adequate for high volume production environments but insufficient for low to medium rapid prototyping environments like SIERA. Hence, the test strategy used in SIERA should eliminate or at least reduce the problems mentioned above. The strategy used in SIERA should be able to support testing at all levels of the system's hierarchy, support existing testability bus standards, produce test vectors, have a facility to initiate and execute tests and provide a means to integrate DFT techniques into the design process. This will result in a complete solution that deals with the design, as

well as, the testing of a system. This approach is contrary to the approaches described in the Previous Work section in Chapter 1, where the authors only presented partial solutions.

Embedded within SIERA is a test environment that fulfills the above requirements, where the designer, has available, testable hardware modules at both the chip and board level that implement a DFT methodology or testability bus standard, software tools that tie these modules together, test languages that describe what DFT modules are used and how to use them during test, and a dynamically reconfigurable custom controller board that is used to control and access the devices (chips and boards) which contain the testable hardware. Specific details and issues regarding the design, implementation, and application of each component of the test environment is the subject of this dissertation. Each component is addressed in the chapters that follow, but a discussion on how we integrate test into SIERA is warranted.

2.2.1 Integrating Test into SIERA

The DFT techniques used in SIERA must tackle test problems at both the chip and board levels. Furthermore, these techniques must not significantly impact performance or area. While preserving these objectives, test is integrated into SIERA in two phases. The first phase involves identifying the building blocks that comprise a chip or board. For example, there are three fundamental units of logic that used to implement a chip, namely combinational logic, registers, and random access memories. At the board level, devices are categorized as either Boundary Scan or non-Boundary Scan components. After the fundamental system components have been identified, specific test methodologies, in particular, those described in Chapter 1, are chosen for each component in the second phase. At the chip level, the registers are configured as a Scan Path and used to test the combinational logic, while memory is tested using the BIST technique. The Boundary

Scan devices are chained together forming a Boundary Scan path where these devices are used to test the board interconnect, as well as, provide access to chip level Scan Path and BIST functions. Dedicated test hardware modules have been developed that implement Scan Path, BIST, and Boundary Scan using the hardware module generation feature in SIERA. Special languages and software tools have also been developed to support these modules, all of which are arranged in a test library. Figure 2-4 illustrates how these features are integrated into SIERA. The design, implementation, and some examples of these modules and the Test and Diagnosis system are discussed in the next Chapter 4, while the software tools that support this hardware are discussed in Chapter 5.

2.3 Summary

An overview of SIERA, the system design methodology, the hardware module generation environment, our system test strategy, the test environment and how it's integrated into SIERA have been presented. The test methodologies used in SIERA are only intended to assist the designer in two ways: (1) to help verify the complete functionality of their system; and (2) to isolate the faulty device down to the logic gate level. Moreover, our approach allows us to deal with testability as part of the design process not as a post-design process.

The test methodologies used in SIERA were chosen because of their ease of implementation and suitability in a rapid prototyping environment. Our approach does not require that the designer be intimately aware of all of the DFT techniques that exist and their implementation, but rather allows the designer to work at a high level with libraries and software tools to implement the actual test. Finally, the role of the chip level macrocells and the board level modules is to help the designer incorporate testability features into the design to meet system test requirements. With a system designed using

these modules, the designer can isolate system faults down to a single chip. This capability is particularly attractive in terms of repair times and repair costs.

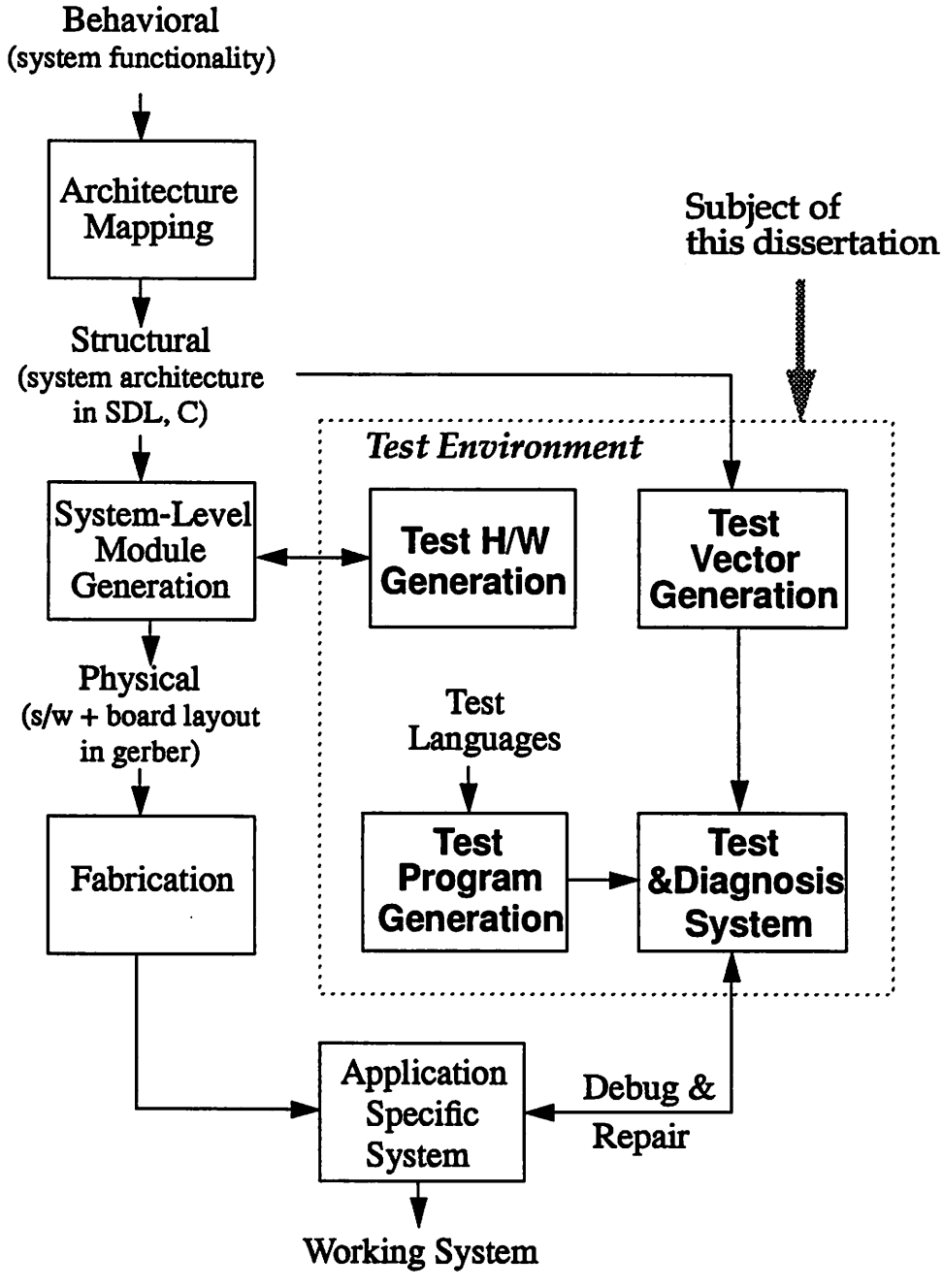


Figure 2-4: Structure of SIERA including test environment.

CHAPTER 3

TEST HARDWARE - CHIP LEVEL

In 1985, an ad hoc group composed of key semiconductor manufacturers formed to establish a solution to the problems associated with board level testing. This concerted effort involved the producers of both chips and board level products. The solution they came up with eventually led to the development of a standard chip level test architecture called Boundary Scan that, not only, solves the board level test problem, but also allows designers to add test features, like Scan Path and BIST, to meet their own requirements. To be compatible, a chip must have certain basic test features which are outlined in the standard specification.

This chapter presents an overview of the Boundary Scan standard, addresses implementation issues associated with integrating Scan Path and BIST with Boundary Scan and examines design versus test cost trade-offs.

3.1 The Boundary Scan Standard: An Overview

Two continuing trends are increasingly making it more difficult to test printed circuit boards:

1. Increasing complexity - As chips become more complex, so does the task of generating tests for boards that use them. For functional testing, test generation times are significantly longer, due to the need to propagate test data through some chips while tests are applied to others. Test lengths also increase as complexity increases.
2. Greater miniaturization - The use of multi-chip modules as well as surface mount packaging technology, particularly when coupled with double-sided component mounting reduces board geometries making boards more difficult to probe using traditional bed-of-nails access.

The purpose of the Boundary Scan [IEEE90a] standard is to provide the basis for solutions for these problems. Boundary Scan solved these problems by eliminating the need to physically probe a component's I/O pins by implementing an electronic probe inside the component's I/O pins. This section describes the principal features of the Boundary Scan Macrocell.

The standard architecture requires of three major circuit blocks shown in Figure 3-1 and described below:

TAP Controller - a finite state machine that responds to control signals supplies through the test access port (TAP) and generates the clocks and control signals required for correct operation.

Instruction Register - an n-bit shift register whose contents determine which test is to be executed

Test Data Register - an n-bit shift register that applies the test stimuli or conditioning values required by a test. At the end of a test, the results in the test data register can be shifted out for examination. This register, for example can be implemented as a Scan Path register.

3.1.1 Test Access Port

These circuit blocks are connected to a TAP which includes four or, optionally, five

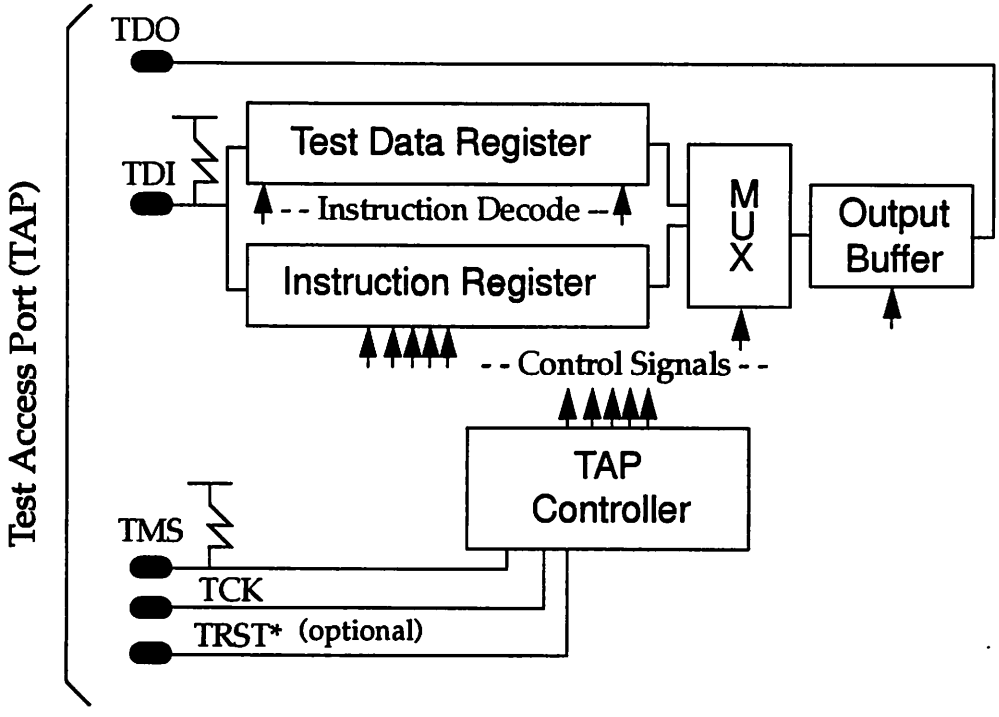


Figure 3-1: Top level view of Boundary Scan test logic.

signals used to control the operation of tests and the application of test data and instructions. The TAP consists of:

Test Clock Input (TCK) - allows test operations to be synchronized between various chips on the board.

Test Mode Select (TMS) - Operation of the test logic is controlled by a sequence of 0s and 1s applied to this input. The sequence on TMS directs the execution of the TAP controller.

Test Data Input (TDI) - Data applied to this serial input is fed either into the instruction register or into the test data register depending on the sequence applied to the TMS pin.

Test Data Output (TDO) - The serial output from the test logic is fed either from the instruction register or from the test data register. During shifting, data applied to TDI will appear at TDO after N clock cycles, where N is the register size. When data is not being shifted through the register, TDO is in a high impedance state.

Test Reset Input (TRST*) - This optional input is used to reset the test logic when a 0 is applied.

The TDI, TMS, and TRST* inputs are equipped with a pull-up resistor so that when they are not driven by an external source, the test logic always sees a logic 1. In the case of the TMS input, this ensures that the TAP controller always starts in the correct state after 5 clock cycles.

3.1.2 TAP Controller

A key goal during the development of the Boundary Scan standard was to keep the number of pins in the TAP to a minimum because chip designers are always reluctant to allocate additional pins for test purposes. The TAP controller achieves this goal with a 16-state finite state machine that implements a serial test protocol. The state diagram for the TAP controller is shown in Figure 3-2. Note that all data register operations end with a '_DR' and all instruction register operations end with a '_IR'. State to state transitions occur on the rising edge of TCK. The 0s and 1s shown adjacent to the state transition arcs indicate the TMS value that must be present together with a rising edge on TCK, for that particular transition to occur. Eight of the sixteen controller states determine operation of the test logic, allowing the following test functions to be performed:

Test-Logic-Reset - All test logic is reset, allowing normal operation of the chip to occur without interference. Regardless of the starting state of the TAP controller, this state can be reached by applying a 1 to the TMS input for five clock cycles. Alternatively, if TRST* is provided, it can be used to asynchronously reset the TAP controller at any point during operation.

Run-Test/Idle - The operation of the test logic in this controller state depends on the instruction in the instruction register. For example, if an instruction activates a self-test, then the self-test will run in this state. If the instruction happens to be one that selects a data register for scanning, then the test logic will be idle.

Capture-DR - Each instruction must identify one or more test data registers that are enabled to operate during test mode when the instruction is selected. In this state, data

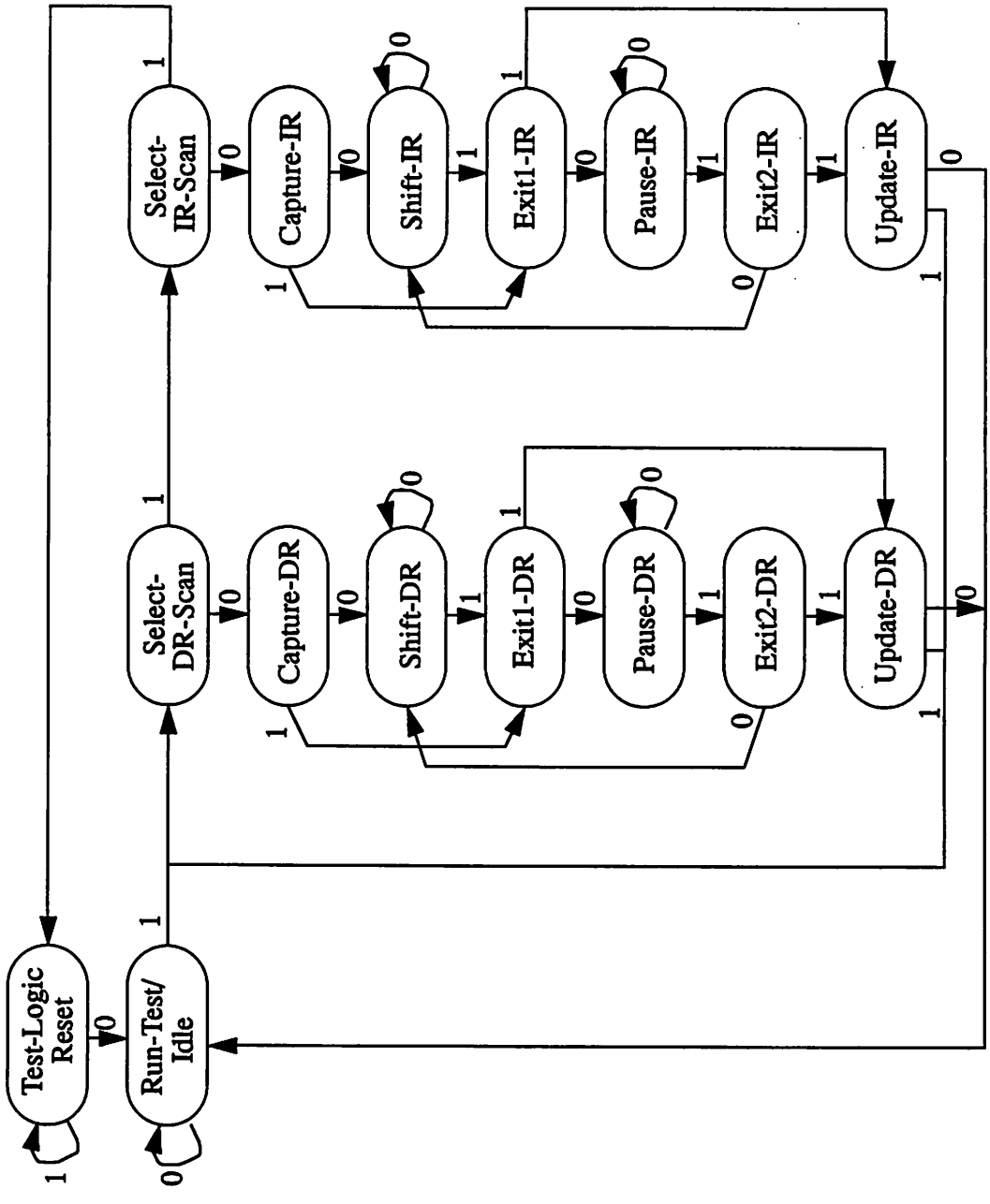


Figure 3-2: TAP controller state diagram.

is loaded from the parallel inputs of the selected test data registers into its shift register paths.

Shift-DR - Each instruction must identify a single test data register that is to be used to shift data between TDI and TDO in this state. Shifting allows the results of the previous test to be examined while applying the next test.

Update-DR - This state signifies the end of the shifting process. Some test data registers may contain latched parallel outputs to prevent signals applied to the system logic or through the chip's I/O pins, from toggling while new test data is shifted into the register.

Capture-IR, Shift-IR, and Update-IR - These states are analogous to Capture-DR, Shift-DR, and Update-IR respectively, but only affect operation of the instruction register. A new instruction is applied in the Update-IR state.

In the Update-DR and Update-IR states, action takes place on the falling edge of TCK, while action takes place on the rising edge of TCK in all of the other states. TDO is active only during the Shift-DR and Shift-IR states. The test logic remains idle in the remaining eight states. The pause states, Pause-DR and Pause-IR, are provided to allow the shifting process to be temporarily halted. The Select-DR-Scan, Exit1-DR, Exit2-DR, Select-IR-Scan, Exit1-IR, and Exit2-IR states are decision points in the state diagram.

3.1.3 Instruction Register

The instruction register provides on the alternate serial paths between TDI and TDO. It operates during the instruction scanning portion of the controller state diagram. The instruction register is a parallel-in, parallel-out shift register. The parallel output is latched so that a new instruction can be shifted in without altering the previous instruction. The latched output is updated from the shift register path in the Update-IR state; at this time, the new instruction becomes current. In the Test-Logic-Reset state, the latched output is reset to load the BYPASS instruction. The instruction register must contain at least two stages. No maximum length is defined, since this will be determined by the number of test

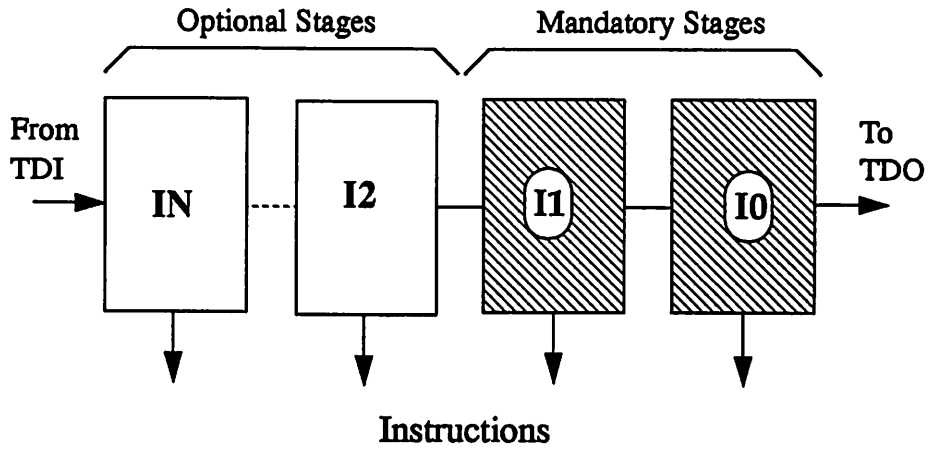


Figure 3-3: Instruction register.

instructions required. A block diagram of the instruction register is shown in Figure 3-3. Stages I1 and I0 must be set to 0 and 1 respectively in the Capture-IR state. These fixed values ease detection and location of faults that may exist in the scan path. In a board design, instruction registers are daisy-chained together in the Shift-IR state so that different instructions can be shifted into each chip in the path. The TAP controller is implemented using the Standard Cell logic design methodology. It was synthesized from a behavioral description written in BDS [Octtools].

Instruction Register Cell Design

The instruction register provides one of the alternate serial paths between TDI and TDO. It operates when the instruction scanning portion of the TAP controller state diagram is entered. The instruction register allows test instructions to be entered into each chip along the board level scan path. These registers are daisy-chained together at the board level in the Shift-IR controller state, so that a different instruction can be loaded into each chip on the path if required. A block diagram of the instruction register cell is shown

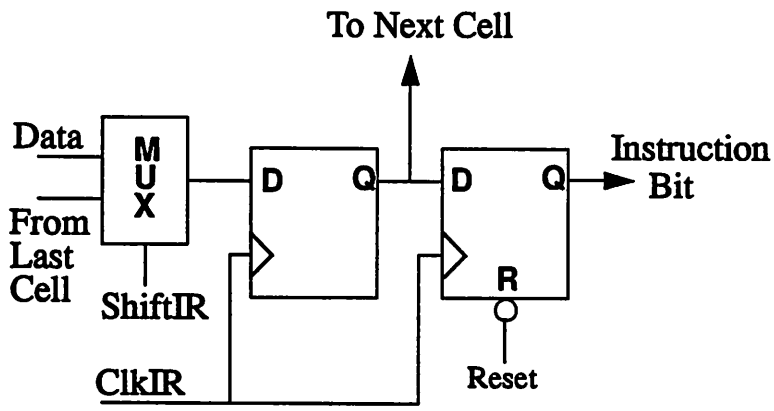


Figure 3-4: Block diagram of instruction register cell.

in Figure 3-4. Each cell has a latched output to which instructions are transferred when they are valid, this assures that the test logic receives only valid instructions. The function of the ShiftIR, ClockIR, and UpdateIR signals are analogous to the ShiftDR, ClockDR, and UpdateIR respectively. When the TAP controller enters the Test-Logic-Reset state, it applies a 0 the Reset* input and forces a 0 to appear at the instruction register's output.

3.1.4 Test Data Register

The Boundary Scan standard specifies the design of three test data registers, two of which must be included in the design. The mandatory test data registers are the bypass and boundary scan registers. The device identification register is optional. All test data Operation of the various test data registers is controlled according to the current instruction. An instruction can place several test data registers into their test mode of operation, but it can select only one register to connect between TDI and TDO in the Shift_DR controller state as shown in Figure 3-5.

Registers that are not used during a test operation are configured such that they do not

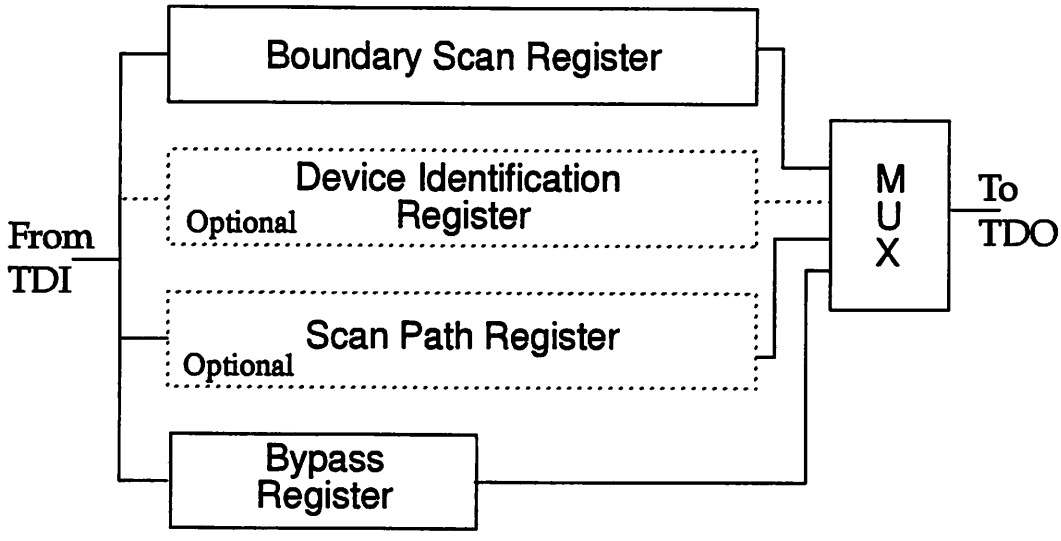


Figure 3-5: Test data registers.

interfere with the operation of the chip's internal logic. Registers that are used during a test operation will load data from their parallel inputs in the Capture-DR state and make any new data available at their latched outputs in the Update-DR state. In other words, the results of a test are sampled in the Capture-DR state, and the new test data is available, at the latest, in the Update-DR state. Any test operations required between the Update-DR and Capture-DR states must occur in the Run-Test/Idle state.And, the register selected by the instruction to be the serial path between TDI and TDO must shift data from TDI towards TDO in the Shift-DR controller state. All other test data registers enabled for test operation will retain their state while shifting occurs.

Boundary Scan Register Cell Design

To comply with the Boundary Scan standard, a chip must contain boundary scan register cells at its input and output pins, as shown in Figure 3-6. These cells should be located:

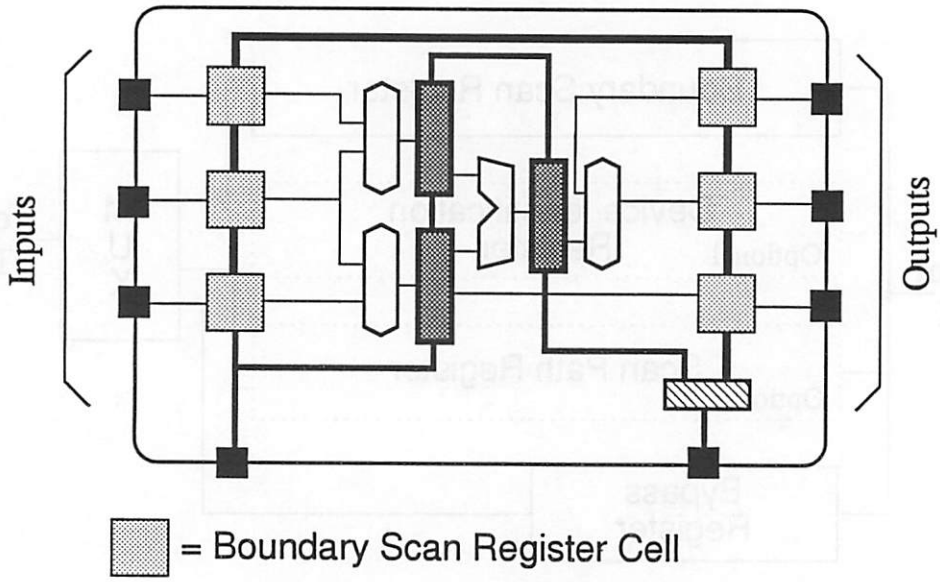


Figure 3-6 : Organization of boundary scan register cells.

- between each input pin (clock or data) and the corresponding input to the chips internal logic;
- between each output from the chip's internal logic and the corresponding output pin;
- and between each tri-state enable or direction control output from the chip's internal logic and the corresponding output driver pin.

A block diagram of an output boundary scan register cell is shown in Figure 3-7. During normal operation, the MODE signal is disabled and the cell becomes transparent and data passes directly to the chips output pin. When an instruction is selected, the ShiftDR and ClkDR signals are asserted until all test data is loaded into the boundary scan chain after which, the UpdateDR signal is asserted and the test data is applied to the output pins. Meanwhile, the mode signal is asserted during the entire test. Test results can then captured at the input pin of an adjacent chip and shifted out for comparison with that of a good circuit.

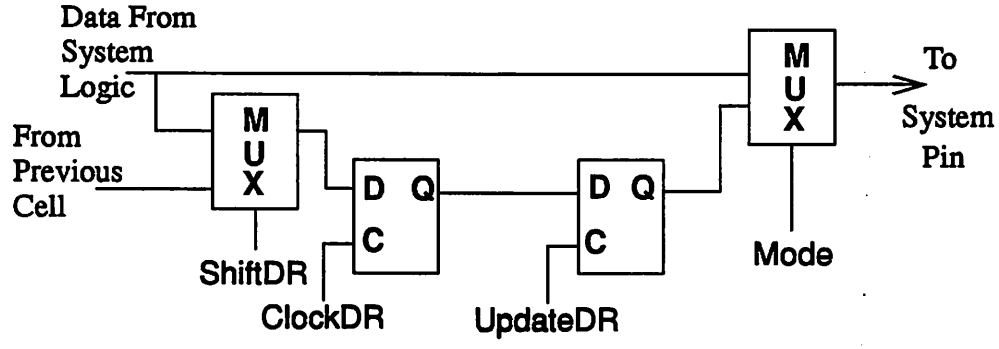


Figure 3-7: Output boundary scan register cell.

These cells operate in conjunction with the cells at the data connections of the chip's internal logic, furthermore, they allow the state of the output driver and the data value driven when the driver is active, to be controlled. The reason for the necessity of these cells is illustrated in Figure 3-8 which shows a board level tri-state bus that can be driven by chip A, chip B, chip C, or chip D. To test the interconnection between these chips, it is necessary to determine whether the bus can be driven to both a logic 0 or logic 1, and whether each chip can drive signals onto the bus independent of the others. Figures 3-9 and 3-10 illustrate how boundary scan cells are used in a tri-state pin and bidirectional, respectively. An additional cell is required to control the state of both the tri-state and bidirectional pins. The CHIP_TEST* signal is provided to prevent the I/O pin from toggling during Scan Path and BIST testing. The EXTEST* signal is provided to prevent the chip's internal logic from changing during component interconnect testing.

3.1.5 Bypass Register Cell Design

The bypass register must also be present in all chips that conform to the standard. It is the shortest path between the TDI and TDO pins and allows data to be shifted through the

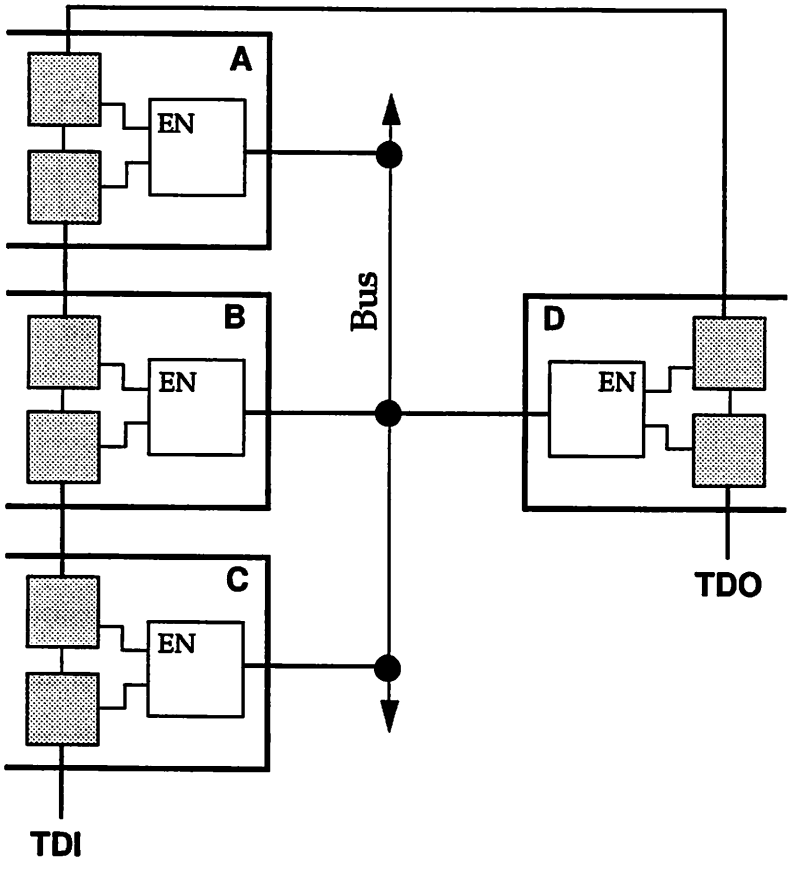


Figure 3-8: Testing a tri-state bus.

chip without interfering with its system operation. The bypass register consists of a one bit shift-register that loads a constant logic 0 in the Capture-DR controller state when the BYPASS instruction is selected. It does not have a parallel data output, therefore, the data present in the register when shifting is completed is unimportant. The importance of this register is illustrated in the following example, consider a board containing 100 Boundary Scan chips all connected into a single serial chain and you need to access the boundary scan register that's located on the 49th chip in the chain, but you do not want to interfere with the operation of the remaining 99 chips. In this case, the required instruction would be loaded into Chip3, with the BYPASS instruction being loaded into the other chips. The

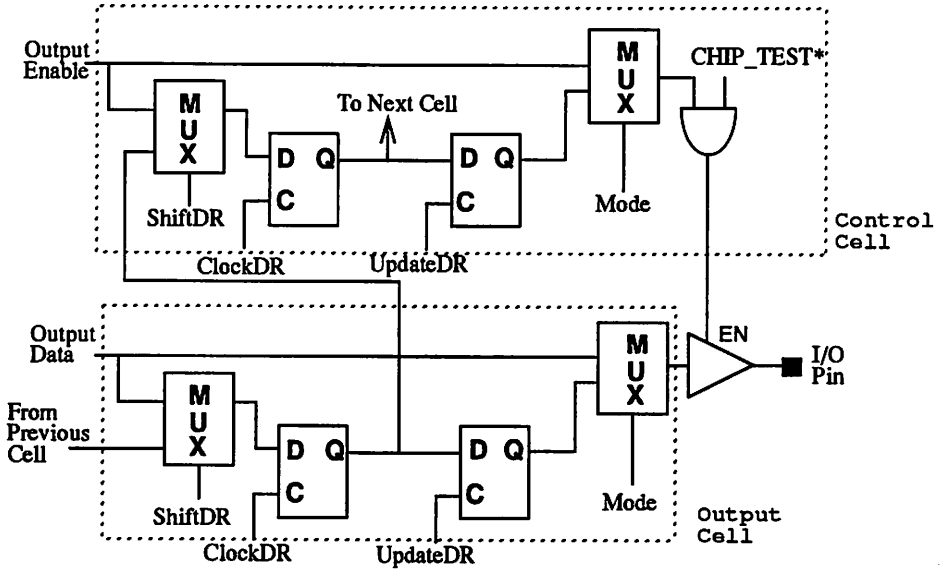
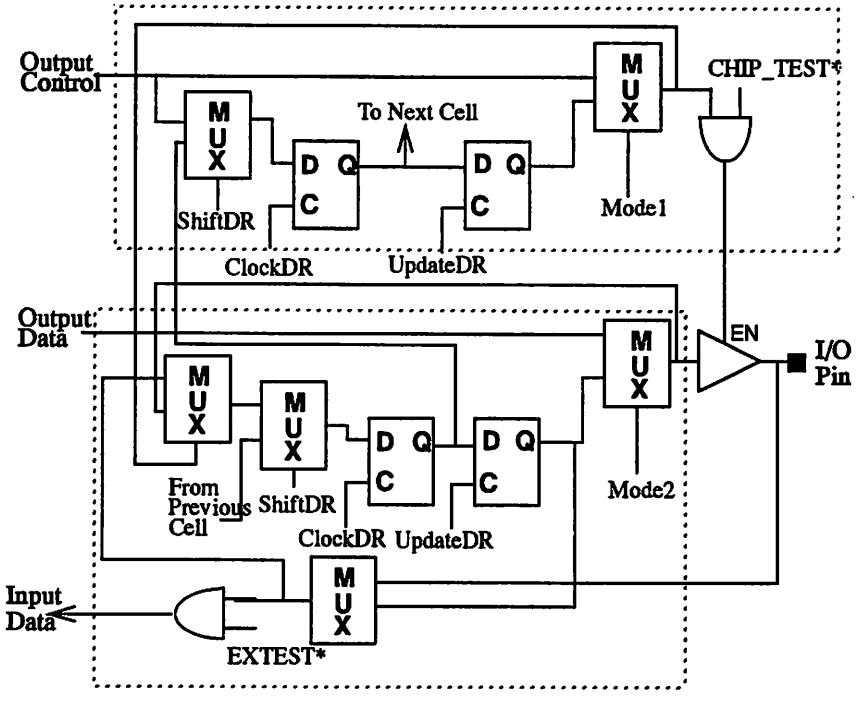


Figure 3-9: Boundary scan cell for tri-state output pin (above).

Figure 3-10 : Boundary scan cell for bidirectional I/O pin (below).



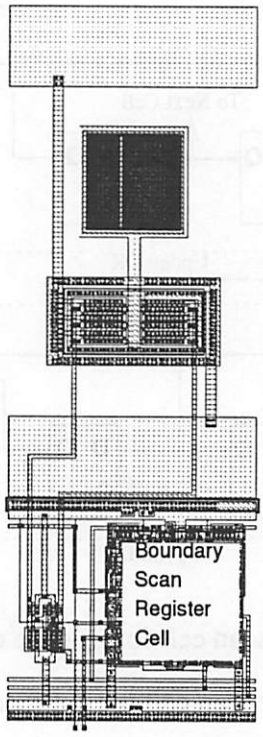


Figure 3-11: Example output pad implementation.

serial bit stream shifted into TDI during the instruction scanning cycle would be: 111....1111CCC...CCC1111...111 where CCC...CCC is the instruction to be loaded into the 49th chip.

3.1.6 Mandatory Instructions

A chip implementation must support several mandatory instructions in order to satisfy the minimum requirements mandated by the standard called EXTEST and SAMPLE. The *Mode* signal in Figure 3-7 is generated by decoding the current instruction and should be asserted (set to 1) when the EXTEST instruction is executed and it should be disabled (set to 0) otherwise. This instruction allows the Boundary Scan register to be used for board interconnect testing in the following way:

- Test data shifted into the Boundary Scan cells located at the chips output pins are driven through the connected pins onto the board interconnections. This process is initiated by executing the EXTEST instruction and then moving to the Shift_DR controller state. One bit of data is shifted into the Boundary Scan register on every rising edge of TCK.
- Shifting is complete when the controller enters the Update_DR controller state. On the falling edge of TCK in this state, the data is transferred from the Boundary Scan register stages onto the latched parallel outputs of each cell. Because the *Mode* signal is asserted by the EXTEST instruction, the test is applied to the board interconnections at this time.
- The test results are captured in the cells at the system input pins. This occurs on the rising edge of TCK in the next Capture_DR controller state.
- The test results are examined by moving back to the Shift_DR controller state. The data held in the Boundary Scan register move one stage towards the TDO pin on each rising edge of TCK.

When the SAMPLE/PRELOAD instruction is executed, the *Mode* signal is disabled allowing the chip to continue its normal operation without interference. This instruction supports two distinct test operations. In the first instance, the Boundary Scan cells at both inputs and outputs load the state of the signal flowing through them between the I/O pin and the chip logic:

- A snap shot of the data flowing through the chip's I/O pins is taken by first executing the SAMPLE/PRELOAD instruction and then moving to the Capture_DR controller state.
- The captured data can be shifted out for examination in the Shift_DR controller state. On each rising edge of TCK, the data held in the Boundary Scan register advance one stage towards TDO.

Applications of the SAMPLE test include debugging of prototype boards.

In the second instance (PRELOAD), data can be shifted into the Boundary Scan cells without interfering with the normal flow of signals between the chip pins and the application logic. This allows the latched parallel outputs in the Boundary Scan cells to be initialized with data before the next instruction is selected. The following events occur before execution of this instruction:

- test data is shifted into the Boundary Scan register by first selecting the SAMPLE/PRELOAD instruction and then moving to the Shift_DR controller state.
- after the data is loaded and shifted in, the scanning sequence is halted by moving to the Update_DR controller state at which time the data is applied for initialization.

By loading suitable data when PRELOAD is selected, the user can ensure that all signals driven off the chip are defined as soon as the EXTEST instruction is selected.

3.2 Designing Chips with Boundary Scan

When incorporating the Boundary Scan architecture into a chip, there are several issues that must be addressed. The decision to incorporate this architecture in a chip first should be considered from a board or system level point of view. If Boundary Scan is to be used as a system requirement, then it is very important to define the instructions at the system or board level first, and then implement the appropriate instructions on the chip. As a minimum, the standard requires that a chip must be able to execute three instructions: BYPASS, SAMPLE, and EXTEST.

To implement the logic required by the standard, the designer must either design the Boundary Scan test circuitry manually or use an existing library module. The module must consist of a minimum of four basic building blocks: a Boundary Scan register, a TAP

controller, an instruction register, and a bypass register.

The standard also contains a number of design requirements that must be followed to ensure that the chip works properly with other Boundary Scan chips. First, a Boundary Scan cell must be placed at each I/O pin except the four test bus signals. Second, the TMS and TDI test bus signals must use a pull-up cell to prevent unstable TAP controller operation. These requirements represent the most common implementation guidelines, and additional rules can be obtained from the actual specification.

3.2.1 Boundary Scan Macrocell

The design of the Boundary Scan architecture should be a structured, parallel effort that complements the natural top-down design style associated with each chip. All this is achieved by a macrocell called JTAG_MACRO that automatically implements this architecture has been developed. It is written in the SDL language and requires that the designer provide parameters such as boundary scan register length. A summary of all the required parameters and their corresponding functions are given in Table 3-1. The designer can also dictate the shape of the Boundary Scan macrocell using the BSrows and SProws parameters. Pointers to the SDL files for the JTAG_MACRO are given in Appendix B.

3.2.2 Boundary Scan I/O Pad Library

To accommodated designs where the designer is constrained to a limited chip core area, a library of input and output pads have been developed that contain boundary scan registers inside them. The serial scan path is constructed by simple abutment of pad cells that form a frame around the border of the chip. A plot of an input boundary scan output pad is shown in Figure 3-11 and Table 3-2 describing boundary scan pad library cells is

Parameter	Function
BSwidth	Determines width of Boundary Scan register
BSrows	Stdcell parameter used to determine the shape of the Boundary Scan register
BSreset	Selects reset option for Boundary Scan register cells
SPwidth	Determines width of Scan Path register
SProws	Stdcell parameter used to determine the shape of the Scan Path register
SPreset	Selects reset option for Boundary Scan register cells
MUXwidth	Determines the width of the multiplexers. Default value is 3, increase by one for each additional register
TAPflag	Selects between Stdcell and PLA controller implementations Stdcell is default implementation.

Table 3-1 : JTAG_MACRO parameter definitions.

given below.

3.2.3 Integrating Scan Path with Boundary Scan

The Scan Path test methodology can be used easily supported by using a private SCANTEST instruction whose opcode is 100. When this instruction is selected, the Test Mode Select signal, which controls movement between controller states, acts like the test mode control for a traditional Scan register, which causes movement between shift and load. Test data is loaded into the Scan register when TMS = 1, and data is shifted when TMS = 0. For chips that employ a single scan path, the TDI and TDO pins become the SCANIN input and SCANOUT output. In this case, the TDO driver must be modified to allow it to be active in the Pause_DR controller state. Multiple scan paths can be supported by multiplexing the serial inputs and outputs onto normal package pins when TMS i= 0 and SCANTEST is selected.

I/O Pad	Function
in_2u	Input Boundary Scan Pad
out_2u	Output Boundary Scan Pad
io_2u	Bidirectional Boundary Scan Pad
tdi	Test Data Input Pad
tdo	Test Data Output Pad
tms	Test Mode Select Pad
tck	Test Clock Input Pad
vdd_2u	Power Pad
gnd_2u	Ground Pad
analog_in_2u	Unbuffered Boundary Scan Pad
space_2u	Space Pad
corner_2u	Pad Frame Corner Pad

Table 3-2 : Listing of Boundary Scan pad library cells.

3.2.4 Integrating BIST with Boundary Scan

In [LeBlanc84], the idea of integrating BIST with Boundary Scan was introduced. Similar to the Scan Path case, the Boundary Scan circuitry can be supplemented to provide test support for BIST applications. By selecting the RUNBIST instruction and placing the TAP controller in the Run_Test/Idle controller state for as many clock cycles as is required to execute the self-test and providing additional circuitry for control, BIST circuitry can be easily controlled through the Boundary Scan test bus interface.

Embedded memory testing is one of many applications that is ideally suited for BIST. In embedded memories, the address, data, and control inputs may not be directly controllable and the data output may not be directly observable at a chip's input and output pins. Further, the test patterns for memories are required to detect a wide variety of complex

faults. These faults are different from classical stuck-at faults. Since, the use of scan based design techniques do not simplify the problem of testing embedded RAMs, it becomes cost-effective to incorporate BIST features in embedded memories to avoid complex time consuming test generation. The BIST feature of embedded memories provides vertical testability of the RAM, not only at the board level, but at the system level as well.

The embedded memory BIST macro requires a counter, exclusive-OR gates, several multiplexers, and a Linear Feedback Shift Register (LFSR) as shown in Figure 3-12. A linear feedback shift register can be formed by exoring the outputs of two or more of the flip-flops together and feeding them back into the input of one of the flip-flops. The counter is used to supply the test patterns to the embedded RAM and to control the testing sequence. Unlike typical BIST techniques which use an LFSR as a pseudo-random pattern generator, the counter provides a deterministic set of patterns necessary for thorough testing of the RAM circuitry. The LFSR in this BIST approach is used to compress the output data from the RAM using signature analysis techniques.

For purposes of illustration, a RAM with 16 addressable locations and 4 bits per location will be considered as shown in Figure 3-13. The basic idea is to use a counter during testing to supply the address to the RAM, to supply data, and also to control the entire test sequence. A 6-bit counter is used in this example. The four lower bits of the counter are used to supply the address and data. The fifth bit is used to control reading and writing of the RAM during the testing sequence such that the entire RAM is written and then read. The sixth bit is used to invert the data going into the RAM during the test. The test proceeds as follows. First the RAM is written with the address such that address location 0 contains the data '0000', address location 1 contains the data '0001', and address location F (Hexadecimal) contains the data '1111'. Thus each address location contains a unique data value. This approach assumes that the number of data bits in each RAM location is

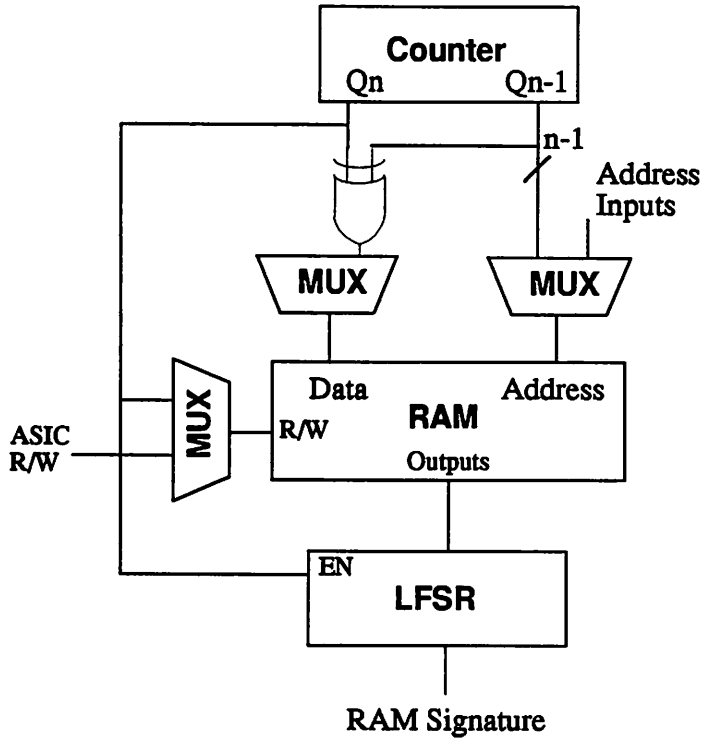


Figure 3-12: Embedded RAM BIST circuit.

greater than or equal to the number of address bits. Next, the RAM is read, via the fifth bit of the counter controlling the read operation and the first four bits controlling the address, beginning with address location 0. When location 0 has been read, any faults in the address decoding circuitry of the RAM will be detected since a failure would have caused either location 0 to be overwritten during the writing sequence or another location to be read during the read operation. When the entire RAM has been read, the sixth bit of the counter is used to invert the data entering the RAM and the entire RAM is rewritten with the complemented data. In this case, address location 0 is written with the data '1111', address location 1 is written with '1110', and so on. Once again the RAM is read under control the fifth bit of the counter. When the read sequence is finished, each bit in the

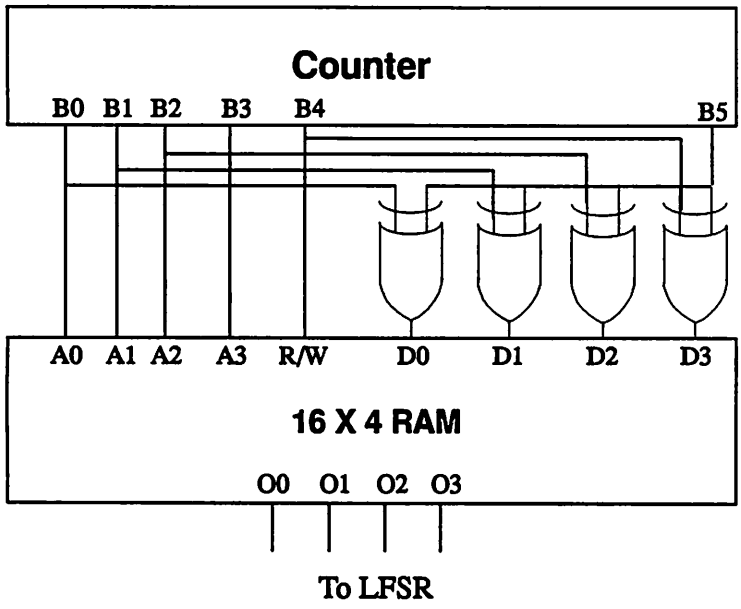


Figure 3-13: RAM BIST example.

RAM has been tested for failures which would render that bit stuck at a logic 0 or 1. Hence, all classical "stuck-at" faults associated with the RAM are detected.

Due to the large number of patterns which can be encountered using the counter testing technique described above, propagating the RAM output data to the chip's outputs can be difficult. By including a data compression technique such as signature analysis, the BIST mechanism is obtained. In the BIST circuit for the RAM, a parallel entry LFSR is most appropriate since the data typically leaves the RAM in parallel. During the write sequences to the RAM the LFSR is disabled from shifting or loading in new data since the output of the RAM may be unknown during a write operation. The LFSR is controlled in this case by the same counter bit output that controls the writing and reading of the RAM. The LFSR is disabled at the end of the test and the resulting signature is read for comparison with a stored correct signature or is compared to a predefined correct signature stored on the chip. The latter case requires the use of a comparator. At present,

the LFSR in this macrocell is generated by hand. The SDL for this macrocell called BIST_MACRO is given in Appendix B.

3.2.5 Chip Implementations

Two prototype chips have been implemented in 2u CMOS technology using the MOSIS fabrication facility. They were designed using the LAGER [Brodersen92][Shung89] silicon assembly system. The layout for TEST_CHIP1 which contains the JTAG_MACRO is shown in Figure 3-14. In this chip, the boundary scan register is embedded within the chip's internal core circuitry, whereas TEST_CHIP2 uses boundary scan pads and its corresponding chip layout is shown in Figure 3-15. Moreover, each chip contains a Scan Path register embedded in its data path that is accessible through the Boundary Scan TAP bus.

3.3 Trade-Offs: Design Costs vs. Test Costs

The impact of Boundary Scan as seen at the chip level is much more profound that at both the board and system levels. This is because Boundary Scan at the chip level affects the I/O pins, which consequently affects the package size, the overall gate count, and the performance due to the additional delay seen at the I/O pins. The Boundary Scan standard requires a minimum of four extra I/O pins. This overhead is always required, but is less obvious in larger package devices. Table 3-3 shows the percent of additional pins per package size. The number of gates required to implement Boundary Scan is primarily driven by the number of chip I/O pins. The reason for this is because the standard requires each functional I/O pin to have a boundary scan register cell. Naturally, chips with low gate counts and a high number of I/O pins will have proportionally more gates to implement the Boundary Scan architecture. The standard also requires a 2-bit Instruction

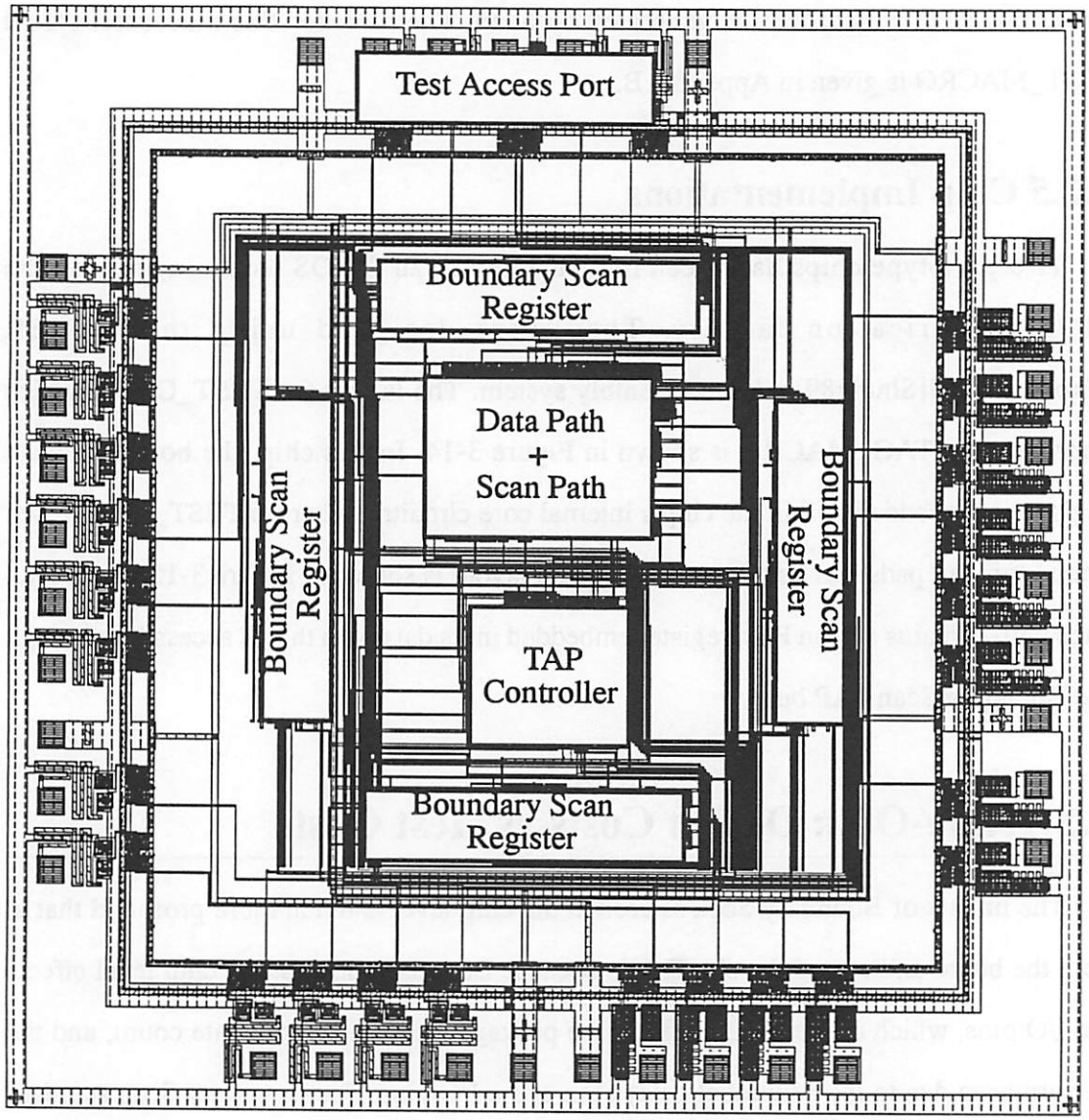


Figure 3-14: Chip layout for TEST_CHIP1.

Register as a minimum, but longer Instruction Registers are allowed to implement additional application specific instructions. The following formula can be used to estimate the gate count overhead (functional I/O pins are only counted):

Gate Count = (TAP) + [(IR) * (IR bit width)] + (Bypass) + [(# I/O pins)* (# gates/pin)].

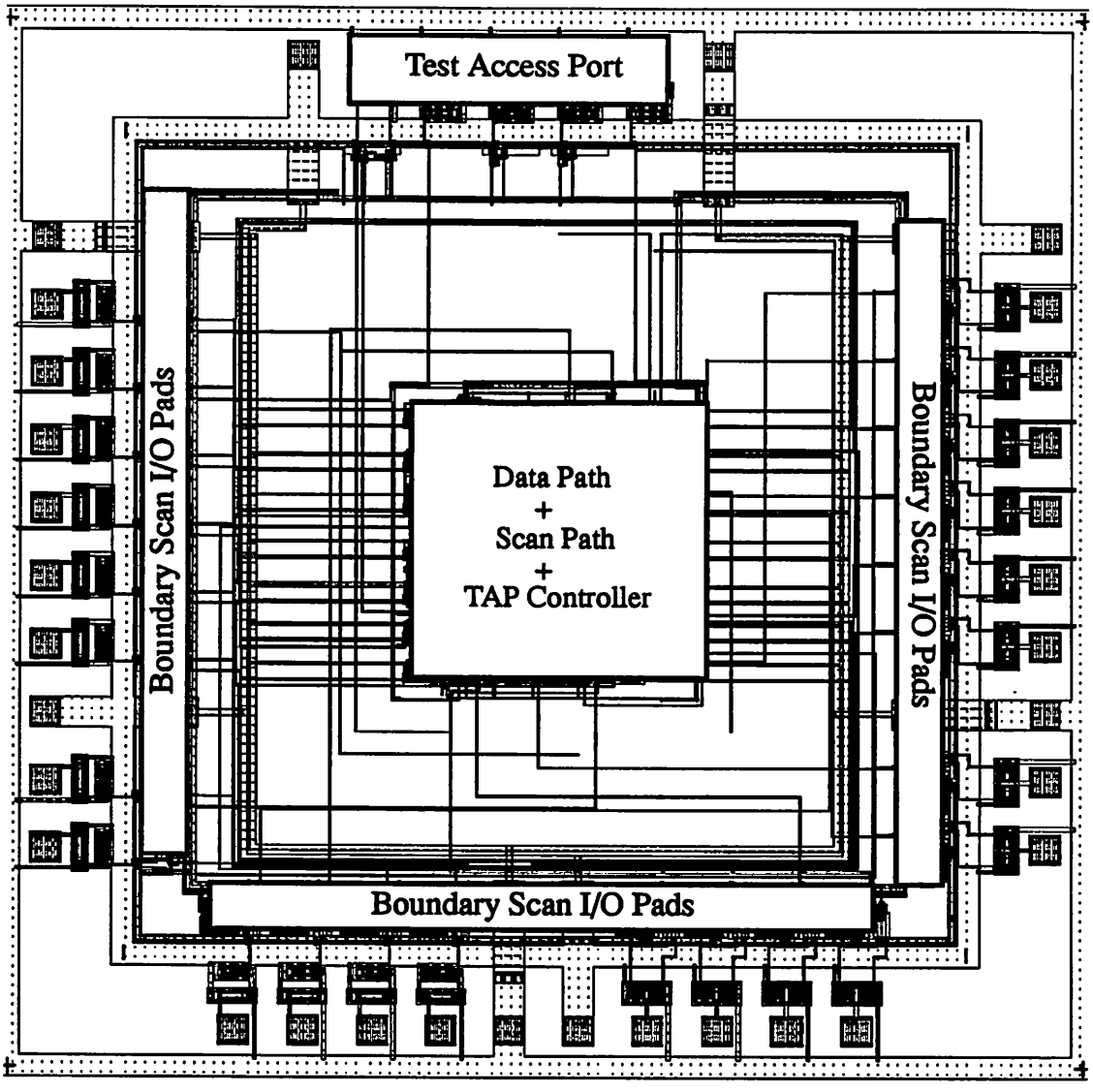


Figure 3-15: Chip layout for TEST_CHIP2.

Boundary scan register cells introduce a propagation delay in the data path that is equivalent to that of a 2-1 multiplexer. A typical value for a 2um CMOS technology is on the order of 3.0 nanoseconds.

Package Size	Percent of Boundary Scan Pins vs. Total Component Pins
24 pins	16.7%
40 pins	10.0%
64 pins	6.3%
100 pins	4.0%
132 pins	3.0%
160 pins	2.5%
208 pins	1.9%

Table 3-3 : Component package /pin ratio.

3.4 Summary

An overview of the Boundary Scan standard was presented in this chapter. Two prototype chips implementing the Boundary Scan architecture and Scan Path have been described. Finally, trade-offs between design and test costs are addressed.

CHAPTER 4

TEST HARDWARE - BOARD LEVEL

Testing at the system or subsystem level is not always accomplished by simply using a set of testable chips unless they are properly integrated at the board level. Traditional board level testing consumes a great deal of time and requires special hardware and complex Automated Test Equipment for each type of board or device. This ultimately results in increased development time.

An innovative approach to the problems associated with traditional board level testing is to incorporate DFT techniques that allow embedded testing to be performed. For example, scanned in values can initialize states before testing, and testing can be done while the component is embedded within a board. Board level testing can made easy with Boundary Scan components. These components can be used to effectively partition and isolate sections of a board for quicker fault isolation. Furthermore, these components can eliminate physical access problems and provide the designer with access to and control of hard to access nodes on the board. Not only do these components perform functions such

as buffers, transceivers, latches, and flip-flops, but they also include components that perform dedicated low level test functions. These low level test functions can be easily combined to create high level test functions. Some of these high level test functions are used in a prototype design called the Test Master Controller board which is used to control and access the Boundary Scan components of a target board.

This chapter deals with the requirements, design, and implementation of the hardware that is used to support board level testing, which includes a Boundary Scan components library, dedicated board level test modules, a custom Test Master Controller board, and a discussion on board level trade-off issues.

4.1 Boundary Scan Component Library

Due to widespread adoption of the Boundary Scan standard by the commercial ASIC industry, many Boundary Scan components are beginning to appear on the market. Since most board designs include octal devices - buffers, latches, transceivers, flip-flops - for bus operations, manufactures have provided families of octal chips that support the Boundary Scan standard. These octals can replace their standard IC counterparts to enhance the testability at the board level. When used in their test mode, these octals offer designers a number of useful test features such as pseudorandom pattern generation and parallel signature analysis.

A variety of Boundary Scan components which are manufactured by a number of ASIC vendors are organized into a test component library. Table 4-1 provides a description of the components and their corresponding function .

As an example, a partial listing of an SDL file for an octal buffer is shown in Figure 4-1. The file provides a black-box footprint of a generic TTL LS244 part where the I/O

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;NAME:xx244.sdl                                        ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;
(parent-cell xx244 (PACKAGECLASS PCB)
(bag JTAG (BSR 18) (IR 8) (BPR 1) (MASTER 0)))
(parameters ;
;pre-defined local variables
(PKGLIST '("DIP" "SOIC" "PLCC") (local))
;user parameters
   (PKGTYPE "SOIC" (assert (memql PKGTYPE PKGLIST)))
   (JTAG 1 (assert (or (= JTAG 0) (= JTAG 1))))
;local variables determined from parameters
(PKGCODE (list-index PKGTYPE PKGLIST) (local))
;oct2rinf variables
(PARTNAME (sel_jtag "74BCT8244" "74LS244")) (PHYSNUMBER
(sel_jtag (sel_pkg "L-GEN24" "L-GENSO24WB" "L-PLCC28SA")
(sel_pkg "L-GEN20" "L-GENSO20WB" "L-PLCC28SA")))
;info for interconnect test pattern generation
(PARTTYPE "DIGITAL"))
(layout-generator                                          NONE)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;NETS AND TERMINALS ;;;;;;;;;;
(net A ((parent (term A (PINNUMBER (pin (sel_jtag
 (sel_pkg (23 22 21 20 19 17 16 15)
 (23 22 21 20 19 17 16 15) (6 5 4 3 2 27 26 25))
 '(2 4 6 8 11 13 15 17))))

(TERM(net TDI (CONDITIONAL (= JTAG 1))
 ((parent (term TDI (PINNUMBER (sel_pkg 14 14 24))
 (TERMTYPE SIGNAL) (DIRECTION INPUT)))))
TYPE SIGNAL) (DIRECTION INPUT)) (width 8))))

(end-sdl)
```

Figure 4-1: Partial SDL file for xx244.

Boundary Scan Component	Component Function
TI SN74BCT8244	Scan Test Device with Octal Buffer
TI SN74BCT8245	Scan Test Device with Octal Bus Transceiver
TI SN74BCT8373	Scan Test Device with Octal D-type Latch
TI SN74BCT8374	Scan Test Device with Octal D-type Flip-Flop
TI SN74ACT8990	Boundary Scan Test Bus Controller Chip
TI SN74ACT8994	Boundary Scan Digital Bus Monitor Chip
TI SN74ACT8997	Boundary Scan Path Linker Chip
TI SN74ACT8999	Boundary Scan Path Selector Chip
TI TMS320C40	Floating-Point DSP for Parallel Processing with Boundary Scan
TI TMS320C50/51	Fixed-Point DSP Chip with Boundary Scan
TI TMS29F816	Boundary Scan Flash EEPROM Chip

Table 4-1 : Listing of Boundary Scan devices.

terminals are the same but the pin mappings are different for a given package type. In this example the JTAG contains all of the Boundary Scan implementation specific information for this particular device. This information includes the size of the Boundary Scan (BSR), Instruction (IR), and Bypass registers (BPR) and a variable indicating whether the chip is a Boundary Scan slave (MASTER 0) or master (MASTER 1) device.

The local variable PKGLIST is used to define the various package types that are available for the given part. PKGCODE is a Lisp function that checks to see if the package type value, which in this case can be a dual-in-line or small outline or leadless chip carrier package, assigned to PKGTYPE by the user is a valid one. The other user parameter, JTAG, is used to distinguish between a non-Boundary Scan and a Boundary Scan component. By default, values for both user parameters are chosen for the user to bias

designs to use Boundary Scan components housed in surface mount packages and most importantly, this is the first step toward automating the process.

The PARTNAME and PHYSNUMBER variables contain information that is required by the board layout generation tool, while the Lisp functions sel_jtag and sel_pkg are used to select the correct part name and part number based on the chosen package type. For example, if the PLCC package type chosen and JTAG is 1, the sel_pkg and sel_jtag will select part name "74BCT8244", part number "L-PLCC28SA" and generate the correct pin mappings for this package. The CONDITIONAL property is used to create the additional nets and terminals for a Boundary Scan component. The TERMTYPE and DIRECTION properties to facilitate netlist checking.

Module Name	Module Function
1149.n Master Controller	A software programmable test master controller that can be configured to implement any one of the IEEE 1149 serial test protocols. This feature is achieved by using a Xilinx FPGA.
Local Boundary Scan Master	A Boundary Scan test bus controller module that supports efficient transfer of serial data and control to and from target devices on the local serial test bus.
Real-Time Monitor	Provides a method of monitoring embedded digital signals paths between components on a board. Can be used to reveal timing-sensitive and/or intermittent failures that are otherwise undetectable without the use of external test equipment.

Table 4-2 : Listing of board level test modules.

4.2 Board Level Test Modules

As mentioned in the previous section, the 1149 controller module consists of three dedicated test modules which perform specific testing functions. These board level test modules are intended to be used as building blocks for higher level testing functions.

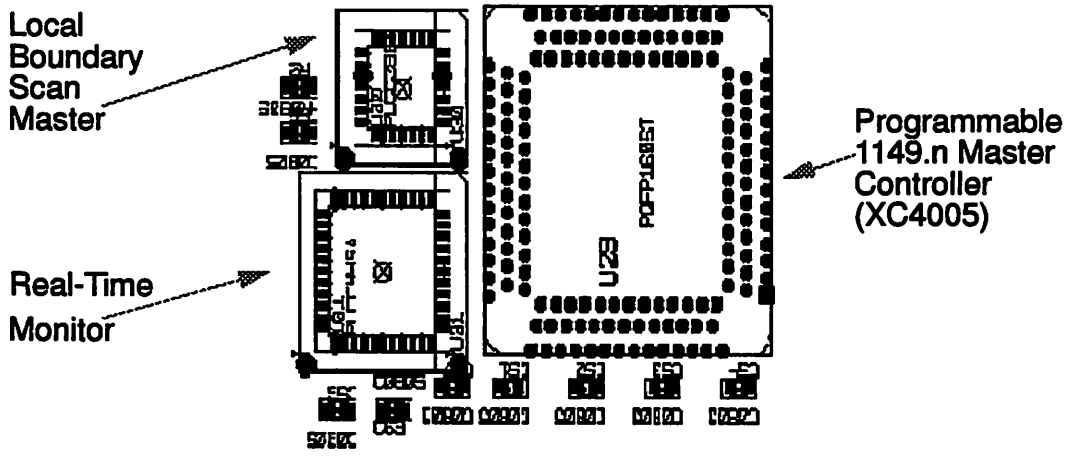


Figure 4-2: Test Master Controller module layout.

Therefore, a designer can enhance the testability of their board design by simply adding one or more of these modules to their design. The modules can also be accessed and controlled via Boundary Scan test bus. The functionality of each of the modules is determined by groupings of one or more of the test components contained in the Test Component Library described in the following section.

These modules have been created using the tools described in the previous chapter. A library of these reusable board level modules has been created and is listed in Table 4-2 along with their corresponding functions. Although the number of library elements is small, it will continue to grow as the demand for modules with more sophisticated test functions increases. The modules are usually specified in a hierarchical SDL file describing the structural interface between its primitive test components. The SDL file also contains floorplanning information. The layout for the 1149 Master Controller module is shown in Figure 4-2. This module does not require any parameters or placement information because, as with the other two modules, the primitive test components that

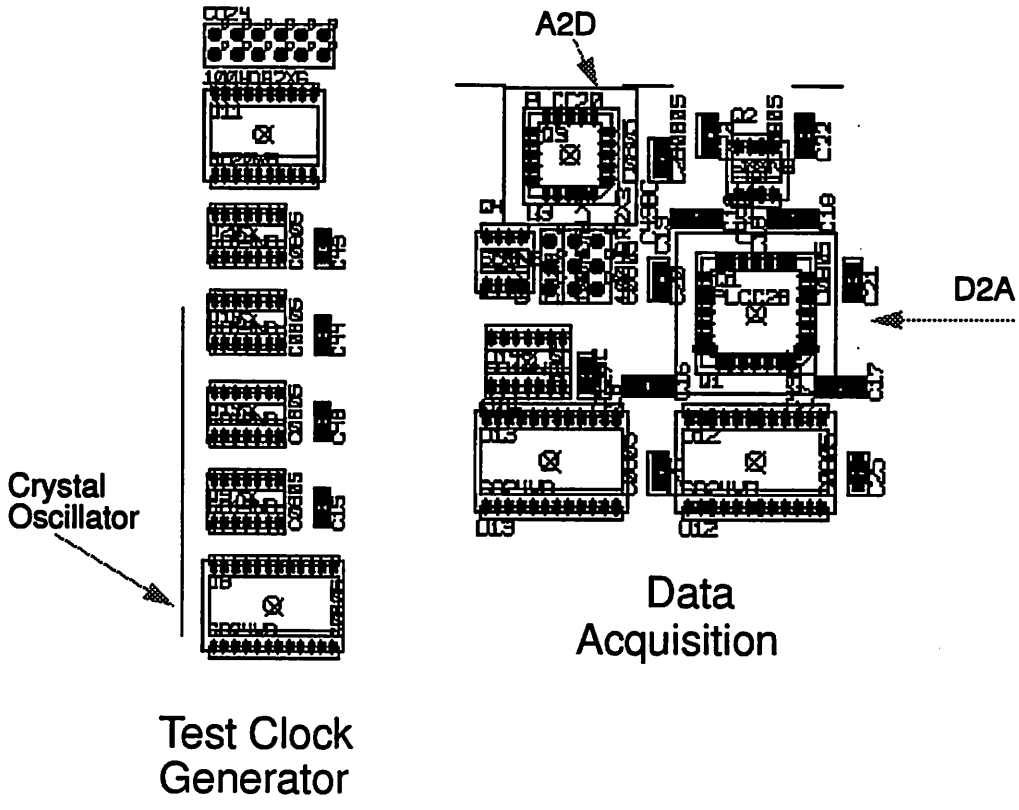


Figure 4-3: Data acquisition and clock generator modules.

make up the modules have already been pre-placed. Other useful modules include the data acquisition and the clock generation both of whose layouts are shown in Figure 4-3. Paths to the SDL files for these modules can be found in Appendix B.

4.3 Guidelines for Prototype Design and Implementation

A set of guidelines that should be followed are given below. Other factors such as board layout limit the type of fabrication and repair process one can use. These guideline are listed below:

1. choose components that either make the testing of the components themselves easier or enhance the testability of the modules or boards that use them. Example features that

make components themselves highly testable include Scan Path and BIST. Boundary Scan is an example feature that enhance board level testability;

2. if a programmable device is to be used in the design, it must be possible to set the device into a known state by applying a signal to one of its inputs. Ideally, the device should be provided with a single asynchronous or synchronous reset input which, when the correct signal value is applied, causes the device to set to a known state;
3. avoid asynchronous design. Asynchronous devices cause significant test and reliability problems and are best avoided completely. If this is not possible, restrict asynchronous devices to a small part of the design which can be isolated from the remainder of the components during test;
4. if necessary, to ease the repair process, all components should be oriented in the same direction. This will also lead to a more reliable design when using a solder flow process.
5. use single-sided component mounting, whenever possible, on plated-through-hole boards. This assembly style uses through-hole or surface mount components. Where components must be mounted on both sides of an assembled board it is essential that all of the components mounted on the bottom side be Boundary Scan components, since this eliminates the need to physically probe their pins;
6. and every board must have one central TAP connected to a 2 x 5 row right-angle connector.

4.4 The Test Master Controller Board

The Test Master Controller (TMC) board [Kornegay91] [Kornegay92] is used to control the test process of a target board by accessing each components's DFT structures via Boundary Scan bus. The TMC transmits test data to and from every component under test in the system. It is also intended to be used embedded in a system as illustrated in Figure 4-7. It also receives instructions and data, which are provided by the user, from the CPU board through the UNIX workstation. These instructions determine which tests are to be executed for the targeted chips on the application board. After a test has been executed, the results are gathered and uploaded to the UNIX workstation where they can be analyzed. Further, it can access a chip's DFT structures through a Boundary Scan

interface. Finally, it can be dynamically reconfigured to support other testability bus standards that use a 5 wire serial test access port.

The architecture consists of the five board level modules shown in Figure 4-4. This architecture was designed with modularity and reusability in mind. Breaking the architecture into smaller, more manageable parts makes testing systems using this architecture, easier. The modules are designed to be reused in other systems. A description of the modules which were created using the module generation environment described in the previous chapter is given below:

VME Bus Interface Logic - implements the VME bus protocol which orchestrates communication between the TMC and the CPU boards.

Control Register - contains execution specific control information required to configure the TMC to operate in one of its test modes.

Status Register - contains all of the system specific status information such as test completion signals, boundary scan path integrity checking information, and other information pertinent to proper system operation.

Clock Generator - a jumper programmable clock generator that produces a two phase non-overlapping clock that operates at clock rates up to 50MHz.

1149.n Test Module - a software programmable test controller module that consists of three smaller modules: a data acquisition, memory, and 1149.n controller.

The data acquisition module contains one 12-bit user programmable Analog-to-Digital Converter that operates up to 100KHz, and one 8-bit Digital-to-Analog Converter that also operates up to 100KHz. This module is provided for testing mixed-signal systems. The memory module which consists of 2 245k x 1 bit SRAM, one for storing the test data to be applied to the device under test, and the other for storing the results that are captured at the end of a test. The 1149.n controller module is the heart of the system is also made up of several smaller dedicated test modules which will be described in the next section.

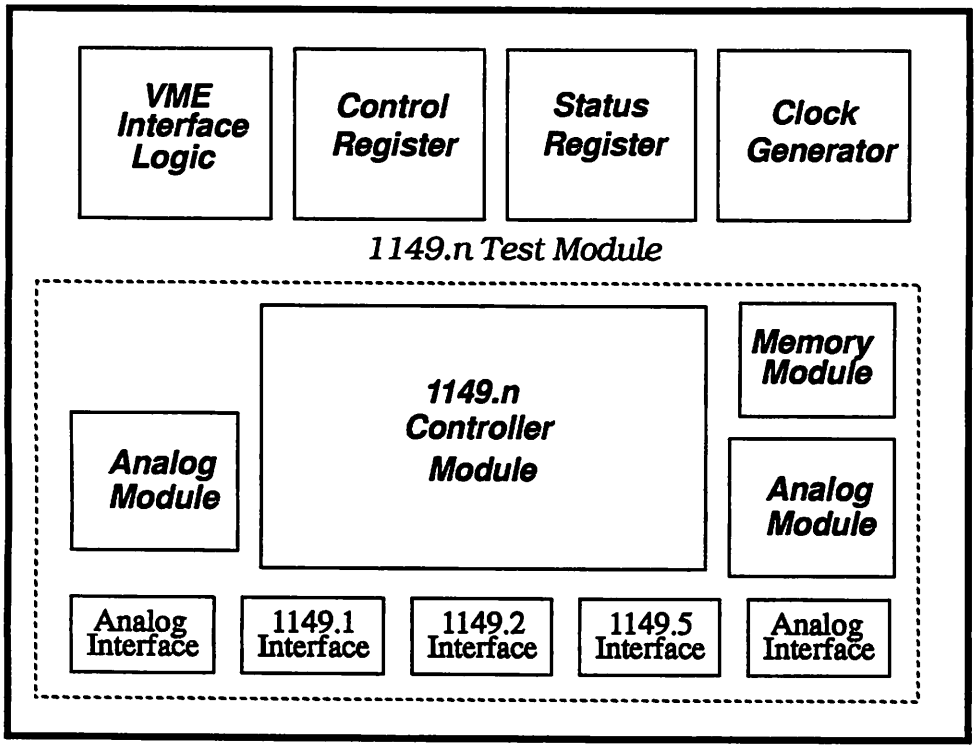


Figure 4-4: Test Master Controller board architecture.

It can be reconfigured, using software, to implement any one or all of the IEEE 1149 [IEEE90a,b][IEEE91] standard bus protocols. In fact, the controller can be configured to perform any custom test protocol provided it uses a five wire serial port. A simplified version of the state diagram for the controller is shown in Figure 4-5. After initialization, the controller begins in the idle state, from which point, it can traverse any one of the branches depending on the value of the test mode (TM) signal. For example, when TM = 0, the controller executes the Boundary Scan bus master protocol. Likewise, the controller will execute any of the other IEEE 1149 bus protocols exercise any BIST features of the devices, or

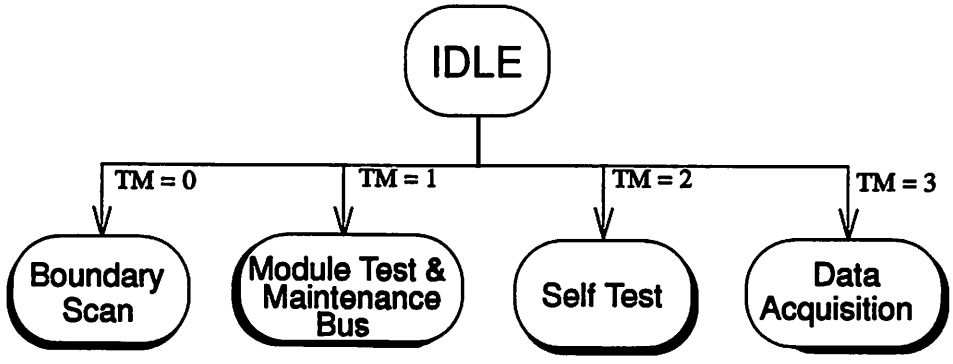


Figure 4-5: Simplified controller state diagram.

apply/capture analog data when TM = 1, 2, or 3. Physical ports exist for the IEEE 1149 buses and the analog module.

4.4.1 TMC Prototype Implementation

The guidelines described in a previous section were strictly adhered to during the design and implementation of the TMC board. Figure 4-6 shows the hierarchy of SDL files. The TMC board was implemented on a 6 layer 6 inch x 9 inch card and contains over 160 surface mount components (chips, capacitors, switches, etc.). The layout of the board is shown in Figure 4-7.

The actual use of the board will depend on whether a centralized or distributed control strategy adopted. In the distributed approach, most of the test functionality is implemented in dedicated hardware that resides on the target boards, whereas, the centralized approach uses the TMC board to implement all board level test functions. The benefits of both of these approaches are outlined below:

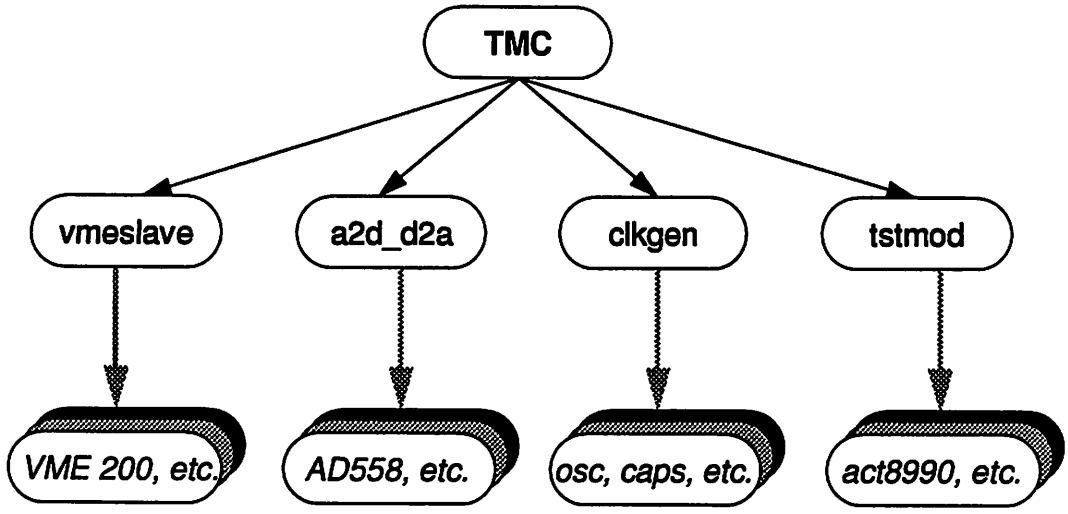


Figure 4-6: Hierarchy of SDL files for TMC board.

Centralized Approach:

- Cost - By centralizing all test functions to a single controller, test sequencing capabilities are not required for each of the target application boards. This can reduce the cost of the test interface and control hardware on each board in a system.
- Simplicity - As the test interface on each board does not contain any board-specific test information, a common test interface can be used on each of the application boards in a system.

Distributed Approach:

- Software - Because distributed test hardware and software allow higher level test functions, less software is required for the TMC board for each of the target application boards. Distributed software can reduce the software development time.

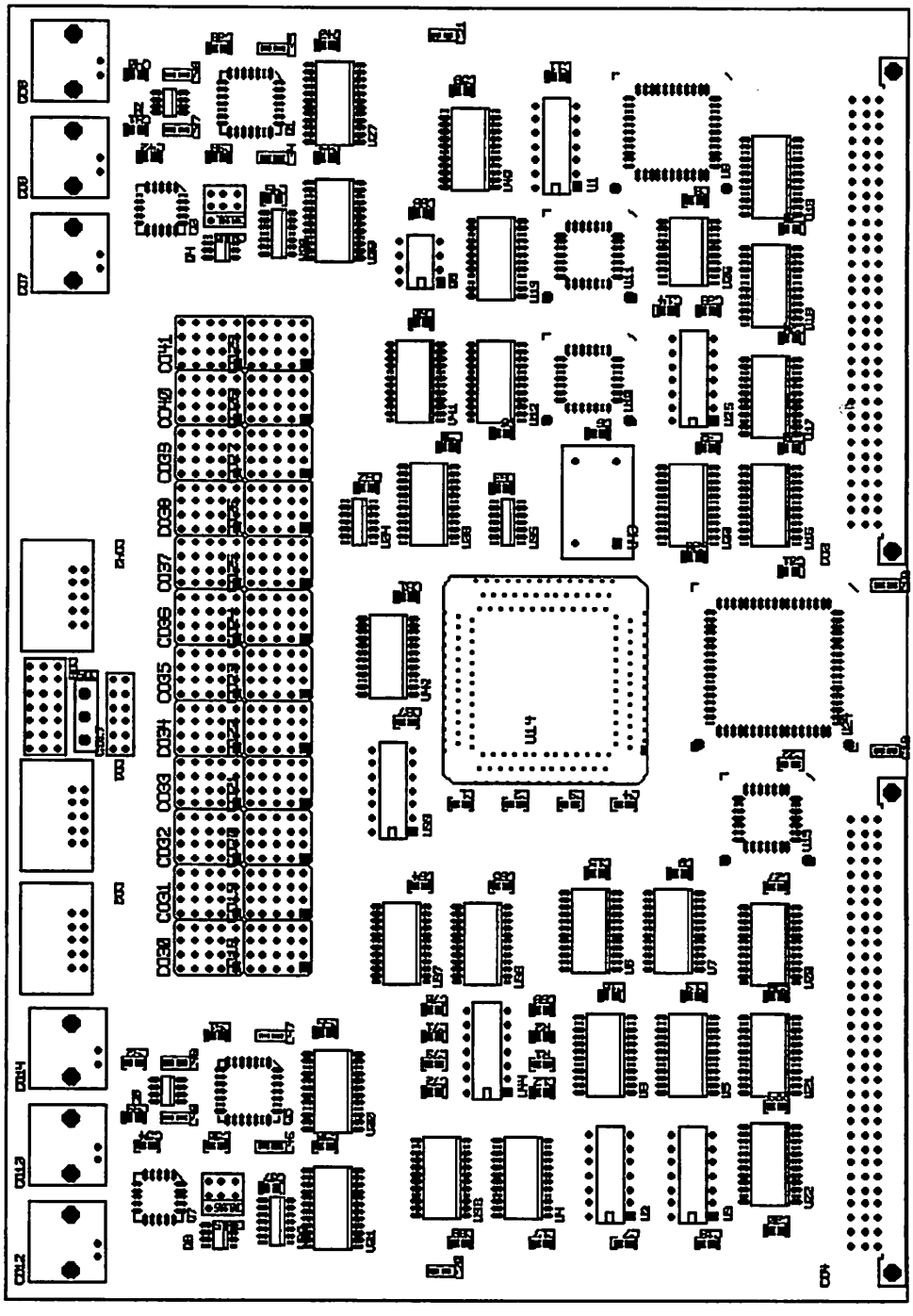


Figure 4-7: TMC board layout.

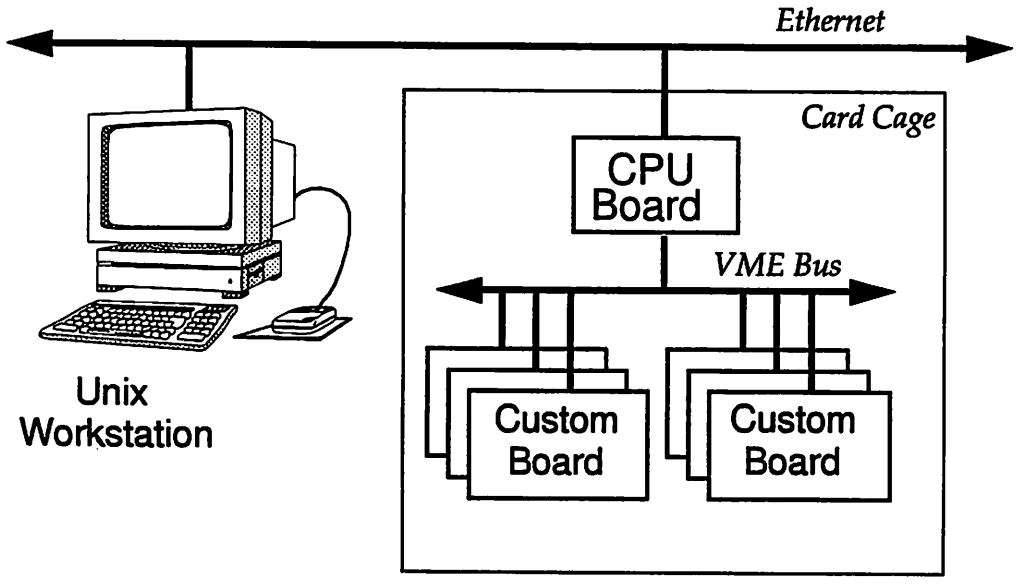


Figure 4-8: System hardware development environment.

- Bus Traffic - Since self-test routines and test patterns are contained on the individual application boards, test bus traffic is restricted to instructions and compressed test results.
- Test Application Speed - The distributed approach facilitates concurrent testing of individual boards, allowing substantial reduction in overall system test time.

4.5 System Level Test Support

When the application boards are finally assembled to form a complete system, they must be tested while it operates in the environment for which it was developed in order to verify its correct operation and diagnose any failures. A high level view of the system hardware development environment that supports SIERA is shown in Figure 4-8. It consists of a VME card cage for housing the application boards, a single-board CPU that runs a real-time customizable operating system kernel, Ethernet board for communicating

with the host workstation, and a UNIX workstation which is used for software development and debugging. To support system level testing, a system must satisfy the following requirements:

1. be capable of accessing chip level test structures;
2. provide control sequences to enable proper execution of chip level test structures;
3. apply test data and collect test results;
4. provide a facility for analyzing test results;
5. be able to test the interconnection between various components on a board via Boundary Scan registers;
6. be compatible with the existing hardware development;
7. operate at system clock rates;
8. be flexible enough to support a variety of testability bus standards such as Boundary Scan;
9. provide access and control of non-Boundary Scan, as well as, analog components for mixed signal applications;
10. be implemented at a low cost.

4.5.1 Test and Diagnosis System

To fulfill the requirements mentioned above, a hierarchical test system called the Test and Diagnosis System (TDS) is described which uses a hierarchy of Boundary Scan test buses embedded into the system's physical hierarchy. In this hierarchy, each chip contains a Boundary Scan interface; all Boundary Scan devices on each board are serially cascaded forming a single scan path where all of the control and test data are applied through a centralized Boundary Scan slave interface; all Boundary Scan slave interfaces on every board are tied to the Boundary Scan master interface on the Test Master Controller board, where test programs direct the execution of all test functions for the entire system; at the next level, the CPU board is used to initialize the Test Master Controller board, which is described in the next section; and finally, at the top-most level, the UNIX workstation

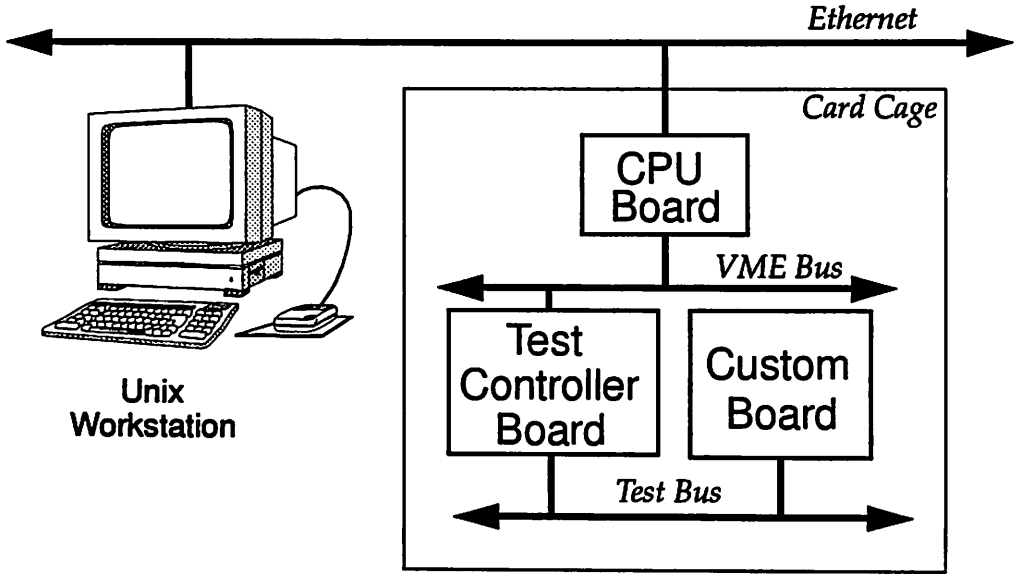


Figure 4-9: Test and Diagnosis system.

provides the user interface for TDS where test vectors are automatically generated and test results are analyzed. A high level diagram of the Test and Diagnosis System is shown in Figure 4-9.

4.6 Board Level Design vs. Test Trade-Off Issues

Design costs are easy to calculate in terms of part costs, manufacturing assembly, design costs, etc. Test costs are also quantifiable but may not be done so easily. Certainly test equipment and test software costs can be readily calculated, but troubleshooting and repair costs are not so obvious. In complex systems, the life cycle costs of the product is dominated by test and maintenance costs, not design and production costs. Therefore, it becomes vitally important to invest time and effort into design for test which will ultimately reduce test and maintenance costs. Some of the design costs such as performance and area costs board level is discussed below.

Partitioning is always a very important consideration for board level testing. Any complex board must have adequate partitioning to allow independent testing of major logic functions. In most designs, partitions can be created by simply replacing the normal buffers and transceivers, which are already required in the design with their Boundary Scan counterparts. By adding these parts, individual functional units on a board such as memory, data bus, or processor can be tested. Repair savings can be achieved through faster troubleshooting and fault isolation of fewer components. By using Boundary Scan devices for partitioning, board failures can be detected and isolated with less probing. Boundary Scan reduces or eliminates the use of test points on a board. Board test logic real estate can be minimized, and, in some cases, board real estate can be gained by using Boundary Scan devices. Board real estate savings can be accomplished by replacing a chip added for test purposes with a Boundary Scan device. Test logic that is added for fault isolation can be efficiently controlled via Boundary Scan test bus.

4.7 Summary

With the system test hierarchy described in this chapter, a user can perform system diagnosis and identify a failed component without removing any parts of the system because all test facilities can be accessed throughout the system hierarchy. By exploiting the module generation facility that already exists in SIERA, testability features can be added to a design with little effort and requiring very little knowledge of how to implement them on the designers behalf. The test hardware modules described in this chapter can perform dedicated test functions that can detect defective components and interconnect on a board. Board level costs associated with the implementation of these features has also been presented.

CHAPTER 5

TEST SOFTWARE: Tools and Languages

As described in Chapter 2, the objective of our test strategy is to integrate test into our system design environment. To realize this strategy requires dedicated software tools. These tools should automate the addition of the test hardware required to implement a DFT methodology, while at the same time, spare the designer of having to know about any implementation specific details. After adding the test hardware, test patterns can then be applied to test the target chip or board interconnect via local test buses. Producing these tests manually is an arduous and tedious task that is very prone to error, especially for large designs. On the other hand, generating test patterns automatically eliminates these problems while producing them efficiently and error free. Furthermore, the test pattern generation task is ideal for automation because many algorithms exist for both combinational circuits and printed circuit board interconnect.

The widespread adoption of the Boundary Scan standard has necessitated the need for a way to simply and effectively describe its implementation specific details in a manner

suitable for software to utilize. The Boundary Scan Description Language (BSDL) [IEEE91b] was developed for this purpose. Two other languages called the Chip Test Language (CTL) [Lien90] and the Module[1] Test Language (MTL) [Lien90] which describe how to use the test features implemented on a chip and board for testing has been developed. These two languages allows the designer to write high level test procedures which are later compiled to produce low level test programs (written in C) which control the operation of Test Master Controller board. With CTL and MTL, test programs can be generated automatically.

This chapter is organized into five sections which cover testability hardware design tools, test generation algorithms and tools used for combinational circuits and board interconnect, and testability hardware description languages and a compiler.

5.1 Testability Hardware Design Tools

To ensure design for testability, the system designer must follow a methodology that addresses testability issues as part of the design process. Much published work on CAD tools, that are now available, support a testability design methodology at the chip level only. However, there also exists a need for tools that support testability design methodologies at the board level. One such tool called JTAGtool described here has been developed to ensure that every Boundary Scan chip, in a board design, is correctly connected to the scan chain.

Some test applications may require the use of a custom test protocol or some new standard comes along requiring a new test protocol, in either case, the architecture of the Test

1. In the context of this work, the term module and board are used interchangeably.

Master Controller (TMC) board is flexible enough to support them. As described in Chapter 3, the TMC uses a programmable device for this purpose. A tool called PLDS [Yu91] is used to map a behavioral representation of the test protocol to the target programmable device, which in this case is a Xilinx Field Programmable Gate Array.

The JTAGtool and the procedure for using PLDS to reconfigure the 1149.n Controller module for the TMC board will be discussed in the sections that follow.

5.1.1 JTAGtool: Boundary Scan Path Routing Tool

The role of JTAGtool is twofold: one, it threads all of the Boundary Scan chips in the design as they appear in the design hierarchy, and two, it generates a file containing the design netlist, which is used in the Module Test Description of the board described later in Section 4.4.3. This tool also eliminates any errors that may otherwise occur when the designer has to manually configure the Boundary Scan path. A block diagram of JTAGtool, which consists of three modules, is shown in Figure 5-1. In the ProcessFacet module, the `structure_instance` view [Shung89] that contains all of the structural information of the design that has been created from a hierarchy of SDL files is flattened down to the PACKAGECLASS property. This will allow JTAGtool to preserve the order of the Boundary Scan chips during creation of the Boundary Scan path. The MakeBScanPath module performs the following tasks:

1. Identifies all of the Boundary Scan master and slave chips present in the design;
2. Cascade all Boundary Scan slave chips in the order in which they appear in the design with the TDI of the first chip connected to a global TDO net and the TDO of the last chip connected to a global TDI net.;
3. The TMS and TCK pins of every slave chip are connected to global TMS and TCK nets;

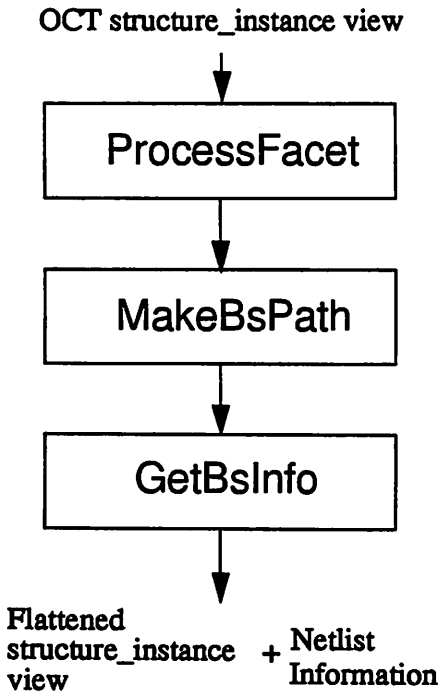


Figure 5-1: Block level diagram of JTAGtool.

4. The global nets TDI, TDO, TMS, and TCK are all connected to a local Boundary Scan master controller chip that is either added automatically by a test module or added manually by the board designer;
5. Finally, the TDI, TDO, TMS, and TCK nets are connected to a 10 pin right angle connector which is to be placed manually by the board designer.

Lastly, the GetBSinfo module extracts all of the pertinent information required for board interconnect test generation such as the board net list.

5.1.2 Test Controller Configuration Tool

One of the most important features of the Test Master Controller board architecture is its reconfigurability. By reconfigurability, we mean hardware that can be changed dynamically or hardware that must be adapted to different user applications. Commercial

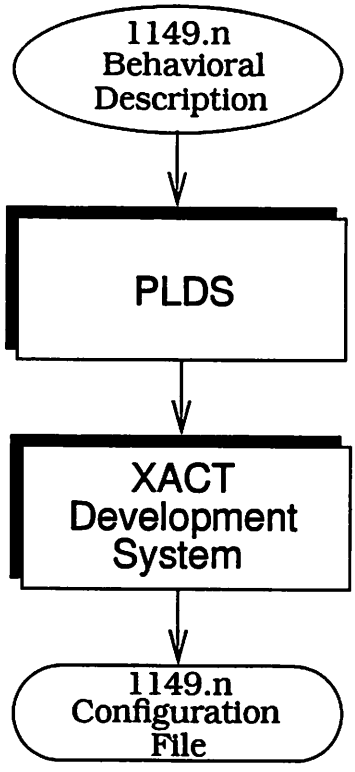


Figure 5-2: Configuration file generation process.

devices such as Field-Programmable Gate Arrays (FPGAs), in particular, the Xilinx XC4000 Logic Cell Array family [Xilinx92], exhibit this feature. These devices can be dynamically reconfigured an unlimited number of times. Xilinx FPGAs comprise three major configurable elements: configurable logic blocks (CLBs), input/output blocks (IOBS), and interconnections. CLBs provide the functional elements for implementing the user's logic. IOBs provide the interface between the package pins and internal signal lines. The programmable interconnect resources provide routing paths to connect the inputs and outputs of the CLBs and IOBs. Reconfiguration is established by programming internal static memory cells that determine the logic functions and their interconnect.

Figure 5-2 illustrates the reconfiguration procedure. The procedure is partitioned into two

steps. In the first step, a file describing the behavior of the 1149.n device to be implemented is used as an input to PLDS whose objective is to provide a solution to efficiently map a high-level description of a design into a set of one or more programmable devices. It provides an interface between the Oct database and commercially available tools supplied by the manufacturer which in this particular case is Xilinx. PLDS produces output files, in Xilinx Netlist Format [Xilinx91], which are then used by the Xilinx XACT Development System [Xilinx91]. Finally, after running the design through the XACT software, an 1149.n configuration file is generated which must be down-loaded to the Test Master Controller board to configure a XC4005 FPGA during system initialization.

5.2 Algorithms for Test Vector Generation

A test for a fault is an input that will produce different outputs in the presence and absence of the fault, thus making the fault effect observable. In a combinational circuit, a specific stuck fault can be tested by a single vector. Stuck faults are not only the simplest faults to analyze, but they also have proved to be very effective in representing the faulty behavior of actual circuits. The simplicity of stuck faults is derived from their logical behavior; so these faults are often referred to as logical faults. Stuck-faults are assumed to affect only the lines between gates. Each line can have two types of stuck faults: stuck-at-1 and stuck-at-0. Thus, a line with a stuck-at-1 fault will always have a logical value 1 irrespective of the correct logical output of the gate driving it. In general, several stuck-faults can be assumed to be simultaneously present in a circuit. A circuit with n lines can have $3^n - 1$ possible stuck line combinations. This is because each line can be in any one of the three states: stuck-at-1, stuck-at-0, or fault-free.

Algorithms for automatic test generation mostly work on the principle of path

sensitization. That is, they attempt to find an input vector that will sensitize a path from the fault site to a primary output. Efficient programs for combinational circuits are available based on well-known algorithms such as D-algorithm [Roth67] and PODEM [Goel81]. They provide the motivation for the use of the Scan path test methodology to convert a sequential test problem to a combinational one. The ultimate goal of test generations is to obtain test vectors of high quality at a reasonable cost. The term fault coverage, which is commonly used to denote test quality, is the ratio of modeled faults detected over the total number of test vectors. For a given fault in a circuit, a test is a set of input stimuli that make the fault effect observable at a primary output. An ideal test generator must be able to find a test for each modeled fault in a circuit at the chip level or a modeled fault in the interconnect between two adjacent chips on a board.

5.2.1 Test Generation Algorithms for Combinational Circuits

A significant theoretical study by Ibarra and Shani [Ibarra75] shows that test generation for combinational circuits belongs to the class of problems called NP-complete, strongly suggesting that no test generation algorithm with a polynomial time complexity is likely to exist. The non-polynomial time complexity here refers to the worst-case effort of test generation in a circuit. Test generation algorithms used in practice appear to be able to achieve slower average time growth by using heuristic search techniques. Goel [Goel81] argues that the time for complete test generation must grow at least as the square of the number of gates in the circuit.

The D-algorithm described in [Roth67] is the most widely used test generation algorithms. In generating a test, the D-algorithm creates a decision structure in which there is more than one choice available at each decision node. Through an implicit enumeration process, all alternatives at each decision node are capable of being examined. For the stuck-at-1

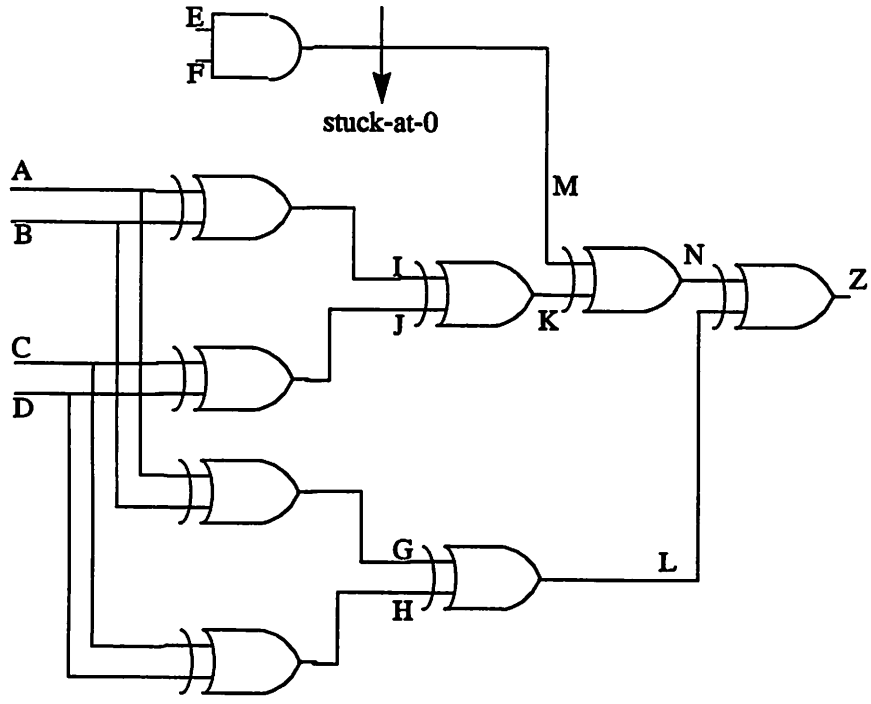


Figure 5-3: Example circuit to illustrate use of D-algorithm.

fault f in Figure 5-3, the D-algorithm typically goes through the following steps:

1. The test for stuck-at-0 requires a logic1 on M for the good circuit. Setting E and F each to 1 results in a D at M, where D designates the correct logic value for a good circuit.
2. Generating a sensitized path from net M to the primary output Z, using recursive intersection of D-cubes, may result in the ordered assignments K = 1 and L = 1 illustrated in the decision diagram in Figure 5-4. Alternative assignments K = 0 and L = 0 are still available for consideration should the present assignments prove futile.
3. The D-algorithm justifies each internal net assignment on a levelized basis. Since the functions P and $\overline{P}$ realized at nets h and i, respectively, are complementary, no justification is possible for the concurrent assignments K = 1 and L = 1. However, in establishing the absence of the justification, the D-algorithm must enumerate 2^3 primary input values before it can correct the bad decision made on L, that is, change the assignment on L from 1 to 0.

A class of circuits for which the D-algorithm performs particularly poorly are those

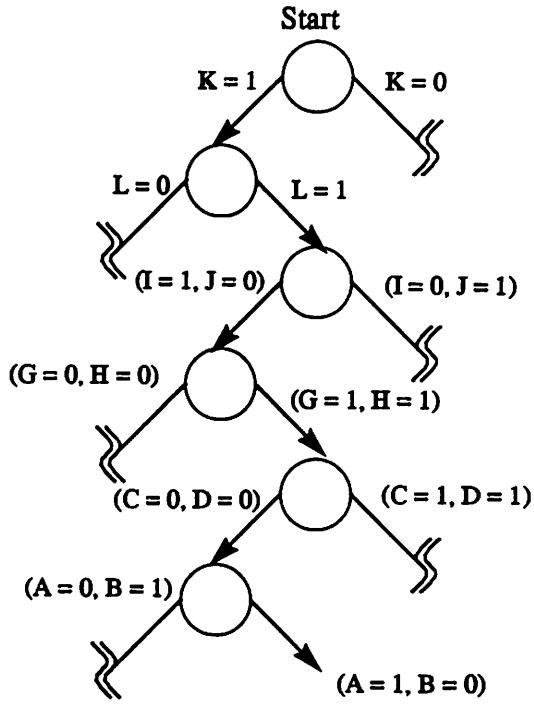


Figure 5-4: Decision tree diagram for D-algorithm.

containing exclusive-or trees. The degradation in performance arises due to excessive amount of backtracking. This motivation has motivated Goel [Goel81] to devise a new test generation algorithm called Path Oriented Decision Making (PODEM) where he uses a branch and bound technique. PODEM implementations are know to run an order of magnitude faster than the D-algorithm on most circuits. The PODEM test generation algorithm is an implicit enumeration algorithm in which all possible primary input patterns are implicitly, but exhaustively, examined as a test for the given fault. The examination of primary input patterns is terminated as soon as a test is found. If it is determined that no primary input pattern can be a test, the fault is untestable. The decision tree structure used in PODEM is shown in Figure 5-5 where all primary inputs are unassigned.

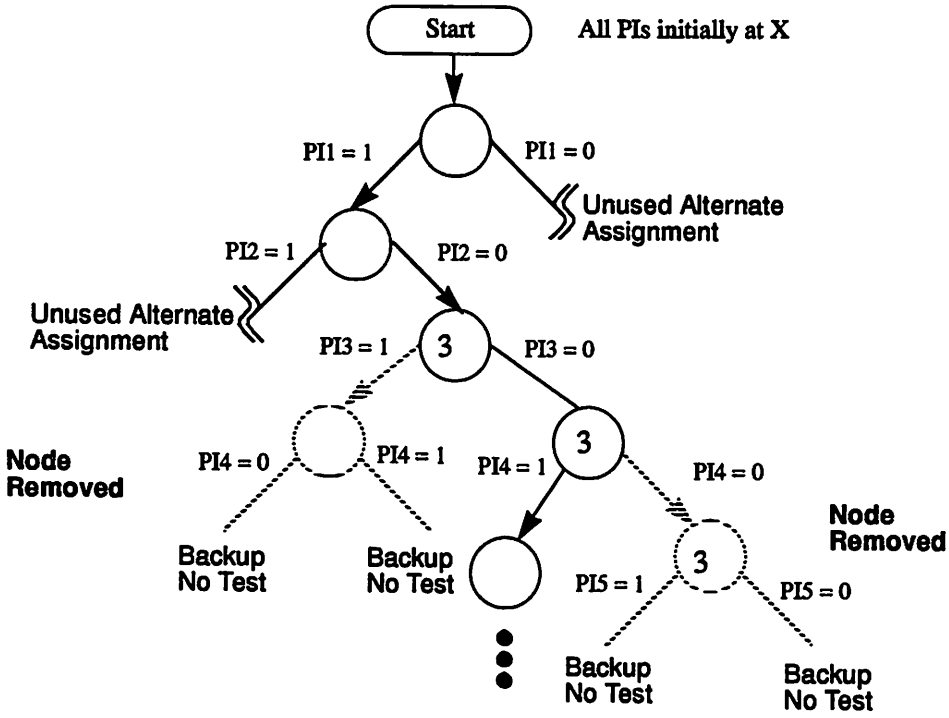


Figure 5-5: Decision tree diagram for PODEM.

An initial assignment ("branch"-in the context of branch and bound algorithms) of either 0 or 1 on a primary input is recorded as an unflagged node in the decision tree. Implications of present Primary assignments uses the five-valued logic described in [Roth67]. The decision tree is an ordered list of nodes with: 1) each node identifying a current assignment of either a 0 or 1 to one primary input, and 2) the ordering reflects the relative sequence in which the current assignments were made. A node is flagged (indicated by a check mark inside the node) if the initial assignment has been rejected and the alternative is being tried. When both assignment choices at a node are rejected, then the associated node is removed and the predecessor node's current assignment is rejected. The last

primary input assignment made is rejected if it can be determined that no test can be generated with the assignments made on the assigned primary inputs, regardless of values that may be assigned to the as yet unassigned primary inputs. The rejection of a primary input assignment results in a "bounding" of the decision tree, in the context of branch and bound algorithms, since it avoids the enumeration of the subsequent assignments to the unassigned primary inputs. In using a branch and bound technique, PODEM solves the test generation problems faced by the D-algorithm.

5.2.2 Test Generation Algorithms for Board Interconnect

Detecting and locating faults on board interconnect has drawn much attention since the emergence of the Boundary Scan standard. Many boards will soon be designed with chips containing the Boundary Scan architecture where during test mode, the chip's I/O pins can be accessed through the Boundary Scan test bus achieving a virtual bed-of-nails capability. Hence, faulty interconnect can be isolated and tested without the need to physically probe the board. Recent work dealing with the problem of generating tests for detecting and locating faults in board interconnect has been reported in [Hansen89][Hassan88][Hassan89][Wagner87]. When testing interconnect on a board, both stuck-at and bridging faults must be considered. Some of these faults are illustrated in Figure 5-6. Since the Boundary Scan path provides direct access to these interconnect, test patterns can be generated which provide 100% coverage of these faults. Because stuck-at faults occur on a variety of bus configurations, different test pattern generation algorithms are required for wired-AND, wired-OR, and tri-state configurations. In [Wagner87], he presents algorithms for generating interconnect test patterns for stuck-at and bridging faults. These algorithms and some of their features are described in the sections that follow.

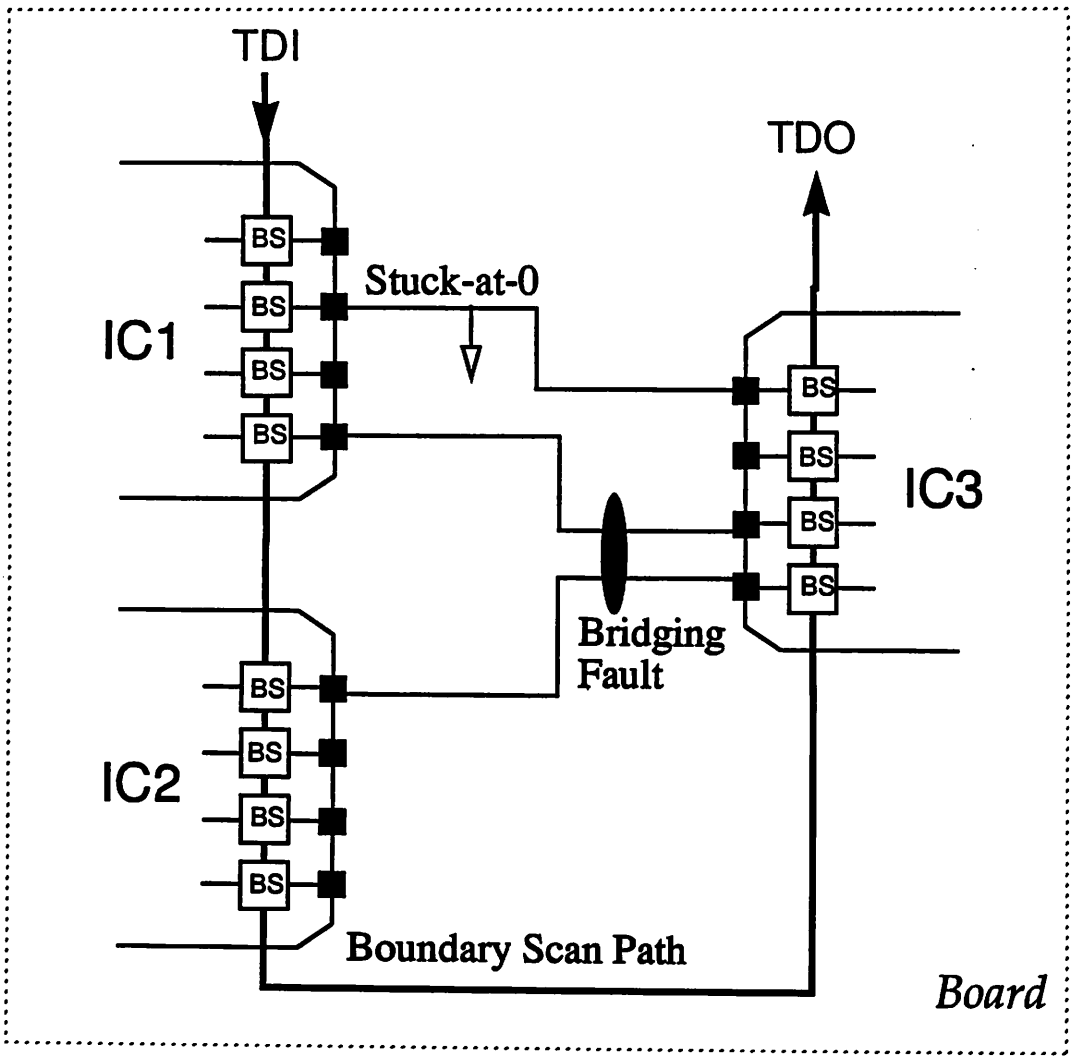


Figure 5-6: Typical printed circuit board interconnect faults.

Testing wired-AND Bus Configurations

As the name implies, the values forced on a wired-AND bus configuration are logically ANDed to obtain the resulting value. Hence, the wired-AND net can be treated in the same way as an AND gate where 100% of all stuck-at faults can be detected with k + 1 test patterns where k is the number of inputs. The test patterns can be divided into k

patterns which test for stuck-at-1 faults and one pattern which tests for all stuck-at-0 faults. The algorithm used for generating tests for a wire-AND configuration is given below.

1. The driver to be tested is set to a logic 0
2. All other drivers on the net are set to a logic 1
3. The data is clocked into the receivers
4. All receivers on the net are examined for a logic 0
5. Repeat steps 1-4 until each driver is tested
6. Every driver is set to a logic 1
7. The data is clocked into the receivers
8. Every receiver is examined for a logic 1

Testing wired-OR Bus Configurations

Generating tests for a wired-OR bus configuration is nearly identical to the wired-AND case. For a wired-OR net with K drivers, 100% of all stuck-at faults can be detected with k + 1 patterns. In this case, the test patterns can be divided into k patterns which test for stuck-at-0 faults and a single pattern which tests for all stuck-at-1 faults. The algorithm used for generating tests for a wired-OR bus configuration is listed below.

1. The driver to be tested is set to a logic 1
2. All other drivers on the net are set to a logic 0
3. The data is clocked into the receivers
4. All receivers on the net are examined for a logic 1
5. Repeat steps 1-4 until each driver is tested
6. Every driver is set to a logic 0
7. The data is clocked into the receivers
8. Every receiver is examined for a logic 0

Testing Tri-State Bus Configurations

When a tri-state bus configuration is used, multiple drivers control one or more receivers as shown in Figure 3-8. Since only a single driver can be enabled at any one time, a special restriction is imposed on the generation of the test patterns. In order to achieve 100% stuck-at fault coverage, each driver on the net must be tested individually for stuck-at-1 and stuck-at-0 faults while the remaining drivers are disabled. Since this requires 2 test vectors per driver, 100% stuck-at fault coverage can be achieved using 2k test vectors where k is the number of drivers on the net. The algorithm for testing this type of bus configuration is provided below.

1. The driver to tested is enabled and set to a logic 1
2. All other drivers are set to a logic 0 and disabled
3. The data is clocked into the receivers
4. The receivers are examined for a logic 1
5. Repeat steps 1-4 until all drivers have been tested
6. The driver to be tested is enabled and set to a logic 0
7. All other drivers are set to a logic 1 and disabled
8. The data is clocked into the receivers
9. All receivers are examined for a logic 0
10. Repeat 7-10 until all drivers have been tested

Bridging Fault Test Pattern Generation

In addition to testing for stuck-at faults, testing for bridging faults must also be considered. A bridging fault occurs when two nets are electrically connected as shown in Figure 5-6. The algorithm for detecting this fault is given below.

1. Enable the drivers on each net
2. Apply a logic 1 to all drivers on the first net
3. Apply a logic 0 to all drivers on the second net

4. Clock the data into the receivers
5. Examine at least one receiver on each net
6. If the data at the receiver of either net does not correspond.

With the data applied to the respective driver, then a bridging fault exists between the nets. This algorithms is good for only two nets. Since a typical board may contain hundreds of interconnection nets, this algorithm must be applied to every possible pair of nets to achieve 100% fault coverage.

5.3 TGS - A Test Vector Generation Tool for Combinational Circuits [USCTG88]

The Test Generation System (TGS) is designed for generating test vectors for combinational circuits described at the gate level. The gate types supported by the system include AND, OR, NAND, NOR, INV (inverter), BUF (buffer), and INPT (input gate). The system provides the following functions:

a. Fault collapsing

b. Test Vector Generation

c. Fault Simulation

d. Integration of a, b, c, to derive a complete set of test patterns. Each of these functions will be described briefly.

The main objective of fault collapsing is to classify the set of all possible stuck faults in order to reduce the total number of tests. Typically, a test for an arbitrary fault detects several other faults in the circuit. The test vector generation process provides a test vector for any given detectable fault (meaning a test can be generated for it). The system uses the PODEM test generation algorithm described in the previous section. Given enough time

PODEM will find a test for the target fault if it is detectable. Fault simulation attempts to identify all faults that can be detected by a given input vector. It provides a list of faults and an associated primary output. If any fault in this list is injected into the circuit, then the logic values of the good circuit and the faulty circuit will differ at the associated primary output under the given input vector. Besides the functions described above, an integrated system which combines fault collapsing, test generation, and fault simulation into one complete test system is provided. The purpose of this system is to execute a complete test generation procedure without any user intervention, once the required input parameters are set up. Figure 5-7 shows high level block diagram of the integrated system.

5.3.1 oct2tgs - OCT to TGS Translator [Bomdica90]

Before combining the combinational logic blocks of a chip with its other functional blocks such as ALUs or RAMs, the combinational blocks must be processed using oct2tgs which is a conversion utility that generates a TGS format circuit description from a flattened OCT `structure_instance` view (default view) or symbolic view of the chip. It decomposes macro cells, such as multiplexers or decoders into primitive logic descriptions which are given in the technology file. Further, it ignores latches, like the scanlatch in the stdcell library, and treats the logic between the latches as an independent logic block and later combines them to form a top level TGS input file. After running this input file through the TGS system to produce test vectors, they can be used to test the combinational logic blocks via Scan Path.

5.4 Testability Hardware Description Languages

Recent adoption of the Boundary Scan standard has prompted the need for development of dedicated languages which describe the testability features implemented

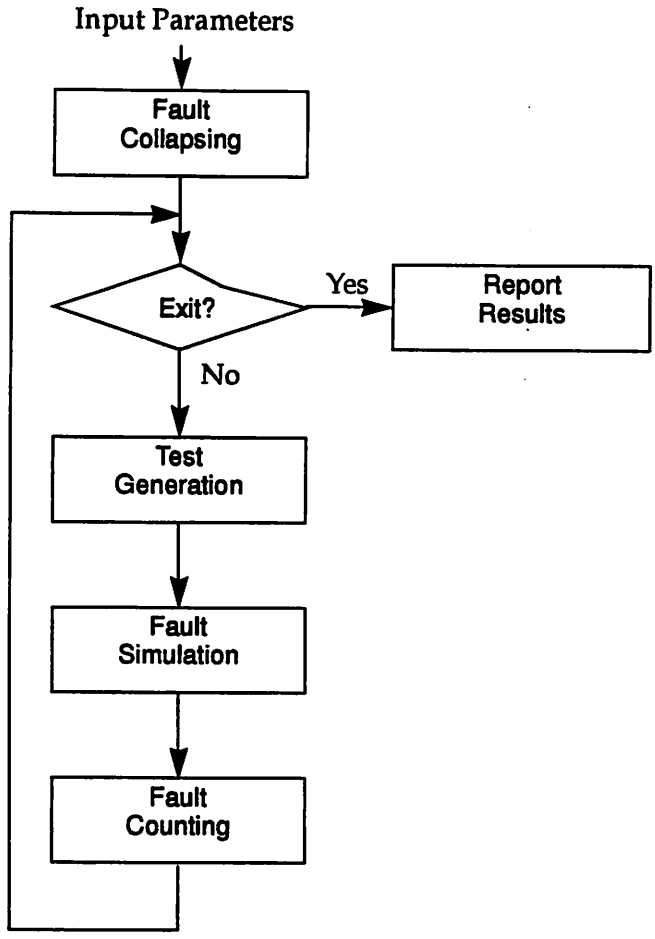


Figure 5-7: Block level diagram of TGS.

at both the chip and board levels. Not only should these languages describe implementation specific details, but they should also describe how to use these features to test chips and boards that have them. To accommodate these needs, several languages have been developed. These include the Boundary Scan Description Language (BSDL) [IEEE91b] which is proposed as a supplement to the Boundary Scan standard and two other languages developed by Lien called Chip Test Language (CTL) [Lien90] and Module Test Language (MTL) [Lien90]. BSDL was developed to describe the implementation specific details of a chip containing the Boundary Scan architecture,

whereas, CTL and MTL were developed were developed to describe how to test a chip or board that uses the Boundary Scan architecture. Features of these languages along with some examples will be presented in the following sections.

5.4.1 BSDL - Boundary Scan Description Language

As more commercial chips become available that support the Boundary Scan standard, each will have the problem of how to describe their unique application of the standard. Some sort of description will be necessary for describing these chips. This section describes a language that captures the essential features of an implementation. This language is called the Boundary Scan Description Language [IEEE91b] and is written within a subset of the VHSIC Hardware Description Language (VHDL) [IEEE88]. The goal of the language is to facilitate communication between chip manufacturers, designers, and tools that need to exchange information on the design of the test logic that complies with the standard. The BSDL language allows description of the testability features in Boundary Scan compliant devices. This language can be used by tools that make use of those testability features. Such tools include testability analysis, test generation and failure diagnosis. With additional capabilities provided by VHDL, it is possible to perform simulation, verification, compliance analysis, and synthesis function.

Boundary Scan Features

What are the Boundary Scan that require a description? All Boundary Scan compliant devices must contain three major parts: a Test Access Port, a TAP controller, and a Boundary Scan register all of whose parameters are described in BSDL. The Boundary Scan register consists of Boundary Scan register cells which are associated with a chip's input, output, bidirectional, and tri-state pins. The Test Access Port contains either four or five dedicated signals, namely TCK, TMS, TDI, TDO and the optional TRST*. It must

also contain a TAP controller, an instruction register, and a Bypass register. The controller implements a minimum set of mandatory instructions which control the operation of the Boundary Scan test logic.

Language Elements

The language consists of a case-insensitive free-form multi-line terminated syntax which is a subset of VHDL. Comments are any text appearing between a "--" and the end of a line. BSDL is composed of three sections which are the entity, package, and package body. An entity is the basis for describing a chip within VHDL. An example of a BSDL file for a TI ACT74SN8244 is given in Appendix A. Within the entity, the Boundary Scan parameters of a chip are described. The 1149.1 related definitions come from a pre-written, standard VHDL package and package body. The definitions for a Boundary Scan package and package body can be found in [IEEE91b]. The package information is directly related to the Boundary Scan standard and development of new standards would require new packages to be created.

The Entity Description

An entity describes a chip's I/O port and important attributes of the chip. For BSDL, an entity has the following structure:

```
entity chipname is -- an entity for chipname
[generic parameter]
[logic port description]
[use statement(s)]
[package pin mappings]
[scan port identification]
[TAP description]
[Boundary Scan Register description
end chipname;
-- End description
```

Generic Parameter

The generic parameter is a VHDL construct used to pass data into a VHDL model. In

BSDL, it is intended as a method for selecting among several packaging options that a chip may have. Each option may have a different mappings between the pins of the package and the bonding pads of the chip. This is called the logical-to-physical relationship of the signals of the chip. The description of the Boundary Scan architecture of the chip is done using logical signals. Applications such as board testing will need to know how the logical structure of the chip maps onto a set of physical pins. For this, a VHDL generic parameter is used. It must have the name shown in order for the software to distinguish it from other parameters that might be passed to the entity. It has the following form:

```
generic(PHYSICAL_PIN_MAP:string:="undefined");
```

Logical Port Description

The port description uses the VHDL port list. It is used to assign meaningful symbolic names to the chip's I/O pins. The inclusion of non-digital pins such as power, ground, or analog signals in the BSDL port description is optional, however, they are recommended for completeness. The logical port description has the following form:

```
<logical port description>:==port(<pin spec>; {<PinID>});
<pin spec>::=<identifier list>:<mode><pin type> <identifier
list>:=<VHDL identifier> {,<VHDL identifier>}
<mode>:== in | out | buffer | inout | linkage
<pin type>:==bit | Bit vector {<range>}
<range:=<numberic constant> to
<numeric constant> downto <numeric constant>
```

The value of mode indicates the direction of signal flow and linkage is used for power, ground, or analog pins.

Use Statement(s)

The use statement identifies the VHDL package required for defining attributes, types, constants, and other items that will be referenced.

Package Pin Mapping

VHDL attribute and constant statements are used to show the package pin mapping. These are shown by example:

```
attribute PIN_MAP of chipname:entity is PHYSICAL_PIN_MAP;
constant dw_package:PIN_MAP_STRING:=<MapString>;
```

Attribute PIN_MAP is a string that is set to the value of the parameter PHYSICAL_PIN_MAP, described above. VHDL constants are then written, one for each package variation, that describe the mapping between the logical and physical pins of the chip. An example of a mapping is:

```
"CLK:1,DATA:(6,7,8,9,15,14,13,12),CLEAR:10,"&
"Q:(2,3,4,5,21,20,19,18), VCC:22, GND:11"
```

The symbol on the right of the colon is the physical pin associated with the port signal. It may be a number or an alphanumeric identifier because some packages like Pin Grid Arrays (PGAs) use coordinate indentifiers like A07 or H13. If signals like DATA are <PinVector>'s in the definition, then a matching list of pins enclosed in parenthesis are required. The physical pin mapped onto DATA[5] is pin 15 in the example above.

Scan Port Identification

Five attributes define the scan port of the chip. These signals are shown below:

```
attribute TAP_SCAN_IN of TDI:signal is true;
attribute TAP_SCAN_OUT of TDO:signal is true;
attribute TAP_SCAN_MODE of TMS:signal is true;
attribute TAP_SCAN_RESET of TRST:signal is false;
attribute TAP_SCAN_CLOCK of TCK:signal is true is
(17.5e6, BOTH);
```

Here, signal names TDI, TDO, TMS, TRST*, and TCK must appear in the port description. The TAP_SCAN_RESET attribute is optional but the others must be specified for correct implementation. The TAP_SCAN_CLOCK attribute is a record with a real number field that gives the maximum operating frequency for TCK. The second field is an

enumerated type with values LOW and BOTH which specify with state(s) the TCK signal may be stopped in without data loss in the Boundary Scan mode.

TAP Description

The next major part of the Boundary Scan architecture that must be described is the chip dependent characteristics of the TAP. It may have four or five control signals, already identified. It may have a user specified instruction set and a number of data register and options.

The Instruction Register may have any length 2 bits or longer and is required to support certain opcodes and some of these have mandatory bit patterns. A circuit designer may add optional instructions and/or new instructions with completely dedicated functions. An instruction may have several bit patterns. Unused bit patterns will default to the BYPASS instruction. The standard also has provisions for private instructions. The characteristics of the instruction register that are captured with the language are length, opcodes, capture, disable, private, and usage. Some examples of these are given below:

```
attribute INSTRUCTION_LENGTH of Chip1:entity is 4;
attribute INSTRUCTION_OPCODE of Chip1:entity is
   "Extest (0000)," & "Bypass (1111)," & "Sample (0001),";
attribute INSTRUCTION_CAPTURE of Chip1:entity is "0101";
attribute INSTRUCTION_DISABLE of Chip1:entity is "Hi_Z";
attribute INSTRUCTION_PRIVATE of Chip1:entity is "Secret";
```

The instruction_length attribute defines the length of all opcode bit patterns. The instruction_opcode attribute is a BSDL string containing the opcode identifiers and their associated bit patterns. The rightmost bit in the pattern is closest to TDO. The standard mandates the existence of EXTEST, BYPASS, and SAMPLE instructions with mandatory bit patterns for the first two. The instruction_capture attribute string determines what bit pattern is loaded into the instruction register when the TAP controller enters the Capture_IR state. The optional instruction_disable attribute identifies an opcode that

makes a Boundary Scan chip disappear. In this mode, the tri-sate outputs are disabled and the BYPASS register is placed between TDI and TDO. The optional instruction_private attribute identifies opcodes that are private and potentially unsafe for access. Software can monitor the instruction register to issue warnings or errors in a private instruction is loaded during run time. The optional instruction_usage is a means for describing static design parameters of a Boundary Scan implementation. The standard contains two instructions whose details of operation are not statically defined, which are RUNBIST and INTEST. The instruction_usage provides additional information about the operation of an instruction. The types of information needed are: register, identification, result identification and clocking information. Below are examples for describing the RUNBIST and INTEST instructions:

```
attribute INSTRUCTION_USABE of chipname: entity is
Runbist (registers Boundary, Signature;" &
   "result 0011010110000100;" &
   "clock TCK in Run_Test_Idle;""length (clock 4000 cycles)," &
   "Intest (clock TCK shifted)";
```

The RUNBIST usage shows that two registers are used, the Boundary Scan register and a second register called Signature. When the test is complete, the result shifted out from Signature should match the given pattern. The test is run by clocking TCK for 4000 cycles while in the Run_Test_Idle controller state. The INTEST usage shows that shifting of the internal Scan register occurs every TCK.

Register Access

All TAP instructions must place a shiftable register between TDI and TDO. User-defined instructions may access the Boundary Scan register, IDCODE register, or BYPASS register. The standard allows additional data registers in the design. These are referenced by user-defined TAP instructions. It is important for software to know the existence and length of these registers and their corresponding instruction. An attribute

has also been provided for this purpose. The attribute for this is:

```
attribute REGISTER_ACCESS of chipname:entity is
   "Boundary (Secret, User1)," &
   "Bypass (Hi_Z, User2)";
```

In this example, Secret, User1, User2, and Hi_Z must be previously defined user instructions. This ability to identify register access allows software to know the length of a scan sequence, which is dependent on the current instruction.

Boundary Scan Register Description

The Boundary Scan register is an ordered list of Boundary Scan cells, numbered 0 to N with cell 0 closest to TDO. These cells vary in design and purpose. Cells must be identified before thay are referenced in the Boundary Scan register description. Three attributes are required to define a register. Examples of their usage are given below:

```
attribute BOUNDARY_CELLS of chipname: entity is "BC_1, MyCell";
attribute BOUNDARY_LENGTH of chipname: entity is 3;
attribute BOUNDARY_REGISTER of chipname: entity is "
   0 (BC_1, IN, input, X)" & "
   1 (BC_1, *, control, 0)" & "
   2 (MyCell, OUT, output3, X, 1, 0,Z)";
```

The first attribute defines the cells used to construct the register. The second attribute defines the number of cells in the Boundary Scan register. This third attribute is a string containing a list of elements, each with two fields. The first field is merely the cell number, which must be between 0 and LENGTH-1. The second is a set of subfields within the parentheses. There can be from four to seven subfields labeled: cell, port, function, safe, ccell, disval, and rslt. All cells at least contain the first four subfields. Only cells providing data for device outputs that can be disabled contain the remaining three subfields. These three determine how to disable the output. The cell subfield identifies the cell design used. The port subfield identifies the port signal that is driven or received by this cell. The function subfield indicates the primary function of the cell. The safe subfield gives the

value that a designer prefers to be loaded into the cell, while the ccell subfield identifies the cell number of the cell that serves as an output enable. The disval subfield determines the value that ccell must have to disable the output driver and the rslt subfield determines the state of the driver when it is disabled.

Defining a Boundary Scan Register Cell

A cell is defined as a VHDL constant. It is an array of records with the range of the array unspecified, but implicit from the number of records given in the constant definition. An example of a Boundary Scan register cell called C_Ex_1 that supports EXTEST, SAMPLE, and INTEST. It loads a '1' during the EXTEST if the cell is used for an input, output or control function, where output2 is a 2 state output function and output3 is a 3 state output function. During INTEST, as an input, it reloads the cell with the data value that was shifted into it. The description for this cell is given below:

```
constant C_Ex_1: CELL_INFO:= (
   (Output2, Extest, One),
   (Output3, Extest, One),
   (Output2, Sample, PI),
   (Output3, Sample, PI),
   (Output2, Intest, PI),
   (Output3, Intest, PI),
   (Control, Extest, One),
   (Input, Extest, One),
   (Control, Sample, PI),
   (Input, Sample, PI),
   (Control, Intest, PI),
   (Input, Intest, PI));
```

This is only an overview of the BSDL language. It is an extensible language for defining the basic testability features of a device implemented with the Boundary Scan standard architecture. It is specifically designed for describing implementations, in a way such that they can be exploited by software tools. For additional information, the reader is referred to [IEEE91b]. BSDL descriptions for some common parts are given in Appendix A.

5.4.2 CTL - Chip Test Language

The Chip Test language [Lien91] was developed to be a superset of BSDL. It too, has a VHDL-like syntax. As mentioned in the BSDL section, BSDL can be used to describe the Boundary Scan architecture features implemented on a chip, however, it does not describe how to utilize these features to test a chip. CTL describes how to utilize the Boundary Scan features for testing through the use of a Test Procedure, which provides the information required for testing a chip. A chip test description (CTD) is written in CTL which includes a Test Procedure in addition to the original BSDL description. The incorporation of the Test Procedure is achieved by adding a VHDL attribute called TEST_PROC. The example below demonstrates how a Test Procedure is incorporated in a CTL file.

```
attribute TEST_PROC of chip1: entity is "Test_Begin" &
TDM 1 = FULLSCAN;" & "REG=SCANPATH,VECFILE=tstvec1,
RESFILE=results1;" & "REG=BOUNDARY,VECFILE=tstvec2,
RESFILE=results2;" &
CLOCK = FCK 1.0 CYCLES_IN RUN_TEST_IDLE;" & "Test_End";
```

It is also possible to forgo using this attribute and describe the Test Procedure in a separate file, where the quotes and & are excluded. For example, the example above can be re-written in a file called chip1.ctp as follows:

```
TEST_PROC of chipname: entity is TEST_BEGIN
TDM <tdm_id> = FULLSCAN; REG=   <reg1>,   VECFILE=<file1>,
RESFILE=<file2>;
CLOCK = FCK <number1> CYCLES_IN RUN_TEST_IDLE; TEST_END
```

A Test Procedure consists of one or more test sequences which will test different parts of a chip according to a predefined test methodology called a Testable Design Methodology (TDM) [Breuer84]. According to Breuer, a TDM deals with the entire process of designing an easily testable structure, developing the test programs, where appropriate, and testing the structure using external and/or built-in-test hardware. Some example TDMs include Scan-Path and Built-In-Self-Test. TDMs are categorized into two types:

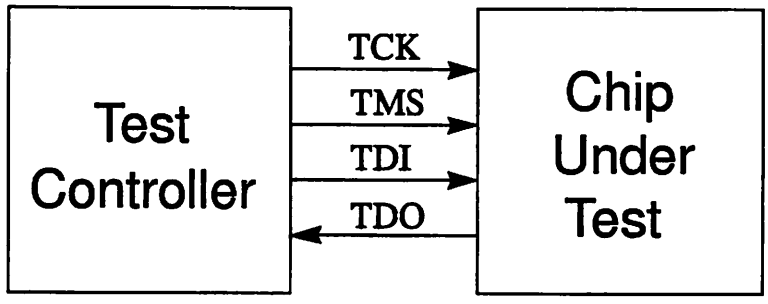


Figure 5-8: Test control model used in CTL.

template-based and user-defined. A template-based TDM is used to describe the procedure for testing a chip designed with a commonly used TDM such as FULLSCAN (Scan Path). A user-defined TDM is used to describe an arbitrary procedure which is written by the designer using the C programming language. The Test Procedure in the example above uses the FULLSCAN TDM which executes a Scan Path test procedure. The tdm_id identifies the TDM in the CTL file. The selected scan register is reg1, which must be previously defined in the CTL file. The test vectors are stored in file1 and the test results are stored in file2. After a test vector is scanned into the Scan Path register, it is necessary to apply number1 cycles of clock FCK to the chip under test before test results are available for scanning out. Note that the test bus must be kept in the RUN_TEST_IDLE controller state during the application of clock FCK. The test control model used by CTL is shown in Figure 5-8. This control model consists of a Boundary Scan Master controller which orchestrates the test process via Boundary Scan test bus and the device under test. In summary, TDMs are dedicated test functions that execute the test sequences required

for implementing the desired test. Template-based TDMs that are currently supported include: FULLSCAN, RUNBIST, and INTEST. The function of these TDMs are described below:

FULLSCAN TDM

The FULLSCAN TDM is used to test a chip designed with the Scan Path technique, where all flip-flops on the chip are made scannable and are cascaded to form a serial scan chain. It executes a Scan Path test in the following steps:

1. Load a test vector into the scan chain by shifting s times where s is the length of the Scan Path register.
2. Repeat for t-1 times t is the number of tests vectors Update the scan chain by running the functional clock for one cycle. Scan out the previous result while scanning in the next test vector.
3. Grab the last result by shifting s times.

RUNBIST TDM

A chip that utilizes BIST hardware can be tested using the RUNBIST public instruction defined in the Boundary Scan standard. Once the RUNBIST instruction is loaded into the instruction register of the TAP, the self-test procedure can be executed simply by applying the test clock, TCK, during the RUNT_TEST_IDLE controller state. In the RUNBIST TDM example below, the result of the test is stored in the reg1 register. Value1 represents the expected good circuit result.

```
TDM <tdm_id> = RUNBIST;
CLOCK = TCK <number1> CYCLES_IN RUN_TEST_IDLE; EXPECTED_RESULT
<reg1> = <value1>;
```

INTEST TDM

A chip can also be tested using the INTEST instruction defined in the Boundary Scan standard. The INTEST instruction must be loaded into the instruction register of the Boundary Scan architecture before execution is started. This TDM differs from

FULLSCAN in that only the Boundary Scan register is included in the scan chain and no internal scan path registers are used. The procedure for testing a chip using the INTEST TDM is described in the following steps:

1. Repeat for t times a. Shift a test vector into the boundary scan register.
2. Apply one or more functional clock cycles.
3. Scan out the results of the Boundary Scan register.

The INTEST TDM is listed as follows:

```
TDM <tdm_id> = INTEST; VECFILE = <file1>, RESFILE = <file2>;
CLOCK = FCK <number1> CYCLES_IN RUN_TEST_IDLE;
```

User-defined TDM

Some additional C functions have been developed to assist the designer in describing user-defined TDMs. A brief description of these TDMs are given below:

ScanIR (outS);

This TDM is used to load the Boundary Scan instruction register with the contents of string *outS* while the previously instruction is scanned out at this time. The format of the instruction is determined by the chip designer except for those instructions defined in the Boundary Scan standard.

ScanDR (outS);

This function is similar to the ScanIR except that the target register is one of the chip's test data

registers which is determined by the contents of the instruction register. When scanning in a new data string, the results of the previous string are scanned out.

Bring2State (i);

This function is used to change the TAP controller state from its current state to state i

which can be any one of the following states: Run_Test_Idle, Test_Logic_Reset, Scan_DR, Pause_DR, Scan_IR, or Pause_IR.

RepeatState (i, n);

This function allows the TAP controller to remain in state *i* for *n* consecutive clock cycles. Note that this function can only be applied to the Pause_DR, or Pause_IR states.

RunTest (n);

This function is used to keep the TAP controller in the Run_Test_Idle state for *n* consecutive clock cycles when exercising the BIST circuitry implemented on the target chip.

The formal definition of the CTL syntax is done using YACC [YACC78] and can be found in [Lien91].

5.4.3 MTL - Module Test Language

The Module Test Language (MTL) is a high level language that can be used to describe how to test a module. It has been developed such that it requires very little knowledge of testing, on behalf of the designer, to use it. The test control model used in MTL is shown in Figure 5-9, where a board consists of one to many Boundary Scan chips which are serially cascaded forming a scan chain and a Boundary Scan master controller chip which can access the chip's Boundary Scan circuitry through two Boundary Scan rings. The test clock TCK which is connected to each chip is not shown.

A module test description (MTD) which contains all the information required for testing the board must accompany each board design A board test program is automatically generated from the board test description and chip test descriptions of the parts that make up the board using a tool call *m2c* which will be described in the next section. An MTD

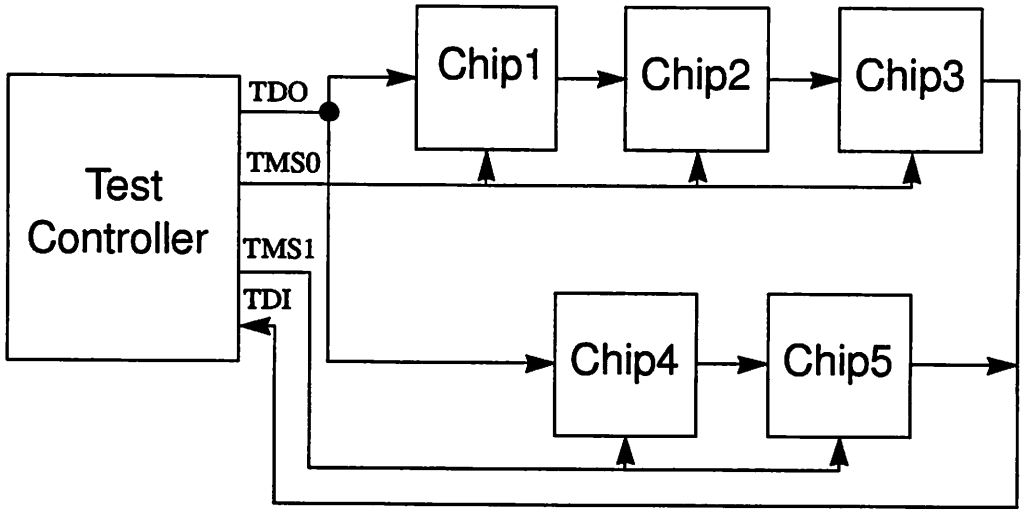


Figure 5-9: Test control model used in MTL.

consists of the following parts: library_id, device_list, test bus configuration, net_list, and test_procedure. The library_id points to the directory containing the CTDs. The device_list ties every chip used on the board with its corresponding CTD in the library. An example device list consisting of two devices is given below:

```
device_list = (Chip1 adder) (Chip2 multiplier);
```

The test bus configuration describes how the chips on the board are configured and how they are connected to a Boundary Scan master controller chip via Boundary Scan test bus. The the chips may be configured in a ring, star, or combination of both topology. In a MTD, a test bus is modeled as a multiple ring topology which can be mapped into any one of these topologies. A ring configuration is formed when all chips exist in the same path, while a star configuration is formed when every ring contains only one device. The test bus for Figure 5-9 is given below.

```
test bus =
ring 0: Chip1 => Chip2 => Chip3,
ring 1: Chip4 => Chip5;
```

The net_list describes how the chips on a board are physically connected. Each terminal of a chip is specified by two names, the first name specifies the chip name, while the second specifies the I/O pin number. An example consisting of two nets is given below.

```
net_list =
net 1: (Chip1 inp1) (Chip2 outp1),
net 2: (Chip2 inp1) (Chip3 inp1) (Chip1 outp1);
```

The test procedure contains the necessary information required for testing a board. This information is represented in terms of standard C code and some test-specific functions. These functions assume no knowledge about the test controller and can be translated to low level functions which are fully supported by a library of test functions written in C.

These I/O functions are used to control the Test Master Controller board or the Local Boundary Scan master controller chip located on the target board. With every chip on a board containing the Boundary Scan architecture, it is possible to write a test procedure for testing the target board using only test-specific functions. Hence, a designer with little knowledge of testing can easily write a test procedure thus greatly reducing test program development time. The only case where testability knowledge is required are situations involving boards which contain a mix of Boundary Scan and non-Boundary Scan components. A brief description of the test-specific function that are used to describe a test procedure are listed below.

Testchip (chip_id);

A Boundary Scan master controller chip or the Test Master Controller board can test a chip, identified by *chip_id*, by executing this function. To impalement this test, the MTL compiler, m2c, uses the chip test description (written in CTL for each chip), which contains one or more test procedures.

Testchip (chip_id) Use TDM (tdm_id);

This function allows part of the target chip to be tested using a TDM, designated by *chip_id* and *tdm_id* respectively. With this function, parts of a chip can be tested in different time intervals where one or more test sessions are executed in each interval.

TestInet ();

This function performs the interconnect test on the target board where every net connecting at least two boundary scan devices is tested. The algorithms used for test pattern generation are described in the section on Test Vector Generation. This function is only used to detect the presence of faults on a net, not for diagnosis.

DiagnosisInet ();

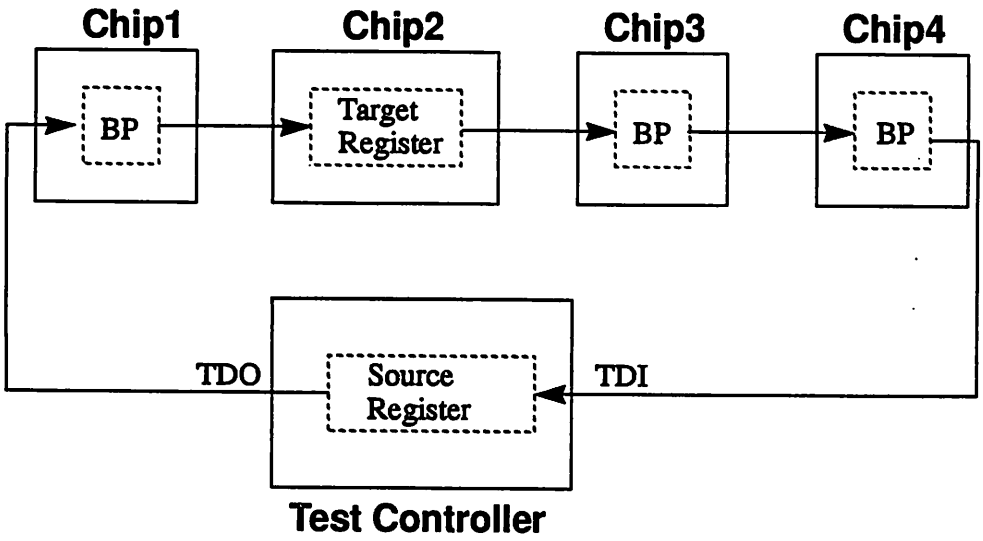
This function is used to diagnose faulty nets on the target board. This function uses a universal test set that includes a walking ones sequence, a walking zeroes, all zeroes vector, and all ones vector. According to [Lien91], all diagnosable faults can be identified using this function.

SampleRing (ring_id);

A snap shot of the current stated of the target ring, designated by *ring_id*, can be achieved using this function. The value returned by this function is a string of 0's and 1's representing the current state of the chips in the Boundary Scan ring.

ScanIR (ring_id, outS);

This function sends instructions to the instruction registers of all the devices under test that are part of the test bus ring designated by *ring_id*. When sending instructions, all chips in the selected scan ring receive a new instruction which is contained in the string *outS*.



Note: BP = Chips in Bypass mode

Figure 5-10: Example configuration with several chips bypassed.

ScanDR (ring_id, preDR, postDR, outS);

This function is similar to the ScanIR except that it sends a string of 0's and 1's contained in *outS* to the selected test data register. For situations where it may be necessary to bypass every chip except the one you're currently sending data to, preDR and postDR are used to indicate the number of bypass registers that appear before and after the test data register of the target chip. The number of shifts required for transmitting data to the target chip is calculated automatically using preDR and postDR. This situation is best illustrated in Figure 5-10. The formal definition of the Module Test language syntax can be found in [Lien91].

5.5 m2c - MTL to C Language Compiler

The m2c compiler generates a test program for the target board from its MTL description and the CTL descriptions of each chip on the board. m2c produces a test program written in ANSI C format that can be used to test the board and a file which describes the board interconnect. A block diagram of m2c shown in Figure 5-11. It consists of a parser module, a template-based TDM module, an interconnect test and diagnosis module, a shift adjustment module, and a test program generation module. Some of the more important modules are described below.

5.5.1 Template-based TDM Module

Meta-procedures that can generate C programs from a template-based TDM are provided. They require a test procedure as an input and they generate a C program for executing the test process of the selected TDM. These meta-procedures consist of *callfullscan, callintest, and callrunbist* where each procedure generates programs for the Fullscan, INTEST, and RUNBIST TDMs respectively. All information required for generating a test program must be provided to these meta-procedures. For example, the following information listed in Table 5-1 is required when using the meta-procedure callfullscan. These values are extracted directly from the CTL and MTL descriptions.

5.5.2 User-defined TDM Module

A user defined TDM is a C program including some test-specific functions. It is necessary to translate them into normal C statements that can be readily executed by the Test Master Controller board or Boundary Scan master controller chip. Because of the differences between the two control models used in CTL and MTL, the test-specific functions are modified to reflect these differences. For example, the test-specific function

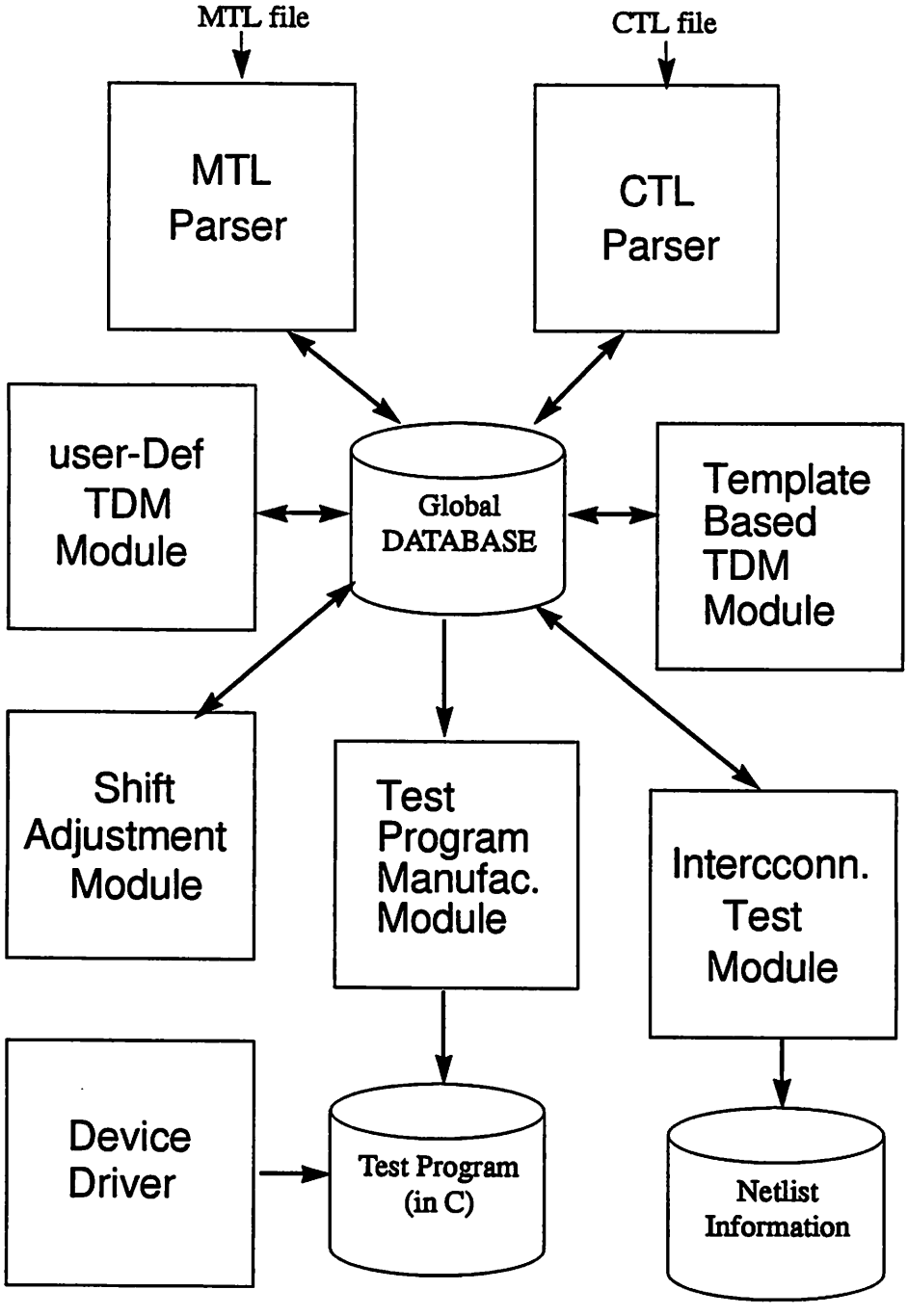


Figure 5-11: Top level view of m2c.

Parameter	Description
chipID	the name of the chip under test
chipType	the type name of the chip under test
ringID	the test bus ring where the chip under test is located
pre	the number of cells between the chip under test and the controller when shifting data
post	the number of cells between the controller and the chip under test when shifting data
ipre	the number of cells between the chip under test and the controller when shifting an instruction
ipost	the number of cells between the controller and the chip under test when shifting an instruction
regIns	the instructions that are used to select the register
vecFID	test data input file name
expFID	test data output file name

Table 5-1 : Information required by meta-procedure *callfullscan*.

scanIR (outS) is converted to *scanIR (RingID, ipre, ipost, outS)* so that the same string of data *outS* can now be sent to the correct chip under test. The parameters of the latter scanIR function are computed by the Shift Adjustment Module described next.

5.5.3 Shift Adjustment Module

In some cases, it may be desirable to send and receive test data to one specific chip in the scan ring while keeping the other chips in the bypass mode. When exchanging data with a single chip in a scan ring, the number of shifts must be adjusted so that data is sent to and received from the chip under test correctly. The number of shifts is determined by the various cases listed in Table 5-2. (Note that the parameter len in the table is the size of the source Boundary Scan register.)

Cases	*pre > post*	*pre < post*	*pre = post*
Number of Shifts	*len + pre*	*len + post*	*len + post*
Number of leading zeroes added when applying test data	*pre - post*	*0*	*0*
Number of trailing zeroes added when applying test data	*post*	*post*	*post*
Number of leading bits ignored when receiving test data	*pre*	*pre*	*pre*
Number of trailing bits ignored when receiving test data	*0*	*post - pre*	*0*

Table 5-2 : Cases used for shifting calculation.

Case 1: *pre > post*

In this case, the total number of shifts is len + pre. It is necessary to add *pre - post* leading zeroes to the output string before loading it into the memory of the Test Master Controller board, such that after shifting, the output string will be properly loaded into the target chip's selected test data register. It is also necessary to discard the first *pre* number of bits so that the correct string is received by the target chip's test data register.

Case 2: *pre < post*

In this case, the total number of shifts is *len + post*, hence, no leading zeroes are require. However, it is necessary to ignore the first *pre* bits received by the memory of the Test Master Controller board. It is also necessary to ignore the last *post - pre* bits received by the memory so that the contents of the source Boundary Scan register can be collected properly.

Case 3: *pre = post*

In this case the total number of shifts is *len + post* and requires no leading zeroes. However, it is necessary to ignore the first *pre* number of bits captured by the memory of

the Test Master Controller board so that the source data can be collected properly.

5.5.4 The genTarget Module

The purpose of this module is to reduce the manual operation in compiling test programs, therefore, reducing errors that may occur during the compilation of the test program on the host system, which in this case is a UNIX workstation. *genTarget* generates a makefile that can produce the executable code that runs the test program.

5.5.5 Interconnect Test Module

This module performs the testing and diagnosis of the board interconnect. It consists of four major components which include net_list generation, test generation, test application and results analysis. The net_list generation component extracts the net list of the board from the MTL file where the drivers and receivers of these nets are all part of the Boundary Scan registers of the chips on the target board. The CTL files for these chips are then read to determine the physical configuration of the Boundary Scan registers. A mapping mechanism is then used to map the terminals of a net to their corresponding physical location in the Boundary Scan register. This mapping information is stored in a file called infofile.net which is later used for interconnect testing. In the test generation part, a test set that can identify all diagnosable faults is generated.This implements the interconnect test generation algorithms described in the Section 4.6.1.A test schedule for applying test vectors and gathering test results is produced by the test application part. Finally, in the results analysis part, test results are compared with their corresponding test vectors for analysis. Faults are detected based on the results of this analysis.

5.5.6 Device Driver

The device driver consists of two C functions reading and writing test data to and from

the Test Master Controller board or a Boundary Scan master controller chip. These functions, called read and write, are hardware dependent in that the device address is determined by the physical location of the TMC or local Boundary Scan master chip. Each of the functions have two arguments namely *addr*, which is the physical address location of the target device on the Test Master Controller board, and *datafile*, which is a pointer to a file where data can be read from or written to.

5.6 Summary

The tools and languages presented in this chapter are intended to relieve the system designer of all of the mundane tasks associated with testing a system. This tasks include Boundary Scan path threading, test pattern generation for both the chip and board levels, and test program generation. The JTAGtool completes the work started by the test modules, which automatically adds Boundary Scan test components, by making sure that every Boundary Scan chip is connected to the scan chain. Furthermore, JTAGtool extracts net list information which is used later during interconnect test generation. PLDS is used to dynamically configure the local controller on the Test Master Controller board to implement a desired test protocol.

PODEM is an efficient test pattern generation tool for combinational logic which can only be used if the Scan Path method is employed. It produces tests that will detect classical stuck-at faults with a high percentage of fault coverage. Board interconnect tests are generated by the Interconnect Module of the m2c test program compiler. This module implements algorithms that provide 100% stuck-at and bridging fault coverage of board interconnects. The Boundary Scan Description Language provides a simple, complete, and automated way of describing implementations. It is specifically designed for describing the numerous options that may be exercised in such implementations. The Chip

Test and Board Test Languages provide a complete testability framework that hides the details of serial scan protocols from the user. These languages control and track the state of the Boundary Scan hardware so that the user can view the target hardware system from a high-level perspective - the same way that a computer's operating system makes the details of computer operation transparent from the user.

CHAPTER 6

PROTOTYPE TESTING

The use of chips incorporating the Boundary Scan standard offers significant advantages when testing prototype systems Prototype systems primarily are developed to allow designers to prove a design concept or implementation before committing it to full production, where the cost to correct problems can be prohibitive. With the test hardware and software tools described in the two previous chapters, designers can verify such items as the basic design concept, theory of operation, board layout, parts selection, etc. Furthermore, the proposed test hardware and software tools reduce the amount of engineering effort required to test a system. However, in order to obtain the full benefits of the test hardware and software requires some knowledge of how to properly use them in a design as well as how to apply them for testing.

An overview of the Boundary Scan chips that provide a means for thorough testing of digital system including how they may be used in a design along with their respective test applications are presented in this chapter. After a discussion on the traditional test methods

used for prototype verification, a structured debug/test procedure is presented that outlines steps a designer can follow in order to properly use these Boundary Scan chips for prototype testing. This is followed by some functional and interconnect test examples which includes test procedures, Chip Test Language and Module Test Language files. Finally, the benefits of Boundary Scan versus traditional test methods and the lessons learned from this prototype testing experience.

6.1 Chips that Simplify Board Level DFT

By placing Boundary Scan chips at critical nodes on a board and in key signal paths, the boundary scan path can be used to provide access to critical nodes on the board. When the system is operating normally, the test circuitry is disabled and devices perform their normal function. During test operations, the chip's I/O boundary is controlled by dedicated test circuitry.

The procedure for implementing test functions varies according to the operation performed. In general, the user will preload one or more data registers, execute an instruction via the instruction register, capture data in the Boundary Scan register, and then scan out the resulting data from the register for comparison with some expected value. The remainder of this section describes the chips, listed in Table 4-2, and the test functions they perform with examples to illustrate how these chips can be used to build in testability are presented.

6.1.1 System Controllability, Observability, and Partitioning Octal Chips

System Controllability, Observability, and Partitioning Environment (SCOPE) octals [TI90b,c,d,e] are standard logic chips that contain Boundary Scan which are intended to

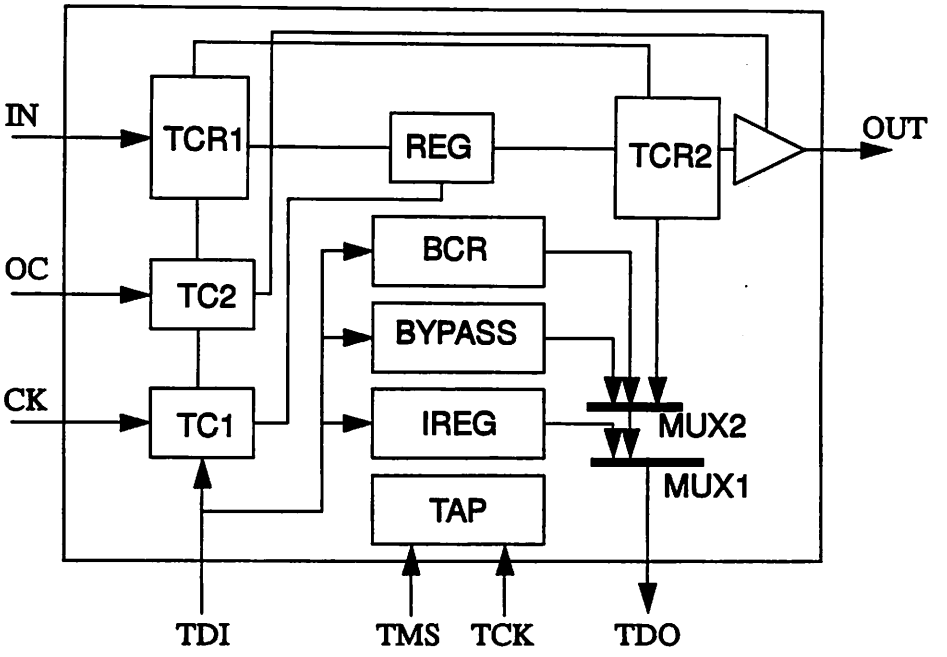


Figure 6-1: Block diagram of octal register architecture.

be place in a system design to greatly enhance overall testability in areas from design and prototype debug to final test and field service of production systems. These devices can be substituted for their non-testing counterparts in a variety of board level design application such as: pipeline registers, board I/O buffers, address and data buffers/transceivers, and finite state machine designs.

Along with the normal function associated with each octal, four pins are added to support the Boundary Scan standard. A block diagram of the SN74BCT8374 octal register type is shown in Figure 6-1. The functional architecture of the 'BCT8374 consists of an 8-bit register (REG), eight data inputs (IN), eight data outputs (OUT), a clock input (CK), and a tri-state output control input (OC). The Boundary Scan architecture consists of a Test

Access Port, an IREG, and a data register section. The data register consists of a bypass register, a boundary control register (BCR), and a Boundary Scan register. The Boundary Scan register consists of Test Cells 1 and 2 (TC1, TC2), and Test Register Cells 1 and 2 (TCR1, TCR2). The other octals have a similar architecture placed around buffer, transceiver, and latch functions. The Boundary Scan register provides the mandatory test features required for compatibility as well as special test features.

Normal Mode Operation

During normal operation, the Boundary Scan register is transparent, allowing input and output signals to pass freely through the test cells, enabling the chip to perform its intended function. While in normal operation, the Test Access Port can receive control from the TMS and TCK inputs to shift data through the chip from its TDI input to the TDO output. Three test instructions can be executed while the device is in this mode: SAMPLE, BYPASS, and a special SCOPE self-test instruction. The SAMPLE and BYPASS functions were described in Chapter 3. While the SAMPLE instruction at first may appear very attractive, the user must know when to sample in order to obtain meaningful data. The self-test instruction executes a self check of each SCOPE cell in the Boundary Scan register.

Test Mode Operation

When placed in an off-line test mode, the normal operation of the SCOPE octal is inhibited. In test mode, instructions can be shifted into the chip to perform all mandatory Boundary Scan instructions, as well as, an extended set of test instructions develop specifically for the SCOPE octals. Prior to loading these instructions, the Boundary Scan register should be set so that a desired test control pattern is applied to the REG inputs, tri-state buffers, and chip outputs. The step ensures that the chip will be in a known state

when the test mode is entered.

When EXTEST or INTEST instruction is loaded into the chip, Boundary Scan register cells TC1, TC2, TCR1, and TCR2 are set to allow simultaneous observation of signals appearing at their I/O pins. During EXTEST and INTEST execution, the Test Access Port receives external input to cause the Boundary Scan register to capture date on CK, OC, and IN inputs as well as the internal REG outputs. While captured data is shifted out, the next test control pattern is shifted in via TDI input. The Boundary Scan register outputs remain in their present state during the shift operation. The process of capturing data, shifting the Boundary Scan register to extract stored data and loading new test data, followed by the application of the new test from the register outputs, is repeated until the test is complete.

Test Extensions

To support extended testing features required additional instructions. The benefits of developing an extended test architecture is that every octal will share a consistent test instruction set, compatible test modes, and reduced complexity in the development of test software tools. These extended instructions are described below:

Control Boundary to High-Impedance

When this instruction is loaded into an octal, the outputs are placed in a high-impedance state and the bypass register is selected. This instruction is designed primarily to facilitate a blend of in-circuit testing and Boundary Scan testing. By disabling the outputs of the device, an in-circuit tester can drive the inputs of another device coupled to the output of the octals without damaging the octal's buffers. While this instruction is in effect, the bypass register is selected to provide a minimum data register scan length through the chip.

Control Boundary to a Logic 0 or 1

When this instruction is loaded into an octal, the Boundary Scan register outputs are set to a prescanned combination of logic 1's and 0's and the bypas register is selected. This instruction allows the test cells to output a control pattern to the REG inputs, tri-state buffers, and chip output. This places the chip in a preferred state while testing neighboring components.

Boundary Read

The contents of the Boundary Scan register can be shifted out when this instruction is executred.This instruction differs from EXTEST or SAMPLE in that the capture operation that normally occurs during the Capture_DR controller state is replaced with a data register hold operation which causes the Boundary Scan register to capture their present state instead of the data values sitting on their inputs. This instruction allows a signature that has been collected in the Boundary Scan register to be shifted out for inspection.

Run Test

This instruction was developed to support BIST approaches. Run Test is a generic instruction that executes the boundary BIST operation setup by control bits programmed in the BCR, shown in Figure 6-1. The BCR control bit settings must be set up via a scan operation prior to loading the Run Test instruction. Run Test will execute during the RUN_TEST/IDLE controller state. The length of a particular Run Test test operation is determined by the number of TCK inputs applied. The Run Test instruction has four operational modes: 16-bit Parallel Signature Analysis (PSA) of the IN inputs, 16-bit Pseudo-Random Pattern Generation (PRPG) form the OUT outputs, simultaneous PSA and PRPG, and simultaneous SAMPLE of IN inputs and TOGGLE of OUT outputs.

During a 16-bit PSA Run Test mode, the 8-bit TCR1 and TCR2 cells are tied together to

form a 16-bit Linear Feedback Shift Register (LFSR). The parallel inputs to the TCR1 are enabled to accept data from the IN bus and the parallel inputs to TCR2 are disabled. In this configuration TCR2 acts as an 8-bit LFSR extension to TCR1. During test, the parallel inputs from the IN bus are compressed into the 16-bit LFSR on the rising edge of TCK. Linking TCR1 to TCR2 allows the octal to receive an extended sequence of 8-bit patterns from the IN bus. At the end of the PSA mode, the 16-bit signature can be shifted out of TCR1 and TCR2 for inspection. While TCR1 and TCR2 are collecting the signature, the outputs of TC1 and TC2 remain in their present state. TC2 can be set to enable or disable the OUT buffers during a test.

During the 16-bit PRPG Run Test mode, TCR1 and TCR2 are tied together to form a 16-bit LFSR as described in the 16-bit PSA test. During the 16-bit PRPG test mode, both parallel inputs to TCR1 and TCR2 are disabled so that both act only as LFSRs. During test, the parallel output from TCR2 drives pseudorandom patterns to the OUT bus one each falling edge of TCK. Since the width of the OUT bus is 8 bits, individual patterns will be repeated during every 256 pattern output sequence. However, the test circuit will produce 256 sets of unique 256 pattern output sequences. TC2 is set to enable the OUT buffers during this test.

During the simultaneous PSA and PRPG Run Test mode, TCR1 and TCR2 operate as two separate 8-bit LFSRs. The parallel inputs to TCR1 are enabled to accept data from the IN bus and the parallel inputs to TCR2 are disabled. During test, TCR2 outputs pseudorandom patterns to the OUT bus on the falling edge of TCK. glue logic residing at the chip's I/O pins can be quickly tested using the Run Test instruction.

During the simultaneous Sample Inputs/Toggle Outputs Run Test mode, TCR2 outputs alternating data patterns to the OUT bus on the falling edge of TCK, and TCR1 accepts

data input form the IN bus on the rising edge of TCK. By adjusting the frequency of TCK, this test can be used to measure the propagation delays through external logic residing between the OUT and IN buses.

Boundary Self-Test

The Boundary Scan register is selected in the scan path. This operation tests the logic in the Boundary Scan cells by loading the complement of the current logic value in the cells. By loading a known value in the register, executing CELLTST, and inspecting the resulting data through a scan operation, the integrity of the register can be verified.

Boundary Toggle Outputs

Functional outputs are toggled on each falling edge of TCK.

Verifying Board Interconnect

Perhaps one of the simplest examples of how Boundary Scan can be used to improve the testability of a board is by verifying the board interconnects (detecting "stuck-at" faults) between chips on a board or between two boards in a system. Figure 6-2 shows two 'BCT8244s being used to buffer signals between two separate parts of the board. They could be on either side of an edge connector, separated by board traces, or in any number of other configurations. The procedure for verifying the interconnect for this example is described below:

1. Initialize the scan path through a reset operation.
2. Scan[1] all zeroes into the output Boundary Scan cells of U1. This can be done with any of several instructions.
3. Scan the EXTEST instruction into both U1 and U2.
4. Capture the Boundary Scan register of U2.

1. Scan means put the Test Access Port in the appropriate shift state and serially load data through the TDI pin.

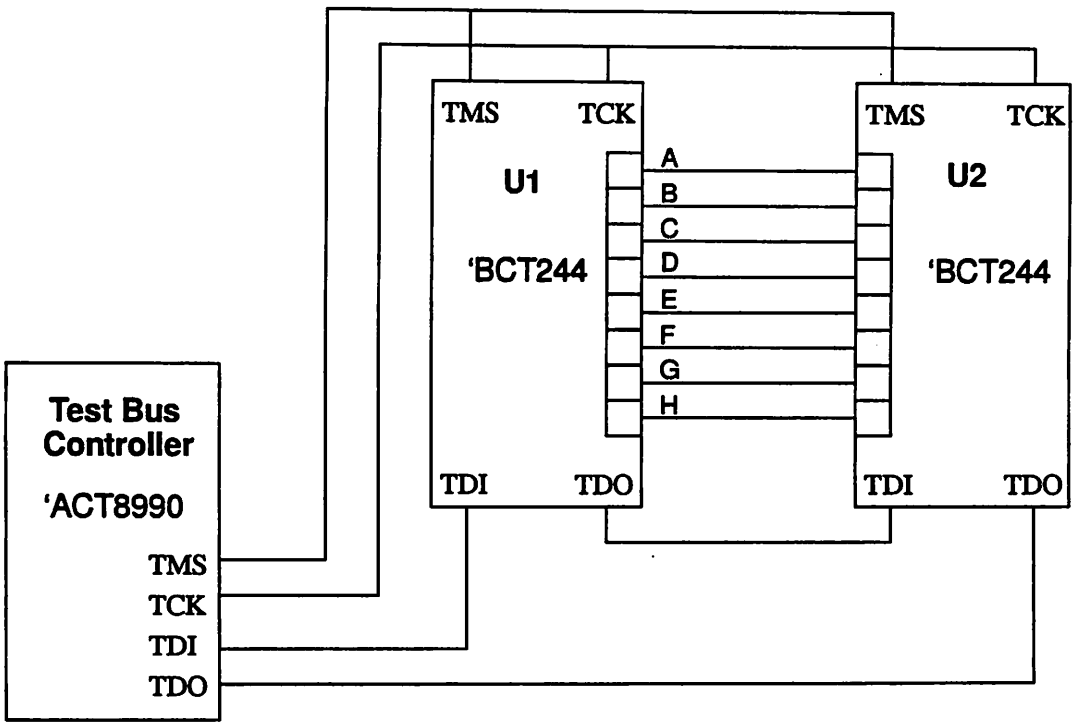


Figure 6-2: Using two 'BCT244's to verify PCB interconnect.

5. Scan out the captured contents of U2's input Boundary Scan cells for inspection, while scanning in the next pattern into the output Boundary Scan cells of U1.
6. Repeat steps 4 and 5 for each pattern.

Step 1 can be accomplished by applying g a logic 1 to the TMS pin, by scanning all 0's into the Boundary Scan register, or by a power-down/power-up sequence. During step 2, load the output Boundary Scan cells of U1 with the data that will be applied through the functional outputs. The EXTEST instruction in loaded by putting U1 and U2 into the Shift_IR controller state and scanning in its opcode (00000000) into the instruction register of both chips. During the previous controller state, U1 will force through its outputs, the data loaded in step 2. Since EXTEST places the Boundary Scan register

between TDI and TDO, going to the Capture_DR state (step 4) will load the input Boundary Scan cells of U2 with the data appearing at its functional inputs.

The Test Access Port controller of both U1 and U2 are placed in the Shift_DR controller state in step 5. The scan path is now 36 bits long, 18 bits from u1's Boundary Scan register and 18 bits for U2's Boundary Scan register. The data from the input Boundary Scan cells of U2 is scanned out to be stored and/or examined. Since all 0's from U1 were forced, the expected value of the data captured at the input Boundary Scan cells of U2 is also all 0's. During this same shift, the output Boundary Scan cells of U1 are loaded with the next pattern. When passing through the Update_DR controller state after shifting is complete, the output Boundary Scan cells of U1 will force the next test pattern is applied. As an example, assume that some wiring defects in the circuit of Figure 6-2 have caused an open between U1 and U2 on signal C, and a short circuit between signals E and F. Table 6-1

Table 6-1 : Shorts/Opens Verification.

Pattern #	Pattern Forced by U1 (A-G)	Pattern Captured by U2 (A-G)
1	00001111	00*/*01111
2	11110000	11110000
3	00110011	00110011
4	11001100	11*/*01100
5	01010101	01*/*10001
6	10101010	10100010

lists six patterns U1 can apply to check for any opens and shorts that may exist between U1 and U2, or shorts between any two pins, along with the data they would capture.

Glue Logic Testing

Pseudo-random pattern generation and parallel signature analysis can be used to verify the logic implementation of a design. During a pseudo-random generation or parallel signature analysis operation, the Boundary Scan registers of a device are configured as linear feedback shift registers and will perform either a pattern generation or data compression operation on every TCK cycle. By loading a known seed value (other than all zeroes) into the Boundary Scan register and knowing the algorithm used, the user can determine (NOTE: Those bits indicating the presence of a defect appear in *italic* type) the patterns that will be generated and/or signature resulting from the data compression of the inputs. In order to exercise the system logic shown in Figure 6-3, the Boundary Scan cells of U1 can be configured to output pseudo-random patterns and the input Boundary Scan cells of U2 configured to compress data by performing the following operations:

1. Initialize the scan path.
2. Load the Boundary Scan registers of both U1 and U2 with the seed values to be used during PRPG and PSA. Any value but all zeroes is acceptable.
3. Scan the SCANN instruction into both U1 and U2.
4. Scan the PRPG data into U1's Boundary Scan register and the PSA data in U2's Boundary Scan register.
5. Scan in the Run Test instruction into U1 and U2.
6. Go to the Run_test/Idle controller state.
7. Execute the PSPG and PSA instruction for the desired number of clock cycles.
8. Scan U2 with the Boundary Read instruction.
9. Scan out the contents of U2's input Boundary Scan cells and compare the resulting signature with the expected values.

To illustrate this concept, assume that a seed value of all ones is loaded into the Boundary Scan register of both U1 and U2 during step 2, and there is no logic between U1 and U2 (i.e., U2's 1A1 = U1's 1Y1, U2's 1A2 = U1's 1Y2, etc.) During Steps 3-5 the octals are

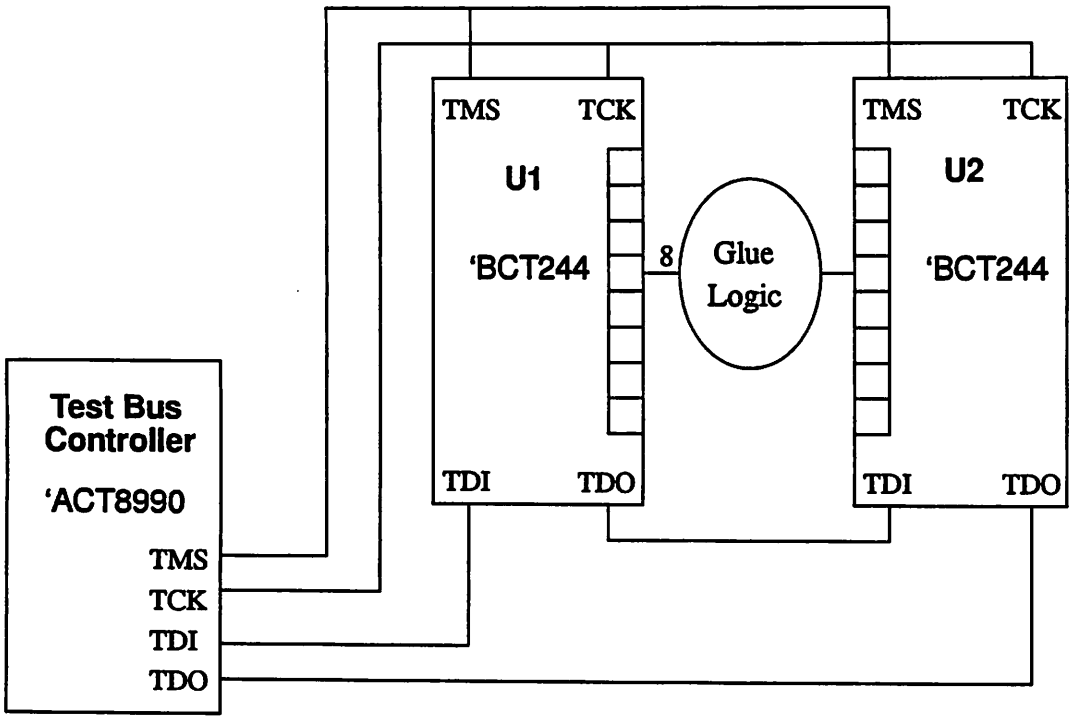


Figure 6-3: Logic verification using Boundary Scan.

loaded with the proper data and instruction to perform the pattern generation and data compression operations. SCANCN places the boundary control register between TDI and TDO so the simultaneous PSA/PRPG code (11) can be loaded using the Shift_DR controller state. Run Test is loaded into the instruction register, and tells the octals to examine their Boundary Scan registers and execute the specified test. In this example, the simultaneous PSA/PRPG function is being used. Figure 6-4 shows the configuration used during this operation. After generating sufficient patterns to test the logic, the signature in U2's input Boundary Scan cells must be examined. This is accomplished using the Boundary Read instruction. It is important that this instruction rather than EXTEST,

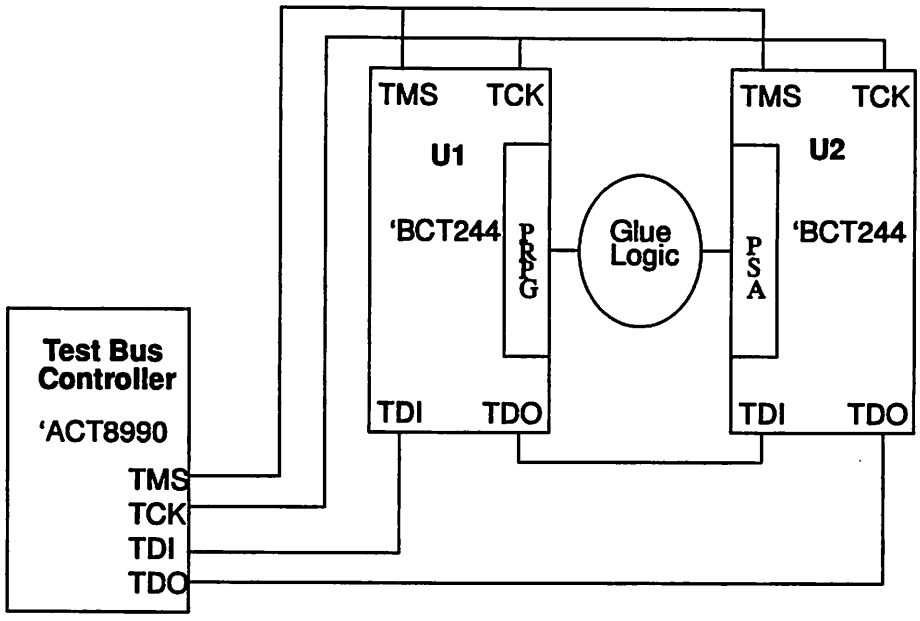


Figure 6-4: Simultaneous pseudo-random pattern generation and parallel signature analysis.

INTEST, or SAMPLE be used, because while all of these instructions will place the Boundary Scan register between TDI and TDO for the ensuing Shift_DR controller state, the other instructions will preload the input Boundary Scan cells with the current input data during the Capture_DR state and overwrite the signature. Boundary Read does not preload the Boundary Scan register during Capture_DR, so the signature is preserved. Table 6-2 shows the pseudo-random patterns generated, and the resulting signature after each pattern, for the first 15 TCK cycles. The first pattern, applied during the falling edge of TCK in Update_IR, is the seed value of all 1's, and the first signature (generated on the first rising edge of TCK after entering Run_Test/IDLE) is based on that value. On the first falling edge of TCK after entering Run_Test/IDLE, the signature is generated on the rising edge of TCK as the controller state changes from Run_Test/Idle to Select_DR_Scan. Although this example contains no logic between U1 and U2, the same principles are

Cycle	Pattern After TCK (1Y1-1Y4, 2Y1-2Y4)	Signature After TCK (1A1-1A4, 2A1-2A4)
1	11111111	1000000000
2	01111111	00111111
3	00111111	10100000
4	10011111	01101000
5	01001111	00011010
6	00100111	10001000
7	00010011	10000100
8	00001001	11001100
9	10000010	10100001
10	10100001	00101000
11	01010000	10111100
12	00101000	11001010
13	10010100	00101111
14	11001010	11110001
15	11100101	11110010

Table 6-2 : PRPG/PSA sequence.

applicable in more complex cases. This method can also be used to verify address decoding to memories, or to apply patterns to the input of a complex chip. By placing the octals in the critical paths, the user can control the signals being applied to any node on the board.

Partitioning for Test

Using octals to buffer key signals can allow for effective partitioning of a board during test to remove unconnected or unwanted components. Partitioning a board into separate stand-alone test cells can reduce the number of patterns required to test the section(s) of

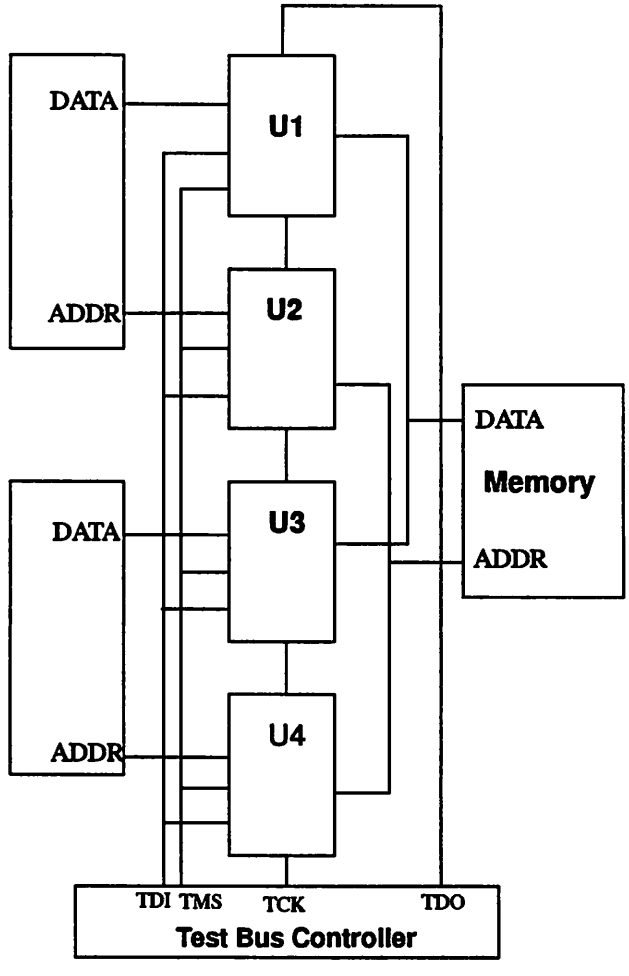


Figure 6-5: Partitioning for test example.

interest. In Figure 6-5, octals are used to partition a shared-memory configuration in which a digital signal processor and graphics signal processor share the same memory. The 'BCT8245s (U1 and U3) are used to buffer data transmission between the processors and memory, and the 'BCT8373s (U2 and U4) are used as address latches. The four octals are connected in a serial scan path with common TCK and TMS signals. Using the octals in

this configuration creates many testing possibilities that go far beyond simple interconnect testing. For example, one or both processors can be effectively removed from system operation by scanning and executing the Control Boundary to High-Impedance instruction on the units buffering it, which will cause the functional outputs of the octals to go into the high-impedance state. This instruction also protects the octals if some type of fixtured testing, such as bed-of-nails test, is to be used by ensuring that the functional outputs are not back driven.

The status of the octal inputs and outputs can also be captured at any time by using the SAMPLE instruction. This operation does not disturb the normal operation of the chips and will not affect the system, but will capture the logic levels at all inputs and outputs. This information can be scanned out and compared against an expected value. The SAMPLE instruction is also useful for preloading the Boundary Scan register prior to another test operation, since the octals will continue to function in a normal mode during the preload scanning.

The circuit in Figure 6-5 would also allow the contents of any or all memory locations to be written to or read from. A memory location can be verified by using the EXTEST instruction to force an address using U2 or U4 and capture the data appearing using U1 or U3. This illustrates the ability of the chips to both control and observe the signals to which they are connected. If the entire contents of the memory are known, the PRPG and PSA operations could unload the entire memory using a minimum number of clock cycles, with the final signature being scanned out for inspection.

6.2 The Digital Bus Monitor Chip

The Digital Bus Monitor (DBM) chip can be included in a board design to provide a

method of monitoring embedded digital signal paths between chips like data buses. The DBM is capable of monitoring digital signal paths while the board is either on-line and operating normally or is in an off-line test mode. The benefit of on-line monitoring is that it can be used to reveal timing sensitive and/or intermittent failures that are otherwise undetectable without the use of external test equipment and mechanical probing fixtures.

One of the advantages in using DBM chips to construct board level BIST structures is that the monitoring capability is embedded in the board design and can be used throughout the life cycles of the board prototype testing, system integration, system test, and real-time diagnostics. Another advantage in using these chips is that it does not significantly impact the performance of the board circuitry. Since the signals to be monitored don not pass through the DBM chip but are only input to the chip, no significant performance penalty is paid when using these chips. In the example shown in Figure 6-6, two chips operate together via address, data, and control interface paths to perform a desired function. In normal operation, Chip1 outputs address and control to Chip2 to pass data between the two chips. Two DBM chips are included in the circuit for address and data bus monitoring. The DBM chips are connected via Boundary Scan test bus and the two-wire event qualification bus. The address and data bus to be monitored are input to the DBMs via observability data inputs (ODIs). The control outputs from Chip1 are input to the DBM chips via clock inputs (CK) to allow them to operate synchronously with the circuit during on-line monitoring.

The test circuitry residing behind the ODI input pins consists of a RAM buffer, and a test cell register. The memory buffer provides storage for multiple ODI input patterns. The test cell register operates as either a Boundary Scan register or PSA register. The memory buffer and test cell register can be operated together or separately, as required by the given test operation. The ODI inputs to the test cell register can be masked individually to allow

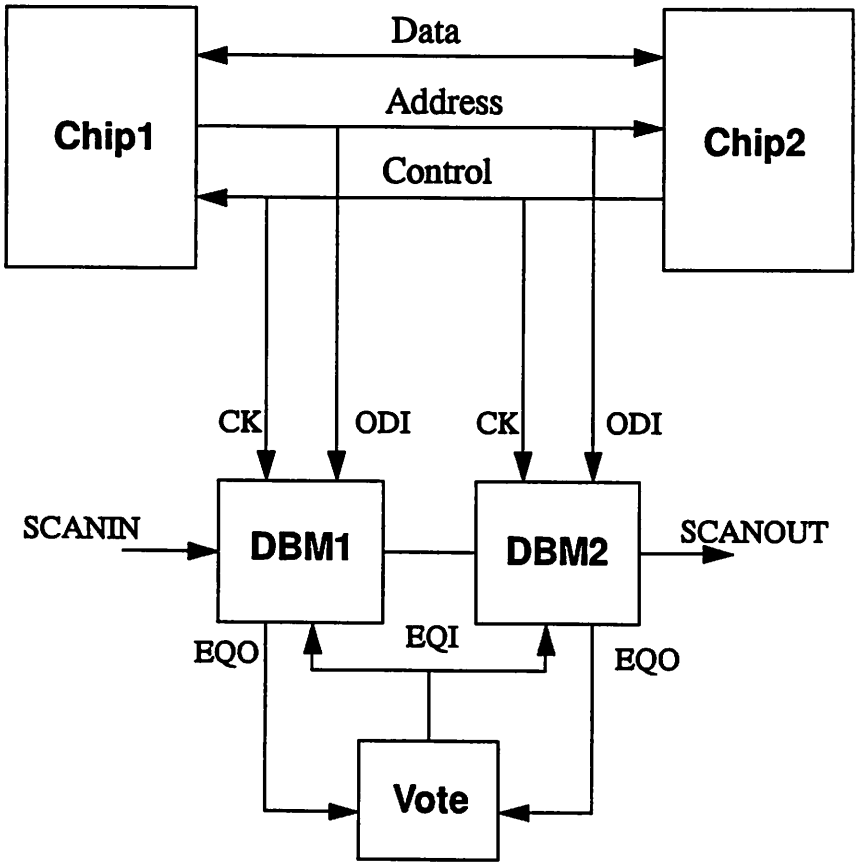


Figure 6-6: Digital bus monitor example.

diagnosing which input or groups of inputs caused a multiple-input PSA operation to fail. When the circuit in Figure 6-6 is placed in an off-line test mode, Chip1 can be made to output data on its address and data bus. The data and address output from Chip1 can be stored into DBM1 and DBM2, respectively, via ODI inputs. After the data have been stored, they can be shifted out for inspection via Boundary Scan path. Similarly, Chip2 can be made to output data on its data bus, to be stored and shifted out for inspection by DBM2. In the off-line test mode, control to store data and operate the scan path is input via Boundary Scan test bus.

When the circuit in Figure 6-6 is on-line and functioning normally, chips DBM1 and DBM2 can continue to monitor the data and address buses suing an internal Event Qualification Module resident in each Digital Bus Monitor chip. During on-line monitoring, the Event Qualification Module outputs control to store the data appearing on the observability data inputs. This module operates synchronously with the control signal input to the Digital Bus Monitor's clock inputs. To determine when to store data, the Event Qualification Module includes comparator logic that can match the data appearing on the observability data inputs against predetermined expected data pattern(s). The compare operation performed on each observability data input can be masked individually to eliminate input signals not required for event qualification. The Event Qualification Module has protocols that allow it to perform different types of event-qualified monitoring operations. The type of monitoring operation to be performed (for example, RAM storage) determines the type of protocol used.

To expand the event-qualification capability, many Digital Bus Monitor chips can be connected via the two-wire event-qualification bus to allow qualification of a test operation to be distributed over a range of chips. During expanded event qualification, each DBM operates to output a match condition on its Event-Qualification Output pin. The match signals from these chips are combined via a voting circuit to produce a global matching signal. The global matching signal is input to each chip via and Event-Qualification Input signal. When a global match signal is received, the internal Event Qualification Module initiates a test monitor operation. In some cases, it may be required to qualify a monitor operation further using external signals. In this case, the external signals are input to the voting circuit to allow finer resolution as to when a monitor operation is performed.

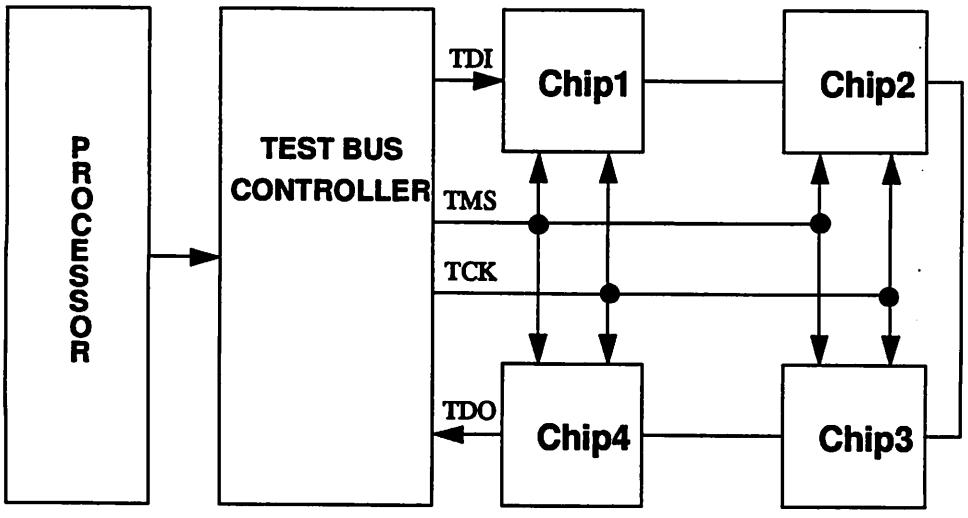


Figure 6-7: Test bus controller example.

6.2.1 The Test Bus Controller Chip

The Test Bus Controller chip is a Boundary Scan test bus controller that supports efficient transfer of serial data and control to and from target chips sitting on the test bus. Figure 6-7 shows an example application of the Test Bus Controller chip which provides the hardware link between a host processor and target chips residing on the scan path. To the processor, it is a peripheral mapped into a particular area of the processor's external memory space. It's processor interface consists of a 16-bit bidirectional data bus; inputs for address, read/write, and chip select signals; and an interrupt output signal.

The chip's test bus interface consists of five signals which are Test Data Output signal, Test Data Input, Test Mode Select, Test Clock Input, and Test Clock Output. The two Test Mode Select outputs allow the it to support separate scan paths. The Test Data Output

signal is a serial data output from the chip that drives the Test Data Input pin of the first target chip in the scan path. The Test Data Input signal is the serial data input to the Test Bus Controller chip that receives the data from the last target chip in the scan path. The Test Mode Select signals are serial control outputs from the chip that drive Test Mode Select inputs of target devices in the scan path. These outputs conforms to the protocol described in the Boundary Scan standard to cause target chips on the scan path to shift data. The Test Clock Input signal is generated externally and is distributed via Test Clock Output to each target device in the scan path. In addition to the required Boundary Scan test bus signals, the interface includes an output signal for initialization of target chips and input signals for receiving test related interrupts from target chips.

Before a scan operation, it receives the parallel data input from the processor (e.g. VME CPU board) that is to be transmitted serially to the target chips in the scan path. Also, the processor inputs a count value into an internal counter, specifying the number of serial data bits to be transferred. After the data and count values have been set up, it receives a command from the processor to initiate the scan operation. During scan operations, it outputs serial data and control signals to the target chips via the Test Data Output and Test Mode Select output signals and receives serial data from the target chips via Test Data Input. By reading status bits from the Test Bus Controller chip, the processor determines when it requires additional read and write operations to maintain the flow of serial data to the target chips in the scan path. When the it's internal counter reaches a minimum value, it outputs an interrupt to the processor, indicating that the required number of serial data bits has been shifted through the scan path.

In addition to controlling scan operations, it simplifies the execution of BIST features incorporated in the target chip. The Boundary Scan test bus protocol state diagrams includes a Run_Test/Idle state in which BIST operations may be executed. If a target chip

has BIST capabilities, an instruction invoking the BIST operation can be scanned into the device. After the target device receives the BIST instruction, execution of the test occurs when it transitions the test bus into the Run_Test/Idle state. As defined in the standard specification, the length of a BIST operation is defined by the number of clock inputs applied while the test bus is in the Run_Test/Idle controller state. To simplify the execution of BIST operations, the Test Bus Controller chip contains two types of run test commands. Both command types are executed while the test bus is in the Run_Test/Idle controller state. The first type uses the internal counter to count the number of clocks applied while the test bus is in the Run_Test/Idle controller state, supporting the BIST procedure detailed in the standard specification. When the counter reaches a terminal value, it transitions the test bus from its current state into either a scan or pause state to terminate the execution of the BIST operation. The second type uses its interrupt inputs to determine the length of time the test bus is in the Run_Test/Idle controller state. This command differs from the first in that it assumes the target chip(s) have additional test pins from which an end-of-test interrupt may be issued to it. When it receives this interrupt, it transitions the test bus from the Run_Test/Idle controller state to either a scan or pause state to terminate the BIST operation.

6.3 Prototype Testing using the TMC Board: A User's Perspective

A prototype system is normally a low cost, low volume production of the end product. Most of the money spent during prototype development is devoted to building the prototype and verifying the design concept; very little usually is allocated for testing. Prototype testing usually is accomplished through simple functional operation and verification of the system as a whole. Functional test of system hardware and software

typically are performed on the entire system or parts of the system, possibly with some unavailable function being emulated externally. When failure are detected during this design verification process, a significant amount of time is spent determining the cause of the failure and correcting it. This debugging process is a manual, often time consuming, task that does not guarantee immediate success. Several hours may be spent rerunning the system after swapping boards and components that are believed to have caused the failure but, in fact, did not. Mixing design verification with fault verification can be costly in the long run if these processes do not complement one another.

Design verification and fault verification can be less painful by using devices that support the Boundary Scan standard and partitioning the system into small, easy to test functions. The use of Boundary Scan allows each partitioned function to be verified and tested independently, thereby reducing the time spent locating the cause of a failure. Boundary Scan provides an increase in the controllability and observability of internal circuit nodes, which is mandatory when isolating system hardware faults and verifying operation of system software. Standard chips implementing Boundary Scan are very beneficial in this situation. The use of such chips supports a hierarchical test philosophy in which the same test capabilities and test programs can be reused at each level of system integration (chip, board, system).

6.3.1 Traditional Test Methods

During prototype system design, testing issues are often far from the minds of the designers. If test is an issue, ad-hoc testability techniques sometimes are implemented. The design primarily is concerned with how to implement a given piece of the system and have it interface correctly with the rest of the system, as well as, how the system is going to function when all the pieces are put together. The designer is also concerned with

getting the system software working and verified on the system. The complexity of the test problem comes to the surface when all the pieces are assembled and the system does not function properly. Determining why the system does not function as intended can be a major problem. Typical problems include:

- A manufacturing problem (mis-wire, wrong component, etc.)
- A design error (incorrect design implementation)
- A software problem (incorrect algorithm)
- A hardware failure (bad part, etc.)

There are many approaches to identify the problem, but no real test strategy exists for debugging. The e designer usually determines the process for obtaining a functional system. The equipment used in the traditional test and debug process will very depending on whether a whole system, subsystem, or a few boards are being checked out. Types of equipment that are needed include:

1. Hardware and software emulator
2. Logic analyzer
3. Oscilloscope
4. Multimeter
5. Specially designed debug boxes (special test equipment)
6. Logic probes

Depending on the design, the equipment cost may be very expensive and hard to justify for a low volume system. User expertise and related training costs, if necessary, are other factors to consider. A typical approach taken to verify/debug/test the hardware and software in a prototype system is:

1. Make a visual inspection of all the boards to check for any obvious problems; for example, wrong parts on the board.

2. Do a continuity test of VCC and GND to check for shorts. This always should be performed on each board before placing it in the system to avoid the possibility of damaging the whole system when power is applied.
3. At this point, there are many different options, depending on what the designer decides is the best approach. One approach is to run the hardware and software, making necessary patches (either hardware or software), to get around parts not currently available. If everything runs as intended, it is assumed that no problem exists. If not, there are may ways to proceed which include the following:
4. - Lower the hardware complexity by removing boards and patching around the boards
5. - Lower the software complexity by changing the software
6. - Execute the software in an single step mode and attempt to identify the source of the problem by using a logic analyzer and/or oscilloscope.

Other approaches like swapping boards can induce faults into the system other than those faults associated with the system under test. The traditional methods described above usually do not involve a structured design verification and testing strategy. Concurrent verification of hardware and software makes it difficult to isolate between hardware and software faults. The increasing complexity and density of today's systems may require costly equipment for verification and test, furthermore, board real-estate must be allocated for probing.

6.3.2 Structured Debug/Test Procedure

The use of the Test Master Controller board and the test software described in Chapter 4 coupled with intelligent use of Boundary Scan devices altogether provide the designer with a structured procedure for verifying, debugging, and testing systems that can be reused for future systems. This procedure uses a building block approach whereby individual parts of the system are verified and then may be used to verify the remaining parts. The procedure is shown in Figure 6-8 and can be divided into the following three phases:

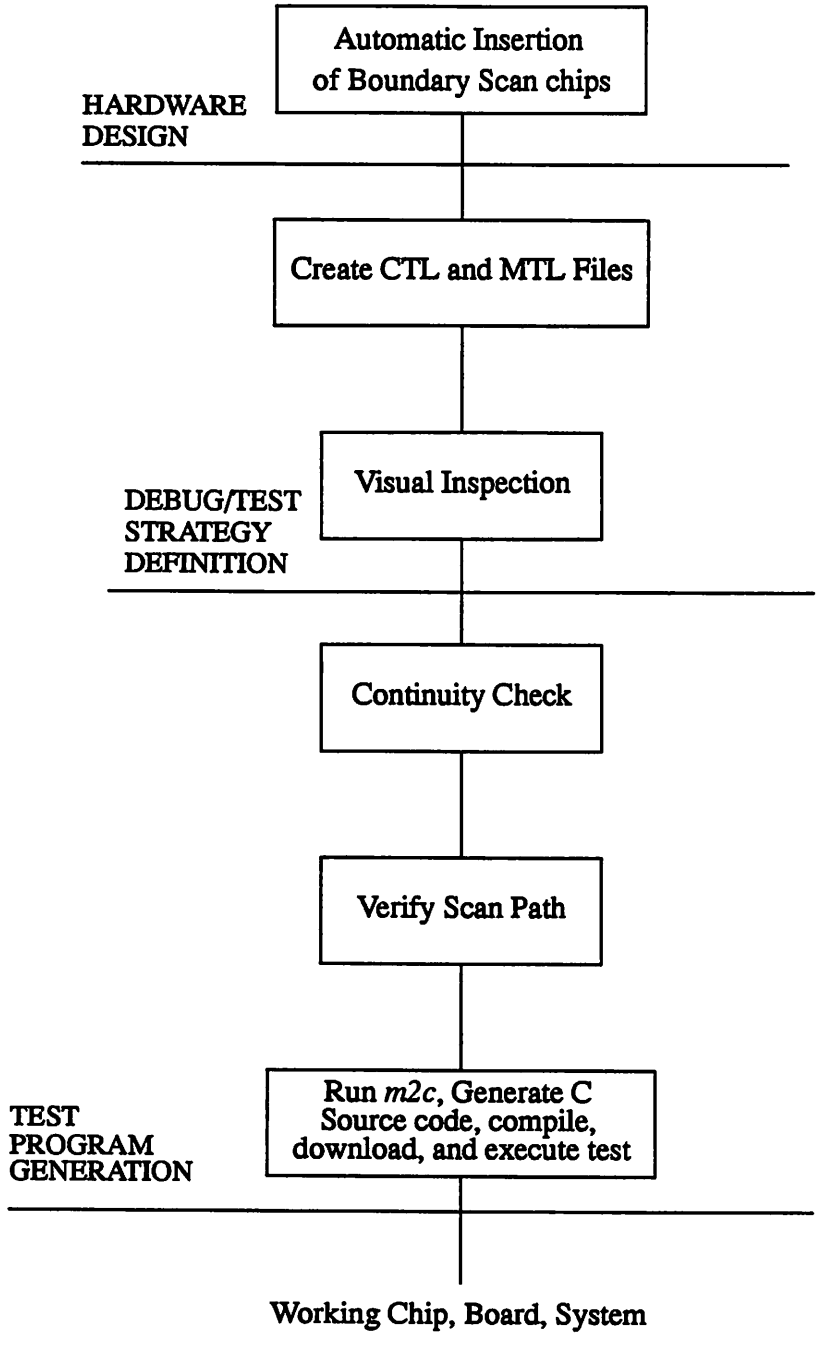


Figure 6-8: Block diagram of structured debug/test procedure.

- Hardware design
- Define test strategy
- Automatic test program Generation and execution

The hardware design phase consists of automatically incorporating Boundary Scan chips into the design to partition it into easily testable functions. During the test strategy definition phase, the designer decides on the best test strategy for their board(s) and then creates CTL and MTL files that will implement their strategy. The CTL and MTL files are then processed, by the *m2c* program, to produce the source C code for the test program. Next, the test program manufacturing module is used to create a make file which gets processed by the Unix Make utility to create the executable code. Finally, the executable code is transferred to the VME CPU board to control the Test Master Controller board during a test.

6.3.3 Test Master Controller Board Prototype

The Test Master Controller board was developed to fulfil two primary objectives:

1. to control the test process of target slave boards containing Boundary Scan
2. to study the benefits of using standard chips incorporating Boundary Scan as a tool for hierarchical test. in a board design

Both of these objectives can be fulfilled by implementing a prototype where Boundary Scan is an essential design element. Scannable buffers, latches, and flip-flops were used to partition the design into sub-functions that could be easily verified. A Boundary Scan test bus controller chip and data bus monitoring chips were also used in the prototype design. The implementation of these chips in the Test Master Controller board prototype is shown in Figure 6-9 where 28 Boundary Scan chips consisting of buffers, latches, and flip-flops were placed strategically on data, address, and control signals, Thus it is divided into five

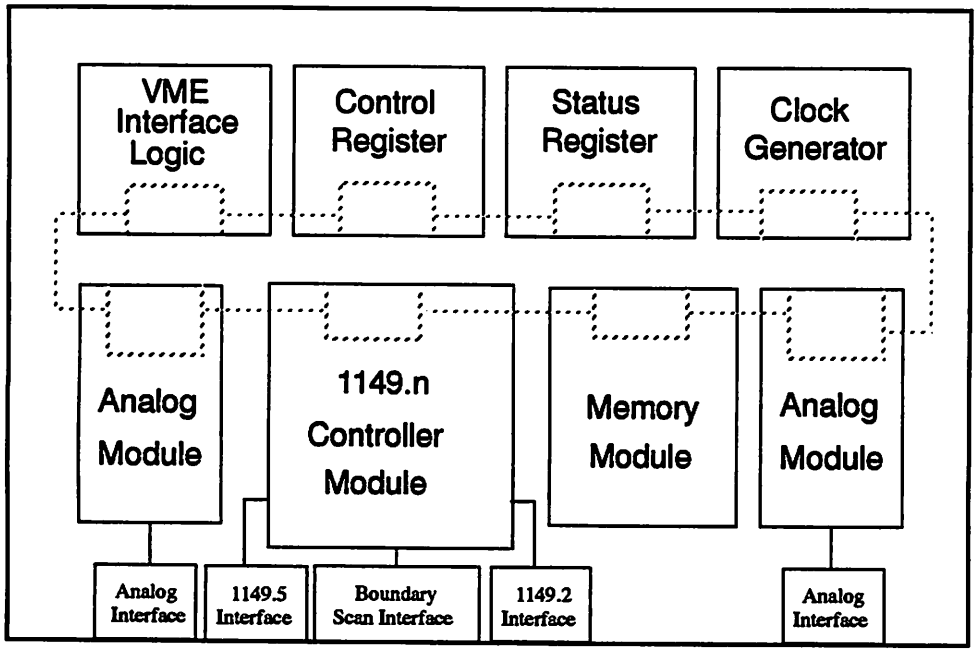


Figure 6-9: Configuration of scan path on TMC prototype.

distinct sub-functions, each individually accessible through the local Boundary Scan interface. These sub-functions are the VME interface logic unit, the clock generator, the memory module, the analog module, and the 1149.n controller module.

6.3.4 Functional and Interconnect Test Example

The debug/test procedure described above is demonstrated in the following example. This example uses an octal buffer chip and a digital bus monitor chip, all containing Boundary Scan, to verify interconnect between an address buffer and a digital bus monitor chip. To generate test programs for this example requires five files: tmc.mtl., buf_2.ctl, buf_3.ctl, buf_4.ctl, and act8994.ctl. These files are listed below.

MTL file for Test Master Controller Board

```
Module = TMC;
LIB = lib (3);
DEVICE_LIST = (Chip1 buf_2) (Chip2 buf_3) (Chip3 buf_4) (Chip4 dbm);
TEST_BUS =
 RING 0: Chip1 => Chip2 => Chip3 => Chip4;
NET_LIST =
 NET 1: (Chip1 Y[1]) (Chip4 D[0]),
 NET 2: (Chip1 Y[2]) (Chip4 D[1]),
 NET 3: (Chip1 Y[3]) (Chip4 D[2]),
 NET 4: (Chip1 Y[4]) (Chip4 D[3]),
 NET 5: (Chip1 Y[5]) (Chip4 D[4]),
 NET 6: (Chip1 Y[6]) (Chip4 D[5]),
 NET 7: (Chip1 Y[7]) (Chip4 D[6]);
 NET 8: (Chip2 Y[0]) (Chip4 D[7]),
 NET 9: (Chip2 Y[1]) (Chip4 D[8]),
 NET 10: (Chip2 Y[2]) (Chip4 D[9]),
 NET 11: (Chip2 Y[3]) (Chip4 D[10]),
 NET 12: (Chip2 Y[4]) (Chip4 D[11]),
 NET 13: (Chip2 Y[5]) (Chip4 D[12]),
 NET 14: (Chip2 Y[6]) (Chip2 D[13]),
 NET 15: (Chip2 Y[7]) (Chip2 D[14]),
 NET 16: (Chip3 Y[0]) (Chip3 D[15]);
MODULE_TEST
#include <stdio.h>;
#include "comp.h"
main ()
{
    testinet (); /*test the entire interconnect */
    testchip (Chip1); /*test the entire chip */
}
END_TEST
```

The test bus is organized into one Boundary Scan ring. There are a total of 28 chips in the scan ring. Only four chips are used in this example, the remaining 24 chips are bypassed. The test procedure for this test consists of the testinet test procedure that implements an interconnect test for the chips listed in the device_list.

CTL Files

BSDL descriptions of the TI8244 and digital bus monitor chips are given in Appendix A. Hence only the test procedure part of the CTL files are given below. The test procedures for the two other buffer chips are almost identical differing in test data only. The test procedure for the digital bus monitor chip differs in length of Boundary Scan register as well as test data. The example test procedure presented is intended to verify the functionality of the devices in addition to verifying the interconnection between the chips involved in the test.

```
Test_Begin
TDM 0 = USER_DEFINE;
#include <stdio.h>
#include <string.h>
#define IR 1
#define DR 0
top 0
 {
 char outs[32], ins[32];
 char *p1, *p2;
 sprintf(out2, "00000000"); /* OPCODE for EXTEST INSTRUCTION*/
 scanIR(outs); /* SCAN CONTENTS of STRING to INSTRUCTION REGISTER */
 sprintf(ousts, "0000000010101010");
 strcpy(ins, scanDR(outs));
 /* load D = 10101010 into Chip1and GET PREVIOUS TEST RESULT */
 p1 = ins+24; /* Advance Pointer to Beginning of result String */
 p2 = outs+14 /* Advance Pointer to Beginning of Data String */
 if (strncmp(p1, p2, 8) != 0) {
 printf(error in 10101010 test.\n";
 exit(1);
 }
 else printf("buf_2 Tested OK.\n");
 Test_End
```

Generated Test Programs

The generated test program for the Test Master Controller board consists of seven files, namely comp.h, tmcmain.c, buf_2tops.c, driver.c, inetpc.c, template.c, and

infofile.net. The file comp.h defines all of the variables used in by the main program tmcmain.c The prefix tmc of the tmcmain.c file is extracted from the tmc.mtl file. The file tmcmain.c contains four functions. The first one, inetpc is used to test the interconnect. The second one, buf_2tops.c is used to test Chip1 using the user-defined TDM. The third one, driver.c, contains all of the functions required to control the Test Master Controller board during a test. The file template.c contains all of the template-based TDM functions. The inetpc procedure uses the information provided in the inforile.net for test generation. The buf_2tops.c file is a translated version of the user-defined procedure for testing Chip1.

6.4 Benefits of Boundary Scan vs. Traditional Methods

The key benefits of using Boundary Scan versus using traditional methods for design verification, debugging, and testing are shown in Table 6-1 and discussed below. Incorporating Boundary Scan into prototype designs will provide the designer with a structured approach to design verification, debugging, and testing of prototype systems, which typically does not exist using traditional methods. As the complexity and density of designs increase, the need for more equipment will be required when using traditional methods. The need for different types of equipment was shown to decrease when Boundary Scan was implemented into the design. The equipment required in debug and test of the prototype system described in the previous section included Boundary Scan octal chips incorporated into the design. and a dedicated Test Master Controller board and test software. In traditional methods, the software generated by designers during design verification and debug is usually discarded after verification of the system. If the software is not discarded, it rarely can be used for board and system testing. In contrast, the hierarchical test method used on the prototype system allowed the software generated for board level debug to be reused to generate system level tests. The controllability and

observability in the traditional debug and test methods usually requires the attachment of emulators, logic analyzers, or oscilloscopes to the device under test. Controllability and observability can be easily achieved with Boundary Scan chips. With this increased internal visibility, fault isolation and system software debug are accomplished more easily. The use of Boundary Scan octal chips, requires the use of four I/O pins and a slightly larger chip foot print a board. In the traditional approach, the amount of space used is dependent on the ad-hoc testability added to the design. Space also needs to be allocated so that boards can be probed.

Trade-offs	Traditional	Boundary Scan
Approaches	Usually not structured Usually starts after design has gone through manufacturing	Structured approach Starts during design
Test Equipment	Complexity/density of designs require more test equipment to test and debug systems All "Hands On"	Limited equipment needed to test and debug systems. No need for expensive special test equipment "Hands On" limited
Software	Tests generally not reusable	Reusable test software go from debug -board
Internal Visibility	Fault isolation depends on ad-hoc testability Manual probing required Required external hardware to look at internal nodes	Fault isolation increased No manual probing required Has internal node visibility
Board Real Estate	Additional real estate required to support ad-hoc testability and space to allow for probing	Additional real estate, four I/O pins, and a larger board package footprint

Table 6-3 : Differences in Traditional and Boundary Scan Test Methods.

6.5 Lessons Learned

Major benefits of using boundary scan versus using traditional methods for design verification, debug, and test were discovered through development of the TMC prototype. Most of the lessons learned fall into one of the following categories: fault-isolation, design partitioning and test access, ease of use, and scan path design. A discussion of these categories and specific lessons, design rules, and observations that were made is provided below.

6.5.1 Fault Isolation (Traditional vs. Boundary Scan)

Using Boundary within the memory module, the memory was loaded and verified. This allowed the problem to be isolated to a design error within the memory enable logic. Another example of how the Test Software and Boundary Scan were successful in isolating various system problems was in testing the 1149.n Test Controller module, where a miswired bus enable signal was creating bus contention on data line. Boundary Scan helped isolate the problem quickly, without requiring any manual probing of the board. In addition to design errors that were isolated, several component and signal failures also were isolated.

6.5.2 Design Partitioning and Test Access

It is very important that Boundary Scan be implemented at functional partitions, especially the board-to-board interface, to allow functions to be exercised and isolated completely. Also, Boundary Scan access to key control signals, such as microprocessor HOLD signals, is important to avoid bus contention when attempting to drive buses with Boundary Scan registers.

By providing controllability and observability of each board's backplane signals through

Boundary Scan chips, each board could be tested independently as it arrived from fabrication. Emulation of board interfaces, whenever required, can be controlled through backplane Boundary Scan chips. Using this method, board-to-board dependencies can be eliminated, and each board could be verified stand-alone. This way, faults caused by backplane wiring can be detected and isolated easily.

6.5.3 Ease of Use (Test Software)

Using the test software to debug/test the TMC board proved to be less time-consuming than taking the traditional approach. For example, I was able to write CTL and MTL files and finish debugging the TMC in 1 week. Once these files were complete enough to verify all TMC functionality, they can be reused later to test the other TMC board.

6.5.4 Scan Path Design

While there are obvious benefits to using Boundary Scan within a design, there are also problems that can be created if it is implemented correctly. Instances where this may occur deal with scan clock control and changing of scan path lengths. These problems can be eliminated through careful adherence to scan path design rules.

Some BIST designs may require explicit clock control over independent scan paths and, therefore, mandate that scan clock be gated. Careful design should be used when such gating is necessary. This gating may cause incorrect data to be gathered during a scan cycle. While the Boundary Scan specification allows scan clocks to be disabled, it toes not encompass the system design considerations necessary to ensure the operation of a chip's additional test capabilities, such as those available on the SCOPE octals. Another problem of concern is the collapsing and expanding of scan rings within a design. For example, collapsing and expanding of scan rings should not be performed arbitrarily with respect to

the test bus controller. Changing the scan path length without notifying the test bus controller will cause the controlling software to become confused and possible not recover. All scan path length changes should be done in conjunction with the local test bus controller chip.

6.6 Summary

Chips incorporating Boundary Scan impose a minimal real estate overhead and change the process of design verification and test, which make it beneficial to the designer. By using chips that support the Boundary Scan standard in a prototype system, some of the problems and questions associated with the verification and testing of prototype systems were solved. In addition to solving test problems, the verification and test process was simplified. Finally, the structured debug/test procedure eliminates the traditional ad hoc techniques used in the past.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

The work presented in this dissertation not only automates the process of incorporating testability into the SIERA integrated system design environment, but it also provides dedicated hardware and software for controlling the test circuitry that has been added to each level of the system's hierarchy.

The test hardware incorporation was automated by hardware module generators. These generators relieve the designer of having to know how to implement a specific DFT methodology and they guarantee correct implementation. The test circuitry that is added to an already existing design supports the JTAG Boundary Scan standard described in Chapter 3, as well as, traditional test methodologies such as Scan Path and Built-In-Self-Test. Test issues associated with each level of system integration can be easily dealt with using the hierarchical test strategy and debug procedure described in Chapters 2 and 5 respectively. Hence, prototypes designed with our integrated CAD system can be functionally verified in a timely fashion.

The mundane tasks of writing test programs for a target system are automated by the test program generation software. The test program generation software extracts the necessary information required to generate a test program from several high level testability description languages.

Various issues related to the incorporation of test into the system design flow, test hardware implementation, Boundary Scan path routing, test vector generation, and testability hardware description languages are addressed in this work. The research presented here can be categorized into two major areas: test hardware and test software. The contributions made in each of these areas as a result of the work presented in this dissertation is described in the sections that follow. Additionally, some of the open problems for future research are discussed, which in some cases are related problems that were outside the scope of this work but nonetheless, arise as a direct consequence of the contributions.

7.1 Test Hardware

The test hardware includes circuitry or components that must be either added to a chip or a board in order to improve its testability, as well as, a custom board for controlling the hardware that is added to a design. This hardware impacts the circuit's area, pin count and delay, which must be balanced against the gains achieved in using them. Hence, a means of determining these costs would be of vital use. Moreover, increases in chip area and/or logic complexity increase power consumption and decrease yield rendering tools that determine the effect of these costs equally useful. Although there are costs associated with implementing chip level hardware, they are minimal when compared to the expense of testing a complex board or system where testability was not considered at all.

7.1.1 Boundary Scan Macrocell

The boundary scan macrocell automatically generates the minimum circuitry required by the JTAG Boundary Scan standard from a set of designer provided parameters. The designer need only provide parameters that determine the boundary scan register length and the overall shape of the macrocell. It is implemented in a scalable CMOS standard cell technology and is described in the SDL language. The macrocell also has provisions for supporting one internal scan path. A future enhancement to the macrocell must include provisions for supporting chips that contain multiple internal scan paths.

7.1.2 Boundary Scan I/O Pads

Boundary Scan I/O pads have also been developed for situations where a chip design may be constrained by core area and are fixed number of I/O pins. In this case, the Boundary Scan cells become part of the I/O pad circuitry. This provides a designer with more design options when he or she is considering using Boundary Scan. Placing the Boundary Scan cells in the I/O pads as opposed to placing them inside the chips core circuitry reduces the delay costs associated with their implementation. At present, pads only exist in 2μ CMOS technology, but in the future, pads for sub-micron technologies need to be developed. Again, care must be taken in their design to ensure that they do not impact circuit area or performance. Furthermore, these cells can be modified to support AC parametric tests like delay faults.

7.1.3 Boundary Scan Components Library and Test Modules

The elements of the Boundary Scan components library only constitute a fraction of the devices that are currently available as new devices are introduced. It is anticipated that as other testability bus standards like the Module Test and Maintenance Bus (P1149.5)

evolve, controllers and interface chips that support it will become available too. When they do, they too must be added to the existing test components library. The components of this library can then be combined forming dedicated test modules, that implement higher level system test functions. A direction one would take in the future would be to determine what is the smallest combination of components that will provide the most amount of test functionality.

7.1.4 Test Master Controller Board

The design and implementation of a generic test master controller board that can be configured using software to implement a variety of standard testability bus protocols, Boundary Scan in particular, was presented in Chapter 3. With this board, it is possible to test and diagnose defective components and interconnects at the chip, board, subsystem, and system level via Boundary Scan test bus. It is intended to be used in a system that employs a hierarchy of testability buses. Some of most important attributes of this board are its:

- dynamic reconfigurability - its functionality is determined by software during initialization which also makes it extremely flexible.
- compatibility - fits in well with our system hardware development environment
- low cost - compared to conventional Automatic Test Equipment
- and better performance - capable of running test at system speeds where most Automatic Test Equipment cannot.

7.2 Test Software

The software tools described in Chapter 4 support a hierarchical test strategy for integrating test into our system design environment. These tools perform tasks such as

automatic test hardware generation, automatic test pattern generation for combinational logic and board interconnect, and automatic test program generation from high level languages.

7.2.1 Testability Hardware Design Tools

Several tools that address testability issues as part of the design process have been developed. Most importantly, these tools relieve the designer of the mundane and redundant tasks required to implement a design for test methodology. For example, the JTAGtool tackles issues associated with board level scan path chaining, as well as, perform all of the pre-processing required for board interconnect testing, while PLDS is used to generate the input file required by the Xilinx XACT software from a high level behavioral description of the test controller.

At present, the JTAGtool only deals with issues associated with single scan path rings, however, it cannot support multiple scan path rings or hybrid ring/star scan path configurations. Supporting multiple and hybrid scan path configurations presents new questions which should be addressed in future work such as:

- what is the most efficient way to configure the rings such that they impose minimal impact on test application time, ease implementation, and test development?
- and for densely populated boards, how should components be placed where they impose minimal impact on routing?

Another issue that's independent of the scan path configuration problem is what affect does Boundary Scan components have on system reliability?

7.2.2 Test Vector Generation Tools

The test vector generation tools described in Chapter 4 produce efficient test patterns for testing combinational logic and board interconnect. Using the Scan Path test methodology affords us two distinct advantages: 1) it allows us to control and monitor internal logic nodes and 2) it simplifies the test generation problem. Assuming a chip is designed with the Scan Path test methodology, the Test Generation System uses PODEM, a well known combinational logic test generation algorithm, to produce tests.

The algorithms used to generate tests that will detect wiring faults in board interconnect implement some variation of a simple marching scheme, where a logic value (1 or 0) is written to and read from each Boundary Scan cell in the scan path. Though this scheme produces a simple test vector set, the time required to apply the test is O (N^2) where N is the number of bits in the scan path. It is obvious from this that as N increases so does the test application time. This would warrant future investigation into algorithms that will produce a minimal size test vector set in order to reduce the test application time without any compromise in diagnostic resolution. Also, with system operating at higher and higher speeds, these algorithms can perhaps be enhanced to produce tests that will detect line delay faults. Another issue that should be addressed is how can use the results of a test to drive the repair process.

7.2.3 Testability Hardware Description Languages

BSDL is a way to provide a consistent description of chip designs complying with the Boundary Scan standard. By writing BSDL files, designers can specify the exact implementation of a chip's Boundary Scan features. CTL complements BSDL by describing how to use these chips for chip level testing. MTL, on the other hand, is used to describe how to use these chips for board level testing. However, since CTL and MTL are

confined to the chip and board levels, a system level or hierarchical test description language that encompasses the chip, board, and system levels would be extremely useful. One such language that is being strongly considered for standardization is called Hierarchical Scan Description Language (HSDL) developed by Texas Instruments. HSDL extends and complements BSDL by describing how devices are connected at the board, system, or MCM level.

7.2.4 Test Program Generation

A major advantage to automating the test program generation process is the reduction in the time required to develop them. The generation process starts with the preparation of test description files, written in high level languages, for each chip and board. Tools have been provided to translate these test descriptions into a test program that runs on the TMC board to control the test hardware implemented on the target board.

Although test program generation reduces test program development time, it takes time to manually create the test description files it uses. Hence, automatic generation of these files would reduce the overall test development time even further. This can be accomplished by generating them as a by-product of the chip and board synthesis.

BIBLIOGRAPHY

1. [Abadir85] M. Abadir, M. Breuer, "A Knowledge Based System for Designing Testable VLSI Circuits", IEEE Design and Test, August 1985.
2. [Abadir89] M. Abadir, "TIGER: Testability Insertion Guidance System", Proc. Int'l Conf. on Computer Aided Design, November 1989, pp. 562-565.
3. [Agrawal84] V. Agrawal, S. Jain, D. Singer, "A CAD System for Design for Testability, VLSI Design, October 1984, pp. 46-54.
4. [Archambeau88] E. Archambeau, K. Van Egmond, "Built-In Test Compiler in an ASIC Environment", Proc. Int'l Test Conf., September 1988, pp. 657-664.
5. [Avra87] L. Avra, "A VHSIC ETM-Bus Compatible Test and Maintenance Interface", Proc. Int'l Test Conf., pp. September 1987, pp. 964-971.
6. [Beenker89] F. Beekner, R. Dekker, R. Stans, M. van der Star, "A Testability Strategy for Silicon Compilers", Proc. Int'l Test Conf., August 1989, pp. 660-669.
7. [Bomdica90] A. Bomdica, "oct2tgs - Users Manual", Mississippi State University, March 1990.
8. [Bozorgui80] S. Bozorgui-Nesbat, E. McCluskey, "Structured Design for Testability to Eliminate Test Pattern Generation", 10th Annual Fault-Tolerant Computing Symposium, October 1980, pp. 158-163.
9. [Breuer84] M. A. Breuer, "A Methodology for the Design of Testable VLSI Chips" ,IEEE Design and Test of Computers, 1984.
10. [Breuer85] M. A. Breuer, "On-Chip Controller Design for Built-In-Test", Technical Report CRI-88-04, Dept. of EE-Systems, USC, December 1985.
11. [Brodersen92] R. W. Brodersen, (ed.), "Anatomy of a Silicon Compiler", Klewer Academic Publishers, 1992.

12. [Budde88] W. O. Budde, "Modular Testprocessor for VLSI Chips and High-Density PC Boards", Trans. on CAD, October 1988, pp. 1118-1124.
13. [Dutt90] N. Dutt, D. Gajski, "Design Synthesis and Silicon Compilation", IEEE Design and Test of Computers, December 1990, pp. 8-23.
14. [Eichelberger77] E. Eichelberger, T. Williams, "A Logic Design Structure for LSI Testing", Proc. 14th Design Automation Conf., June 1977, pp. 462-468.
15. [Emori90] M. Emori, T. Aikyo, Y. Machida, J. Schikatani, "ASIC CAD System Based on Hierarchical Design for Testability", Proc. Int'l Test Conf., September 1990, pp. 404-409.
16. [Fasang85] P. Fasang, J. Shen, M. Schuette, W. Gwaltney, " Automated Design for Testability of Semicustom Integrated Circuits", Proc. Int'l Test Conf., November 1985, pp. 558-564.
17. [Funatsu75] S. Funatsu, N. Swkatsuki, T. Arima, "Test Generation Systems in Japan", 12th Design Automation Conf., June 1975, pp. 112-114.
18. [Fung86] H. Fung, S. Hirschhorn, "An Automatic DFT System for the Silc Silicon Compiler, IEEE Design and Test, February 1986, pp. 45-57.
19. [Gheewala89] T. Gheewala, "CrossCheck: A Cell Based VLSI Testability Solution", Proc. Design Automation Conf., June 1989.
20. [Goel81] P. Goel, "An Implicit Enumberation Algorithm to Generate Test for Combinational Logic Circuits", IEEE Trans. on Comp., March 1981, pp. 215-222.
21. [Hallenbeck89] J. Hallenbeck, J. Cybrynski, N. Kanopoulos, T. Markas, N. Vassanthavada, "The Test Engineer's Assistant: A Support Environment for Hardware Design for Testability", IEEE Computer, April 1989, pp. 59-68.
22. [Hansen89] P. Hansen, "Testing Conventional Logic and Memory Clusters using Boundary Scan Devices as Virtual ATE Channels", Proc. Int'l. Test Conf., August 1989, pp. 166-173.
23. [Harrison86] D. S. Harrison, et. al., "Data Management and Graphics Editing in the Berkeley Design Environment", Proc. Int'l Conf. Computer-Aided Design, November 1986, pp. 24-27.
24. [Hassan88] A. Hassan, J. Rajski, V. K. Agrawal, "Testing and Diagnosis of Interconnects using the Boundary Scan Architecture", Proc. Int'l. Test Conf., September 1988, pp. 126-137.
25. [Hassan89] A. Hassan, V.K. Agrawal, J. Rajski, B Dostie, "Testing of Glue Logic Interconnects using Boundary Scan Architecture, Proc. Int'l. Test Conf., August 1989, pp. 700-711.
26. [Ibarra75] O. H. Ibarra, S. K. Sahni, "Polynomially Complete Fault Detection Problems", IEEE Trans. on Comp., March 1975, pp. 242-249.
27. [IEEE88] IEEE Std. 1076-1987, "IEEE Standard VHDL Language Reference", IEEE Standards Board, March 1988.
28. [IEEE90a] IEEE Std. 1149.1-1990, "IEEE Standard Test Access Port and Boundary Scan Atchitecture", February 15, 1990.
29. [IEEE90b] IEEE Std. P1149.5/D0.2, "Standard Backplane Module Test and Maintenance (MTM) Bus Protocol", April 1990.
30. [IEEE91a] IEEE Std. P1149.2/D0.2, "Extended Digital Serial Subset", February 1991.

31. [IEEE91b] IEEE Std. P1149.1b/D1,"Supplement (B) to Standard Test Access Port and Boundary Scan Architecture", October 7, 1991.
32. [Kornegay90a] K. T. Kornegay, R. W. Brodersen, "A VME Based Test Controller Board for TSS", First Great Lakes Symposium on VLSI, February 1990.
33. [Kornegay90b] K. T. Kornegay, "A Test Controller Board for TSS", M. S. Report, Memorandum No. UCB/ERL M91/4, January 1991.
34. [Kornegay91] K. T. Kornegay, R. W. Brodersen, "A VME Based Test Controller Board", Preseted at Int'l. Test Conf., November 1991.
35. [Kornegay92] K. T. Kornegay, R. W. Brodersen, "An Architecture for a Reconfigurable 1149.n Master Controller Board", Proc. Int'l Test Conf., September 1992.
36. [LeBlanc84] J. LeBlanc, "LOCST: A Built-In-Self-Test Technique", IEEE Design and Test, November 1984, pp. 45-52.
37. [Lien88] J. Lien, M. Breuer, "A Test and Maintenance Controller for a Module Containing Testable Chips", Proc. Int'l. Test Conf., September 1988, pp. 502-513.
38. [Lien89] J. Lien, M. Breuer, "A Universal Test and Maintenance Controller for Modules and Boards", IEEE Trans. on Industrial Electronics, May 1989, pp. 231-240.
39. [Lien91] J. Lien, "Design of Hierarchically Testable and Maintainable Systems", Ph.D Thesis, July 1991, University of Southern California, CEng Technical Report 91-19.
40. [Maunder90] C. Maunder, R. Tulloss, "The Test Access Port and Boundary Scan Architecture", IEEE Computer Society Press, 1990.
41. [McCluskey85a] E. McCluskey, "Built-In-Self-Test Techniques", IEEE Design and Test", April 1985, pp. 21-28.
42. [McCluskey85b] E. McCluskey, "Built-In-Self-Test Structures", IEEE Design and Test", April 1985, pp. 29-36.
43. [Muris90] M. Muris, "Integrating Boundary Scan Into An ASIC Design Flow", Proc. Int'l Test Conf., September 1990, pp. 472-477.
44. [Octtools] Berkeley CAD Group, "OCTTOOLS Reference Manuals", EECS Dept., U. C. Berkeley, 1991.
45. [Rabaey86] H. Rabaey, S. Pope, R. W. Brodersen, "An Integrated Automatic Layout Generation System for DSP Circuits", IEEE Trans. on CAD, July 1985.
46. [Racal] Racal-Redac Inc, "VISULA-PLUS User's Guide".
47. [Roth67] J. Roth, W. Bouricius, P. Schneider, "Programmed Algorithms to Compute Tests and to Detect and Distinguish Between Failures in Logic Circuits", IEEE Trans. Electronic Comp., October 1967, pp. 567-580.
48. [Samad86] M. Samad, J. Fortes, "Explanation Capabilities in DEFT - A Design for Testability Expert System", Proc. Int'l Test Conf., September 1986, pp. 954-963.
49. [Samad89] A. Samad, M. Bell, "Automating ASIC Design for Testability - The VLSI Test Assistant", Proc. Int'l Test Conf., August 1989, pp. 819-828.
50. [Shung89] C. S. Shung, et.al., "An Integrated CAD System for Algorithm-Specific IC Design", IEEE Trans. on CAD, April 1991.
51. [Srivistava92] M. Srivastava, "Rapid-Prototyping of Hardware and Software in a Unified Framework", Ph.D. Thesis, June 1992.
52. [Stroud86] S. K. Jain, C. E. Stroud, "Built-In-Self-Ttest of Embedded Memories", IEEE Design & Test, October 1986, pp. 27-37.

53. [Sun91] J. Sun, M. Srivasstava, R. W. Brodersen, "SIERA: A CAD Environment for Real-Time Systems", 3rd IEEE/ACM Physical Design Workshop, May 1991.
54. [Swan89] G. Swan, Y. Trivedi, D. Wharton, "CrossCheck - A Practical Solution for ASIC Testability", Proc. Int'l Test Conf., August 1989, pp. 903-908.
55. [TI90a] Texas Instruments, "SCOPE Testability Products Applications Guide", 1990.
56. [TI90b] Texas Instruments, Scan Test Devices with Octal Bus Tranceivers Data Sheet.
57. [TI90c] Texas Instruments, Scan Test Devices with D-Type Edge Triggered Flip-Flops Data Sheet.
58. [TI90d] Texas Instruments, Scan Test Devices with Octal Buffer Data Sheet.
59. [TI90e] Texas Instruments, Scan Test Device with Octal Latch Data Sheet.
60. [USCTG88] USC Test Group, "Test Generation System (TGS) User's Manual -- Version 1.0", USC, Dept. of Elec. Eng.-Sys., Technical Report No. CENG 89-03.
61. [VHSIC86] VHSIC Phase 2, "Interoperability Standards ETM-bus Specification", Version 2.0, December 31, 1986.
62. [VxWorks] Wind River Systems, "VxWorks Programmers Guide".
63. [Williams73] M. Williams, J. Angell, "Enhanced Testability of Large Scale Integrated Circuits via Test Points and Additional Logic", IEEE Trans. on Computers, January 1973.
64. [Williams83] T. Williams, K. Parker, "Design for Testability - A Survey", Pro. of the IEEE, January 1983, pp. 98-112.
65. [Wagner87] P. Wagner, "Interconnect Testing with Boundary Scan", Int'l Test Conf., September 1987, pp. 52-57.
66. [Xilinx91] Xilinx Inc., "User's Guide and Tutorials", 1991.
67. [Xilinx92] Xilinx Inc., "The XC4000 Data Book", 1992.
68. [Yau90] C. Yau, "The Boundary Scan Master: Target Applications and Functional Requirements", Proc. Int'l Test Conf., September 1990, pp. 311-315.
69. [Yacc78] S. C. Johnson, "Yacc: Yet Another Compiler-Compiler", UNIX Programmer's Manual, AT&T Bell Laboratories, 1978.
70. [Yu91] R. Yu., "PLDS: Prototyping in LAGER using Decomposition and Synthesis", M. S. Report, U. C. Berkeley, May 1991, ERL Memo. No. UCB/ERL M91/53.

APPENDIX A: BSDL Files

```
-- BSDL description of Texas Instruments 74bct8244 Octal D Flip-Flop
--
--   source         :   /sccs/scan/src/s.74bct8244
--
--   revision       :   13.1
--
--   date           :   04/22/91-22:30:35
--

entity ttl74bct8244 is
  generic (PHYSICAL_PIN_MAP : string := "DW_PACKAGE");

  port (G1_BAR:in bit; Y:out bit_vector(1 to 8); A:in bit_vector(1 to
8);
        GND, VCC:linkage bit; G2_BAR:in bit; TDO:out bit; TMS, TDI,
TCK:in bit);

  use STD_1149_1_1990.all;    -- Get Std 1149.1-1990 attributes and
definitions

  attribute PIN_MAP of ttl74bct8244 : entity is PHYSICAL_PIN_MAP;

          constant     DW_PACKAGE:PIN_MAP_STRING:="G1_BAR:1,
Y:(2,3,4,5,7,8,9,10), " &
          "A:(23,22,21,20,19,17,16,15)," &
          "GND:6, VCC:18, G2_BAR:24, TDO:11, TMS:12, TCK:13, TDI:14";

          constant     FK_PACKAGE:PIN_MAP_STRING:="G1_BAR:9,
Y:(10,11,12,13,16,17,18,19)," &
          "A:(6,5,4,3,2,27,26,25)," &
          "GND:14, VCC:28, G2_BAR:7, TDO:20, TMS:21, TCK:23, TDI:24";
```

```
  attribute TAP_SCAN_IN    of TDI : signal is true;
  attribute TAP_SCAN_MODE  of TMS : signal is true;
  attribute TAP_SCAN_OUT   of TDO : signal is true;
  attribute TAP_SCAN_CLOCK of TCK : signal is (20.0e6, BOTH);

  attribute INSTRUCTION_LENGTH of ttl74bct8244 : entity is 8;

  attribute INSTRUCTION_OPCODE of ttl74bct8244 : entity is
       "BYPASS (11111111, 10001000, 00000101, 10000100, 00000001),"
&
       "EXTEST (00000000, 10000000),"  &
       "SAMPLE (00000010, 10000010),"  &
       "INTEST (00000011, 10000011),"  &
       "TRIBYP (00000110, 10000110),"  &     -- Boundary Hi-Z
       "SETBYP (00000111, 10000111),"  &     -- Boundary 1/0
       "RUNT   (00001001, 10001001),"  &     -- Boundary run test
       "READBN (00001010, 10001010),"  &     -- Boundary read normal
       "READBT (00001011, 10001011),"  &     -- Boundary read test
        "CELLTST(00001100, 10001100),"  &      -- Boundary selftest
normal
        "TOPHIP (00001101, 10001101),"  &      -- Boundary toggle out
test
       "SCANCN (00001110, 10001110),"  &     -- BCR Scan normal
       "SCANCT (00001111, 10001111)";        -- BCR Scan test

   attribute INSTRUCTION_CAPTURE of ttl74bct8244 : entity is
"10000001";
  attribute INSTRUCTION_DISABLE of ttl74bct8244 : entity is "TRIBYP";

  attribute REGISTER_ACCESS of ttl74bct8244 : entity is
       "BOUNDARY (READBN, READBT, CELLTST),"   &
       "BYPASS (TOPHIP, SETBYP, RUNT, TRIBYP),"   &
      "BCR[2] (SCANCN, SCANCT)";   -- 2-bit Boundary Control Register

  attribute BOUNDARY_CELLS of ttl74bct8244 : entity is "BC_1";
  attribute BOUNDARY_LENGTH of ttl74bct8244 : entity is 18;

  attribute BOUNDARY_REGISTER of ttl74bct8244 : entity is
      -- num cell  port    function safe [ccell disval rslt]
        "17 (BC_1, G1_BAR, input,    X)," &  -- Merged Input/Control
        "17 (BC_1, *,      control, 1)," &  -- Merged Input/Control
        "16 (BC_1, G2_BAR, input,    X)," &  -- Merged Input/Control
        "16 (BC_1, *,      control, 1)," &  -- Merged Input/Control
        "15 (BC_1, A(1),   input,    X)," &
        "14 (BC_1, A(2),   input,    X)," &
        "13 (BC_1, A(3),   input,    X)," &
        "12 (BC_1, A(4),   input,    X)," &
        "11 (BC_1, A(5),   input,    X)," &
        "10 (BC_1, A(6),   input,    X)," &
        "9  (BC_1, A(7),   input,    X)," &
        "8  (BC_1, A(8),   input,    X)," &
        "7  (BC_1, Y(1),   output3, X,  17, 1, Z)," &  -- cell 17 @ 1
-> Hi-Z.
        "6  (BC_1, Y(2),   output3, X,  17, 1, Z)," &
        "5  (BC_1, Y(3),   output3, X,  17, 1, Z)," &
        "4  (BC_1, Y(4),   output3, X,  17, 1, Z)," &
        "3  (BC_1, Y(5),   output3, X,  16, 1, Z)," &  -- cell 16 @ 1
-> Hi-Z.
        "2  (BC_1, Y(6),   output3, X,  16, 1, Z)," &
        "1  (BC_1, Y(7),   output3, X,  16, 1, Z)," &
        "0  (BC_1, Y(8),   output3, X,  16, 1, Z)";
```

```
end ttl74bct8244;

-- BSDL description of Texas Instruments 74bct8245 Octal Transceiver
--
--   source        :  /sccs/scan/src/s.74bct8245ab
--
--   revision      :  13.1
--
--   date          :  04/22/91-22:30:36
--

entity ttl74bct8245ab is
  generic (PHYSICAL_PIN_MAP : string := "DW_PACKAGE");

-- This BSDL description for the '8245 treats the 'A' signals as
inputs
-- and the 'B' signals as outputs, that is, as a unidirectional
device.

  port (DIR:in bit; B:out bit_vector(1 to 8); A:in bit_vector(1 to
8);
        GND, VCC:linkage bit; G_NEG:in bit; TDO:out bit; TMS, TDI,
TCK:in bit);

  use STD_1149_1_1990.all;    -- Get Std 1149.1-1990 attributes and
definitions

  attribute PIN_MAP of ttl74bct8245ab : entity is PHYSICAL_PIN_MAP;

  constant DW_PACKAGE:PIN_MAP_STRING:="DIR:1, B:(2,3,4,5,7,8,9,10), "
&
          "A:(23,22,21,20,19,17,16,15)," &
          "GND:6, VCC:18, G_NEG:24, TDO:11, TMS:12, TCK:13, TDI:14";

            constant      FK_PACKAGE:PIN_MAP_STRING:="DIR:9,
B:(10,11,12,13,16,17,18,19)," &
          "A:(6,5,4,3,2,27,26,25)," &
          "GND:14, VCC:28, G_NEG:7, TDO:20, TMS:21, TCK:23, TDI:24";

  attribute TAP_SCAN_IN    of TDI : signal is true;
  attribute TAP_SCAN_MODE  of TMS : signal is true;
  attribute TAP_SCAN_OUT   of TDO : signal is true;
  attribute TAP_SCAN_CLOCK of TCK : signal is (20.0e6, BOTH);

  attribute INSTRUCTION_LENGTH of ttl74bct8245ab : entity is 8;

  attribute INSTRUCTION_OPCODE of ttl74bct8245ab : entity is
       "BYPASS (11111111, 10001000, 00000101, 10000100, 00000001),"
&
       "EXTEST (00000000, 10000000)," &
       "SAMPLE (00000010, 10000010)," &
       "INTEST (00000011, 10000011)," &
       "TRIBYP (00000110, 10000110)," &     -- Boundary Hi-Z
       "SETBYP (00000111, 10000111)," &     -- Boundary 1/0
       "RUNT   (00001001, 10001001)," &     -- Boundary run test
       "READBN (00001010, 10001010)," &     -- Boundary read normal
       "READBT (00001011, 10001011)," &     -- Boundary read test
        "CELLTST(00001100, 10001100)," &      -- Boundary selftest
normal
```

```
        "TOPHIP (00001101, 10001101),"  &      -- Boundary toggle out
test
       "SCANCN (00001110, 10001110),"  &     -- BCR Scan normal
       "SCANCT (00001111, 10001111)";        -- BCR Scan test

  attribute INSTRUCTION_CAPTURE of ttl74bct8245ab : entity is
"10000001";
  attribute INSTRUCTION_DISABLE of ttl74bct8245ab : entity is
"TRIBYP";

  attribute REGISTER_ACCESS of ttl74bct8245ab : entity is
       "BOUNDARY (READBN, READBT, CELLTST),"   &
       "BYPASS (TOPHIP, SETBYP, RUNT, TRIBYP),"   &
      "BCR[2] (SCANCN, SCANCT)";   -- 2-bit Boundary Control Register

  attribute BOUNDARY_CELLS of ttl74bct8245ab : entity is "BC_1";
  attribute BOUNDARY_LENGTH of ttl74bct8245ab : entity is 18;

  attribute BOUNDARY_REGISTER of ttl74bct8245ab : entity is
      -- num cell  port    function safe [ccell disval rslt]
        "17 (BC_1, DIR,    input,    1),"  &  -- Sets direction A to B
        "16 (BC_1, G_NEG,  input,    X),"  &  -- Merged Input/Control
        "16 (BC_1, *,      control, 1),"  &  -- Merged Input/Control
         "15 (BC_1, *,       internal,X),"  &  -- Treat this as an
internal cell
         "14 (BC_1, *,       internal,X),"  &  -- Treat this as an
internal cell
         "13 (BC_1, *,       internal,X),"  &  -- Treat this as an
internal cell
         "12 (BC_1, *,       internal,X),"  &  -- Treat this as an
internal cell
         "11 (BC_1, *,       internal,X),"  &  -- Treat this as an
internal cell
         "10 (BC_1, *,       internal,X),"  &  -- Treat this as an
internal cell
         "9  (BC_1, *,       internal,X),"  &  -- Treat this as an
internal cell
         "8  (BC_1, *,       internal,X),"  &  -- Treat this as an
internal cell
        "7  (BC_1, B(1),   output3, X,  16, 1, Z)," &  -- cell 16 @ 1
-> Hi-Z.
         "7  (BC_1, A(1),   input,    X),"  &  -- Merged Input/Output
         "6  (BC_1, B(2),   output3, X,  16, 1, Z)," &
         "6  (BC_1, A(2),   input,    X),"  &  -- Merged Input/Output
         "5  (BC_1, B(3),   output3, X,  16, 1, Z)," &
         "5  (BC_1, A(3),   input,    X),"  &  -- Merged Input/Output
         "4  (BC_1, B(4),   output3, X,  16, 1, Z)," &
         "4  (BC_1, A(4),   input,    X),"  &  -- Merged Input/Output
         "3  (BC_1, B(5),   output3, X,  16, 1, Z)," &
         "3  (BC_1, A(5),   input,    X),"  &  -- Merged Input/Output
         "2  (BC_1, B(6),   output3, X,  16, 1, Z)," &
         "2  (BC_1, A(6),   input,    X),"  &  -- Merged Input/Output
         "1  (BC_1, B(7),   output3, X,  16, 1, Z)," &
         "1  (BC_1, A(7),   input,    X),"  &  -- Merged Input/Output
         "0  (BC_1, B(8),   output3, X,  16, 1, Z)," &
         "0  (BC_1, A(8),   input,    X)";     -- Merged Input/Output

end ttl74bct8245ab;
```

```
-- BSDL description of Texas Instruments 74bct8245 Octal Transceiver
--
--   source        :  /sccs/scan/src/s.74bct8245ba
--
--   revision      :  13.1
--
--   date          :  04/22/91-22:30:37
--

entity ttl74bct8245ba is
  generic (PHYSICAL_PIN_MAP : string := "DW_PACKAGE");

-- This BSDL description for the '8245 treats the 'B' signals as
inputs
-- and the 'A' signals as outputs, that is, as a unidirectional
device.

  port (DIR:in bit; A:out bit_vector(1 to 8); B:in bit_vector(1 to
8);
        GND, VCC:linkage bit; G_NEG:in bit; TDO:out bit; TMS, TDI,
TCK:in bit);

  use STD_1149_1_1990.all;    -- Get Std 1149.1-1990 attributes and
definitions

  attribute PIN_MAP of ttl74bct8245ba : entity is PHYSICAL_PIN_MAP;

  constant DW_PACKAGE:PIN_MAP_STRING:="DIR:1, B:(2,3,4,5,7,8,9,10), "
&
          "A:(23,22,21,20,19,17,16,15)," &
          "GND:6, VCC:18, G_NEG:24, TDO:11, TMS:12, TCK:13, TDI:14";

            constant      FK_PACKAGE:PIN_MAP_STRING:="DIR:9,
B:(10,11,12,13,16,17,18,19)," &
          "A:(6,5,4,3,2,27,26,25)," &
          "GND:14, VCC:28, G_NEG:7, TDO:20, TMS:21, TCK:23, TDI:24";

  attribute TAP_SCAN_IN    of TDI : signal is true;
  attribute TAP_SCAN_MODE  of TMS : signal is true;
  attribute TAP_SCAN_OUT   of TDO : signal is true;
  attribute TAP_SCAN_CLOCK of TCK : signal is (20.0e6, BOTH);

  attribute INSTRUCTION_LENGTH of ttl74bct8245ba : entity is 8;

  attribute INSTRUCTION_OPCODE of ttl74bct8245ba : entity is
       "BYPASS (11111111, 10001000, 00000101, 10000100, 00000001),"
&
       "EXTEST (00000000, 10000000)," &
       "SAMPLE (00000010, 10000010)," &
       "INTEST (00000011, 10000011)," &
       "TRIBYP (00000110, 10000110)," &     -- Boundary Hi-Z
       "SETBYP (00000111, 10000111)," &     -- Boundary 1/0
       "RUNT   (00001001, 10001001)," &     -- Boundary run test
       "READBN (00001010, 10001010)," &     -- Boundary read normal
       "READBT (00001011, 10001011)," &     -- Boundary read test
        "CELLTST(00001100, 10001100)," &      -- Boundary selftest
normal
        "TOPHIP (00001101, 10001101)," &      -- Boundary toggle out
test
       "SCANCN (00001110, 10001110)," &     -- BCR Scan normal
       "SCANCT (00001111, 10001111)";       -- BCR Scan test
```

```
  attribute INSTRUCTION_CAPTURE of ttl74bct8245ba : entity is
"10000001";
  attribute INSTRUCTION_DISABLE of ttl74bct8245ba : entity is
"TRIBYP";

  attribute REGISTER_ACCESS of ttl74bct8245ba : entity is
       "BOUNDARY (READBN, READBT, CELLTST),"   &
       "BYPASS (TOPHIP, SETBYP, RUNT, TRIBYP),"   &
      "BCR[2] (SCANCN, SCANCT)";   -- 2-bit Boundary Control Register

  attribute BOUNDARY_CELLS of ttl74bct8245ba : entity is "BC_1";
  attribute BOUNDARY_LENGTH of ttl74bct8245ba : entity is 18;

  attribute BOUNDARY_REGISTER of ttl74bct8245ba : entity is
      -- num cell  port    function safe [ccell disval rslt]
        "17 (BC_1, DIR,    input,    0)," &  -- Sets direction B to A
        "16 (BC_1, G_NEG,  input,    X)," &  -- Merged Input/Control
        "16 (BC_1, *,      control, 1)," &  -- Merged Input/Control
         "15 (BC_1, *,       internal,X)," &  -- Treat this as an
internal cell
         "14 (BC_1, *,       internal,X)," &  -- Treat this as an
internal cell
         "13 (BC_1, *,       internal,X)," &  -- Treat this as an
internal cell
         "12 (BC_1, *,       internal,X)," &  -- Treat this as an
internal cell
         "11 (BC_1, *,       internal,X)," &  -- Treat this as an
internal cell
         "10 (BC_1, *,       internal,X)," &  -- Treat this as an
internal cell
         "9  (BC_1, *,       internal,X)," &  -- Treat this as an
internal cell
         "8  (BC_1, *,       internal,X)," &  -- Treat this as an
internal cell
        "7  (BC_1, A(1),   output3, X,  16, 1, Z)," &  -- cell 16 @ 1
-> Hi-Z.
         "7  (BC_1, B(1),   input,    X)," &  -- Merged Input/Output
         "6  (BC_1, A(2),   output3, X,  16, 1, Z)," &
         "6  (BC_1, B(2),   input,    X)," &  -- Merged Input/Output
         "5  (BC_1, A(3),   output3, X,  16, 1, Z)," &
         "5  (BC_1, B(3),   input,    X)," &  -- Merged Input/Output
         "4  (BC_1, A(4),   output3, X,  16, 1, Z)," &
         "4  (BC_1, B(4),   input,    X)," &  -- Merged Input/Output
         "3  (BC_1, A(5),   output3, X,  16, 1, Z)," &
         "3  (BC_1, B(5),   input,    X)," &  -- Merged Input/Output
         "2  (BC_1, A(6),   output3, X,  16, 1, Z)," &
         "2  (BC_1, B(6),   input,    X)," &  -- Merged Input/Output
         "1  (BC_1, A(7),   output3, X,  16, 1, Z)," &
         "1  (BC_1, B(7),   input,    X)," &  -- Merged Input/Output
         "0  (BC_1, A(8),   output3, X,  16, 1, Z)," &
         "0  (BC_1, B(8),   input,    X)";     -- Merged Input/Output

end ttl74bct8245ba;


-- BSDL description of Texas Instruments 74bct8373 Octal D Latch
--
--  source       :  /sccs/scan/src/s.74bct8373
--
--  revision     :  13.1
```

```
--
--  date          :  04/22/91-22:30:38
--

entity ttl74bct8373 is
  generic (PHYSICAL_PIN_MAP : string := "DW_PACKAGE");

  port (CLK:in bit; Q:out bit_vector(1 to 8); D:in bit_vector(1 to
8);
        GND, VCC:linkage bit; OC_NEG:in bit; TDO:out bit; TMS, TDI,
TCK:in bit);

  use STD_1149_1_1990.all;    -- Get Std 1149.1-1990 attributes and
definitions

  attribute PIN_MAP of ttl74bct8373 : entity is PHYSICAL_PIN_MAP;

  constant DW_PACKAGE:PIN_MAP_STRING:="CLK:1, Q:(2,3,4,5,7,8,9,10), "
&
          "D:(23,22,21,20,19,17,16,15)," &
          "GND:6, VCC:18, OC_NEG:24, TDO:11, TMS:12, TCK:13, TDI:14";

            constant      FK_PACKAGE:PIN_MAP_STRING:="CLK:9,
Q:(10,11,12,13,16,17,18,19)," &
          "D:(6,5,4,3,2,27,26,25)," &
          "GND:14, VCC:28, OC_NEG:7, TDO:20, TMS:21, TCK:23, TDI:24";

  attribute TAP_SCAN_IN    of TDI : signal is true;
  attribute TAP_SCAN_MODE  of TMS : signal is true;
  attribute TAP_SCAN_OUT   of TDO : signal is true;
  attribute TAP_SCAN_CLOCK of TCK : signal is (20.0e6, BOTH);

  attribute INSTRUCTION_LENGTH of ttl74bct8373 : entity is 8;

  attribute INSTRUCTION_OPCODE of ttl74bct8373 : entity is
       "BYPASS (11111111, 10001000, 00000101, 10000100, 00000001),"
&
       "EXTEST (00000000, 10000000)," &
       "SAMPLE (00000010, 10000010)," &
       "INTEST (00000011, 10000011)," &
       "TRIBYP (00000110, 10000110)," &     -- Boundary Hi-Z
       "SETBYP (00000111, 10000111)," &     -- Boundary 1/0
       "RUNT   (00001001, 10001001)," &     -- Boundary run test
       "READBN (00001010, 10001010)," &     -- Boundary read normal
       "READBT (00001011, 10001011)," &     -- Boundary read test
        "CELLTST(00001100, 10001100)," &      -- Boundary selftest
normal
        "TOPHIP (00001101, 10001101)," &     -- Boundary toggle out
test
       "SCANCN (00001110, 10001110)," &     -- BCR Scan normal
       "SCANCT (00001111, 10001111)";       -- BCR Scan test

  attribute INSTRUCTION_CAPTURE of ttl74bct8373 : entity is
"10000001";
  attribute INSTRUCTION_DISABLE of ttl74bct8373 : entity is "TRIBYP";

  attribute REGISTER_ACCESS of ttl74bct8373 : entity is
       "BOUNDARY (READBN, READBT, CELLTST)," &
       "BYPASS (TOPHIP, SETBYP, RUNT, TRIBYP)," &
      "BCR[2] (SCANCN, SCANCT)";   -- 2-bit Boundary Control Register
```

```
  attribute BOUNDARY_CELLS of ttl74bct8373 : entity is "BC_1";
  attribute BOUNDARY_LENGTH of ttl74bct8373 : entity is 18;

  attribute BOUNDARY_REGISTER of ttl74bct8373 : entity is
      -- num cell  port    function safe [ccell disval rslt]
        "17 (BC_1, CLK,     input,    X)," &
        "16 (BC_1, OC_NEG, input,    X)," &  -- Merged Input/Control
        "16 (BC_1, *,       control, 1)," &  -- Merged Input/Control
        "15 (BC_1, D(1),    input,    X)," &
        "14 (BC_1, D(2),    input,    X)," &
        "13 (BC_1, D(3),    input,    X)," &
        "12 (BC_1, D(4),    input,    X)," &
        "11 (BC_1, D(5),    input,    X)," &
        "10 (BC_1, D(6),    input,    X)," &
        "9  (BC_1, D(7),    input,    X)," &
        "8  (BC_1, D(8),    input,    X)," &
        "7  (BC_1, Q(1),    output3, X,  16, 1, Z)," &  -- cell 16 @ 1
-> Hi-Z.
        "6  (BC_1, Q(2),    output3, X,  16, 1, Z)," &
        "5  (BC_1, Q(3),    output3, X,  16, 1, Z)," &
        "4  (BC_1, Q(4),    output3, X,  16, 1, Z)," &
        "3  (BC_1, Q(5),    output3, X,  16, 1, Z)," &
        "2  (BC_1, Q(6),    output3, X,  16, 1, Z)," &
        "1  (BC_1, Q(7),    output3, X,  16, 1, Z)," &
        "0  (BC_1, Q(8),    output3, X,  16, 1, Z)";

end ttl74bct8373;

-- BSDL description of Texas Instruments 74bct8374 Octal D Flip-Flop
--
--  source       :  /sccs/scan/src/s.74bct8374
--
--  revision     :  13.1
--
--  date         :  04/22/91-22:30:40
--

entity ttl74bct8374 is
  generic (PHYSICAL_PIN_MAP : string := "DW_PACKAGE");

  port (CLK:in bit; Q:out bit_vector(1 to 8); D:in bit_vector(1 to
8);
        GND, VCC:linkage bit; OC_NEG:in bit; TDO:out bit; TMS, TDI,
TCK:in bit);

  use STD_1149_1_1990.all;    -- Get Std 1149.1-1990 attributes and
definitions

  attribute PIN_MAP of ttl74bct8374 : entity is PHYSICAL_PIN_MAP;

  constant DW_PACKAGE:PIN_MAP_STRING:="CLK:1, Q:(2,3,4,5,7,8,9,10), "
&
         "D:(23,22,21,20,19,17,16,15)," &
         "GND:6, VCC:18, OC_NEG:24, TDO:11, TMS:12, TCK:13, TDI:14";

           constant      FK_PACKAGE:PIN_MAP_STRING:="CLK:9,
Q:(10,11,12,13,16,17,18,19)," &
         "D:(6,5,4,3,2,27,26,25)," &
         "GND:14, VCC:28, OC_NEG:7, TDO:20, TMS:21, TCK:23, TDI:24";

  attribute TAP_SCAN_IN    of TDI : signal is true;
```

```
  attribute TAP_SCAN_MODE  of TMS : signal is true;
  attribute TAP_SCAN_OUT   of TDO : signal is true;
  attribute TAP_SCAN_CLOCK of TCK : signal is (20.0e6, BOTH);

  attribute INSTRUCTION_LENGTH of ttl74bct8374 : entity is 8;

  attribute INSTRUCTION_OPCODE of ttl74bct8374 : entity is
       "BYPASS (11111111, 10001000, 00000101, 10000100, 00000001),"
&
       "EXTEST (00000000, 10000000),"  &
       "SAMPLE (00000010, 10000010),"  &
       "INTEST (00000011, 10000011),"  &
       "TRIBYP (00000110, 10000110),"  &      -- Boundary Hi-Z
       "SETBYP (00000111, 10000111),"  &      -- Boundary 1/0
       "RUNT   (00001001, 10001001),"  &      -- Boundary run test
       "READBN (00001010, 10001010),"  &      -- Boundary read normal
       "READBT (00001011, 10001011),"  &      -- Boundary read test
        "CELLTST(00001100, 10001100),"  &       -- Boundary selftest
normal
        "TOPHIP (00001101, 10001101),"  &      -- Boundary toggle out
test
       "SCANCN (00001110, 10001110),"  &      -- BCR Scan normal
       "SCANCT (00001111, 10001111)";         -- BCR Scan test

   attribute INSTRUCTION_CAPTURE of ttl74bct8374 : entity is
"10000001";
  attribute INSTRUCTION_DISABLE of ttl74bct8374 : entity is "TRIBYP";

  attribute REGISTER_ACCESS of ttl74bct8374 : entity is
       "BOUNDARY (READBN, READBT, CELLTST),"   &
       "BYPASS (TOPHIP, SETBYP, RUNT, TRIBYP),"   &
      "BCR[2] (SCANCN, SCANCT)";   -- 2-bit Boundary Control Register

  attribute BOUNDARY_CELLS of ttl74bct8374 : entity is "BC_1";
  attribute BOUNDARY_LENGTH of ttl74bct8374 : entity is 18;

  attribute BOUNDARY_REGISTER of ttl74bct8374 : entity is
      -- num cell  port    function safe [ccell disval rslt]
        "17 (BC_1, CLK,    input,   X)," &
        "16 (BC_1, OC_NEG, input,   X)," &  -- Merged Input/Control
        "16 (BC_1, *,      control, 1)," &  -- Merged Input/Control
        "15 (BC_1, D(1),   input,   X)," &
        "14 (BC_1, D(2),   input,   X)," &
        "13 (BC_1, D(3),   input,   X)," &
        "12 (BC_1, D(4),   input,   X)," &
        "11 (BC_1, D(5),   input,   X)," &
        "10 (BC_1, D(6),   input,   X)," &
        "9  (BC_1, D(7),   input,   X)," &
        "8  (BC_1, D(8),   input,   X)," &
        "7  (BC_1, Q(1),   output3, X,  16, 1, Z)," &  -- cell 16 @ 1
-> Hi-Z.
        "6  (BC_1, Q(2),   output3, X,  16, 1, Z)," &
        "5  (BC_1, Q(3),   output3, X,  16, 1, Z)," &
        "4  (BC_1, Q(4),   output3, X,  16, 1, Z)," &
        "3  (BC_1, Q(5),   output3, X,  16, 1, Z)," &
        "2  (BC_1, Q(6),   output3, X,  16, 1, Z)," &
        "1  (BC_1, Q(7),   output3, X,  16, 1, Z)," &
        "0  (BC_1, Q(8),   output3, X,  16, 1, Z)";

end ttl74bct8374;
```

```
------National Semiconductor-----------------------------------------
-------

entity scan18245t is
  generic (PHYSICAL_PIN_MAP : string := "SSOP_PACKAGE");

  port (G1_NEG:in bit; A1:inout bit_vector(0 to 8); A2:inout
bit_vector(0 to 8);
        G2_NEG:in bit; B1:inout bit_vector(0 to 8); B2:inout
bit_vector(0 to 8);
       GND:linkage bit_vector(0 to 7); VCC:linkage bit_vector(0 to
3);
       DIR1:in bit; DIR2:in bit; TDO:out bit;
       TMS, TDI, TCK:in bit);

  use STD_1149_1_1990.all;    -- Get Std 1149.1-1990 attributes and
definitions

  attribute PIN_MAP of scan18245t : entity is PHYSICAL_PIN_MAP;

  constant SSOP_PACKAGE:PIN_MAP_STRING:="G1_NEG:54, G2_NEG:31," &
                                  "B1:(2,4,5,7,8,10,11,13,14),
B2:(15,16,18,19,21,22,24,25,27)," &
                                 "A1:(55,53,52,50,49,47,46,44,43),
A2:(42,41,39,38,36,35,33,32,30)," &
          "GND:(6,12,17,23,34,40,45,51)," &
          "VCC:(9,20,37,48),"  &
          "DIR1:3, DIR2:26, TDO:28, TMS:1, TCK:29, TDI:56";

  constant PLCC_PACKAGE:PIN_MAP_STRING:="G1_NEG:54, G2_NEG:31," &
                                  "B1:(2,4,5,7,8,10,11,13,14),
B2:(15,16,18,19,21,22,24,25,27)," &
                                 "A1:(55,53,52,50,49,47,46,44,43),
A2:(42,41,39,38,36,35,33,32,30)," &
          "GND:(6,12,17,23,34,40,45,51)," &
          "VCC:(9,20,37,48),"  &
          "DIR1:3, DIR2:26, TDO:28, TMS:1, TCK:29, TDI:56";


  attribute TAP_SCAN_IN    of TDI : signal is true;
  attribute TAP_SCAN_MODE  of TMS : signal is true;
  attribute TAP_SCAN_OUT   of TDO : signal is true;
  attribute TAP_SCAN_CLOCK of TCK : signal is (25.0e6, BOTH);

  attribute INSTRUCTION_LENGTH of scan18245t : entity is 8;

  attribute INSTRUCTION_OPCODE of scan18245t : entity is
       "BYPASS (11111111),"  &
       "EXTEST (00000000),"  &
       "SAMPLE (10000001),"  &
       "HIGHZ  (00000011),"  &
       "CLAMP  (10000010)";

  attribute INSTRUCTION_CAPTURE of scan18245t : entity is "00111101";
  attribute INSTRUCTION_DISABLE of scan18245t : entity is "HIGHZ";

  attribute REGISTER_ACCESS of scan18245t : entity is
       "BYPASS (HIGHZ,CLAMP)";  -- HIGHZ and CLAMP

  attribute BOUNDARY_CELLS of scan18245t : entity is "BC_1, BC_4";
  attribute BOUNDARY_LENGTH of scan18245t : entity is 80;
```

```
attribute BOUNDARY_REGISTER of scan18245t : entity is
    -- num cell  port    function safe [ccell disval rslt]
      "0  (BC_1, A2(8),    output3, X,  73, 0, Z)," &
      "1  (BC_1, A2(7),    output3, X,  73, 0, Z)," &
      "2  (BC_1, A2(6),    output3, X,  73, 0, Z)," &
      "3  (BC_1, A2(5),    output3, X,  73, 0, Z)," &
      "4  (BC_1, A2(4),    output3, X,  73, 0, Z)," &
      "5  (BC_1, A2(3),    output3, X,  73, 0, Z)," &
      "6  (BC_1, A2(2),    output3, X,  73, 0, Z)," &
      "7  (BC_1, A2(1),    output3, X,  73, 0, Z)," &
      "8  (BC_1, A2(0),    output3, X,  73, 0, Z)," &
      "9  (BC_1, A1(8),    output3, X,  77, 0, Z)," &
      "10 (BC_1, A1(7),    output3, X,  77, 0, Z)," &
      "11 (BC_1, A1(6),    output3, X,  77, 0, Z)," &
      "12 (BC_1, A1(5),    output3, X,  77, 0, Z)," &
      "13 (BC_1, A1(4),    output3, X,  77, 0, Z)," &
      "14 (BC_1, A1(3),    output3, X,  77, 0, Z)," &
      "15 (BC_1, A1(2),    output3, X,  77, 0, Z)," &
      "16 (BC_1, A1(1),    output3, X,  77, 0, Z)," &
      "17 (BC_1, A1(0),    output3, X,  77, 0, Z)," &
      "18 (BC_4, B2(0),    input,   X)," &
      "19 (BC_4, B2(1),    input,   X)," &
      "20 (BC_4, B2(2),    input,   X)," &
      "21 (BC_4, B2(3),    input,   X)," &
      "22 (BC_4, B2(4),    input,   X)," &
      "23 (BC_4, B2(5),    input,   X)," &
      "24 (BC_4, B2(6),    input,   X)," &
      "25 (BC_4, B2(7),    input,   X)," &
      "26 (BC_4, B2(8),    input,   X)," &
      "27 (BC_4, B1(0),    input,   X)," &
      "28 (BC_4, B1(1),    input,   X)," &
      "29 (BC_4, B1(2),    input,   X)," &
      "30 (BC_4, B1(3),    input,   X)," &
      "31 (BC_4, B1(4),    input,   X)," &
      "32 (BC_4, B1(5),    input,   X)," &
      "33 (BC_4, B1(6),    input,   X)," &
      "34 (BC_4, B1(7),    input,   X)," &
      "35 (BC_4, B1(8),    input,   X)," &
      "36 (BC_1, B2(8),    output3, X,  72, 0, Z)," &
      "37 (BC_1, B2(7),    output3, X,  72, 0, Z)," &
      "38 (BC_1, B2(6),    output3, X,  72, 0, Z)," &
      "39 (BC_1, B2(5),    output3, X,  72, 0, Z)," &
      "40 (BC_1, B2(4),    output3, X,  72, 0, Z)," &
      "41 (BC_1, B2(3),    output3, X,  72, 0, Z)," &
      "42 (BC_1, B2(2),    output3, X,  72, 0, Z)," &
      "43 (BC_1, B2(1),    output3, X,  72, 0, Z)," &
      "44 (BC_1, B2(0),    output3, X,  72, 0, Z)," &
      "45 (BC_1, B1(8),    output3, X,  76, 0, Z)," &
      "46 (BC_1, B1(7),    output3, X,  76, 0, Z)," &
      "47 (BC_1, B1(6),    output3, X,  76, 0, Z)," &
      "48 (BC_1, B1(5),    output3, X,  76, 0, Z)," &
      "49 (BC_1, B1(4),    output3, X,  76, 0, Z)," &
      "50 (BC_1, B1(3),    output3, X,  76, 0, Z)," &
      "51 (BC_1, B1(2),    output3, X,  76, 0, Z)," &
      "52 (BC_1, B1(1),    output3, X,  76, 0, Z)," &
      "53 (BC_1, B1(0),    output3, X,  76, 0, Z)," &
      "54 (BC_4, A2(0),    input,   X)," &
      "55 (BC_4, A2(1),    input,   X)," &
      "56 (BC_4, A2(2),    input,   X)," &
      "57 (BC_4, A2(3),    input,   X)," &
```

```
        "58 (BC_4, A2(4),     input,    X)," &
        "59 (BC_4, A2(5),     input,    X)," &
        "60 (BC_4, A2(6),     input,    X)," &
        "61 (BC_4, A2(7),     input,    X)," &
        "62 (BC_4, A2(8),     input,    X)," &
        "63 (BC_4, A1(0),     input,    X)," &
        "64 (BC_4, A1(1),     input,    X)," &
        "65 (BC_4, A1(2),     input,    X)," &
        "66 (BC_4, A1(3),     input,    X)," &
        "67 (BC_4, A1(4),     input,    X)," &
        "68 (BC_4, A1(5),     input,    X)," &
        "69 (BC_4, A1(6),     input,    X)," &
        "70 (BC_4, A1(7),     input,    X)," &
        "71 (BC_4, A1(8),     input,    X)," &
        "72 (BC_1,    *,     control, 0)," &
        "73 (BC_1,    *,     control, 0)," &
        "74 (BC_4, G2_NEG,     input,    X)," &
        "75 (BC_4, DIR2, input, X)," &
        "76 (BC_1,    *,     control, 0)," &
        "77 (BC_1,    *,     control, 0)," &
        "78 (BC_4, G1_NEG,     input,    X)," &
        "79 (BC_4, DIR1, input, X)" ;

end scan18245t;


-- Motorola 68040 BSDL description                                    900618
--
--  source        :  /sccs/scan/src/s.68040
--
--  revision      :  13.3
--
--  date          :  12/05/91-07:40:19
--

 entity MC68040 is

   generic(PHYSICAL_PIN_MAP:string := "PGA_18x18");

   port(TDI:    in      bit;
        TDO:    out     bit;
        TMS:    in      bit;
        TCK:    in      bit;
        TRST:   in      bit;
        RSTO:   buffer  bit;
        IPEND:  buffer  bit;
        CIOUT:  out     bit;
        UPA:    out     bit_vector(0 to 1);
        TT:     inout   bit_vector(0 to 1);
        A:      inout   bit_vector(0 to 31);
        D:      inout   bit_vector(0 to 31);
        LOCKE:  out     bit;
        LOCK:   out     bit;
        R_W:    inout   bit;
        TLN:    out     bit_vector(0 to 1);
        TM:     out     bit_vector(0 to 2);
        SIZ:    inout   bit_vector(0 to 1);
        MI:     buffer  bit;
        BR:     buffer  bit;
        TS:     inout   bit;
        BB:     inout   bit;
```

```
        TIP:    out     bit;
        PST:    buffer  bit_vector(0 to 3);
        TA:     inout   bit;
        TEA:    in      bit;
        BG:     in      bit;
        SC:     in      bit_vector(0 to 1);
        TBI:    in      bit;
        AVEC:   in      bit;
        TCI:    in      bit;
        DLE:    in      bit;
        PCLK:   in      bit;
        BCLK:   in      bit;
        IPL:    in      bit_vector(0 to 2);
        RSTI:   in      bit;
        CDIS:   in      bit;
        MDIS:   in      bit;
        EGND:   linkage bit_vector(1 to 23);
        EVDD:   linkage bit_vector(1 to 12);
        IGND:   linkage bit_vector(1 to 12);
        IVDD:   linkage bit_vector(1 to 7);
        CGND:   linkage bit_vector(1 to 2);
        CVDD:   linkage bit_vector(1 to 6);
        PGND:   linkage bit_vector(1 to 3);
        PVDD:   linkage bit_vector(1 to 2)
      );

   use STD_1149_1_1990.all;

   attribute PIN_MAP of MC68040 : entity is PHYSICAL_PIN_MAP;

 -- 18x18 PGA Pin Map

   constant PGA_18x18 : PIN_MAP_STRING :=
       "TDI:   S3, "  &
       "TDO:   T2, "  &
       "TMS:   S5, "  &
       "TCK:   S4, "  &
       "TRST:  T3, "  &
       "RSTO:  R3, "  &
       "IPEND: S1, "  &
       "CIOUT: R1, "  &
       "UPA:   (Q3, Q1), "  &
       "TT:    (P3, P2), "  &
       "A:     (L18, K18, J17, J18, H18, G18, G16, F18, E18, F16, P1,
N3, "  &
       "        N1,  M1,  L1,  K1,  K2,  J1,  H1,  J2,  G1,  F1,  E1,
G3, "  &
       "        D1,  F3,  E2,  C1,  E3,  B1,  D3,  A1), " &
       "D:     (C3,  B3,  C4,  A2,  A3,  A4,  A5,  A6,  B7,  A7,  A8,
A9, "  &
      "        A10, A11, A12, A13, B11, A14, B12, A15, A16, A17, B16,
C15,"  &
       "        A18, C16, B18, D16, C18, E16, E17, D18), " &
       "LOCKE: R18, "  &
       "LOCK:  S18, "  &
       "R_W:   N16, "  &
       "TLN:   (Q18, P18), "  &
       "TM:    (N18, M18, K17), "  &
       "SIZ:   (P17, P16), "  &
       "MI:    Q16, "  &
       "BR:    T18, "  &
```

```
        "TS:    R16, "  &
        "BB:    T17, "  &
        "TIP:   R15, "  &
        "PST:   (T15, S14, R14, T16), "  &
        "TA:    T14, "  &
        "TEA:   S13, "  &
        "BG:    T13, "  &
        "SC:    (T12, S12), "  &
        "TBI:   S11, "  &
        "AVEC:  T11, "  &
        "TCI:   T10, "  &
        "DLE:   T9, "  &
        "PCLK:  R9, "  &
        "BCLK:  R7, "  &
        "IPL:   (T8, T7, T6), "  &
        "RSTI:  S7, "  &
        "CDIS:  T5, "  &
        "MDIS:  S6, "  &
        "EGND:  (S2,  Q2,  N2,  L2,  H2,  F2,  D2,  B2,  B4,  B6,  B8,
B10,"  &
       "        B13, B15, B17, D17, F17, H17, L17, N17, Q17, S17, S15),
"  &
        "EVDD:  (R2,  M2,  G2,  C2,  B5,  B9,  B14, C17, G17, M17, R17,
S16)," &
        "IGND:  (T4,  R4,  L3,  K3,  C7,  C9,  C11, K16, M16, R13, R11,
S10)," &
        "IVDD:  (R5,  M3,  C8,  C10, C12, L16, R12), "  &
        "CGND:  (C6,  C13), "  &
        "CVDD:  (J3,  H3,  C5,  C14, H16, J16), "  &
        "PGND:  (S9,  R10, R6), "  &
        "PVDD:  (S8,  R8)  ";

 -- Other Pin Maps here when documented

   attribute TAP_SCAN_IN of TDI:signal is true;
   attribute TAP_SCAN_OUT of TDO:signal is true;
   attribute TAP_SCAN_MODE of TMS:signal is true;
   attribute TAP_SCAN_CLOCK of TCK:signal is (10.0e6, BOTH);
   attribute TAP_SCAN_RESET of TRST:signal is true;

   attribute INSTRUCTION_LENGTH of MC68040:entity is 3;

   attribute INSTRUCTION_OPCODE of MC68040:entity is
     "EXTEST   (000),"       &
     "HI_Z     (001),"       &
     "SAMPLE   (010, 011),"  &
     "SHUTDOWN (100, 101),"  &
     "BYPASS   (111, 110)";

   attribute INSTRUCTION_CAPTURE of MC68040:entity is "001";
   attribute INSTRUCTION_DISABLE of MC68040:entity is "HI_Z";

   attribute REGISTER_ACCESS of MC68040:entity is
     "BYPASS (SHUTDOWN, HI_Z)  ";

   attribute BOUNDARY_CELLS of MC68040:entity is
     "BC_2, BC_4";

   attribute BOUNDARY_LENGTH of MC68040:entity is 184;

   attribute BOUNDARY_REGISTER of MC68040:entity is
```

```
   --num  cell   port      function safe ccell dsval rslt
     "0   (BC_2, RSTO,     output2,  X), "              &
     "1   (BC_2, IPEND,    output2,  X), "              &
     "2   (BC_2, CIOUT,    output3,  X,  156, 0, Z), "  &  -- 156 = io.0
     "3   (BC_2, UPA(0),   output3,  X,  156, 0, Z), "  &
     "4   (BC_2, UPA(1),   output3,  X,  156, 0, Z), "  &
     "5   (BC_2, TT(0),    output3,  X,  156, 0, Z), "  &
     "6   (BC_4, TT(0),    input,    X), "              &
     "7   (BC_2, TT(1),    output3,  X,  156, 0, Z), "  &
     "8   (BC_4, TT(1),    input,    X), "              &
     "9   (BC_2, A(10),    output3,  X,  150, 0, Z), "  &  -- 150 =
io.ab
     "10  (BC_4, A(10),    input,    X), "              &
     "11  (BC_2, A(11),    output3,  X,  150, 0, Z), "  &
     "12  (BC_4, A(11),    input,    X), "              &
     "13  (BC_2, A(12),    output3,  X,  150, 0, Z), "  &
     "14  (BC_4, A(12),    input,    X), "              &
     "15  (BC_2, A(13),    output3,  X,  150, 0, Z), "  &
     "16  (BC_4, A(13),    input,    X), "              &
     "17  (BC_2, A(14),    output3,  X,  150, 0, Z), "  &
     "18  (BC_4, A(14),    input,    X), "              &
     "19  (BC_2, A(15),    output3,  X,  150, 0, Z), "  &
   --num  cell   port      function safe ccell dsval rslt
     "20  (BC_4, A(15),    input,    X), "              &
     "21  (BC_2, A(16),    output3,  X,  150, 0, Z), "  &
     "22  (BC_4, A(16),    input,    X), "              &
     "23  (BC_2, A(17),    output3,  X,  150, 0, Z), "  &
     "24  (BC_4, A(17),    input,    X), "              &
     "25  (BC_2, A(18),    output3,  X,  150, 0, Z), "  &
     "26  (BC_4, A(18),    input,    X), "              &
     "27  (BC_2, A(19),    output3,  X,  150, 0, Z), "  &
     "28  (BC_4, A(19),    input,    X), "              &
     "29  (BC_2, A(20),    output3,  X,  150, 0, Z), "  &
     "30  (BC_4, A(20),    input,    X), "              &
     "31  (BC_2, A(21),    output3,  X,  150, 0, Z), "  &
     "32  (BC_4, A(21),    input,    X), "              &
     "33  (BC_2, A(22),    output3,  X,  150, 0, Z), "  &
     "34  (BC_4, A(22),    input,    X), "              &
     "35  (BC_2, A(23),    output3,  X,  150, 0, Z), "  &
     "36  (BC_4, A(23),    input,    X), "              &
     "37  (BC_2, A(24),    output3,  X,  150, 0, Z), "  &
     "38  (BC_4, A(24),    input,    X), "              &
     "39  (BC_2, A(25),    output3,  X,  150, 0, Z), "  &
   --num  cell   port      function safe ccell dsval rslt
     "40  (BC_4, A(25),    input,    X), "              &
     "41  (BC_2, A(26),    output3,  X,  150, 0, Z), "  &
     "42  (BC_4, A(26),    input,    X), "              &
     "43  (BC_2, A(27),    output3,  X,  150, 0, Z), "  &
     "44  (BC_4, A(27),    input,    X), "              &
     "45  (BC_2, A(28),    output3,  X,  150, 0, Z), "  &
     "46  (BC_4, A(28),    input,    X), "              &
     "47  (BC_2, A(29),    output3,  X,  150, 0, Z), "  &
     "48  (BC_4, A(29),    input,    X), "              &
     "49  (BC_2, A(30),    output3,  X,  150, 0, Z), "  &
     "50  (BC_4, A(30),    input,    X), "              &
     "51  (BC_2, A(31),    output3,  X,  150, 0, Z), "  &
     "52  (BC_4, A(31),    input,    X), "              &
     "53  (BC_2, D(0),     output3,  X,  151, 0, Z), "  &  -- 151 =
io.db
     "54  (BC_2, D(1),     output3,  X,  151, 0, Z), "  &
     "55  (BC_2, D(2),     output3,  X,  151, 0, Z), "  &
```

```
   "56  (BC_2, D(3),     output3,  X,  151, 0, Z), "  &
   "57  (BC_2, D(4),     output3,  X,  151, 0, Z), "  &
   "58  (BC_2, D(5),     output3,  X,  151, 0, Z), "  &
   "59  (BC_2, D(6),     output3,  X,  151, 0, Z), "  &
 --num  cell   port      function safe ccell dsval rslt
   "60  (BC_2, D(7),     output3,  X,  151, 0, Z), "  &
   "61  (BC_2, D(8),     output3,  X,  151, 0, Z), "  &
   "62  (BC_2, D(9),     output3,  X,  151, 0, Z), "  &
   "63  (BC_2, D(10),    output3,  X,  151, 0, Z), "  &
   "64  (BC_2, D(11),    output3,  X,  151, 0, Z), "  &
   "65  (BC_2, D(12),    output3,  X,  151, 0, Z), "  &
   "66  (BC_2, D(13),    output3,  X,  151, 0, Z), "  &
   "67  (BC_2, D(14),    output3,  X,  151, 0, Z), "  &
   "68  (BC_2, D(15),    output3,  X,  151, 0, Z), "  &
   "69  (BC_2, D(16),    output3,  X,  151, 0, Z), "  &
   "70  (BC_2, D(17),    output3,  X,  151, 0, Z), "  &
   "71  (BC_2, D(18),    output3,  X,  151, 0, Z), "  &
   "72  (BC_2, D(19),    output3,  X,  151, 0, Z), "  &
   "73  (BC_2, D(20),    output3,  X,  151, 0, Z), "  &
   "74  (BC_2, D(21),    output3,  X,  151, 0, Z), "  &
   "75  (BC_2, D(22),    output3,  X,  151, 0, Z), "  &
   "76  (BC_2, D(23),    output3,  X,  151, 0, Z), "  &
   "77  (BC_2, D(24),    output3,  X,  151, 0, Z), "  &
   "78  (BC_2, D(25),    output3,  X,  151, 0, Z), "  &
   "79  (BC_2, D(26),    output3,  X,  151, 0, Z), "  &
 --num  cell   port      function safe ccell dsval rslt
   "80  (BC_2, D(27),    output3,  X,  151, 0, Z), "  &
   "81  (BC_2, D(28),    output3,  X,  151, 0, Z), "  &
   "82  (BC_2, D(29),    output3,  X,  151, 0, Z), "  &
   "83  (BC_2, D(30),    output3,  X,  151, 0, Z), "  &
   "84  (BC_2, D(31),    output3,  X,  151, 0, Z), "  &
   "85  (BC_4, D(0),     input,    X), "              &
   "86  (BC_4, D(1),     input,    X), "              &
   "87  (BC_4, D(2),     input,    X), "              &
   "88  (BC_4, D(3),     input,    X), "              &
   "89  (BC_4, D(4),     input,    X), "              &
   "90  (BC_4, D(5),     input,    X), "              &
   "91  (BC_4, D(6),     input,    X), "              &
   "92  (BC_4, D(7),     input,    X), "              &
   "93  (BC_4, D(8),     input,    X), "              &
   "94  (BC_4, D(9),     input,    X), "              &
   "95  (BC_4, D(10),    input,    X), "              &
   "96  (BC_4, D(11),    input,    X), "              &
   "97  (BC_4, D(12),    input,    X), "              &
   "98  (BC_4, D(13),    input,    X), "              &
   "99  (BC_4, D(14),    input,    X), "              &
 --num  cell   port      function safe ccell dsval rslt
   "100 (BC_4, D(15),    input,    X), "              &
   "101 (BC_4, D(16),    input,    X), "              &
   "102 (BC_4, D(17),    input,    X), "              &
   "103 (BC_4, D(18),    input,    X), "              &
   "104 (BC_4, D(19),    input,    X), "              &
   "105 (BC_4, D(20),    input,    X), "              &
   "106 (BC_4, D(21),    input,    X), "              &
   "107 (BC_4, D(22),    input,    X), "              &
   "108 (BC_4, D(23),    input,    X), "              &
   "109 (BC_4, D(24),    input,    X), "              &
   "110 (BC_4, D(25),    input,    X), "              &
   "111 (BC_4, D(26),    input,    X), "              &
   "112 (BC_4, D(27),    input,    X), "              &
   "113 (BC_4, D(28),    input,    X), "              &
```

```
   "114 (BC_4, D(29),    input,     X), "              &
   "115 (BC_4, D(30),    input,     X), "              &
   "116 (BC_4, D(31),    input,     X), "              &
  "117 (BC_2, A(9),     output3,  X,  150, 0, Z), "  & -- 150 = io.ab
   "118 (BC_4, A(9),     input,     X), "              &
   "119 (BC_2, A(8),     output3,  X,  150, 0, Z), "  &
--num  cell    port      function safe ccell dsval rslt
   "120 (BC_4, A(8),     input,     X), "              &
   "121 (BC_2, A(7),     output3,  X,  150, 0, Z), "  &
   "122 (BC_4, A(7),     input,     X), "              &
   "123 (BC_2, A(6),     output3,  X,  150, 0, Z), "  &
   "124 (BC_4, A(6),     input,     X), "              &
   "125 (BC_2, A(5),     output3,  X,  150, 0, Z), "  &
   "126 (BC_4, A(5),     input,     X), "              &
   "127 (BC_2, A(4),     output3,  X,  150, 0, Z), "  &
   "128 (BC_4, A(4),     input,     X), "              &
   "129 (BC_2, A(3),     output3,  X,  150, 0, Z), "  &
   "130 (BC_4, A(3),     input,     X), "              &
   "131 (BC_2, A(2),     output3,  X,  150, 0, Z), "  &
   "132 (BC_4, A(2),     input,     X), "              &
   "133 (BC_2, A(1),     output3,  X,  150, 0, Z), "  &
   "134 (BC_4, A(1),     input,     X), "              &
   "135 (BC_2, A(0),     output3,  X,  150, 0, Z), "  &
   "136 (BC_4, A(0),     input,     X), "              &
  "137 (BC_2, TM(2),    output3,  X,  156, 0, Z), "  & -- 156 = io.0
   "138 (BC_2, TM(1),    output3,  X,  156, 0, Z), "  &
   "139 (BC_2, TM(0),    output3,  X,  156, 0, Z), "  &
--num  cell    port      function safe ccell dsval rslt
   "140 (BC_2, TLN(1),   output3,  X,  156, 0, Z), "  &
   "141 (BC_2, TLN(0),   output3,  X,  156, 0, Z), "  &
   "142 (BC_2, SIZ(0),   output3,  X,  156, 0, Z), "  &
   "143 (BC_4, SIZ(0),   input,     X), "              &
   "144 (BC_2, R_W,      output3,  X,  156, 0, Z), "  &
   "145 (BC_4, R_W,      input,     X), "              &
   "146 (BC_2, LOCKE,    output3,  X,  156, 0, Z), "  &
   "147 (BC_2, SIZ(1),   output3,  X,  156, 0, Z), "  &
   "148 (BC_4, SIZ(1),   input,     X), "              &
   "149 (BC_2, LOCK,     output3,  X,  156, 0, Z), "  &
   "150 (BC_2, *,        controlr, 0), "              &  -- io.ab
   "151 (BC_2, *,        controlr, 0), "              &  -- io.db
   "152 (BC_2, MI,       output2,  X), "              &
   "153 (BC_2, BR,       output2,  X), "              &
   "154 (BC_2, *,        controlr, 0), "              &  -- io.2
   "155 (BC_2, *,        controlr, 0), "              &  -- io.1
   "156 (BC_2, *,        controlr, 0), "              &  -- io.0
  "157 (BC_2, TS,       output3,  X,  156, 0, Z), "  &  -- 156 = io.0
   "158 (BC_4, TS,       input,     X), "              &
  "159 (BC_2, BB,       output3,  X,  155, 0, Z), "  &  -- 155 = io.1
--num  cell    port      function safe ccell dsval rslt
   "160 (BC_4, BB,       input,     X), "              &
  "161 (BC_2, TIP,      output3,  X,  155, 0, Z), "  &  -- 155 = io.1
   "162 (BC_2, PST(3),   output2,  X), "              &
   "163 (BC_2, PST(2),   output2,  X), "              &
   "164 (BC_2, PST(1),   output2,  X), "              &
   "165 (BC_2, PST(0),   output2,  X), "              &
  "166 (BC_2, TA,       output3,  X,  154, 0, Z), "  &  -- 154 = io.2
   "167 (BC_4, TA,       input,     X), "              &
   "168 (BC_4, TEA,      input,     X), "              &
   "169 (BC_4, BG,       input,     X), "              &
   "170 (BC_4, SC(1),    input,     X), "              &
   "171 (BC_4, SC(0),    input,     X), "              &
```

```
    "172 (BC_4, TBI,      input,    X), "              &
    "173 (BC_4, AVEC,     input,    X), "              &
    "174 (BC_4, TCI,      input,    X), "              &
    "175 (BC_4, DLE,      input,    X), "              &
    "176 (BC_4, PCLK,     input,    X), "              &
    "177 (BC_4, BCLK,     input,    X), "              &
    "178 (BC_4, IPL(0),   input,    X), "              &
    "179 (BC_4, IPL(1),   input,    X), "              &
  --num  cell   port     function safe ccell dsval rslt
    "180 (BC_4, IPL(2),   input,    X), "              &
    "181 (BC_4, RSTI,     input,    X), "              &
    "182 (BC_4, CDIS,     input,    X), "              &
    "183 (BC_4, MDIS,     input,    X)  ";

attribute DESIGN_WARNING of MC68040: entity is
    "A non-standard clocking protocol on BCLK must be observed " &
    "when entering Boundary Scan Test Mode.";

 end MC68040;
```

APPENDIX B: Test Hardware And Software Organization

Most of the software tools and hardware libraries described in this thesis. This appendix describes the organization and location of the test hardware and software. test programs, SDL files for the new parts and hardware modules can be found in the following directories.

I. Test Hardware

This directory is organized into two parts: chip level hardware and board level hardware. The files and directories in each of the subdirectories are listed below.

a. ~siera/testhw/chiplevel/bscan
   This directory contains chip level macro that generates all of the circuitry required to support the Boundary Scan architecture described in Chapter 3.

b. ~siera/testhw/boardlevel/TMCboard
   This directory contains all SDL files for the Test Master Controller Board.

c. ~siera/testhw/boardlevel/SDL.MODULES
This directory contains all of the dedicated board level test modules described in Chapter 4.

d. ~siera/testhw/boardlevel/SDL.LEAF
This directory contains leafcells for all of the programmable parts used in the TMC board design.

e. ~siera/testhw/boardlevel/NEWPARTS
This directory contains SDL files for all of the components used in the TMC board design.

f. ~siera/testhw/boardlevel/BSDL.FILE
This directory contains Boundary Scan descriptions of all of the Boundary Scan components.

The guidelines listed below are provided to ensure correct operation of the JTAG_MACRO described in Chapter 3.

1. LSB of Boundary Scan Register must connect to BSRin
2. LSB of Internal Scan Register must connect to INTin
3. MSB of Internal and Boundary register must connect to tdi
4. INPUT_MODE signal must drive MODE signal of input BSR registers
5. OUTPUT_MODE signal must drive MODE signal of output BSR registers
6. phi1 must be connected to the global clock
7. tdi, tdo, and tms must be brought out to the package pins
8. tdi and tms must be driven by unbuffered pads because they contain internal pullups
9. tdo must drive a tristate pad controlled by EN
10. SCAN is the control signal for the internal scan register
11. rstb is the active low reset signal used to reset the IR, and TAP controller
12. CLK_DR, Shift_DR, and Upd_DR are control signals for the BSR register
13. oe_in is used for tristate or bi-directional pads, this is the user generated control signal
14. The pad order must be followed when using the bspads, I/O .....TDI.TMS.TDO....I/O, otherwise use any preferred ordering.

II. Test Software

The test software directory is also partitioned into two parts: chip level tools and board level tools. The two chip level tools are oct2tgs and JTAGtool. The man page for oct2tgs

already exists in ~lager/SUN4_2.0/LagerIV while the man page for JTAGtool is listed below.

```
JTAGtool(1)        SIERA Tools Users Manual        JTAGtool(1)

NAME
     JTAGtool - threads all of the Boundary Scan components
     contained in an OCT structure_instance view of a board
     design.

SYNOPSIS
     JTAGtool [-=V vov_flags] [-=E on_error] [-q] [-L Logfile]
     [-d] cell

DESCRIPTION
     JTAGtool is used to thread all of the Boundary Scan chips in
     a board design in the order they appear in the design hierarchy
     and it also generates a file containing the design netlist which
     is to be used later for debugging purposes. The input must be
     it must be a hierarchical OCT view.

INPUT
     The input is taken as the cellname indcated on the command
     line. If the view and/or facet are omitted, the contents
     facet of the SIV is used.

     JTAGtool requires that the OCT views contain the following
     properties:

     When using the structure_instance view:

     i.   The cell:structure_instance:contents facet must have
          the direction properties attached to all the formal
          terminals. Also, the formal terminals should contain
          the nets they are connected to.
 OPTIONS
        -=V:          reserved to control design manager VOV
                      (not used yet)
       -=E:         cause fatal errors to core dump (on_error = "core")
                      or exit(on_error = "exit")
        -q:           turn off terminal messages
        -L:           Log file name
        -d:           debug mode
        cell:         name of cell:structure_instance to be processed


SIERA Release 1.0   Last change: Aug 1992                      1

JTAGtool(1)         SIERA Tools Users Manual         JTAGtool(1)

 AUTHOR
     Kevin T. Kornegay (UC Berkeley)
```

The procedure for using the *M2C* tool is described below.

Using M2C

Before using this tool, the user must prepare an MTL description of the board under test and a CTL description for all of its Boundary Scan components.

An MTL file consists of a part that describes the configuration of the board under test and another part that describes how the board is to be tested using the test specific statements described in Chapter 5.

The CTL file consists of a BSDL description of the part and one or more template-based or user-defined TDM procedures.

Once the CTL and MTL files are prepared, the user can generate a test program by executing *M2C* boardname, where boardname.mtl is the name of the MTL file. Then execute *genTarget* to create the test program. The test program which is written in C is placed in a directory called workDIR. Compile the executable files using the Unix make utility on the boardname.mak file produced by *genTarget*. Pop up a window on the target VME card cage and load the executable file on the VME CPU board and execute the test program. *M2C* and *genTarget* are located in ~siera/testsw/bin.Examples are located in ~siera/testsw/examples.