Rapid Prototyping of Application-Specific Signal Processors (RASSP)

RASSP Design For Testability (DFT) Methodology

Version 1.0

September 15, 1995

ADVANCED TECHNOLOGY LABORATORIES

LOCKHEED MARTIN

ADVANCED TECHNOLOGY LABORATORIES

LOCKHEED MARTIN

Rapid Prototyping of Application-Specific Signal Processors (RASSP)

RASSP Design For Testability (DFT) Methodology

Version 1.0

Submitted By:

Lockheed Martin RASSP Team
Lockheed Martin Advanced Technology Laboratories
Bldg. A&E 2W
1 Federal Street
Camden, New Jersey   08102

Contract Number:

DAAL01-93-C-3380

Date:

September 15, 1995

Notes for Users

Version 1.0 is being released for trial use by the Lockheed Martin ATL RASSP team and designated government agencies. It represents the next step towards comprehensive Design-for-Testability methodology that introduces new concepts for adoption into the RASSP environment and that helps identify and direct the needs for new automated design-for-test tools which enable the rapid prototyping of testable electronic digital systems.

It is expected that this document will be updated periodically as concepts are refined and any deficiencies corrected during the trial use and implementation phases.

Please send all comments to:

Diana Brucoli
Lockheed Martin Advanced Technology Laboratories
One Federal Street, A&E/2NW
Camden, NJ 08102
609-338-4247 (voice)
609-338-4155 (fax)
dbrucoli@atl.lmco.com

Table of Contents

List of Illustrations

List of Tables

1.0 INTRODUCTION

1.1 Purpose

The RASSP Design-For-Testability (DFT) Methodology enables designers to create systems that can be cost-effectively tested throughout their life cycles. Designs that adhere to this methodology are made testable on the basis of various design for testability (DFT) and built-in-self-test (BIST) techniques. The methodology covers various aspects of test and diagnosis at the chip, MCM, board and system levels, including test requirements capture; test strategy development; DFT and BIST architecture development; DFT and BIST design and insertion; test pattern generation; test pattern evaluation; and test application and control. This methodology provides the designer with a process for introducing testability requirements and constraints early in the design cycle and for addressing DFT and BIST issues hierarchically at the chip, multichip module (MCM), board, and system levels. The payback for early testability emphasis includes lower test cost throughout the life cycle of the product, reduced design cycle time, improved system quality, and enhanced system availability and maintainability.

1.2 Scope

The hierarchical design-for-test methodology described in this document provides RASSP designers and test engineers with a process for introducing test requirements and strategies early in the design cycle. It then provides them with procedures for addressing DFT and test strategy selection and implementation issues at the chip, MCM, board, and system levels. The methodology embodies several incremental layers of DFT strategy selection and implementation activities which operate on successive refinements of the design, starting from the conceptual (system specification) level down to the physical implementation level. The impact of test decisions on meeting requirements is determined at all levels of the design. While the methodology is not driven by specific test related tools, knowledge of the existing tools has been used to ensure the practicality of the methodology. In addition, examples of tools are discussed that can be used to automate various phases of the DFT processes.

As is true with the RASSP program in general, discussions of analog and mixed signal systems are beyond the scope of the document.

1.3 Intended Audience

This document is intended for RASSP designers, manufacturing test engineers, and field diagnostics engineers. Although, the prime focus is on the rapid design and prototyping of digital signal processor systems, the methodology could also be used on other system application projects (e.g., computers, telecommunications, etc.)

1.4 Document Organization

This document is organized into six basic sections. This section defines the purpose and scope of the document, identifies how the document is to be used and its relationship to other RASSP documents, and discusses the test problem and the DFT solution. Section 2 gives an overview of RASSP goals and discusses requirements for the DFT Methodology, along with some simplifying concepts. Section 3 is the focal point of the document, describing the details of the RASSP methodology, as well as the relationship and contribution of the DFT methodology to the RASSP goals and methodology. This section also addresses the many types of testing throughout the life cycle phases of the system, while discussing applicable forms of DFT. Sections 4, 5, and 6 provide a summary, glossary, and references respectively.

1.5 Relationship to Other Key Documents

There is a close relationship between this document and the overall RASSP Methodology Overview Document. DFT and test activities are incorporated into the overall methodology document at a high level. This document describes these activities in more detail and how they interface with other RASSP design activities.

There is also a close relationship between this document and the RASSP Testability Architecture Description. While the DFT Methodology discusses DFT processes, the Testability Architecture Description discusses a prescribed testability architecture that is based on current DFT techniques used in custom design and COTS design environments. The prescribed architecture is both compatible with and derivable from the DFT Methodology. Since the Testability Architecture reflects the details of the test related features of the RASSP Model Year Architecture, all three documents have a strong relationship.

This document builds upon a wealth of test related knowledge captured in various military test standards and testability handbooks, such as MIL-STD-2165A and its associated handbook, MIL-HDBK-XX47 (2/28/92). Although the RASSP DFT methodology does not strictly follow the detailed task descriptions of MIL-STD-2165A, the methodology supports many of the philosophies of the standard, such as the following:

a. Establishment of sufficient, achievable, and affordable test strategies and architectures using built-in features and external automated test equipment.
b. Integration of testability into systems during the architecture and detailed design processes.
c. Evaluation of the extent to which the design meets its testability requirements.
d. Inclusion of testability in the design review processes.

The use of VHDL and WAVES to capture and exchange test related information is an integral part of the RASSP DFT Methodology. In addition to the documents that describe these various standards, a preliminary draft manuscript for the documentation of military electronic computers with the VHDL handbook describes DoD requirements for the use of VHDL and how it applies to test related information (Chapter 8). The methodology described, herein, adheres to the guidance provided in that handbook as appropriate.

1.6 Use of this Document and Tailoring to Organizations and/or Projects

Activities and tasks described are intended to be tailored to the particular needs of the system being designed and to the development, manufacturing and field environments. Certain tasks described may not be applicable in some cases. The methodology employs a hierarchical refinement process. The number of iterations in that refinement process will vary depending on the hierarchical decomposition of the system. The selection of test strategies and test architectures is largely constraint driven in the RASSP methodology; and hence, the solution is inherently tailored to system requirements and application and environment constraints. However, based on the degree of use of COTS in the system, and the selected test strategy and architecture, tailoring of specific process steps may be required.

1.7 The Test Problem and the DFT Solution

The general, design/engineering, production and field test problems are discussed. Then the DFT solution is introduced.

1.7.1 The General Test Problem

Much of the problem of testing electronic systems today relates to the complexity of integrated circuits and, in turn, the complexity of the multichip modules, boards, and systems incorporating those ICs. The explosion in circuit complexity was first documented in the early 1970s by Gordon Moore of Intel, who showed that the transistor density per chip was doubling approximately every three years. That trend continued in the 1970s and into the 1980s. However, as the 1990s approached and the new decade began, the increase in density actually accelerated, such that today, it is not unusual to see a doubling of density within a one year period or less.

Today, VLSI logic and memory chips can contain millions of transistors. In addition, it is now possible to incorporate both digital and analog circuitry on the same chip. Boards and multichip modules can contain tens or even hundreds of components, making their complexity an order of magnitude or more greater than that of the integrated circuit.

Test problems associated with circuit complexity are further complicated by packaging complexity. The use of surface mount technology, ball-grid arrays, fine component and interconnect spacing, multichip modules, double-sided boards, multilayer boards, piggy back boards, and conformal coating, all contribute to poor accessibility of nodes for physical test probing.

1.7.2 The Design/Engineering Test Problem

During roughly the two decades spanning the mid-1960s to the mid-1980s, the use of logic analyzers, and in the case of microprocessor-based systems, the additional use of in-circuit emulators, have prevailed for debugging prototypes. The complement of tools has typically been selected on the basis of its ability to help detect and analyze design and assembly problems, as well as to aid in the integration of hardware and software. However, increased emphasis on shortening the time to first customer delivery, improving quality, managing the complexity of designs more effectively, and reducing overall life cycle costs, has resulted in a growing interest in more efficient ways of performing design verification and prototype test.

Integration of hardware and software configuration items has always represented significant risk. Unanticipated iterations between hardware and software components crop up which can result in significant schedule delays. Compounding this problem is the integration of real-time software into multiprocessor systems. Small timing differences can surface which compound over time resulting in missed tasks and/or system halt. Debugging (detection and isolation) these events/problems can add weeks to months depending upon the system size and the type of problem.

At the same time, circuit and packaging technology have thwarted attempts to continue using traditional logic analyzers and in-circuit emulators. For example, the advent of surface mount technology has made the I/O signals on ICs inaccessible for probing and clipping. Additional accessibility problems exist with systems that are tested after boards are conformally coated, after the unit's enclosure is sealed, or perhaps when the entire unit is subjected to an environmental test chamber, in which the only access is via connectors and cables.

Another problem facing traditional hardware debug approaches is that the growth in pin count has made emulator and analyzer sockets and connectors complex, expensive, and cumbersome. Complicating the problem is the fact that the tremendous increase in processor speeds has forced analyzers and in-circuit emulators into a realm of speed that is not easily supported by their external cabling techniques. Transmission line effects come into play, and signals in the test interface become subject to noise or crosstalk problems.

Still another contributing problem is that the use of emulators in the past has required the development of a processor model and interface for each new processor used. This recurring expense has been very difficult to justify, since the typical life span of usage of such processors has decreased over the years. Finally, the application of in-circuit emulators to multiprocessor based designs has always been very difficult, if not impossible.

The net result of these problems plaguing traditional methods is that designers today are finding it extremely difficult to perform adequate hardware debug and to verify correct hardware and software integration.

1.7.3 The Production Test Problem

Increases in design complexity, performance and physical constraints poses a serious challenge from the production testing point of view. Testing high performance integrated circuits and high density boards or MCMs is a difficult task because of their performance, complexity and physical and/or electrical constraints. System clock frequencies are surpassing the capabilities of Automatic Test Equipment. Traditional test methods such as in-circuit probing cannot be applied to new technologies such as Ball Grid Arrays, Surface Mounted parts and MCMs unless test points are provided or new methodologies, such as boundary-scan, are applied.

Test generation is the process of generating test stimuli and predicting expected responses, while test verification is the process of determining the fault coverage or effectiveness of the tests through such techniques as fault simulation and physical fault injection. The problem with both processes is that the time and cost required to perform them grows as the size of the circuit grows. For example, it has been shown (Ibarra and Sahni) that for a digital circuit, the time required for test generation or fault simulation grows exponentially as a function of the number of gates in the circuit. Fortunately, through clever programming techniques, tailored algorithms, expert system methods, hardware simulation accelerators, and the use of DFT, the increase in test generation and fault simulation time has been brought down to a more manageable n^2 or n^3 relationship, where "n" is the number of gates in the digital circuit. Even with such improvement, the cost of test generation and verification is still growing rapidly.

Another factor related to the production test problem is the cost of test equipment. It is not unusual to spend half a million to over a million dollars for a board tester and easily more than a million for a fully configured VLSI (chip) tester. Even ignoring the absolute dollar amount paid for the production tester, since it's cost can be amortized over the total volume of product manufactured, it is remarkable to consider the amount and complexity of test equipment required just to test an IC.

1.7.4 The Field Test Problem

The same problems outlined above for production test also apply to field test. However, the following additional constraints exist:

a. The cost of procuring and maintaining test equipment, as well as the cost to develop and maintain TPSs, is outstripping budgets. As was the case with production testers, it is not unusual to spend hundreds of thousands, if not millions, of dollars for the development or purchase of an intermediate or depot level tester.
b. The cost of maintenance in general is ever increasing. It is no longer possible to support the cost of three levels of maintenance in the field.
c. It is no longer practical to require highly skilled troubleshooters in the organizational levels to perform diagnosis of system problems.
d. The cost of stocking spares (logistics) is driven by LRU cost and false alarms. Failures identified in the field which cannot be duplicated at the depot (i.e., LRU was good and/or system backplane was out of spec.) must be minimized.

The net result is that users are requiring built-in-test and diagnostics be supplied with electronic equipment as a requirement on par with size, weight and power limitations.

1.7.5 The DFT Solution

Traditionally, test related design activities have occurred near the end of the design cycle. This approach, Design-then-Test, causes delays and sometimes forces expensive design iterations. To cost-effectively and rapidly produce high quality ICs, boards/MCMs and systems, systems and components must be designed with test as a critical requirement. A design paradigm, known as "design for testability," is increasingly being used in quality conscious organizations with a Design-to-Profit and total customer satisfaction focus.

The key solution to alleviate this growing test problem is to utilize inherent structures and incorporate additional test features into the design itself to facilitate test and diagnosis. These design-for-testability features include internal scan and boundary-scan techniques, BIST hardware and software, and test buses and controllers. DFT features such as boundary-scan and BIST can eliminate the need for expensive testers and go/no-go or diagnostic test generation by providing a mechanism for accessing and testing internal circuitry using built-in stimulus generators and response evaluators. These features can also result in shortened design cycle times, reduction in manufacturing test cost, improvement in test effectiveness, a quality improvement in the shipped product, reduction in life cycle maintenance cost, etc.

However, DFT and BIST are not free. They may add to the size of the circuits, they may introduce delays, they may add to the fabrication or assembly costs, they may reduce raw manufacturing yield, and have other potential impacts. Weighing the pros and cons of various test strategies is highly dependent on the design itself, the target application, the available tools and equipment, etc. Key test strategy decisions have to be made early in the design cycle and then continuously refined and implemented during the design process.

Numerous test methods are needed at various points during the manufacturing and life cycle maintenance (e.g., functional test, interconnect test, structural component test, at-speed test, etc.) Also many hardware options exist to support and facilitate these tests, ranging from scan to full BIST. Designers need guidance in selecting the optimal test strategies and how to utilize these methods cost-effectively throughout the system hierarchy.

At one extreme, we could require that every chip incorporate DFT and BIST. However, in most cases, only a subset of the COTS chips may incorporate testability features, while others may have no DFT features. This situation cannot be avoided because RASSP designs will very likely contain COTS items that have varying degrees of testability. The problem then becomes that of exploiting the partial DFT features in testing the system as much as possible and still maintain a high level of test quality. Over time, it is quite possible that the needs and demands of the RASSP program, as well as the rest of the industry, may drive requirements for incorporating greater levels of testability and BIST in COTS equipment.

1.8 Introductory Concepts

Key concepts are introduced to precisely define their usage within the RASSP DFT Methodology. The broadened scope of "testing" is covered leading into a description of the core test cycle (Detection/Isolation/Correction - D/I/C) used in each process step. The importance of test requirements compliance tracking is discussed. A life cycle view of testing is reviewed. In keeping with the expanded definition of "testing" and the concept of the test cycle, the definition of "design for testability" is broadened. Finally, key terms are defined so that they may be used in the detailed process descriptions.

1.8.1 A Broadened Scope for the Concept of Testing

In the DFT Methodology, the term "testing" has been expanded to encompass any process which detects and isolates and corrects anomalies (e.g., design flaws, manufacturing defects, field defects) in any phase of the life cycle. The definition of testing is being broadened to enable the concept of design for testability to be expanded so that an investment in DFT provides the greatest contribution to achieving the RASSP 4x improvement goals in cycle time, quality, and cost.

1.8.2 The Test Cycle

1.8.2.1 The Concept of the Test Cycle

While the broadened definition of testing covers many forms of testing from design verification to manufacturing test to field diagnostics or depot testing, one common subprocess exists, which is referred to in this document as "the test cycle." As depicted in Figure 1-1, the test cycle consists of three functions: detection, isolation, and correction.

– "Detection" is the process of determining the presence of an anomaly, including design flaws, manufacturing defects, and field defects.

– "Isolation" is the process of determining the location of the cause of the anomaly. In some cases, it is desirable to isolate the anomaly to a functional area, such as might occur in a fault-tolerant system capable of reconfiguring around failed functional areas. In other cases, it may be desirable to isolate to a physical entity, such as a factory or field replaceable item.

– "Correction" is the process of removing and replacing the item causing the anomaly, such as would occur in a manufacturing or field repair process. Depending upon context, it is also considered to be the process of removing the damaging effect of the item causing the anomaly, such as would be the case in a fault-tolerant system.

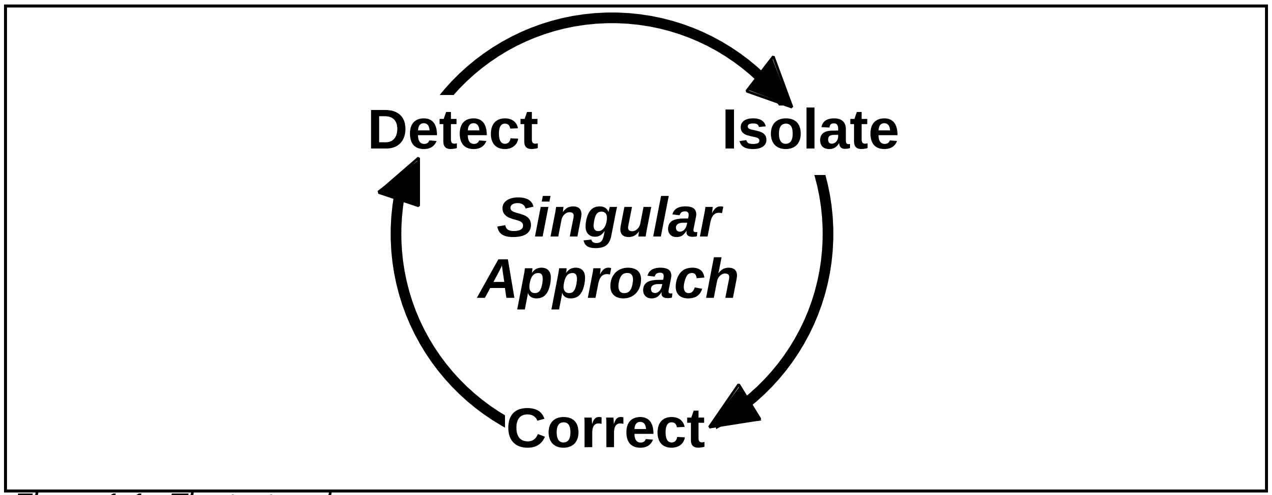


Figure 1-1. The test cycle.

There are other test related functions that have widespread application in the life cycle of a system, such as test control, initialization, and recording and reporting of anomalies. While these are important functions that will be discussed later in this document, they are not as fundamental and not necessarily common to every test process in every phase of the life cycle, as the functions defined in the "test cycle."

1.8.2.2 Test Requirements Compliance Tracking

One of the major philosophies espoused in this document is the concept of continuous, test requirements compliance tracking through overlapping processes of prediction, verification, and measurement (see Figure 1-4). "Prediction" is considered to be the process of determining compliance to test requirements prior to the availability of a complete, detailed design. It is accomplished through comparison with similar library elements or through analytic techniques, such as circuit level testability measures or topological dependency models. Prediction is performed from the System Definition process, through the Architecture Definition process, and into the early stages of the Detailed Design process.

The latter part of the Architecture Definition process, along with the entire Detailed Design process, and the early part of the Manufacturing process, are covered by the "Verification" process of test requirements compliance tracking. Verification is performed when a detailed design is available, and terminates during the Manufacturing stage, as higher volumes of physical product become available for the “measurement” process. "Verification" makes use of such techniques as deterministic fault simulation, probabilistic fault grading, and closed form proofs, for such techniques as exhaustive type BIST approaches.

Finally, the "Measurement" compliance tracking process covers the latter part of the Detailed Design process, along with the entire Manufacturing and Field Support processes. "Measurement" compliance tracking process applies when test performance measurements can be made on actual hardware and software, such as physical prototypes, manufactured or fielded systems.

The three processes of prediction, verification, and measurement together provide feedback to track test requirements compliance and to allow correction of deficiencies. In addition, the same feedback is used to correlate the results of the tools used in the three overlapping compliance tracking processes to assess their adequacy and permit improvement in the tracking processes themselves.

Details of the compliance tracking processes, along with examples of applicable tools, are discussed in Section 3 of this document.

1.8.3 A Life Cycle View of Testing

Consistent with the expanded definition of testing and the concept of the test cycle, the DFT Methodology views testing as a life cycle process. As shown in Figure 1-2, the test cycle is applied in every phase of the life cycle, and it is, of course, applied iteratively within each phase

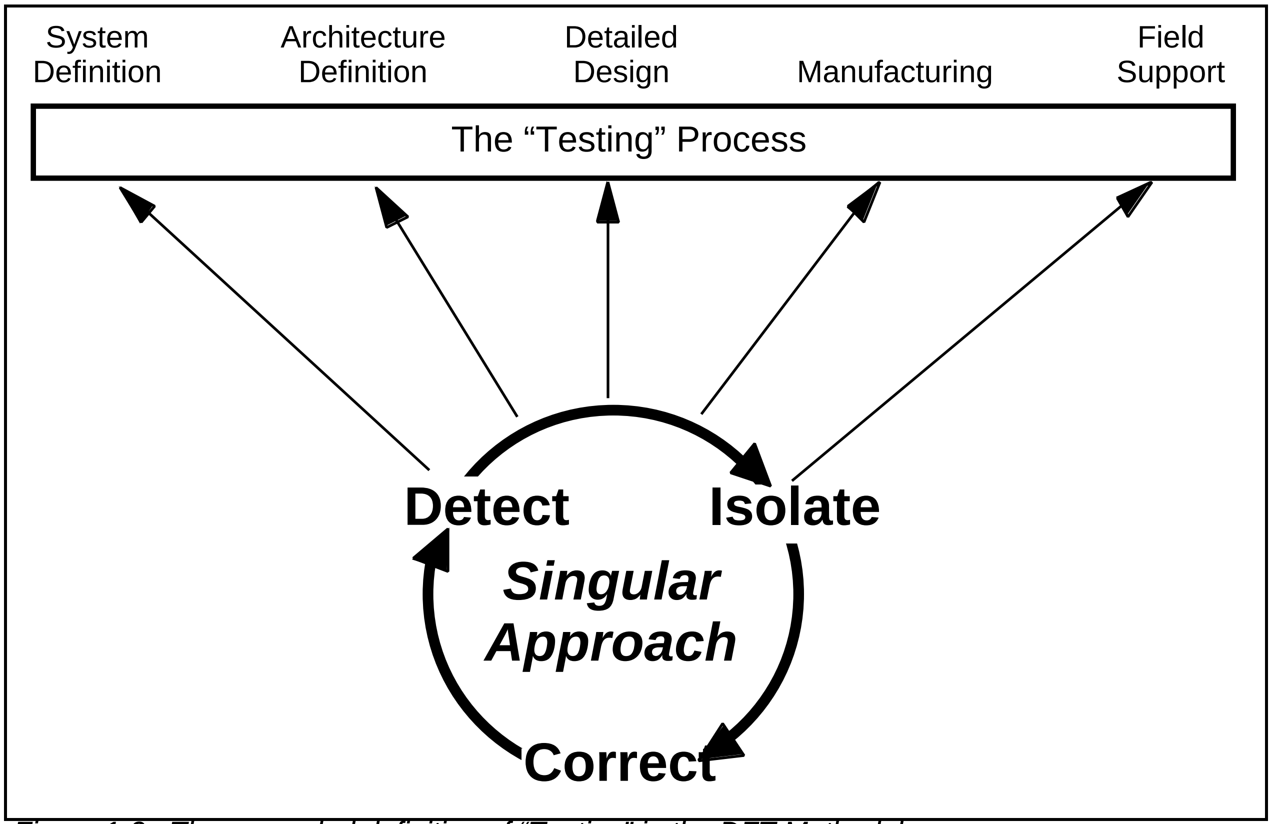


Figure 1-2. The expanded definition of "Testing" in the DFT Methodology.

as well. Figure 1-3 provides some examples of the application of the expanded notion of testing and the test cycle. As shown, the test cycle might manifest itself as a mental process during the review of a requirements specification, as automated processes occurring during simulation or ATE based testing, and as a built-in process through the execution of a BIST capability in an IC.

Figure 1-4 depicts the RASSP DFT Methodology embedded into the overall Methodology and the test architecture embedded in the Model Year Architecture. In the RASSP System Definition process, the test cycle can be applied to unearth inconsistencies or errors in requirement specifications. This could be a human mental process or be facilitated by an automated tool. The test cycle is also applied to determine anomalies during the execution of executable specifications (E-Specs).

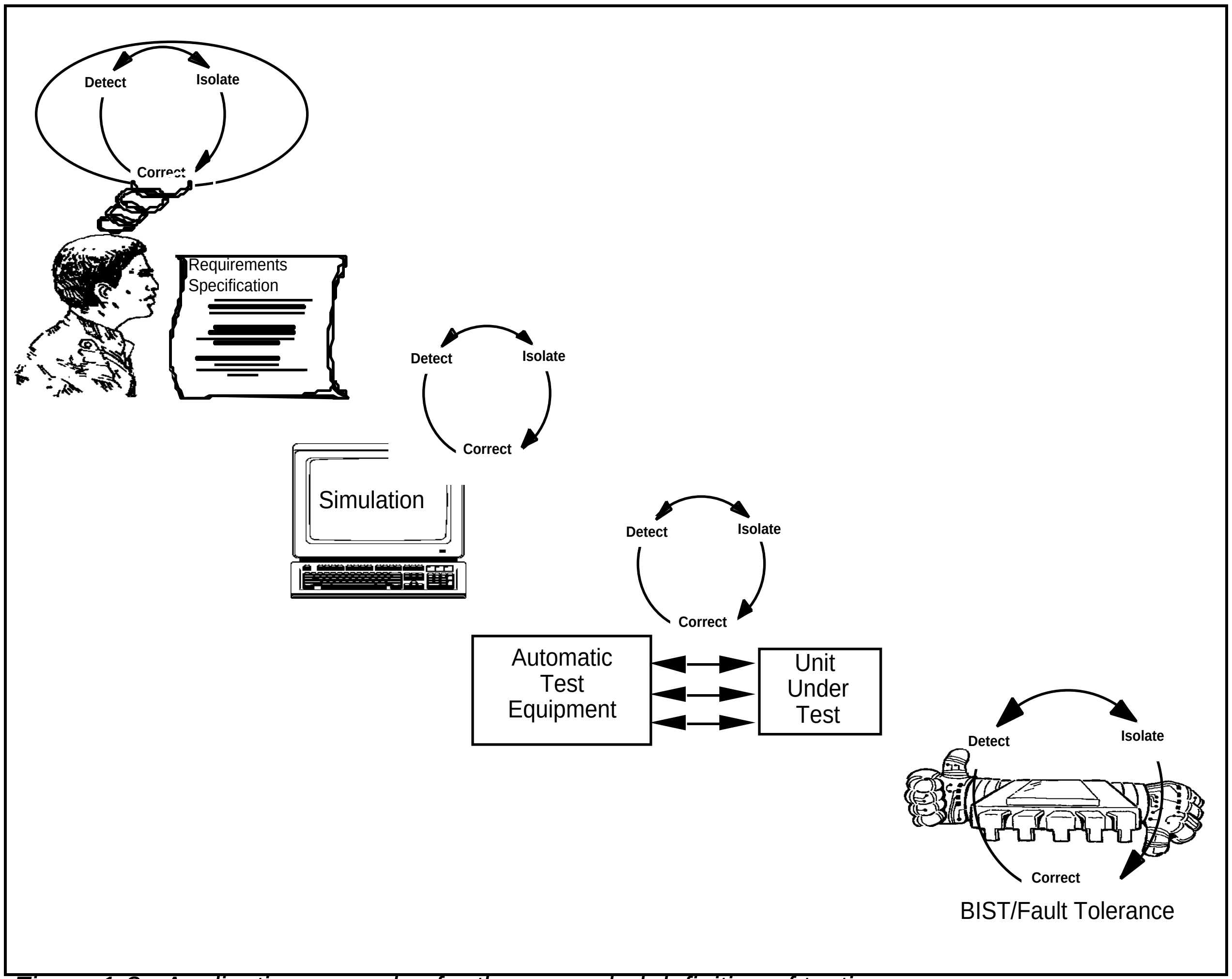


Figure 1-3. Application examples for the expanded definition of testing.

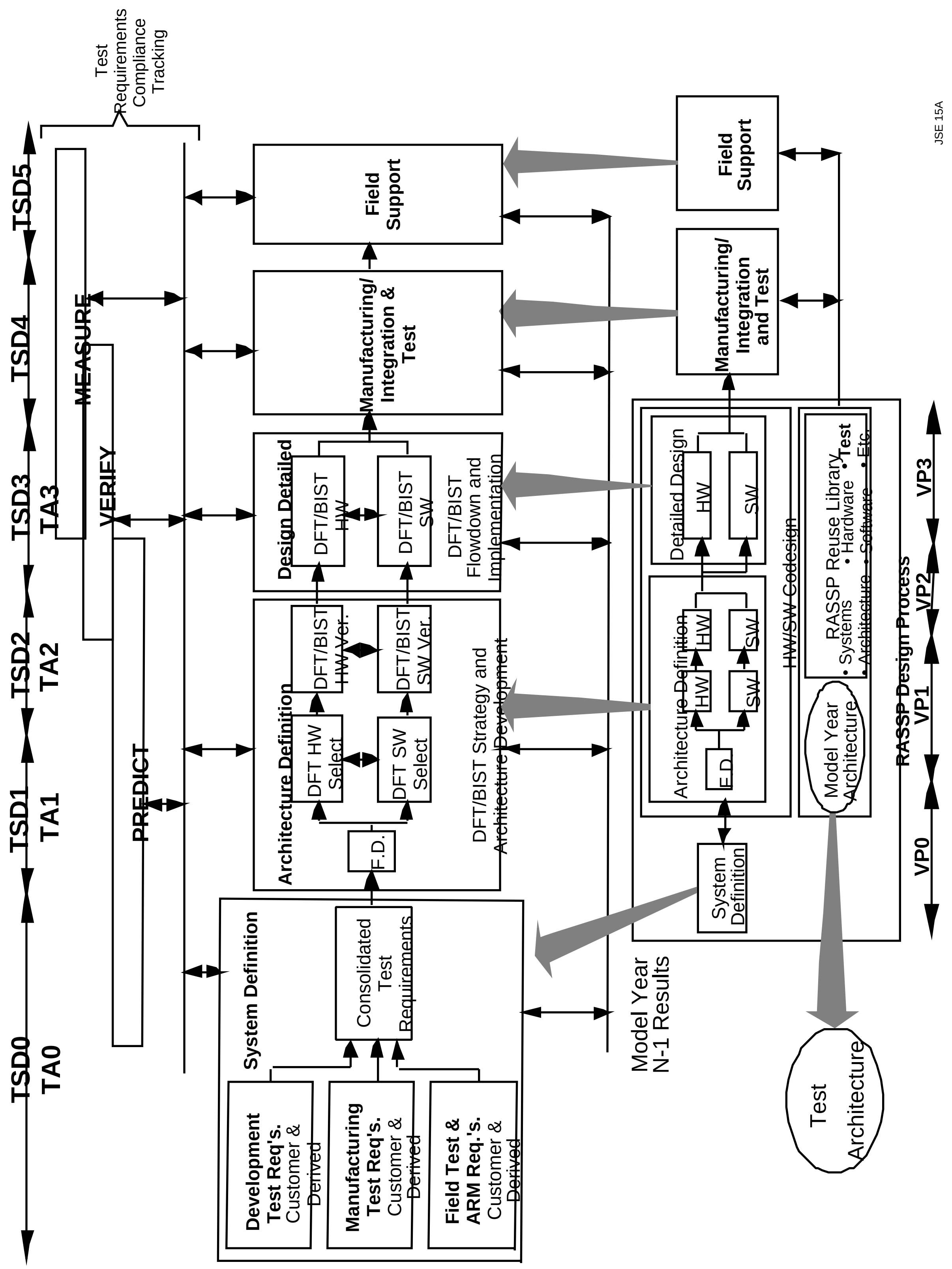


Figure 1-4. Overview of the DFT Methodology.

In the RASSP Architecture Definition process, the test cycle would appear as part of such processes as verification of functional allocation, behavioral and performance simulations, and code verification.

In the RASSP Detailed Design process, the test cycle is used iteratively during such processes as low level simulations and test vector generation and verification, as well as during insertion of BIST.

In the Manufacturing process, the test cycle appears extensively, in all forms of testing, such as functional test, in-circuit test, boundary-scan test, parametric test, environmental stress screening (ESS), etc.

In the Field Support process, the test cycle appears in such processes as power-on BIST, on-line and off-line BIST, depot level testing, etc.

1.8.4 A Broadened Scope for the Concept of Design for Testability

In keeping with the expanded definition of "testing" and the concept of the test cycle, the DFT Methodology uses a broadened definition of "design for testability," as follows:

"Design for Testability" - The incorporation of a capability in any or all embodiment(s) of a system, from requirements specification to architecture level to detailed hardware and software designs, that will facilitate (DFT) or assimilate (BIST) the processes of detecting, isolating, and correcting anomalies (design flaws, manufacturing defects, and field defects), originating from any stage of the system's life cycle. The DFT feature or BIST capability should be usable or invocable at any stage of the life cycle.

The motivation for this expanded definition of DFT comes from three factors:

a. The complexity of systems in all embodiments is increasing dramatically. Therefore, the process of verifying and ensuring the problem-free state of a system throughout its life cycle is becoming increasingly difficult and costly. The integration of DFT into the design process and into many different embodiments of the system facilitates the management of that complexity during the life cycle phases of testing. This contribution will aid in the reaching of all three RASSP goals of cost and cycle time reduction and quality improvement.
b. The cost of correcting a latent problem increases significantly as a system transitions through its life cycle phases. Catching problems earlier can be facilitated by more efficient and effective testing facilitated by DFT embodiments in each phase. This supports the RASSP goals of reducing cycle time and cost.
c. The test cycle is common to all stages of the product. Techniques and tools which support the test cycle, including testability analysis tools, DFT techniques, and test vectors, etc., have the potential for reuse from one life cycle phase to other phases. For example, boundary-scan implementations and PC-based boundary-scan testing may have been instituted to facilitate debug and test generation in the design phase; but it could also be used in the production phase, as well as for the field depot. This notion of stretching the application domain supports the RASSP goals of high degree of reuse to shorten the time to delivery.

While the notion of DFT embodiments being used in each phase of the life cycle and being integrated into all levels of the system hierarchy may seem far fetched at first, a look at the recent trends in testing and testability principles suggests otherwise. In its early days of fruition, DFT was focused almost entirely on board level techniques to solve board manufacturing test problems. The well known ad hoc DFT techniques (partitioning, test points, clock control, breaking feedback loops, etc.) emerged from those pioneering efforts. However, since those early days, the following expansions in DFT scope have occurred:

a. The advent of DFT and BIST techniques implemented within chips, such as internal scan, boundary-scan, circular BIST, STUMPS, and LOCST.

b. The beginnings of DFT and BIST synthesis at the high level design phase (e.g., behavioral level), rather than at the back end of the process.
c. The advent of standardized and hierarchical system test buses, such as IEEE 1149.1 and P1149.5.
d. Applications of DFT principles to software, such as the use of software partitioning, embedded assertions for in-line checks, self-verifying objects, and recovery blocks.
e. Research into DFT structures for application to executable specifications.

These new embodiments of DFT are just a few signs that the industry is moving toward a broader scope for the domain of DFT.

1.8.5 Definitions

While a glossary is provided in Section 5 of this document, several key terms and concepts need to be defined at this point so that they may be used in the sections of the document that follow.

Anomalies - refers to the aggregate of physical faults and design flaws. The hierarchy for physical anomalies is defects, which may cause failures, which are modeled as faults, and may manifest themselves as errors in system operation. These apply to both the manufacturing and field environments. They (from defects on) may be solid, transient, or intermittent. The design anomaly hierarchy is flaws which may manifest themselves as errors in system operation. Hence, the error category is where design anomalies and physical anomalies merge. The term "fault model" applied to physical anomalies is correct. The term "design flaw model" is used to handle design flaws.

BIST - The capability for an item to test itself, with minimal or no external test equipment. BIST may be implemented in hardware or software at any level of packaging. In this document, BIST is considered to be synonymous with "BIT" and "self-test." Furthermore, BIST is considered to encompass "diagnostics" when it includes a fault isolation capability. Thus, BIST may provide detection, isolation (diagnostics), and possibly correction (with fault tolerance).

BITE - Built-In Test Equipment - This term is not used in this document, since it traditionally referred to a specific module for performing the test or BIST function. Today's technology dictates that BIST be a much more distributed function, rather than being centralized in a single module.

Correction - The process of removing the cause (maintenance) or effects (fault tolerance) of a design flaw, manufacturing defect or physical fault.

Debugging - A form of testing associated with the detection, isolation, and correction of design flaws only.

Defect - A physical breakdown of an interconnection, such as a foil trace, connector, or cable, or a physical breakdown of a device, such as a transistor, resistor, capacitor, etc.

Design Flaw - A mistake in the design or implementation of a circuit, assembly or software routine which may result in an error in system operation.

Design-for-Testability - The incorporation of a capability in any or all embodiment(s) of a system, from requirements specification to architecture level to detailed hardware and software designs, that will facilitate (for external testing) or assimilate (for BIST) the processes of detecting, isolating, and correcting anomalies (design flaws, manufacturing defects, and field defects), originating from any stage of the system's life cycle. The DFT feature or BIST capability should be usable or invocable at any stage of the life cycle.

Detection - The process of determining the presence of an anomaly, including design flaws, manufacturing defects, and field defects.

Diagnosing - The phase of testing associated with locating the source of the anomaly.

Diagnostics - The fault isolation capability of DFT and BIST. Diagnostics, when implemented, is considered to be an integral part of DFT and BIST and not a separate capability.

Embodiment - One form or instantiation of a system or its parts.

Error - Incorrect behavior of a system, sub-assembly or logic circuit, due to the effects of a design flaw or the propagation of a physical fault through at least one level of gating.

Failure - Incorrect transistor level behavior of a logic circuit, due to the presence of a defect.

Fault - Incorrect gate level behavior, due to the presence of a failure.

Isolation - The process of determining the location of a design flaw or physical fault in a unit under test.

Test and Maintenance Buses - A hierarchy of standardized buses used for communication of test information between test and maintenance controllers. Examples are the IEEE 1149.1 and 1149.5 buses.

Test and Maintenance Controllers - A hierarchy of functions used to control system test and maintenance activities. The functions communicate through a hierarchy of standardized test and maintenance buses.

Test Means - A vehicle used to detect, isolate, and possible correct an item under test. The nature of the vehicle and item under test depends on the life cycle step and level of system hierarchy at which the test is applied. Examples are logic simulation, BIST, and PC-based testers.

Testing - The process of detecting, isolating, and correcting an anomaly arising from any phase of the system's life cycle.

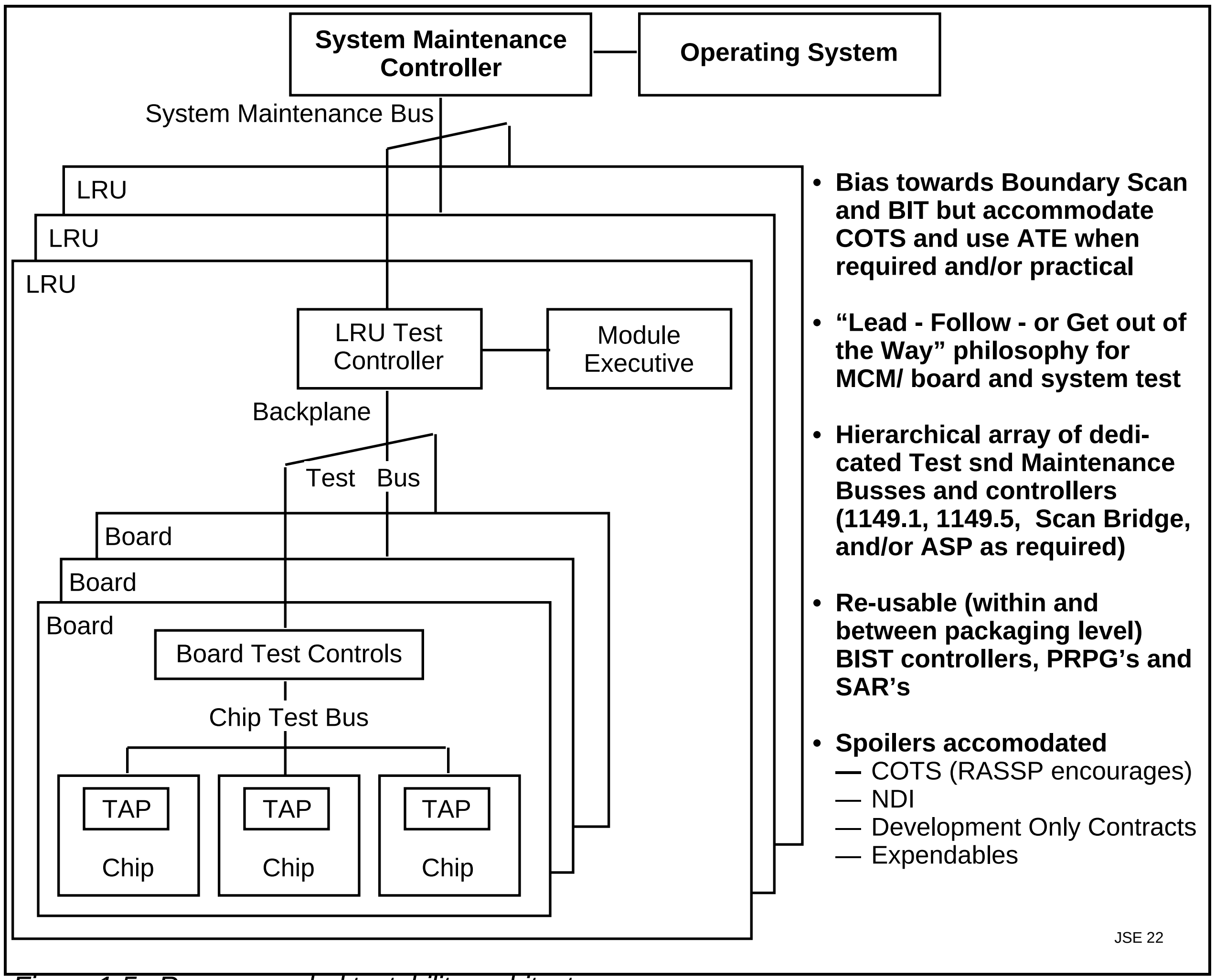
Testability - An attribute of a design at any level of abstraction that reflects the ease with which the item can be tested. Testability is considered poor if any characteristic of the item under test makes it difficult to generate, evaluate, or apply tests. The testability attribute can be predicted, verified, and measured to determine compliance with requirements.

1.9 Overview of the Recommended Testability Architecture

The prescribed RASSP testability architecture (see Figure 1-5) relies heavily on the use of BIST and IEEE 1149.1 boundary-scan incorporated at the IC level. It also relies on the reuse of BIST and boundary-scan tests at all packaging levels from the MCM-level to the system-level. COTS components and/or designs are accommodated by use of the “lead, follow, or get out of the way” philosophy. The “lead” concept suggests insertion or use of a COTS item or re-use item that incorporates BIST or DFT features that can form the basis for a test. The “follow” concept applies to COTS items that may not incorporate BIST or DFT features, but which at least do not interfere with the test (e.g., simply pass the test vectors through to the next stage). Finally, the “get out of the way” concept applies to COTS items which lack testability and BIST, and which must be bypassed during the primary test process and dealt with separately for their own test.

Communication of test data across the packaging hierarchy is accomplished through a system of hierarchical test and maintenance controllers and busses such as IEEE 1149.5.

Further details of the testability architecture can be found in the "RASSP Testability Architecture Description."



- **Bias towards Boundary Scan and BIT but accommodate COTS and use ATE when required and/or practical**
- **“Lead - Follow - or Get out of the Way” philosophy for MCM/ board and system test**
- **Hierarchical array of dedi-cated Test snd Maintenance Busses and controllers (1149.1, 1149.5, Scan Bridge, and/or ASP as required)**
- **Re-usable (within and between packaging level) BIST controllers, PRPG’s and SAR’s**
- **Spoilers accomodated**
  — COTS (RASSP encourages)
  — NDI
  — Development Only Contracts
  — Expendables

Figure 1-5. Recommended testability architecture.

2.0 RASSP GOALS AND DFT METHODOLOGY REQUIREMENTS

The primary goal in the RASSP program is to provide an improvement of at least a factor of four in the time required to conceptualize, design/upgrade, and field signal processor designs, with similar improvements in design quality and life cycle cost. To achieve this goal MMC has developed an overall methodology. The overall RASSP methodology emphasizes the following approaches:

- Concurrent Engineering
- First Pass Success
- Re-use designs and information
- Automation and High level synthesis

The DFT methodology emphasizes these approaches also. In addition, the DFT methodology allows for the possibility of DFT refinement later in the design phase; supports feedback of in-process data from production and field for incorporation into subsequent model years and new designs. Furthermore, it is not driven by existing tools, but uses knowledge of tools to assess the practicality of implementing the methodology.

2.1 Concurrent Engineering

The RASSP methodology emphasizes concurrent design practices between disciplines such as systems, HW and SW design, and test. Concurrent engineering is emphasized by the insertion of a new process step between systems and detailed design—Architecture Definition. This process step brings the differing disciplines together for the up-front trade studies during which 90% of the system cost is determined.

In a concurrent methodology, the expertise of test, manufacturing, support, and reliability engineers is integrated into the design process with a view toward all elements of the product life cycle from conception through disposal, including quality, cost, schedule, and user requirements. Hierarchical DFT analysis and synthesis methodology that is integrated with front end design activities is essential to achieve the goals of concurrent engineering.

2.2 First Pass Success

First pass success of all hardware and software elements is required to eliminate time-consuming and costly design rework cycles. An integrated approach is taken to hierarchical design verification to improve design quality and performance. This process is emphasized by the second part of Architecture Definition. After a preliminary architecture is selected, the architecture is verified by co-simulation of hardware and software configuration items. By this process, risk items are rapidly identified and addressed before detailed design commences.

2.3 Re-use

Maximum reuse of both hardware and software elements is required to achieve 4x decreases in cycle time. Re-use elements include previous designs and Commercial Off the Shelf (COTS) components, boards and systems. The Model Year Architecture is designed to ensure that re-usable elements can be easily incorporated.

Design reuse is a key enabling technology for the model year concept. The DFT methodology should maximize the reuse of all aspects of test related information, such as test requirements; test strategies; DFT/BIST techniques; testable chips, MCMs, boards, and systems; BIST software; and test vectors. It should support the concept of maximum reuse in many dimensions (across packaging levels, across life cycle, etc.). Four dimensions of re-use are defined as shown in Figure 3-2 of Section 3.

2.4 Automation and High Level Synthesis

Task automation and support for high level synthesis is required to help manage design and documentation complexity. The level of design moves up from gates to behavior. Low level design will be accomplished via synthesis and autocode generation tools. Designers working with designs at the behavioral level have decreased insight into the lower levels where test is normally inserted and graded. Hence, the capability to specify, synthesize and grade test features must move up to the behavioral level also.

A system level perspective of test must be taken from the onset. The methodology must support the capture and management of system test requirements. The implementation of the methodology should embody user-friendly tools for assisting designers in selecting a practical global test strategy, and then implementing and tracking that strategy in detail through the various levels of the design hierarchy.

Ideally, test requirements and design requirements can be integrated prior to synthesis; and hence, a testable-by-construction methodology is realized. An alternative approach is to synthesize test features after design synthesis. The RASSP DFT methodology should primarily support the former but provisions should be made to allow the latter, when existing design components without required test features are used, or when tradeoffs, in terms of power, size, and cost, prevent use of DFT features such as full scan.

To control test complexity growth while providing high fault coverage, the DFT techniques employed within the Test Architecture and/or a RASSP design have to satisfy the following characteristics:

– Support Hierarchical Integration - That is, the test capability can be used from ICs to boards to chassis to systems and from design to manufacturing to the field.
– Be Flexible - Support the use of different techniques for different parts of the design to allow for optimization.
– Facilitate and Enforce Re-use - Re-use of all elements of test and DFT. Certain DFT techniques, such as internal scan, boundary-scan, and BIST, are re-usable at higher packaging levels.
– Encourage Interoperability - Support standard test interfaces to enable mixing of components with various test capabilities. The tools used to implement the methodology must be compatible with the rest of the design tools via the use of standards such as VHDL and WAVES to allow for easy integration. The tools must also be easy to use by the designer.

Finally, the number of new test-specific tools required to implement the methodology should be kept to a minimum. Re-use of functional development tools such as requirements tracking, HDL entry, simulators, synthesis and software development should be maximized to the greatest extent possible.

3.0 THE RASSP DFT METHODOLOGY

3.1 Overview of the RASSP DFT Methodology and Its Relationship to the Overall RASSP Methodology

An overview of the RASSP DFT Methodology and its relationship to the overall RASSP Methodology is illustrated in Figure 1-4. It has some important features, as outlined below:

a. It is a methodology embedded in the RASSP methodology and proceeds concurrently. It is deliberately not depicted as a separate process, but rather as an embedded subset of the overall RASSP process.
b. It is driven by test requirements that span the entire life cycle of the system from design to manufacturing to field support.
c. It promotes continuous tracking of compliance to requirements, rather than periodic tracking.
d. It forces attention to reuse in multiple dimensions, as discussed in a later section.

3.1.1 High Level Description of the DFT Methodology

Using Figures 1-4, 3-1a and 3-1b, a high level description of the DFT methodology is presented. The methodology begins with a tangible management commitment. This tangible commitment provides the budget and resources to proceed with the methodology.

System Definition involves test requirements specification. The test requirements come from an integration of customer and derived requirements for the three phases of testing -design, manufacturing, and field support. Specification involves a preliminary, life-cycle cost of test analysis (if such an economics model is available), a test technology assessment, and a design impact analysis to determine the realizability, consistency, and validity of the requirements. Subsequently, during the same step, the three sets of requirements are consolidated into a consolidated requirements specification. The purpose of consolidating the life cycle requirements is to establish a single source of test requirements and, more importantly, to encourage all three test organizations (design, manufacturing, and field) to explore the possibility of a singular test philosophy that can be used throughout the entire life cycle of the system.

The Architecture Definition phase consists of three steps: functional design, architecture selection, and architecture verification. In the functional design step, the consolidated test requirements are used to develop the top level test strategy for the design, manufacturing, and field test phases.

In the architecture selection step, the top level test strategy is used to develop and evaluate various candidate, top level test architectures, by determining which architecture(s) best supports the top level test strategy with the least impact on the candidate functional architectures. Thus, the impact of each test architecture on the candidate functional architectures is assessed and incorporated into the tradeoff and selection process for them. The test requirements, test strategy, and test architecture are then allocated to BIST/DFT hardware and software for one or more of the selected architectures. In addition, candidate DFT and BIST techniques are identified for later implementation, based on the specified requirements. Also in this step, any top level BIST supervisory software development will begin, as will on-line BIST code, since it may have an impact on performance and throughput. Prediction and verification processes begin in this stage as appropriate for compliance tracking.

In the Architecture Verification step, the next level of detail of the selected test architecture(s) are generated and additional details are provided regarding the test architecture impact on the selected functional architecture(s). For example, behavioral and performance simulations will include effects of DFT/BIST techniques, such as the estimated performance degradation due to hardware concurrent fault detection circuits or due to periodic execution of on-line BIST software diagnostics. Prediction and verification processes continue during this stage for compliance tracking.

In the Detailed Design phase, test strategies, architecture, and requirements are flowed down to the detailed design of BIST/DFT hardware and software. The detailed design of the BIST/DFT hardware is performed concurrently and interactively with functional design using automatic or manual insertion and then is reflected into behavioral and structural simulation models, whenever possible. The BIST/DFT detailed design is performed interactively with the functional design, since each impacts the other. Any remaining BIST software (e.g., for power-up or other off-line BIST functions) is implemented. Test vector sets are developed and verified for each packaging level for physical prototype test, production test, and field test. All test vector sets are documented using WAVES. Prediction, verification, and measurement processes are used in this stage for requirements compliance tracking. All test vector sets are documented using WAVES. As the detailed design is verified and debugged, design flaw models are updated to provide more accurate models for the TSDs. Prediction, verification, and measurement processes are used in this stage for requirements compliance tracking.

In the Manufacturing phase, functional and performance testing of the overall prototype is performed to verify compliance to functional and performance requirements, with DFT and BIST hardware and software included. Ongoing production test is performed. Verification and measurement processes are used in this stage for compliance tracking to test requirements. Measurement data is acquired through such techniques as automatic system fault history logging and ATE-based data collection. BIST and tester-based test cost and performance data are captured and encapsulated for the reuse library. Manufacturing defect analysis profiles and distributions are used to update the manufacturing fault model for use in the TSDs.

In the field phase, BIST and DFT capabilities are in use. BIST and DFT functions are also used for lower level (e.g., organizational and depot level) testing. Verification and measurement processes are used in this stage for compliance tracking. As in the manufacturing phase, measurement data is acquired through such techniques as automatic system fault history logging and ATE-based data collection. BIST and tester-based test cost and performance data are captured and encapsulated for the reuse library. Field defect analysis profiles and distributions are used to update the field fault model for use in the TSDs.

Throughout the entire DFT methodology, interfacing is done to the RASSP reuse library to access existing candidates and to add to the library when appropriate. In addition, feedback is being provided continuously from the compliance tracking process back to the responsible persons to assure corrective action is taken. Finally, it is recognized that iterations may be necessary because of the inherent nature of the RASSP methodology.

3.1.2 Relationship of Test Requirements, Test Strategies, and Test Architectures

Figure 3-1a illustrates the relationship between test requirements, test strategies, and test architectures. Figure 3-1b shows test requirements are flowed down the packaging hierarchy and then compliance is verified as strategies and architectures are implemented. Test requirements are requirements specifying the test fault coverage, test time, and other test parameters associated with testing the Unit Under Test (UUT), which may be a chip, MCM, board, or system. Test requirements do not include specification of test means, but instead, are allocated to various test means during the development of "test strategies."

In this methodology, requirements are checked for realizability, consistency, and validity when they are received, generated or specified. This process minimizes the problem where an invalid requirement (such as 99% single stuck-at gate level fault detection for a design with 100% COTS components) flows all the way down to the board level before it is found to be impossible to meet. A test requirement is considered realizable if it can be achieved, given current test technology, and system constraints (size, weight, power, degree of COTS, etc.). A test requirement is consistent, if it does not contradict any other test requirements and is derivable from the higher level requirement from which it flowed. Finally, a test requirement is valid if there is currently a practical and cost effective means to predict, verify, and measure compliance to the requirement. If there is a question about the realizability, consistency, and validity of a test requirement, a mini-spiral can be initiated to further analyze the requirement. A requirement is amended or deleted if it is not consistent, realizable, and valid.

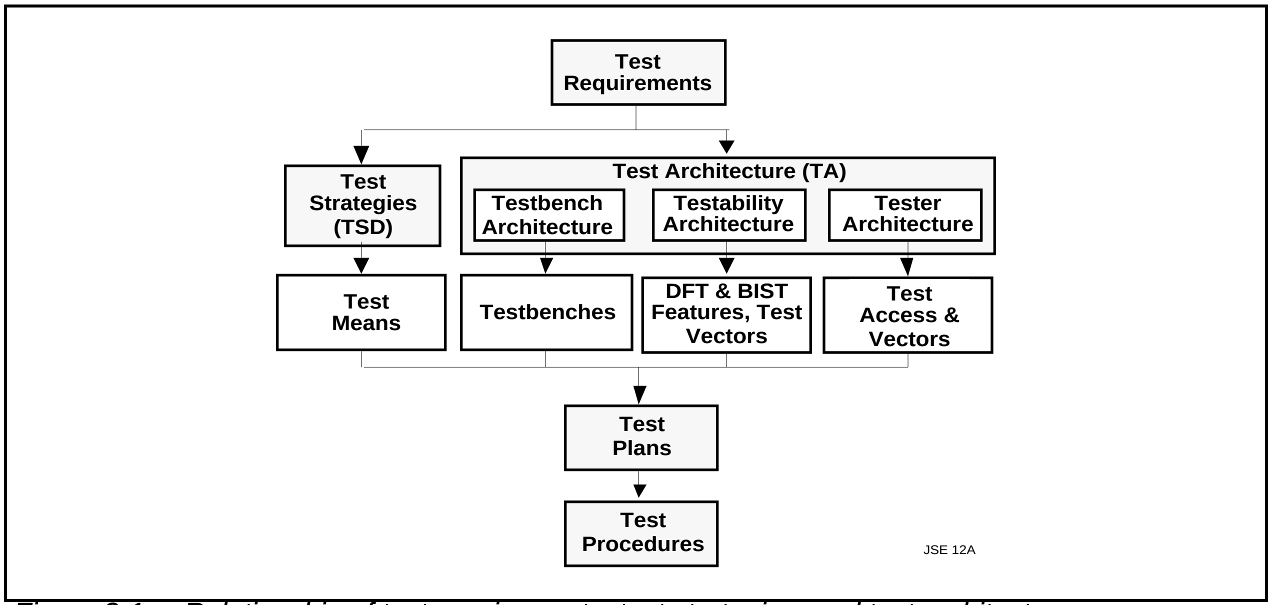


Figure 3-1a. Relationship of test requirements, test strategies, and test architectures.

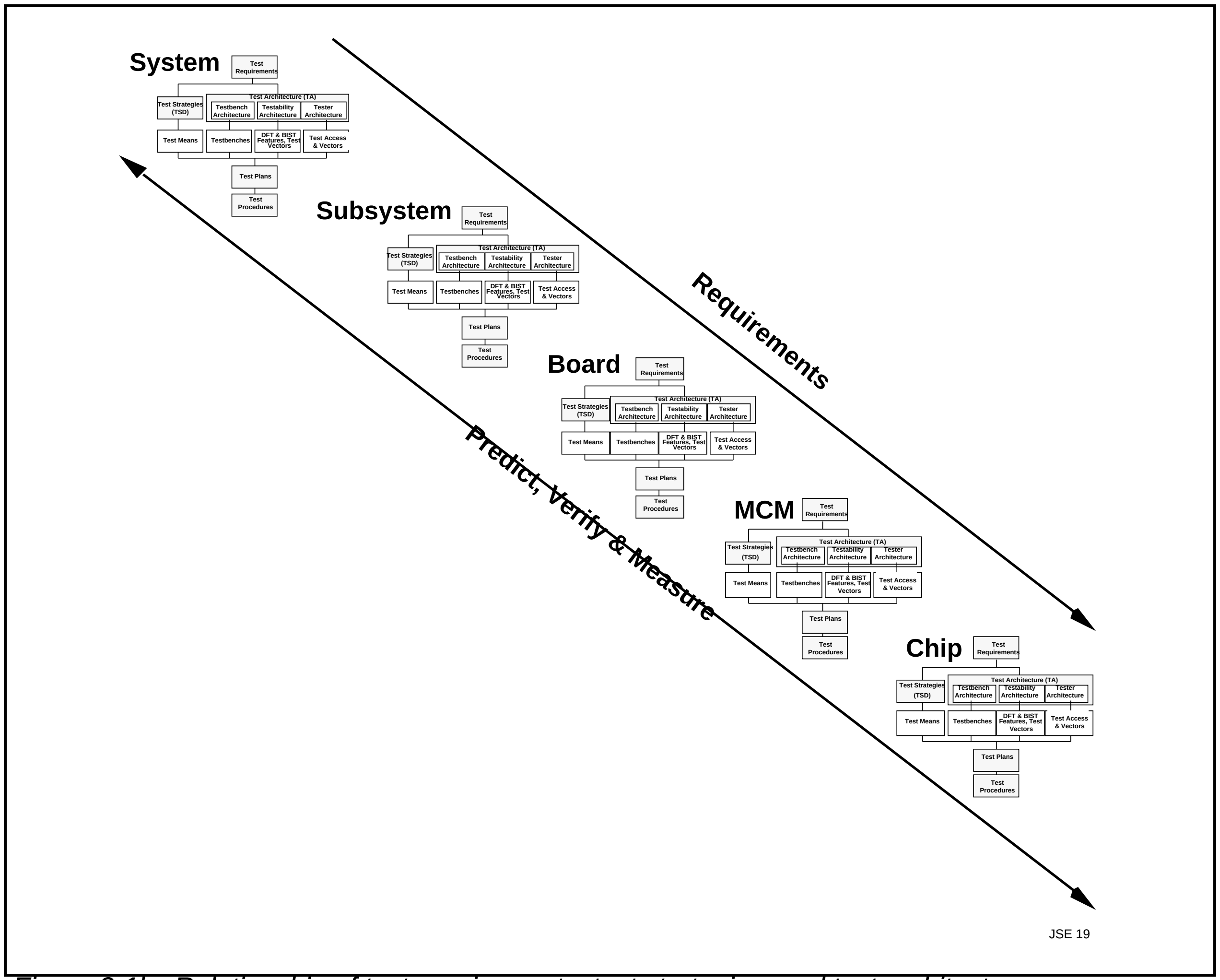


Figure 3-1b. Relationship of test requirements, test strategies, and test architectures across packaging hierarchy.

A test strategy is developed for each life cycle phase of testing (design, manufacturing, and field) and defines the mix of test means by which the UUT will be tested. Test means can be built-in self-test (hardware and/or software based), test equipment, and manual procedures. During the development of the test strategy, test requirements drive the selection and mix of test means and ultimately are allocated to each of the selected test means.

Test strategies, which are specified and managed by the use of “test strategy diagrams” described below, drive the development of test plans and test procedures at each level of the packaging hierarchy. Test plans describe a high level view of testing the UUT, including budget, schedule, resources, capital equipment, and a description of the test flow. Test procedures give the detailed, step by step procedures for testing the UUT. As with test requirements, test strategies are checked for realizability, consistency, and validity when they are generated and documented, before they are flowed down to the next lower level of abstraction.

A test architecture is a suite consisting of the UUT and any test equipment required, based on the developed test strategy. The test architecture consists of the “testbench architecture” which describes the simulation-based testbench for design-verification; the "testability architecture" which describes the DFT and BIST features in the UUT; and the "tester architecture" which describes the configuration of “test equipment” required for any externally based testing. During the development of the testability architecture, the impact on the functional design, such as real estate, I/O pins, performance, power, and reliability, is constantly evaluated to ensure it is acceptable. Test architectures are checked for realizability, consistency, and validity when they are generated and documented, before they are flowed down to the next level of abstraction (as was done with the test requirements and strategies). Note, certain types of testability architectures, such as those based on BIST and boundary-scan, are more robust than others, in that they are applicable across the packaging hierarchy, as well as across all life cycle testing phases.

As the system is decomposed into its constituent elements, test requirements, test strategies, and test architectures are “flowed-down” to each level of hardware packaging and software abstraction level. At each level, compliance to the higher level test requirements, strategies, and architectures is checked. If at a lower level, it is impossible to adhere to the requirements, strategy, or architecture, the previous level or levels are re-examined and analyzed to correct the problems in those higher level requirements, strategy, or architecture. Note, the amount of backtracking required should be minimized and problems should be identified earlier than in normal methodologies, since the strategy and architecture are driven by the requirements and all three are checked for realizability, consistency, and validity when they are generated initially at the front end of the process and also when captured at each successive level.

3.1.3 Supporting Concepts

3.1.3.1 Metrics

"Metrics" are quantitative parameters used for two purposes in the methodology:

a. Requirements specification and compliance tracking
b. Tradeoff analysis and selection

Metrics can be categorized into four major categories: Performance, System Impact, Functional Process Impact and Test Process Impact. A list of metrics for each category follows.

a. Performance (Specified Per Life Cycle Test Phase - Design, Manufacturing, and Field)

    1. Controllability and observability
    2. Fault detection coverage (w/assumed fault model)
    3. Fault isolation coverage (w/assumed fault model)
    4. Error correction coverage (w/assumed fault model)
    5. Average, minimum, and maximum ambiguity group size
    6. Average, minimum, and maximum position of replaceable item
    7. Test pattern set size
    8. Detection time
    9. Isolation time
    10. Correction time
    11. False alarm rate

b. System Impact

    1. Area overhead
    2. Performance impact
    3. I/O overhead
    4. Power overhead
    5. Reliability impact
    6. Maintainability impact
    7. Availability impact

c. Functional Process Impact

    1.1 - 1.n. Impact on time for each functional process step.
    2.1 - 2.n. Impact on cost for each functional process step.

d. Test Process Impact

    1.1 - 1.n. Impact on time for each test process step.
    2.1 - 2.n. Impact on cost for each test process step.

Performance metrics are specified for each life cycle test phase, since the values of the metrics, as well as the associated "fault models" change from design to manufacturing to the field. Some of the metrics can only be applied and/or predicted, verified, and measured at specific steps of the process. This distinction is described in the TMAT table in Appendix A (Table A-1).

For a given model year or system, all of the above metrics could be used for requirements, and none left for tradeoffs. In practice, such an approach may be impractical to manage and/or may be too constraining. In most projects, the performance category above would be used for requirements and the other three categories of metrics would be used for tradeoffs.

As noted in the discussion of requirements in Section 3.1.1 above: a metric cannot be used as a requirement unless compliance can be predicted, verified, and measured. Before ruling out a metric, however, consideration should be given to practical and cost effective means of estimating compliance. For example, if a requirement for deterministic pattern fault detection coverage is specified, it would normally be difficult to determine compliance prior to test pattern generation. However, the use of pseudorandom patterns as a stimulus, graded by a probabilistic fault grader, may be an acceptable prediction prior to performing full fault simulation on a deterministic test pattern set. Note by performing such a procedure, testability information is made available early in the process.

3.1.3.2 The Concept of the Test Metrics/Tool Application Table (TMAT) (Table A-1 in Appendix A)

In the DFT Methodology, the choice of test related metrics, as well as the tools used to predict, validate, and measure values for those metrics, is highly dependent on several factors:

a. The step in the process
b. The level of packaging or abstraction
c. The normal project constraints (schedule, budget, resources)

Since there is unlikely to be a single combination of metric and tool that would apply to all cases of the above factors, for a given step in the Methodology, a selection-based approach is taken, using what is called the Test Metrics/Tool Application Table (TMAT) (see Table A-1). The actual matrix, which a preliminary version appears in the Appendix, provides the preferred metric/tool pair for each process step entered in the methodology. The metrics all fall in the basic categories described in Section 3.1.3.1. Once a process step is entered, the requirements captured, and the test strategy and architecture developed, the TMAT is accessed and the proper metric/tool pairs selected for each prediction, verification, or measurement activity in that process step. As new tools (or even new metrics) emerge, the table can be updated. As the metric/tool pairs are used in the methodology and their cost effectiveness and accuracy are observed, the table can be updated relative to the preferred status of each pair. Hence, the TMAT should be tailored and maintained by a given organization based upon their toolset and class of products being designed.

3.1.3.3 Reuse Concept

One of the key ways in which the DFT Methodology contributes to the achievement of the RASSP goals is through the enforcement of reuse of DFT in four different dimensions. As pictured in Figure 3-2, the four dimensions of reuse are as follows:

a. Across the life cycle phases within a given model year. From system definition to field support, each DFT/BIST embodiment or instantiation is designed for maximum reuse in the phases to follow. Specifics of enforcement are discussed below in the discussion of the test strategy diagram.
b. Across the packaging hierarchy within a given model year. From chip to system, each DFT/BIST embodiment or instantiation is designed for maximum reuse in testing the package levels above and below it. Specifics of enforcement are discussed below (test strategy diagram and domain analysis).
c. Across a single packaging level within a given model year. Each DFT/BIST embodiment or instantiation is designed for maximum reuse in testing other entities within the same package level. For example, a board level BIST technique is examined to see if it can be used for testing other boards in the system.
d. As new model years unfold, the RASSP reuse library is used to provide the outputs of some steps in the process (rather than perform the step from scratch), and outputs of each process step become candidates for encapsulation in the reuse library for future model years or systems. Test related reuse items include, for example, test requirements; test strategies; DFT/BIST techniques for certain logic structures; testable chips, MCMs, etc.; BIST software modules; and test vector sets for certain library elements.

Examples of the types of information that would be useful to store are as listed below. Some of these items (a & e) could be encapsulated in the reuse library and others could be captured in a supplemental test information database.

a. Test item (i.e., test requirements, test strategy diagram, test architecture, test plan, test procedure, POST control program, BIST SW primitive, BISTed ASIC, BISTed SRAM cell, BIST controller, TAP controller, boundary scan register, scannable flip flop, etc.)
b. Description and/or documentation of an item
c. Other functional items that must be interfaced to (i.e., DSP, memory, I/O, etc.)
d. Other DFT/BIST items that must be interfaced to (See a. above)
e. Specification format and language (i.e., template)
f. Typical or specific values of metrics (See metrics section)
g. Applicable test phases (for a given test item, technique, etc.)
h. Applicable test modes (i.e., POST, on-line BIST, etc.)
i. Applicable process step
j. Applicable hardware or software structures (in which to embed item)
k. Applicable hardware or software structures (to apply item to)
l. Compatibility with interface standards
m. Compatibility with types of BIST or test equipmet

n. Tool environment and specific tool required for:
   Implementation
   Requirements Prediction
   Requirements Verification
   Requirements Measurement
o. Case histories (model year(s) or system(s) where used)
p. Literature references

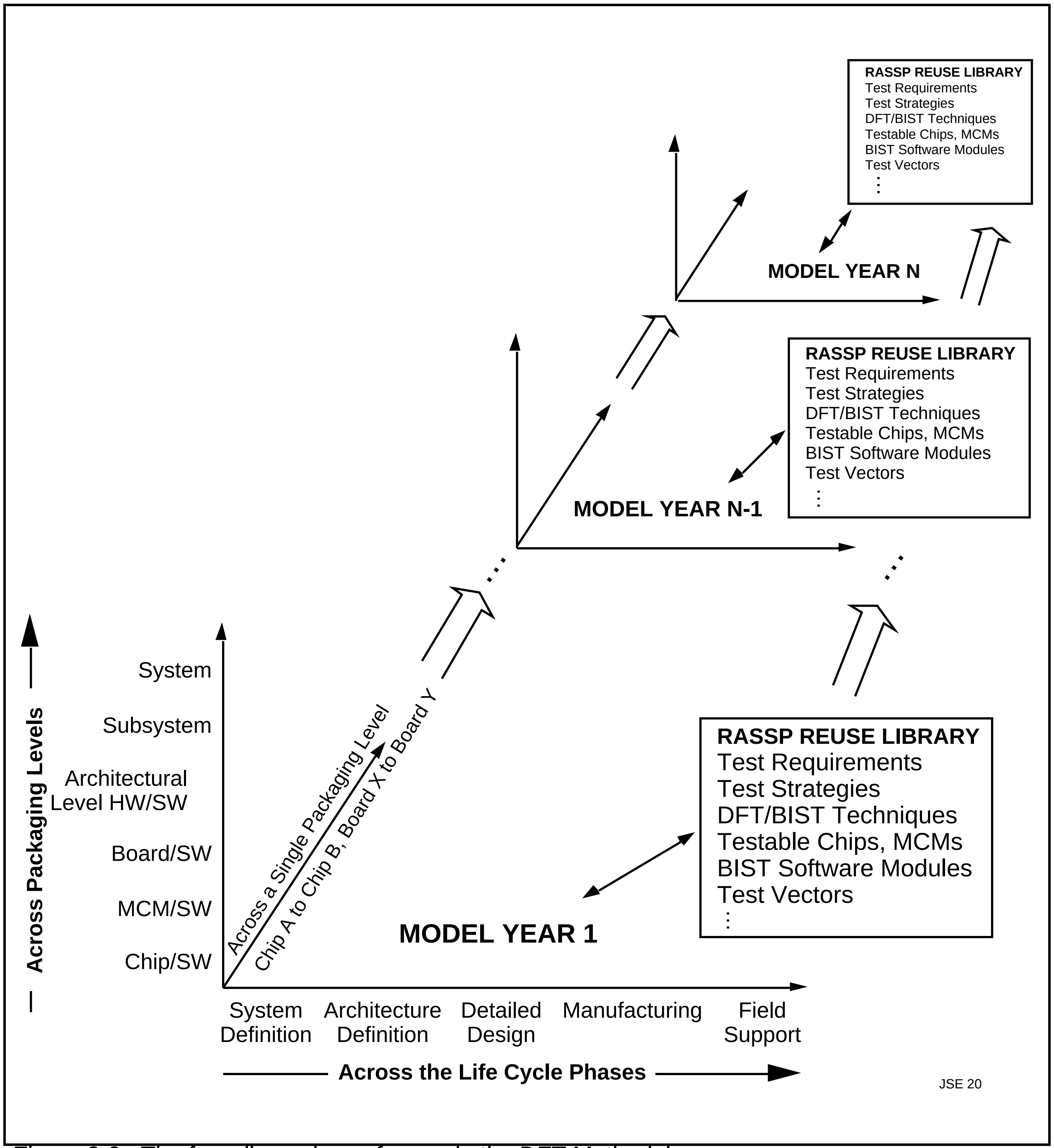


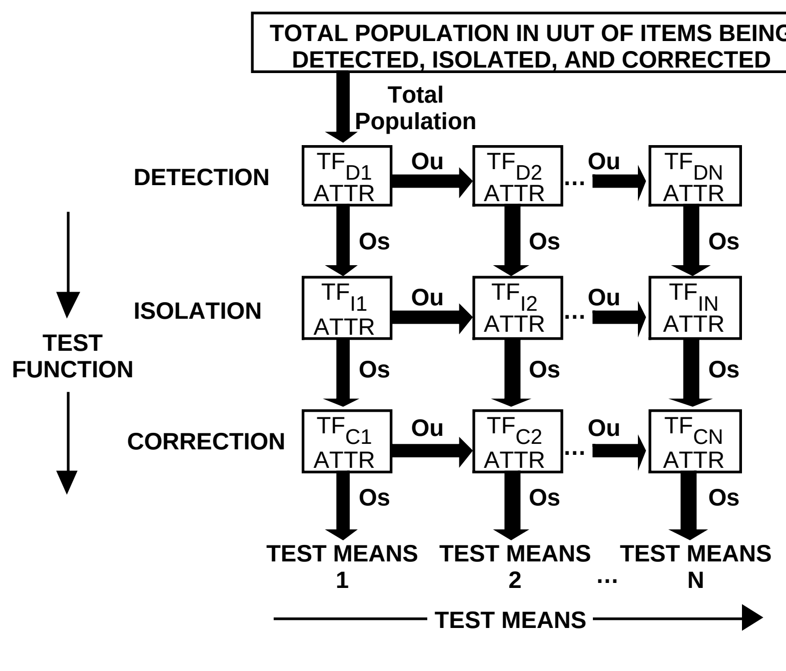
Figure 3-2. The four dimensions of reuse in the DFT Methodology.

3.1.3.4 Test Strategy Diagrams

The test strategy diagram (TSD) is a key construct used in the DFT Methodology to bridge from requirements to implementation, manufacturing, and field; to "knit" all processes together; to provide a means of carrying information between and within steps of the process; and to manage the requirements specification and compliance tracking. The Test Strategy Diagram is used to allocate a given "fault population" onto "test means" for detection, isolation and correction.

A "test means" is a vehicle used to detect, isolate, and possibly correct an item under test. The nature of the vehicle and item under test depends on the life cycle step and level of system hierarchy at which the test is applied. For example, during the architecture definition step, the "vehicle" could be specified in general terms such as peer reviews, simulation, ATE, and BIST, while the "item under test" might be specified as requirements specifications, behavioral model, boards, and chips, respectively. On the other hand, in the detailed design stage, the vehicle might be DSP ASIC team review, structural VHDL simulation, PC-based boundary-scan tester, and circular BIST respectively, while the item under test might be DSP ASIC flowed down manufacturing test requirements, DSP ASIC structural VHDL model, signal processing board prototype, and DSP ASIC in manufacturing test.

The anatomy of a Test Strategy Diagram is shown in Figure 3-3. It consists of a three by "n" array of cells, comprised of one row for each of the three test functions, detection, isolation, and correction. The n columns correspond to n test means, such as BIST, ATE, or manual procedures. *Test means are listed in priority order from left to right*, starting with the means that has the highest fault detection coverage (or lowest cost) to the means on the right that has the lowest fault detection coverage (or highest cost). The input to the array is the total population of items or anomalies being tested for, and that total population is being operated on by the array. Each cell in the array can be thought of as a flow graph, having one input and one or two outputs.



DEFINITIONS

ATTR	Attributes being evaluated (e.g., cost, time, quality, etc.)
Os	Output successful (e.g., detected quantity)
Ou	Output unsuccessful (e.g., undetected quantity)
TEST MEANS	Means of performing test (e.g., types of BIST, types of test equipment, manual procedures, etc.)
TF	Transfer function (e.g., fault coverage percent)

JSE 13

Figure 3-3. Anatomy of a test strategy diagram.

Inside each cell is a transfer function or operator that acts on the input to produce the output(s). The most common transfer function in the TSD examples in this document is the coverage operator, which expresses the degree of detection, isolation, and correction coverage. It is stored in the form a decimal value, displayed as a percentage and it is multiplied times the quantity coming into the cell. Note, the transfer functions of test means are position dependent. That is the transfer functions in the nth column operate on the residual population from the previous (n-1) columns of items or anomalies being tested for.

The implied implementation of the TSD, as shown in the figures in this document, makes use of a three dimensional spreadsheet paradigm, since it is a tool that is available in most companies. However, it is certainly possible that a custom TSD tool could be developed in a compiled high level language (e.g., C) and provide much faster operation, particularly for large systems, as well as provide a more friendly graphical user interface. If the TSD concept were implemented in a three dimensional spread sheet, the three dimensions would be test function (D/I/C), test means (the strategy), and conformance tracking. If a modern spreadsheet such as MS Excel were used books would be used to track conformance across the packaging hierarchy.

In addition to the transfer function, one or more attributes are associated with the cell. Examples of important attributes are test time and cost , real estate penalty, quality levels, etc. Additional ones may be used as appropriate. The values of the attributes are determined a priori and stored in the cell. For example, a test time stored in the upper left cell in Figure 3-3 could represent the time for test means #1 to detect a fault. A cost factor could include a one time (non-recurring) cost for test generation and test evaluation, plus a recurring cost for test application that would be incurred for each unit. The attribute normally is stored as a worst case value, although average values could be used as appropriate.

Early in the methodology process, initial values for transfer functions would be requirements, while the attributes (which might also be requirement values) and the anomaly population density can be obtained from historical data accumulated in the reuse library (described above), can be generated from new estimates (based on similar systems), or can make use of a variable to take the place of a fixed value. As the methodology continues, actual values for transfer functions, anomaly population densities, and attributes can be entered into the appropriate TSDs through the prediction, verification, and measurement processes.

When a TSD cell is "activated," its attribute is factored into (usually by simple addition) the accumulated value of that attribute through each path in the array intersecting with the cell. Thus, test time and cost factors can be summed across test means and test functions. The test cost information can be used for such applications as performing test strategy tradeoffs, manufacturing cost budgeting, or maintenance and logistics planning. Test time information could be used, for example, for input to a fault-tolerant system's availability model.

Each cell in the array has one or two outputs. The quantity exiting from the bottom of the cell represents the "successful" quantity. For example, if the transfer function is fault coverage, the quantity exiting from the bottom of the top left cell in Figure 3-3 represents the detected quantity of faults. The quantity exiting horizontally represents the "unsuccessful" quantity, which in the example cited above would be the undetected quantity of faults. The quantity of unsuccessfully handled faults can be used, for example, as input to a manufacturing defect (quality) level analysis.

The cells on the right side of Figure 3-3 are known as "terminating cells," because they represent the last possible test means to achieve the detection, isolation, and correction. For that reason, such cells have only one output, which is the one associated with the successful quantity. In the case that a TSD lacks a terminating cell in the detection, isolation, or correction functions, the implication is that that fault quantity must be handled during testing at the next higher level of package testing. Therefore, fault quantities emanating from the right side of the TSD must be included in the fault population for the next higher level of package, along with the new fault population associated with that next higher level package. Thus, the TSD concept forces consideration of all possible scenarios of fault detection, isolation, and correction. If 100% of the faults must be detected, isolated, and corrected by some combination of BIST, test equipment, and manual procedures, such as the case in the Integrated Diagnostic concept, the sum of all faults entering the top of the systems TSDs must equal the sum of the outputs exiting at the bottom of all TSDs across all packaging levels.

The TSD has many applications within the DFT Methodology. For example, its application to managing test requirements and associated compliance tracking information is depicted in Figure 3-4. A TSD would be generated to store the system level consolidated test requirements discussed in Section 3.1 above (shown on the bottom left of the figure). The system level requirements TSD can be used to flowdown the requirements to lower levels of the hierarchy, thus producing a requirements TSD at each packaging level. As the prediction, verification, and measurement data are generated during compliance tracking across the life cycle, a TSD is generated for each set of data (P, V, and M) at each packaging level, and automatically is compared against the requirements TSD values to flag compliance violations. A comparison should also be done between the P-data, V-data and M-data, to assess continuously the integrity of the prediction, verification, and measurement methods and tools.

The application of the TSD along the life cycle dimension in Figure 3-4 can be thought of as providing traceability and reuse enforcement. This is accomplished by prohibiting the "collapsing" of the TSD spreadsheet as the life cycle steps are traversed. For example, the establishment of boundary-scan based PC testing during the design phase would automatically be carried forward into the TSDs for manufacturing and the field to force at least an analysis of the potential reuse of the test means in those stages.

The application of the TSD along the vertical (packaging hierarchy) dimension can be considered as providing requirements allocation and reuse enforcement as well. In regard to the latter application, the reuse enforcement is accomplished in a similar way as with the life cycle reuse enforcement: by prohibiting the "collapsing" of the spreadsheets when integrating TSD values up the hierarchy. In this methodology, it is required that a test means be carried up to at least one higher level of packaging for analysis. For example, the use of BIST and boundary-scan in a chip as a chip level test means would appear not only in the chip level TSD for that chip, but those specific test means would reappear in the TSDs of at least the MCM or board in which that chip resides. If that chip becomes the basis of a board level BIST capability, that board level BIST must be considered a test means for the system level TSD. This philosophy forces the evaluation of the contribution of low level test means (especially BIST) at the higher packaging levels.

The application of the TSDs in the diagonal dimension in Figure 3-4 (R, P, V, M) represents the means for managing compliance tracking information. TSDs with dashed or dotted lines represent those that are preliminary or for which only partial information would be available in that life cycle phase.

Figure 3-5 provides an example of a TSD for trading off, analyzing, and finally specifying the field test requirements for a system having both BIST and fault tolerance requirements. This actual example from a fault-tolerant avionics system assumed a fault model consisting of single stuck-at faults as well as single transient or intermittent faults. In this example, failure rate is used as an estimate of relative fault population densities and is shown at the top of the diagram. It is important to note that this value was not derived from MIL-STD 217 MTBF predictions. It was derived from a reliability model that assumed only the target faults defined in the fault model. The total failure rate was allocated between the two types of faults based on some field failure studies that showed similar systems in similar environments tended to experience four times as many transient or intermittent failures as solid (stuck-at) failures.

Each failure rate total is entered into the top of the respective cell arrays. For both types of faults, the detection, isolation, and correction requirements were entered into the cells for each of the three allowed test means: BIST/fault tolerance, test equipment, and manual procedures. The calculations were performed using test coverage and an attribute of test time. Fault coverage numbers are multiplied times the failure rate number to indicate what percentage of the fault

See power point Fig. 3-4

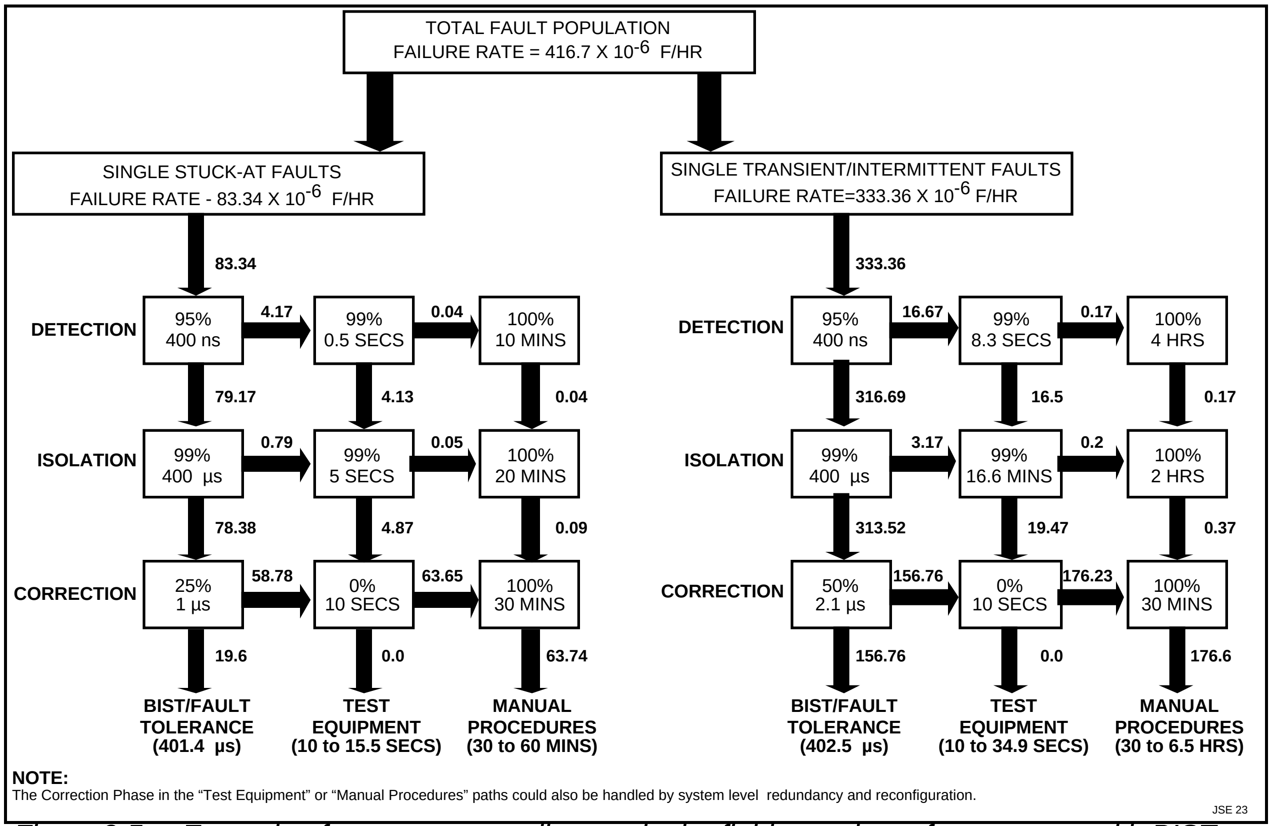

TOTAL FAULT POPULATION
FAILURE RATE = 416.7 X 10-6 F/HR
SINGLE STUCK-AT FAULTS
FAILURE RATE - 83.34 X 10-6 F/HR
SINGLE TRANSIENT/INTERMITTENT FAULTS
FAILURE RATE=333.36 X 10-6 F/HR
83.34
333.36
DETECTION
95% 400 ns
4.17
99% 0.5 SECS
0.04
100% 10 MINS
79.17
4.13
0.04
ISOLATION
99% 400 µs
0.79
99% 5 SECS
0.05
100% 20 MINS
78.38
4.87
0.09
CORRECTION
25% 1 µs
58.78
0% 10 SECS
63.65
100% 30 MINS
19.6
0.0
63.74
BIST/FAULT TOLERANCE (401.4 µs)
TEST EQUIPMENT (10 to 15.5 SECS)
MANUAL PROCEDURES (30 to 60 MINS)
DETECTION
95% 400 ns
16.67
99% 8.3 SECS
0.17
100% 4 HRS
316.69
16.5
0.17
ISOLATION
99% 400 µs
3.17
99% 16.6 MINS
0.2
100% 2 HRS
313.52
19.47
0.37
CORRECTION
50% 2.1 µs
156.76
0% 10 SECS
176.23
100% 30 MINS
156.76
0.0
176.6
BIST/FAULT TOLERANCE (402.5 µs)
TEST EQUIPMENT (10 to 34.9 SECS)
MANUAL PROCEDURES (30 to 6.5 HRS)
NOTE:
The Correction Phase in the "Test Equipment" or "Manual Procedures" paths could also be handled by system level redundancy and reconfiguration.
JSE 23

Figure 3-5. Example of a test strategy diagram in the field test phase for a system with BIST and fault tolerance (requirements version).

population is successfully detected, isolated, and corrected by each of the test means. The test time attribute is activated when the transfer function is applied to the fault population entering a given TSD cell. The TSD allows for the analysis of each scenario of handling faults by any combination of the three allowed test means, providing the possible values of total test time through each scenario path (shown exiting at the bottom of each array column). At a later date, a cost figure was associated with the test time, and a cost of test analysis was performed to provide another dimension of tradeoffs to further refine the requirements. Eventually, after many "what-if scenarios" using the TSD, the requirements as shown in Figure 3-5 were finalized.

3.1.3.5 Board Design Example Used For Illustration In The Methodology Discussions

Figure 3-6 shows a hypothetical board design that will be used to illustrate examples of several concepts in the DFT methodology. Although it is hypothetical, it does represent a reasonable set of functions sometimes used in the digital sections of communications systems. There is a transmit path consisting of an incoming 32 bit parallel bus going to some receivers and a 32 bit register consisting of four octal register chips. The 32 bit register feeds a memory bus, whose read data bus in turn feeds some data path logic that ultimately produces a high speed bit serial data path that exits the board. The receive path is identical except, its signal path is in the opposite direction. It is important to note that the two SRAM arrays (transmit and receive) are separate entities with separate buses. Dedicated DSPs handle signal processing functions for the receive and transmit paths.

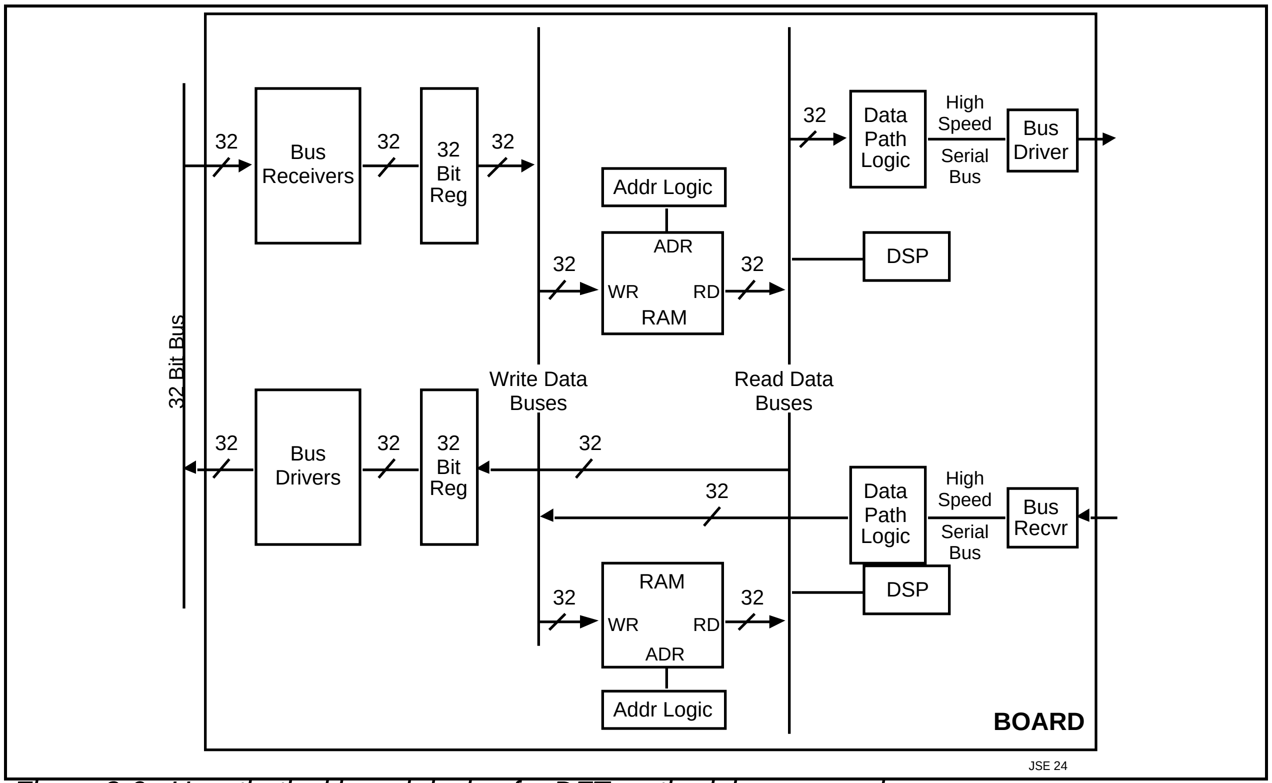


Figure 3-6. Hypothetical board design for DFT methodology examples.

3.2 Details of the RASSP DFT Methodology

Figure B-1a (see Appendix B) depicts the overall process. As shown in Figure B-1a, the process is comprised of five processes:

- System Definition Process
- Architecture Definition Process
- Detailed Design Process
- Manufacturing Process
- Field Support Process

Figure B-1b depicts the subprocess steps within architecture definition (functional design, architecture selection, and architecture verification).

Figure 3-7 shows the development of key data items as the methodology progresses through the process steps.

Descriptions of each process step are included below. The description of each process step is comprised of:

1. Inputs
2. Process
3. Outputs
4. Handling of COTS
5. Risk management/Mini-Spirals
6. Interface to Reuse Library
7. Interface to Overall RASSP Methodology and Tools

Each section begins with a top level description of the activities being performed concurrently in the development of the overall processor system (i.e., normal functional modes or non-test related). A subprocess chart with more detail is included with each section.

	System Definition	Functional Design	Arch. Selection	Arch. Verif.	Detailed Design	Mfg. Integration and Test	Field Support
Requirements	Consolid.; Checked for valid, consistent, and reliable	Implication; Modes and Isolation specs	Flow down to board level	Flow down to DSPs, ASICs and Logic Blocks	Verify and prel. measure	Measure, Update and Feedback	Measure, Update and Feedback
Fault Models	Preliminary	Refined	Refined	Completed	Updates and Feedback	Updates and Feedback	Updates and Feedback
Test Benches	VP0	(prel) VP1	VP1	VP2	VP3	-	-
Test Strategies (TSD)	TSD0	(prel) TSD1	TSD1	TSD2	TSD3	TSD4	TSD5
Test Architecture (TA)	TA0	(prel) TA1	TA1	TA2	TA3	-	-
Test and Diagnostic Plans	System Accept. Test	Subsystem Accept. Test	Prel. Board Accept. Test	Board and Special Chip (i.e. bare die, ASICs)	Chip to System Mfg. and Field	Updates and Feedback	Updates and Feedback
Test and Diagnostic Vectors	System Accept. Test	Subsystem Accept. Test	Prel. Board Accept. Test	Prel. Design Verification	Des. Ver., Prod. and Field	Updates and Feedback	Updates and Feedback
Test and Diagnostic Procedures	-	-	Prel. System and Subsystem Accept. Test	System and Subsystem Accept. Test	Chip to System Mfg. and Field	Updates and Feedback	Updates and Feedback
BIST HW/SW Allocation	-	1st Pass	Preliminary	Completed	Updates and Feedback	-	-
BIST and DFT HW	-	-	High Level Models	Behavioral Models	Structural Models; Synthesis; Implement	Updates and Feedback	Updates and Feedback
BIST SW CP and DFGs	-	On-LIne BIST	On-LIne BIST Target Dependent	On-LIne BIST Equivalent DFGs	Post , On-Line and Off-Line BIST, Self Diagnostics	Updates and Feedback	Updates and Feedback
BIST SW Primitives	As Required	As Required	As Required	As Required	As Required	Updates and Feedback	Updates and Feedback

Figure 3-7. Key data and development items correlated with process step.

3.2.1 System Definition Step

Requirements are derived from customer and/or parent system requirements and maintained during the system definition process. All of the functions including test functions such as BIST are partitioned into processor system and subsystem functions. An overall model of the processor system is developed which is referred to as VP0. This model together with testbenches represents the inputs, outputs and transformations of the inputs by the processor system including latency. Key outputs of the DFT efforts are the preliminary test strategy diagram, TSD0, and testability architecture, TA0. Figure 3-8 shows the DFT steps which occur during the system definition step. The key step, requirements analysis, is shown in more detail in Figure 3-9.

3.2.1.1 Inputs

The inputs to this process step are as follows:

3.2.1.1.1 Inputs from Overall RASSP Processes

iRa.	RASSP Reuse library	Step 2, 5, 10 and 11
iRb.	Information on project commitment to DFT (resources, tools, schedule, budget, etc.)	Step 3
iRc.	Customer requirements for design, manufacturing, and field test; maintenance requirements; quality requirements; availability, reliability, and maintainability requirements	Step 4
iRd.	Historical hardware and software design flaw statistics for previous model years or systems	Step 5
iRe.	Manufacturing defect profiles and statistics for previous model years or systems	Step 5
iRf.	Field failure profiles and statistics for previous model years or systems	Step 5
iRg.	System constraints (size, weight, power, etc.)	Step 6
iRh.	High level system functional description	Step 10
iRi.	Emerging partitioning of system into subsystems	Step 10

3.2.1.1.2 Inputs from DFT Processes

iDa.	The TMAT Table	Step 1
iDb	Testability Architecture Description	Step 10

3.2.1.2 Process

The following activities take place in this process step:

1. Using iDa, access the Test Metrics/Tool Application Table and select metrics/tool pairs for use during this step. Maintain and update these metrics throughout the process. Capture and review final values before exiting the process.

2. Using iRa, check the reuse library for candidate reuse elements for this step (i.e., requirements, fault models, TSD0, TA0, VP0 T&M Controller and Bus models) .

3. Using iRb, assess the presence of sufficient, tangible management commitment to the DFT process (resources, tools, capital equipment, schedule, budget, etc.).

4. Using iRc, capture customer test requirements. Expand and refine the customer test requirements specification, using the template shown in Figure 3-10. This template forces the contractor to remove the ambiguity from the spec and resolve all open issues up front. Negotiations and discussions with the customer would usually be necessary.

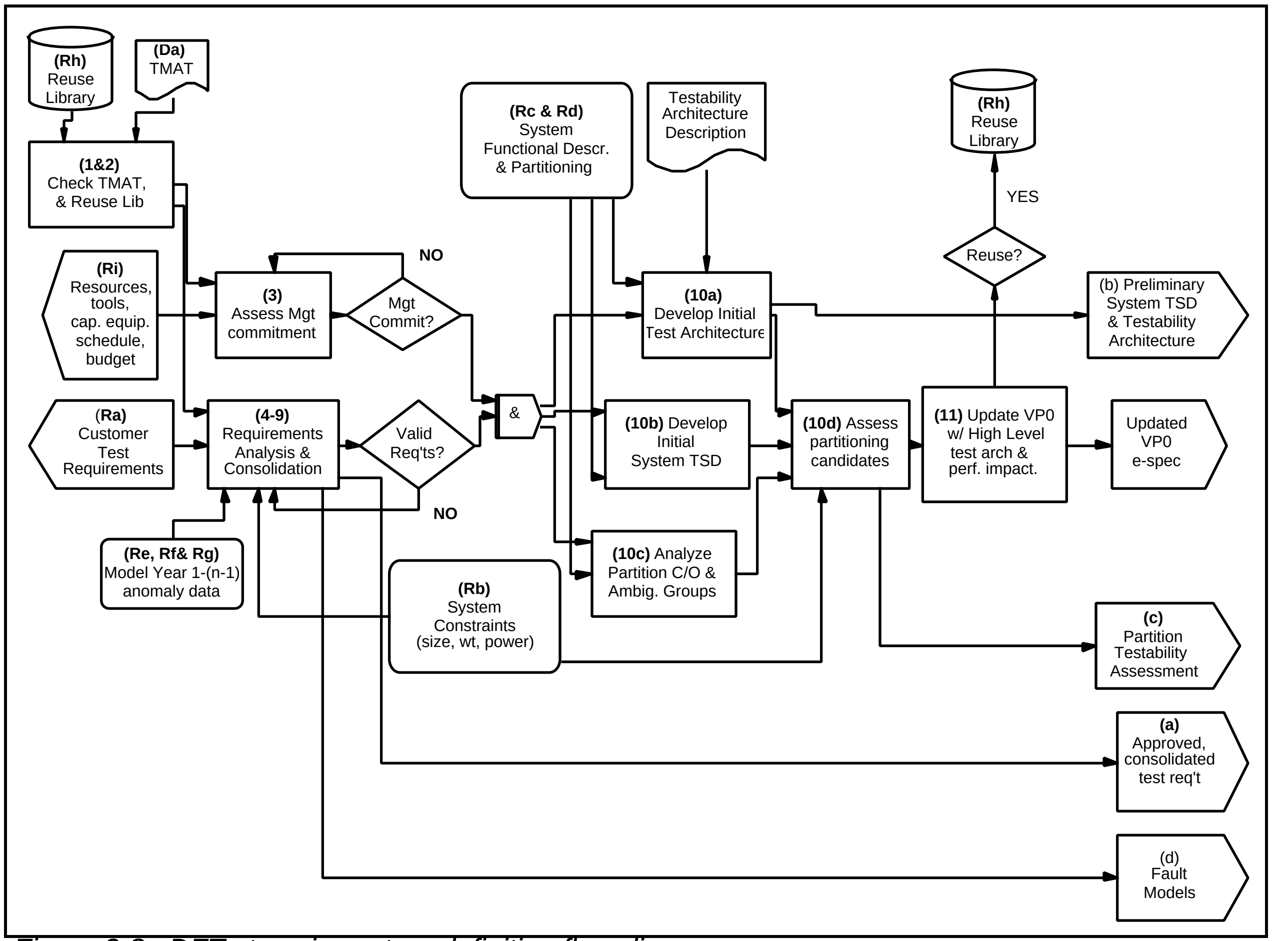

(Rh) Reuse Library
(Da) TMAT
(1&2) Check TMAT, & Reuse Lib
(Rc & Rd) System Functional Descr. & Partitioning
Testability Architecture Description
(Rh) Reuse Library
YES
Reuse?
NO
(Ri) Resources, tools, cap. equip. schedule, budget
(3) Assess Mgt commitment
Mgt Commit?
(10a) Develop Initial Test Architecture
(b) Preliminary System TSD & Testability Architecture
(Ra) Customer Test Requirements
(4-9) Requirements Analysis & Consolidation
Valid Req'ts?
&
(10b) Develop Initial System TSD
(10d) Assess partitioning candidates
(11) Update VP0 w/ High Level test arch & perf. impact.
Updated VP0 e-spec
NO
(Re, Rf& Rg) Model Year 1-(n-1) anomaly data
(Rb) System Constraints (size, wt, power)
(10c) Analyze Partition C/O & Ambig. Groups
(c) Partition Testability Assessment
(a) Approved, consolidated test req't
(d) Fault Models

Figure 3-8. DFT steps in system definition flow diagram.

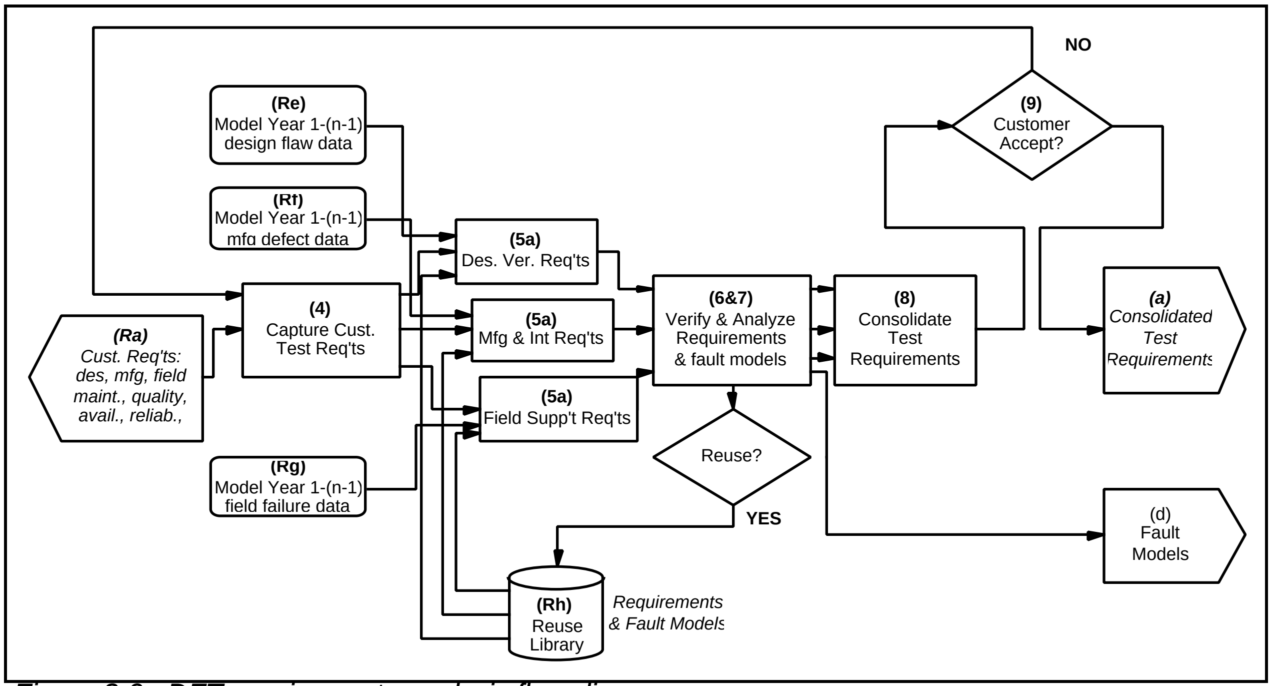

NO
(9) Customer Accept?
(Re) Model Year 1-(n-1) design flaw data
(Rf) Model Year 1-(n-1) mfg defect data
(5a) Des. Ver. Req'ts
(Ra) Cust. Req'ts: des, mfg, field maint., quality, avail., reliab.,
(4) Capture Cust. Test Req'ts
(5a) Mfg & Int Req'ts
(6&7) Verify & Analyze Requirements & fault models
(8) Consolidate Test Requirements
(a) Consolidated Test Requirements
(5a) Field Supp't Req'ts
Reuse?
YES
(Rg) Model Year 1-(n-1) field failure data
(d) Fault Models
(Rh) Reuse Library
Requirements & Fault Models

Figure 3-9. DFT requirements analysis flow diagram.

1. **Requirement Name (e.g., fault detection)**
   a. **Test Phase**
      - Design (e.g., simulation, prototype debug, qualification test, etc.)

- Production (e.g., go/no test, diagnostic test, repair verification, etc.)
- Field (e.g., operational, organizational, intermediate, depot)

b. Test Means

- BIST
- Test Equipment
- Manual procedures
- Mix of above

c. Test Mode

- Power-on test
- On-line concurrent (including interfering vs. non-interfering)
- On-line non-concurrent (including periodic vs. event-driven, operator invoked or software invoked)
- Off-line

d. Degree of Allowable External Support

- Operator may be "in the loop" to help achieve requirement
- Troubleshooter may be "in the loop" to help achieve requirement
- Job performance aid may be used to help achieve requirement

e. Fault Model Assumption

The definition of a "fault." For example, single stuck-at fault, multiple stuck-at fault, delay fault, intermittent fault, transient fault, etc.

f. Quantitative Definition of Metric

An equation defining the metric. For example, fault detection coverage might be defined as the total number of faults detected automatically by BIST, divided by the number of possible faults, with "fault" defined by the fault model above.

g. Prediction and Validation Weighing Factors

The factors used to allocate the requirements and later, to calculate system values from lower level values. For example, failure rate, usage rate, mission criticality etc.

h. Quantitative Requirement

The actual quantitative requirement, calculated by the "quantitative definition" above. For example "98%."

i. Allowable Requirement Compliance Tracking Methodologies

The means that may be used to track compliance to the requirement throughout the life cycle of the system. For example, topological dependency models at the prediction stage, fault simulation at the validation stage, and instrumentation or automatic fault-history logging at the measurement stage.

2. Metric Name (e.g., fault isolation coverage

a.

b.

...

Figure 3-10. Template for specifying test and DFT/BIST requirements.

5. Using iRa, check the reuse library for any test requirements that are lacking within the customer requirements for the three phases of testing (see Section 3.1.3.3). Using the necessary organizations, establish test requirements for each of the three phases of testing, including derived requirements, using the template shown in Figure 3-10.

   As part of this step, using iRd through iRf, begin the development of a "fault model" for each of the three phases of testing, using either the default "fault model" from the reuse library" or establish a set of new fault models, based on the design flaw, manufacturing defect and field defect statistics provided as inputs to this process step. The fault model will be refined in the Architecture Selection Process.

6. Using iRg, verify that all requirements are realizable, consistent, and valid. Realizability should be checked particularly in light of system constraints, iRg. Generate a mini-spiral if necessary for requirements that are questionable.

7. Using the organizations associated with the three phases of testing, consolidate the three groups of test requirements into a single "consolidated test specification" document. This step provides two benefits:

   - It creates a single spec in which all test related requirements exist for the entire life cycle of the system.

   - It forces the organizations responsible for the three phases of test to meet, where they can critique each others specs, fill in transition gaps in requirements, remove ambiguities, and negotiate tradeoffs. The objective should be development of a singular test philosophy and test architecture concept that will satisfy all of the needs of the three test phases (a step that will follow in the Architecture Definition stage). This effort will avoid the traditional situation, where three diverse test strategies, test equipment sets, and test vector sets are developed in three independent processes.

8. Analyze partial or full requirements or analyses results for possible inclusion in the RASSP reuse library.

9. Seek customer buy-off on the consolidated test requirements specification

10. Using iRh and iRi, interact with RASSP subsystem partitioning efforts. More specifically, perform the following:

    a. Using the consolidated test requirements, analyze and develop the preliminary system-level Test Strategy Diagram, TSD0 (see Figure 3-4, Hierarchy of Test Strategy Diagrams). The test strategy diagram allocates the test requirements to BIST, test equipment, and manual procedures. The preference should be the use of BIST and boundary-scan whenever possible, based on the preferred Test Architecture described earlier in this document and illustrated in Figure 1-5. The test strategy is developed using the TSD method discussed earlier and illustrated in Figure 3-11 and previous Figures 3-3 through 3-5. Figure 3-11 shows the top level TSD used to capture the initial test strategy developed in this stage. The default case for this diagram is that the test means is the same for all three phases of testing, unless modified during this process step.

       Initial values for transfer functions in the TSD would be requirements, while the attributes (which might also be requirement values) and the anomaly population density can be obtained from historical data accumulated in the reuse library, can be generated from new estimates (based on similar systems), or can make use of a variable to take the place of a fixed value. As the process continues, actual values for transfer functions, anomaly population densities, and attributes are entered into the appropriate TSDs through the prediction, verification, and measurement processes.

       While different test strategies for the three phases of test is allowed/supported, it is recommended that the three test organizations jointly develop a common test strategy (for example, based on simulation, PC based boundary-scan test, and BIST). While some forms of testing, such as IDDQ, may be specific to one phase of testing, due to a unique fault model assumption, it should still be possible to establish a common overall test strategy for the three phases. Each organization could then map that strategy onto their own TSD to observe its impact for their "fault" model assumptions and test environments.

    b. Develop initial test architecture (TA0), at the system/subsystem stage, including creation of test bus interface and test controller hierarchies, based on the ideal test architecture described earlier.

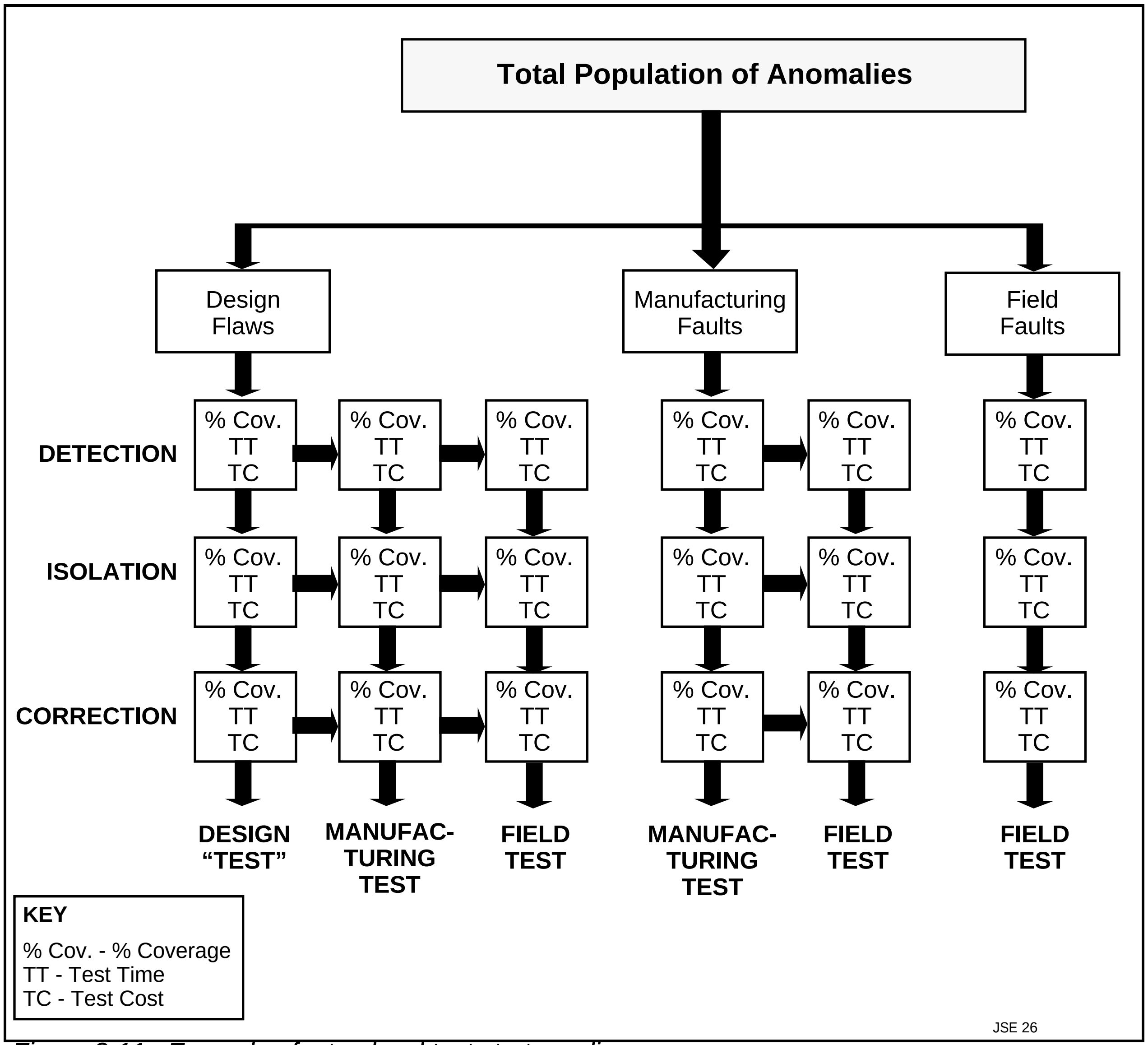


Figure 3-11. Example of a top level test strategy diagram.

c. Use controllability/observability analysis (e.g., using a topological dependency model) to analyze the inherent testability of the proposed system/subsystem partitioning. Assess impact on meeting requirements, and assure adequate fault isolation capabilities, as well as ambiguity group sizes and characteristics.

d. Using the analysis on step (b) above, assess subsystem partitioning candidates on the basis of supporting the preliminary concept of the test architecture and ability to support compliance to test requirements.

11. Reflect preliminary concept of the test architecture (TA0) into the executable specification for VP0. In the system model portion, include DFT and BIST features (e.g., test buses and controllers) where possible in the system functionality data. Incorporate estimated degradation factors in system performance data to accommodate any predicted on-line BIST impact. Include BIST/DFT requirements in physical constraint data, (iRb). In the test bench portion, include proper stimulus and response data to verify BIST/DFT functionality and where possible, to validate test requirements compliance during VP0 simulations. Performance simulations should verify that functional performance, degraded by any estimated on-line BIST impact, still meets performance requirements. Back-annotations of the e-spec should be done, based on more detailed level simulations at later points.

3.2.1.3 Outputs

3.2.1.3.1 Outputs to the Overall RASSP Processes

oRa.	Test related elements for the reuse library.	Step 8, 10 and 11
oRb.	VP0 e-spec updated based on preliminary test architecture concept.	Step 11

3.2.1.3.2 Outputs to DFT Processes

oDa.	Preliminary fault model definitions	Step 5
oDb.	An approved, consolidated test requirements specification.	Step 7
oDc.	Preliminary version of top level test architecture, TA0, and test strategy diagram, TSD0, including allocation of test requirements to BIST, test equipment, and manual procedures for all three phases of testing.	Step 10
oDd.	Testability assessment(s) of subsystem partitions.	Step 10

3.2.1.4 Handling of COTS

a. The impact of COTS on this step is to place practical limitations on the test requirements and subsequent test strategies. For example, it does not make sense today to specify a requirement that can only be achieved with 100% boundary-scan, if a large part of the system will use COTS boards that do not come in boundary-scan versions.

b. The development of the fault model in this step is also affected by the presence of COTS, particularly if the COTS element is a "black box" in terms of its structural composition. Several options exist for handling fault models for black box COTS elements:

   1. Establish a structural fault model, in which the "node" is the lowest level input or output for which a description exists. This level will usually be at the inputs and outputs of blocks at the block diagram level or could be simply the inputs or outputs of the package itself (such as a chip) or board.
   2. Attempt to define a "functional fault model." This is a fault model based on the function, rather than the structure of the item; and it describes the erroneous functional behavior that would occur in the presence of a fault. While this approach will work for simple functions, it is not likely to be practical for large VLSI based components or boards.

c. The presence of COTS may throttle attempts to reach a singular life cycle test strategy. One reason is that black box COTS elements are usually tested functionally (for defect detection and isolation), while most of the rest of the design that is not black box oriented is usually tested using a great deal of structural testing and some functional testing. An analysis of the degree of anticipated COTS usage and the extent to which current COTS incorporates test features, would tend to allow an early, preliminary assessment of what plateau of test architectures might be possible. For example, several "plateaus" can be envisioned:

   1. All COTS with no testability features. This would drive the project toward the traditional test philosophy, which may not even be possible, depending on the complexity and packaging of the COTS circuitry.
   2. COTS with some ad hoc test features, but no BIST. This may at least allow the boards to be tested with traditional in-circuit or functional testers, again depending on the complexity and packaging of the COTS circuitry.
   3. COTS with boundary-scan features but no BIST. This plateau "opens the gate" to possible singular test strategies built around PC-based testing.
   4. COTS with BIST and boundary-scan. This plateau definitely "opens the gate" to singular test strategies built around PC-based testing and BIST.

Both of the latter two plateaus are in the future at this time. In the meantime, preference (from a DFT standpoint) should always be given to COTS items in the latter two categories.

3.2.1.5 Risk Management/Mini-Spirals

This step could stimulate a mini-spiral (See RASSP methodology document.) to explore such things as a quickly emerging test technology that may become available in the project timeframe (e.g., board level BIST insertion). For example, if there are any major breakthroughs in functionally oriented fault modeling or testing for black boxes, that would warrant a mini-spiral, since the black box represents a test risk.

Another application of the mini-spiral could be to assess the consistency, validity, and realizability of a questionable requirement(s).

3.2.1.6 Interface to Reuse Library

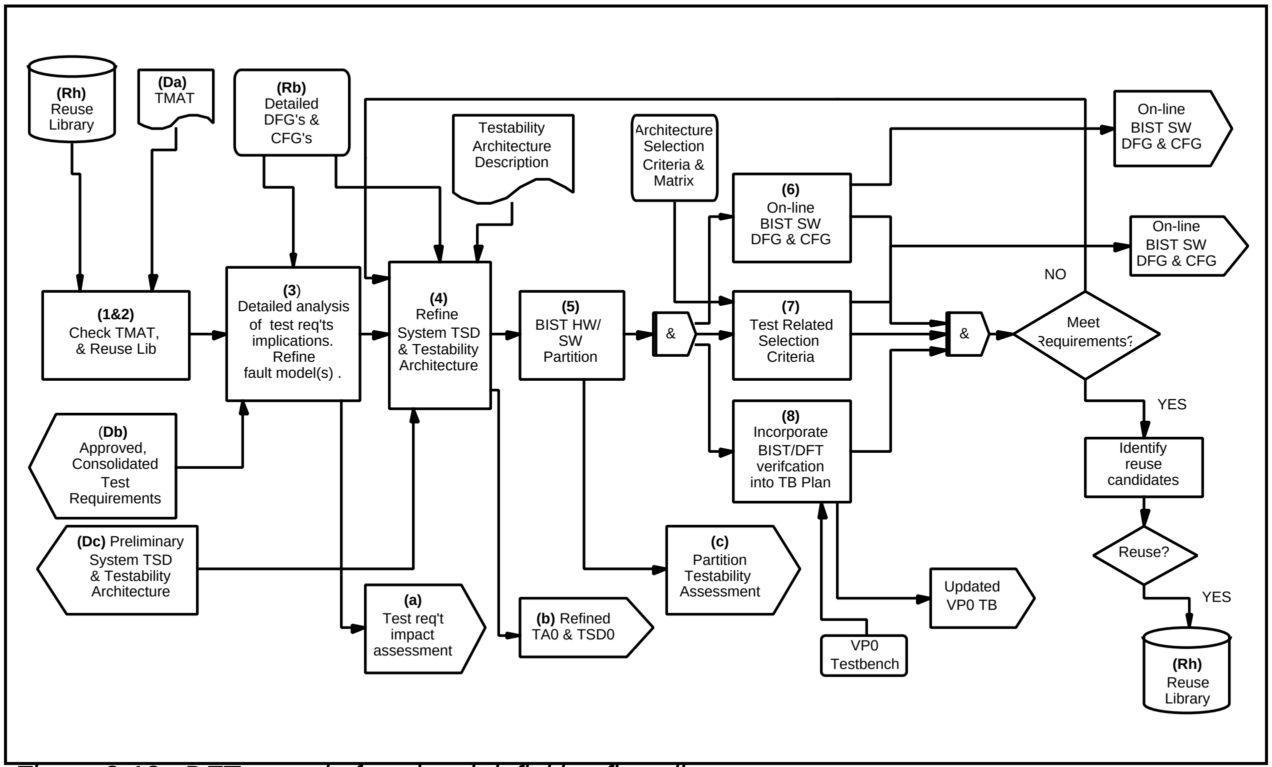
This step could spawn entries in the reuse library in the form of fault models, test requirements specifications and consolidated test requirements specifications.

3.2.1.7 Interface to the Overall RASSP Methodology and Tools

Interface will be to PRICE, RDD100, and RTM.

3.2.2 Architecture Definition - Functional Design Step

The functional design step provides a more detailed analysis of the processing requirements (including BIST), resulting in initial sizing estimates, detailed data and control flow graphs for all required processing modes to drive the hardware/software codesign, and the criteria for architecture selection. The control flow graphs provide the overall signal processor control, such as mode switching (referred to as the command program). Functional simulators support the execution of both the data and control flow graphs. For complex control applications, these simulations can be coupled to ensure that all control is properly executed and results in the proper graph actions (e.g., mode transitions). Figure 3-12 shows the DFT steps which occur during the Functional Definition Step.



(Rh) Reuse Library
(Da) TMAT
(Rb) Detailed DFG's & CFG's
Testability Architecture Description
Architecture Selection Criteria & Matrix
(6) On-line BIST SW DFG & CFG
On-line BIST SW DFG & CFG
On-line BIST SW DFG & CFG
NO
(1&2) Check TMAT, & Reuse Lib
(3) Detailed analysis of test req'ts implications. Refine fault model(s) .
(4) Refine System TSD & Testability Architecture
(5) BIST HW/ SW Partition
&
(7) Test Related Selection Criteria
&
Meet Requirements?
YES
(Db) Approved, Consolidated Test Requirements
(8) Incorporate BIST/DFT verifcation into TB Plan
Identify reuse candidates
(Dc) Preliminary System TSD & Testability Architecture
(c) Partition Testability Assessment
Reuse?
Updated VP0 TB
YES
(a) Test req't impact assessment
(b) Refined TA0 & TSD0
VP0 Testbench
(Rh) Reuse Library

Figure 3-12. DFT steps in functional definition flow diagram.

3.2.2.1 Inputs

3.2.2.1.1 Inputs from Overall RASSP Processes

iRa.	Reuse library.	Step 2
iRb.	In-process detailed data and control flow graphs	Step 6
iRc.	Architecture selection criteria matrix	Step 7
iRd.	Test bench plans and definitions from System Definition process.	Step 8

3.2.2.1 .2 Inputs from DFT Processes

iDa.	The TMAT Table	Step 1
iDb.	An approved, consolidated test requirements specification.	Step 3
iDc.	Preliminary, top level test architecture, TA0, and test strategy, TSD0.	Step 4
iDd.	Testability assessment(s) of subsystem partitions.	Step 4

3.2.2.2 Process

The following steps are followed:

1. Using iDa, access the Metric/Tool Application Table (TMAT) (see Table A-1) and select metrics/tool pairs for use during this step.

2. Using iRa., check the reuse library for candidate reuse elements for this step (i.e., requirements impact analyses, fault models, BIST DFG's, CFG's and architecture-independent primitives, test-related selection criteria, BIST/DFT verification testbenches).

3. Using iDb, perform a more detailed analysis of the test requirements and their implications and refine the underlying fault model(s) accordingly. For example, begin to define the details of the BIST modes of operation (power-on, on-line, off-line, etc.). Begin determination of the details of the fault isolation capabilities, including assessment of the impact of ambiguity group size requirements.

4. Based on the overall top level test architecture, TA0, and test strategy, TSD0,[1] upon the analysis in Step 3 of the consolidated test requirements specification and upon the increased information developed under functional design represented as iRb, develop preliminary test strategy and architecture, TSD1 and TA1, respectively (see Figure 3-4, Hierarchy of Test Strategy Diagrams).

5. Begin first pass at DFT/BIST hardware and software partitioning.

6. Begin development of on-line BIST software, by reflecting it into the detailed DFGs and CFGs, iRb.

7. Using iRc, insert test related criteria (including DFT and BIST) into the architecture selection criteria matrix.

8. Using iRd, incorporate BIST/DFT verification into test bench plans and descriptions.

3.2.2.3 Outputs

3.2.2.3.1 Outputs to the Overall RASSP Processes

oRa.	Updated DFG's and CFG's with on-line BIST SW functions captured.	Step 6
oRb.	Test related criteria for architecture selection criteria matrix.	Step 7
oRc.	BIST/DFT verification approach for test bench plans and descriptions.	Step 8

3.2.2.3.2 Outputs to DFT Processes

oDa.	Preliminary assessment of Test Requirement implications.	Step 3
oDb.	Refined Fault Models	Step 3
oDc.	Test mode and fault isolation specifications.	Step 3
oDd.	Preliminary architecture-level test strategy diagram, TSD1, and testability architecture, TA1.	Step 4
oDe.	First pass at BIST/DFT hardware/software partitioning.	Step 5
oDf	On-line BIST SW functions captured into DFG's and CFG's.	Step 6

3.2.2.4 Handling of COTS

At this step, the general potential areas of COTS may start to be identifiable. Development is started on the "lead, follow, or get out of the way" strategy for COTS at the testability architecture level. This COTS strategy is discussed in more detail in the detailed design phase.

[1] Including allocation of test requirements to BIST, test equipment, and manual procedures for all three phases of testing.

If COTS components such as boards or subsystems will be used and SW BIST is the primary means for test, then the DFG's for SW BIST of these components should be captured, simulated and fault graded to the maximum extent practicable.

3.2.2.5 Risk Management/Mini-Spirals

As before, this step could stimulate a mini-spiral (See RASSP methodology document.) to explore such things as a quickly emerging test or BIST/DFT technology that may become available in the project timeframe (e.g., front end BIST insertion).

3.2.2.6 Interface to Reuse Library

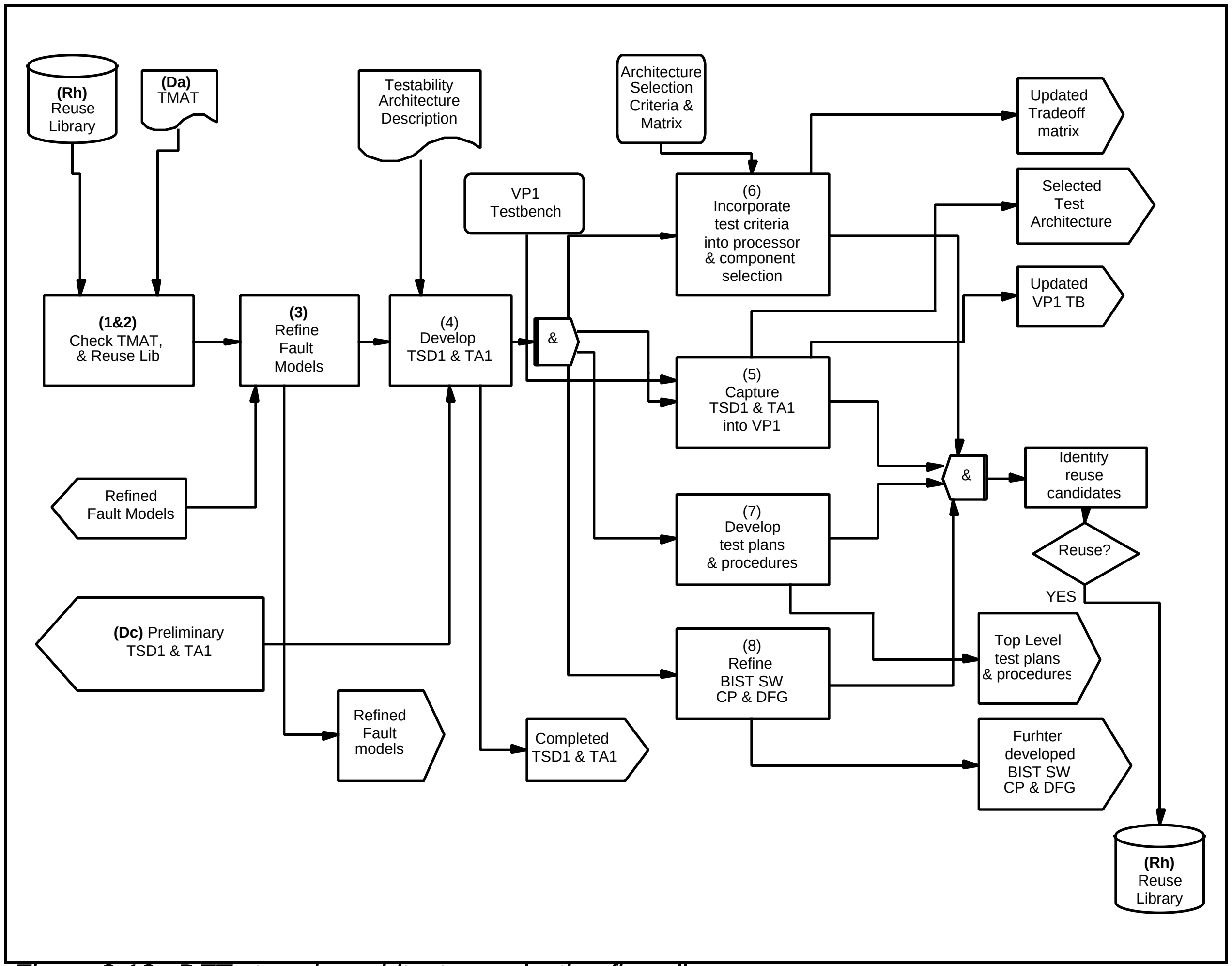
Any of the outputs of the steps in this stage could provide candidates for encapsulation in the reuse library.

3.2.2.7 Interface to RASSP Tools

(TBD)

3.2.3 Architecture Definition - Architecture Selection Step

During architecture selection, various candidate architectures are evaluated through iterative performance simulation and optimized to appropriate levels of detail. A trade-off analysis based on the established selection criteria results in the specification of the detailed architecture, and software partitioning and mapping. As part of the trade-off analysis, information is used from the required cross disciplines such as reliability and testability (either manually or through design advisors) to populate the trade-off matrix. Figure 3-13 shows the DFT steps which occur during the Architecture Selection Step.



(Rh) Reuse Library
(Da) TMAT
Testability Architecture Description
Architecture Selection Criteria & Matrix
Updated Tradeoff matrix
Selected Test Architecture
VP1 Testbench
(6) Incorporate test criteria into processor & component selection
Updated VP1 TB
(1&2) Check TMAT, & Reuse Lib
(3) Refine Fault Models
(4) Develop TSD1 & TA1
&
(5) Capture TSD1 & TA1 into VP1
&
Identify reuse candidates
Refined Fault Models
(7) Develop test plans & procedures
Reuse?
YES
(Dc) Preliminary TSD1 & TA1
(8) Refine BIST SW CP & DFG
Top Level test plans & procedures
Refined Fault models
Completed TSD1 & TA1
Furhter developed BIST SW CP & DFG
(Rh) Reuse Library

Figure 3-13. DFT steps in architecture selection flow diagram.

This portion of the process is heavily dependent on the reuse of architectural (hardware and software) components to provide significant time-to-market improvements. In addition, during architecture selection, all software not represented by the DFG's is designed. Based on the requirements, the non-DFG software may include BIST[2], downloading data and code, and diagnostics. The virtual prototype, VP1, produced during architecture selection is not a full system prototype, since function and performance are simulated independently and may or may not be coupled with the overall control mechanism. Several architectures may be selected for verification during the following step (i.e., primary candidate and risk reduction alternatives).

[2] At the time of this writing, the feasibility of capturing the overall system level BIST function with DFGs and previously developed primitives is being investigated. The goal is to develop BIST functions with the same tools/methodology as the other functions.

Virtual Prototypes (VPx) are specified at the highest level of abstraction (lowest level of detail) of any model in the prototype. The highest level of abstraction of any model in VP1 has performance modeled at least at the Network Architecture Level and functionality modeled at least at the Data Flow Graph level. High risk elements of a design will typically have been developed to a higher level of detail than this specified level. For instance, a high risk ASIC design mini-spiral might have been spun-off to verify that the ASIC's specifications are realistic. Before a VP1 level architecture can be approved, positive feedback from these activities must be received. The amount of detail required depends on what is agreed upon by the IPDT as an acceptable level of risk. For the ASIC design in the example, it could be a behavioral model of the ASIC along with estimates of its size and I/O. However, in another case, a gate level model of a critical path may be developed. For a completely COTS solution, full functional models or even actual hardware may be available, and no performance models needed. In this case, level of detail for VP1 will be much higher than in the custom-part example.

By the end of Architecture Selection (VP1), all large components (processors and custom processors) should be chosen, function defined, timing and sizing estimated. All other logic should reside in behavioral VHDL blocks which correspond approximately to boards or blocks of logic on a board. Communication between the boards should be defined along with approximate mechanical size and pin count off the board. Function (at the level of DFGs) and performance (at the level of performance models) are simulated independently and may or may not be coupled with the overall control mechanism.

The Architecture Selection level test strategy diagram, TSD1, and testability architecture, TA1, are developed concurrently with VP1 for each candidate architecture.

3.2.3.1 Inputs

3.2.3.1.1 Inputs from Overall RASSP Processes

iRa.	Reuse library	Step 2
iRb.	Architecture component test bench definitions.	Step 5
iRc.	Executable specification for VP1	Step 5

3.2.3.1.2 Inputs from DFT Processes

iDa.	TMAT table	Step 1
iDb.	Refined fault model definitions	Step 3
iDc.	Preliminary architecture-level test strategy diagram, TSD1, and test architecture, TA1.	Step 4
iDd.	First pass at BIST/DFT hardware/software partitioning.	Step 4
iDe.	Test mode and fault isolation specifications.	Step 4-7
iDf.	Preliminary assessment of Test Requirements implications	Step 5
iDg.	BIST/DFT verification approach for test bench plans and descriptions.	Step 5
iDh.	Test related criteria for architecture selection criteria matrix.	Step 6

3.2.3.2 Process

The following steps are followed:

1. Using iDa, access the Metric/Tool Application Table (TMAT) (see Table A-1) and select metrics/tool pairs for use during this step.

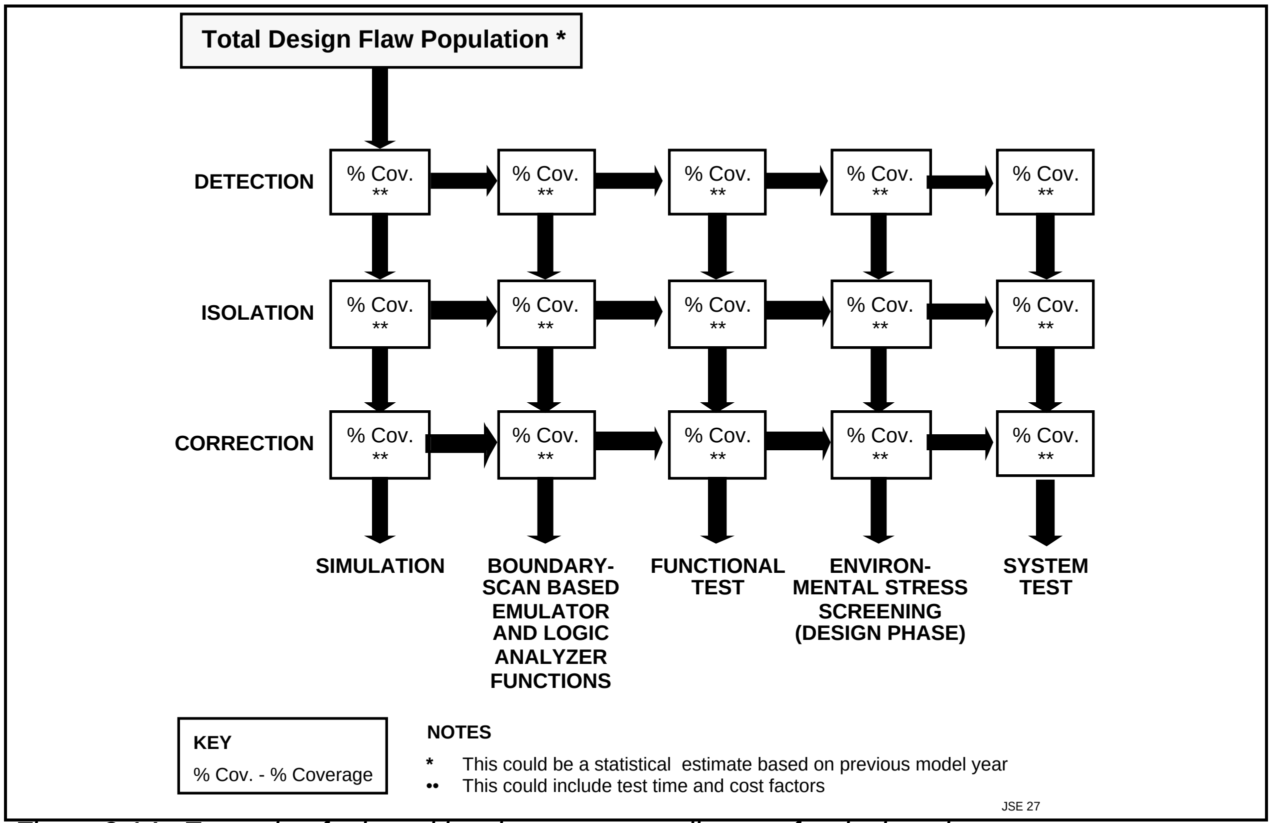
2. Using iRa, check the reuse library for candidate reuse elements for this step.

3. Using iDb, refine the "fault" models for each of the three phases of testing - design, manufacturing and field, as input to the test strategy development. These fault model definitions become the basis for establishing the anomaly populations in the TSDs.

4. Using iDc (TSD1 and TA1), the top level Test Strategy Diagram and Testability Architecture, including captured test requirements, develop the Architecture-selection level test strategy, TSD1 (oDc), and testability architecture, TA1 (oDc), for each candidate functional architecture being considered (see Figure 3-4, Hierarchy of TSD's). The test strategy, TSD1 (oDc), is developed using the TSD method discussed earlier and illustrated in Figures 3-14 through 3-17 and previous Figures 3-3 through 3-5 and 3-11. Figure 3-14 shows the concept of a TSD-based analysis for board testing (debugging) in the design test phase. Figures 3-15 and 3-16 show TSD-based analyses for the manufacturing test phase using respectively, test time and cost (Figure 3-15) and defect or quality levels (Figure 3-16). Figure 3-17 shows a TSD-based analysis used for BIST requirements analysis.

Concurrently, perform the following substeps:

a. Determine the test bus and controller hierarchy for each test architecture.
b. Perform hardware/software tradeoff analysis for each test architecture.
c. Using the first pass allocation in the Functional Design stage, refine allocation of BIST/DFT functions to hardware and software.
d. Prior to simulation, predict inherent testability of each architecture under consideration, using controllability and observability analysis (e.g., topological dependency modeling). Include ambiguity group analysis in this effort. Document results in the top level and architectural level, prediction TSD's within TSD1 (oDc) and in the RASSP architecture tradeoff matrix (oRb).
g. Generate preliminary prediction TSD's for each board type.
f. Develop preliminary requirements for MCM and chip components and document in TSD1.



Total Design Flaw Population *
DETECTION
ISOLATION
CORRECTION
% Cov. **
SIMULATION
BOUNDARY-SCAN BASED EMULATOR AND LOGIC ANALYZER FUNCTIONS
FUNCTIONAL TEST
ENVIRON-MENTAL STRESS SCREENING (DESIGN PHASE)
SYSTEM TEST
KEY
% Cov. - % Coverage
NOTES
* This could be a statistical estimate based on previous model year
•• This could include test time and cost factors
JSE 27

Figure 3-14. Example of a board level test strategy diagram for design phase test strategy development and tradeoff analysis.

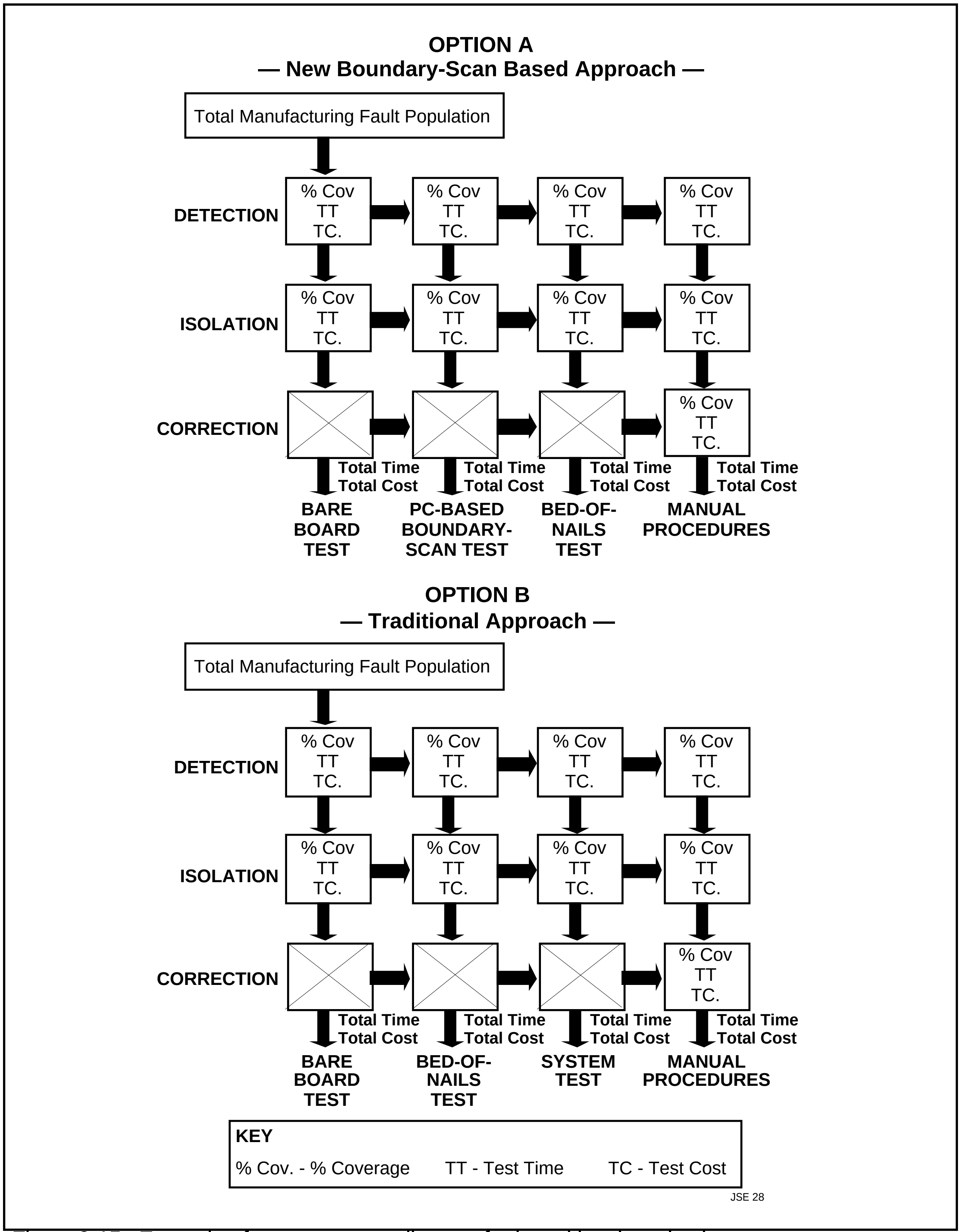


Figure 3-15. Example of a test strategy diagram for board level production test strategy development and tradeoffs.

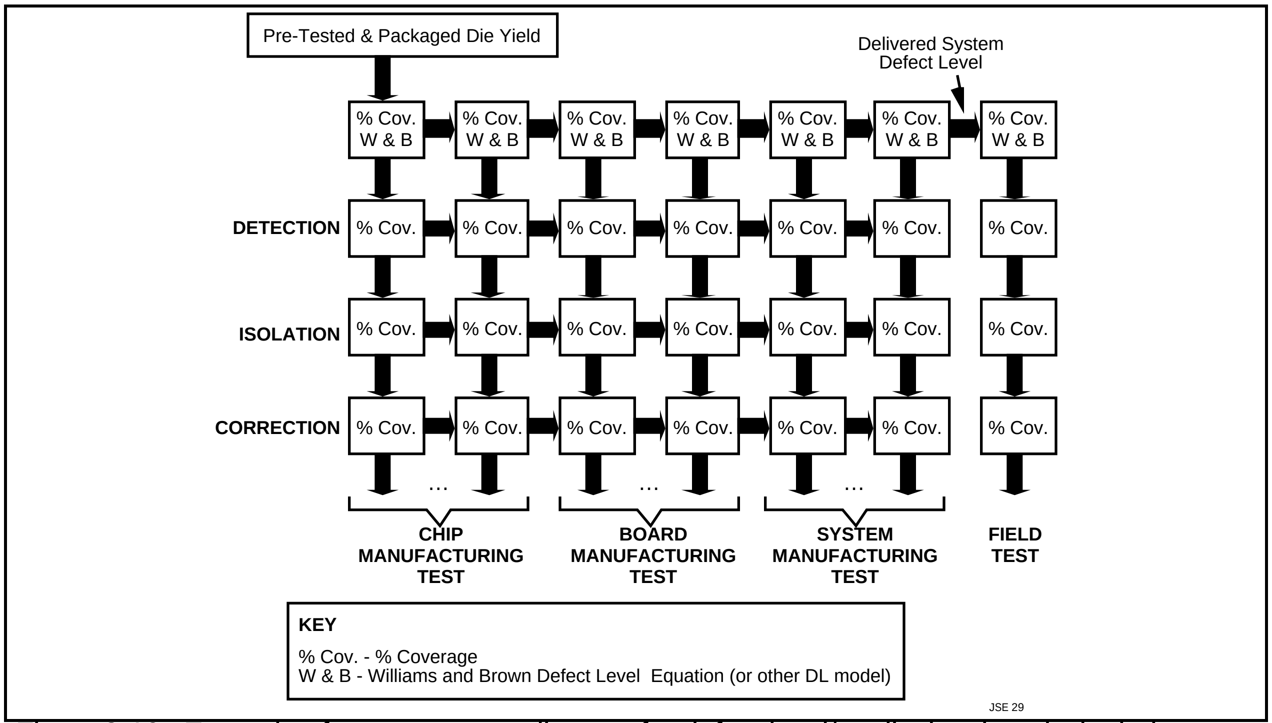

Pre-Tested & Packaged Die Yield
Delivered System Defect Level
% Cov. W & B
DETECTION
ISOLATION
CORRECTION
% Cov.
CHIP MANUFACTURING TEST
BOARD MANUFACTURING TEST
SYSTEM MANUFACTURING TEST
FIELD TEST
KEY
% Cov. - % Coverage
W & B - Williams and Brown Defect Level Equation (or other DL model)
JSE 29

Figure 3-16. Example of a test strategy diagram for defect level/quality level analysis during manufacturing.

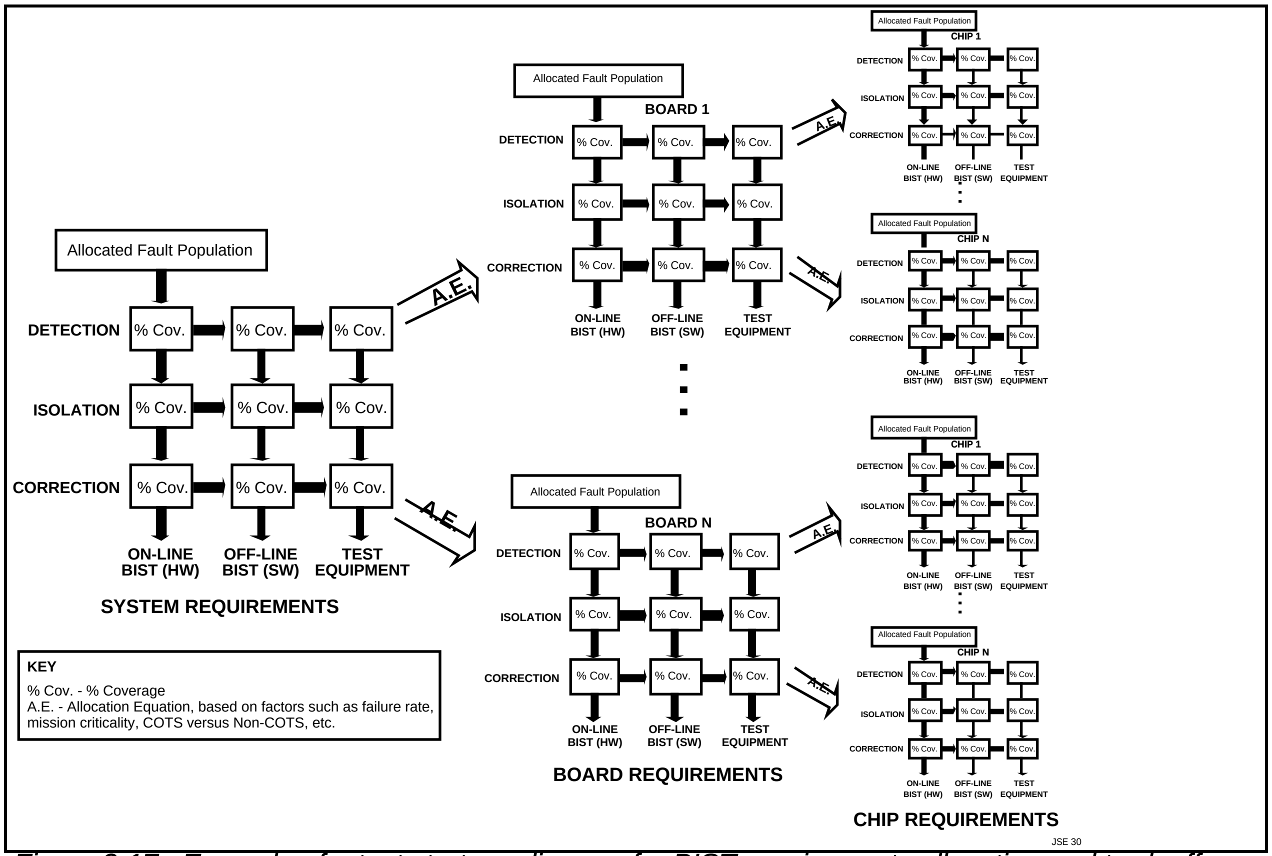


Figure 3-17. Example of a test strategy diagram for BIST requirements allocation and tradeoff analysis for the field test phase

5. Using iRc, reflect test architecture, TA1, into the executable specification for VP1. In the system model portion, include DFT and BIST features where possible in the system functionality data. Incorporate degradation factors in system timing and performance data to accommodate any predicted impact of on-line BIST. Include DFT/BIST requirements in physical constraint data. In the test bench portion, include proper stimulus and response data to verify BIST/DFT functionality and where possible, to verify test requirements compliance during VP1 simulations. Separate performance simulations should verify that functional performance, degraded by any on-line BIST operations, still meets performance requirements. Any behavioral level simulations should verify BIST/DFT hardware/software codesign success. Back-annotations of the e-spec should be done, based on more detailed level simulations at later points. Simulation results for compliance verification, if any at this point, should be inserted into the top level Verification TSD within TSD1 (oDc) and in the RASSP architecture tradeoff matrix (oRb).

6. During processor and component selection (MCAs, ASICs, FPGAs, COTS Interconnect chips, etc.), ensure BIST and DFT are considered as criteria for selection, oRb, (e.g., boundary-scan, scan-based emulation capabilities, BIST, strong interrupt capability, sufficient throughput to accommodate BIST functions, etc.). Processor selection in particular will impact Testability Architecture and strategy, in the case that processors are used for BIST/DFT functions (i.e., software BIST allocation).

7. Using iDc, develop top level test plans and procedures (oDe), based on the architecture level TSD's and Test Architectures being evaluated.

8. Continue with non-DFG BIST software development (oDf). Refine any DFG/CFG based BIST software.

9. Select one test architecture for the selected functional architecture (oRc).

3.2.3.3 Outputs

3.2.3.3.1 Outputs to the Overall RASSP Processes

oRa.	VP1 e-spec updated based on test architecture candidates.	Step 5
oRb.	DFT inputs to architecture tradeoff matrix	Step 4-6
oRc.	Processor selection results, including BIST/DFT considerations.	Step 9

3.2.3.3.2 Outputs to DFT Processes

oDa.	Refined fault model definitions.	Step 3
oDb.	TSD1, Architecture-level test strategy diagram, TSD1, and Test Architecture, TA1, for each processor system architecture candidate.	Step 4, 5
oDc.	Preliminary DFT/BIST hardware/software allocation	Step 4, 5
oDd	Architecture component test bench definitions.	Step 5
oDe.	Preliminary board level test plans and system acceptance test procedures .	Step 7
oDf.	Further developed non-DFG BIST software. Target- dependent DFG/CFG on-line BIST software.	Step 8

3.2.3.4 Handling of COTS

At this step, the "lead, follow, or get out of the way" strategy for COTS at the testability architecture level can be refined. This COTS strategy is discussed in more detail in the detailed design phase.

Another major issue with COTS is the definition of the fault model. At a minimum, it should be possible to define the fault model for COTS boards and devices at least at the I/O pin level. If functional block diagrams are available for the COTS devices, the fault model may be able to be extended down to the periphery of the individual blocks within the device, thus reducing the reliance on "black box" functional test to the testing of the internal blocks within the device. Plans should be initiated to establish how COTS will be handled in the fault simulation activities in the Detailed Design stage.

3.2.3.5 Risk management/Mini-Spirals

As before, this step could stimulate a mini-spiral (See RASSP methodology document.) to explore such things as a quickly emerging test or BIST/DFT technology that may become available in the project timeframe (e.g., front end BIST insertion).

3.2.3.6 Interface to Reuse Library

Any of the outputs of the steps in this stage could provide candidates for encapsulation in the reuse library.

3.2.3.7 Interface to RASSP Tools

(TBD)

3.2.4 Architecture Definition - Architecture Verification Step

Architecture verification is the process of hierarchically simulating both the functionality and increased performance detail of a selected architecture candidate. Up to this point the overall functionality has not been verified. An integrated, simulation-framework supports mixed-domain simulation so that high-level performance and functional simulation can be coupled with ISA or RTL VHDL simulators, hardware emulators or hardware testbeds. The goal is to validate operation of all architectural entities and the interfaces between them before detailed design. Software partitions are autocoded to produce software modules translated from the processor independent library elements to optimized, processor specific implementations. Figure 3-18 shows the DFT steps which occur during the Architecture Verification step.

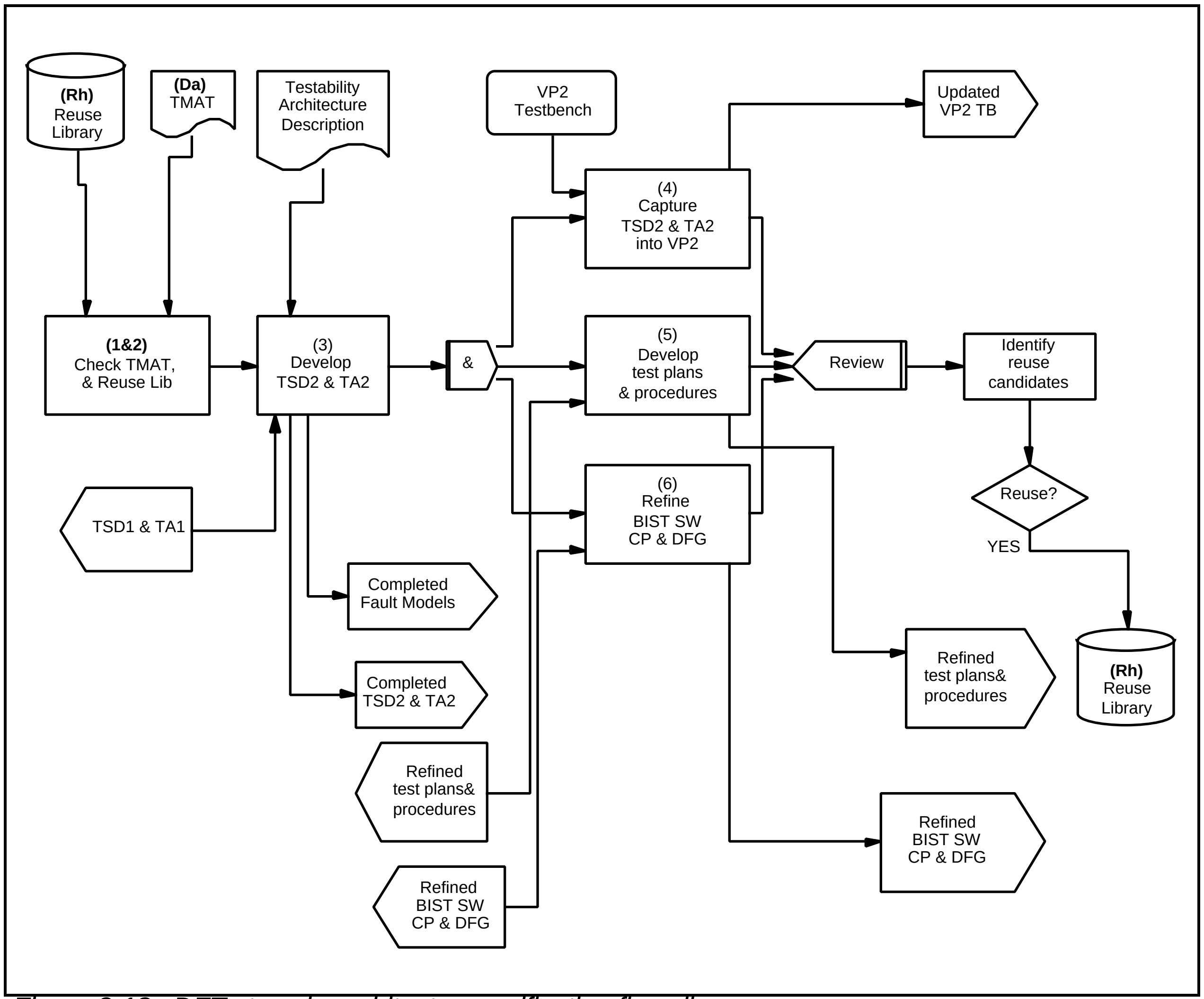

(Rh) Reuse Library
(Da) TMAT
Testability Architecture Description
VP2 Testbench
Updated VP2 TB
(4) Capture TSD2 & TA2 into VP2
(1&2) Check TMAT, & Reuse Lib
(3) Develop TSD2 & TA2
&
(5) Develop test plans & procedures
Review
Identify reuse candidates
Reuse?
YES
TSD1 & TA1
(6) Refine BIST SW CP & DFG
Completed Fault Models
Completed TSD2 & TA2
Refined test plans& procedures
(Rh) Reuse Library
Refined test plans& procedures
Refined BIST SW CP & DFG
Refined BIST SW CP & DFG

Figure 3-18. DFT steps in architecture verification flow diagram.

VP2 for the verified architecture is produced at the end of this step. All boards are broken down into behavioral blocks representing MCMs, ASICs, FPGAs or relatively small logic blocks (i.e., glue logic and/or functions to be implemented in PLDs). This is where ASIC/FPGA trade-off studies are done. Blocks representing small amounts of logic may remain unmapped until module design if necessary. Such things as buffering would not be accounted for at this point although it should be kept in mind for space considerations. Performance and functionality are simulated together with the overall control mechanism. At this point, the highest level of abstraction allowed is a full-behavioral model which incorporates function and timing. All code should be verified during VP2 at least at the HLL level.

This step may be entered a number of times as part of architecture selection but is exited successfully only when all architecture requirements have been achieved. Only one solution (architecture) is output from this step for detailed design.

3.2.4.1 Inputs

3.2.4.1.1 Inputs from Overall RASSP Processes

iRa.	Reuse library	Step 2
iRb.	In-Process VP2 e-spec	Step 4
iRc.	Processor selection results, including BIST/DFT considerations	Step 3 and 4
iRd.	Architecture Tradeoff Matrix	Step 3 and 4

3.2.4.1.2 Inputs from DFT Processes

iDa.	TMAT Table	Step 1
iDb.	Refined fault model definitions.	Step 3
iDc.	TSD1, Architecture-level test strategy diagram, TSD1, and Test Architecture, TA1, for each processor system architecture candidate.	Step 3
iDd	Architecture component test bench definitions.	Step 4
iDe.	Preliminary board level test plans and system acceptance test procedures .	Step 5
iDf.	Further developed non-DFG BIST software. Target- dependent DFG/CFG on-line BIST software.	Step 6
iDg.	Refined DFT/BIST hardware/software allocation	Step 6

3.2.4.2 Process

The following steps are followed:

1. Using iDa, access the metric/tool Application Table (TMAT) (see Table A-1) and select metrics/tool pairs for use during this step.

2. Using iRa, check the reuse library for candidate reuse elements for this step (i.e., fault models, TSD’s, TA’s, testbenches, test plans and procedures, BIST SW CP, DFG, and primitives)

3. Using iDc (TSD1 and TA1), the top level Test Strategy Diagram, and Testability Architecture, develop the Architecture Verification level test strategy, TSD2 (oDc), and testability architecture, TA2 (oDc) (see Figure 3-4, Hierarchy of TSDs).

Concurrently, perform the following substeps:

   a. Extend the test bus and controller hierarchy for each test architecture onto the board/module level.
   b. Using the preliminary allocation in the Architecture Selection stage, refine allocation of BIST/DFT functions to hardware and software.
   d. Prior to simulation, predict testability of each architecture under consideration, using controllability and observability analysis (e.g., topological dependency modeling and boundary scan access analysis). Include ambiguity group and testability analysis in this effort. Document simulation and analysis results in the top level and architectural level, preliminary verification TSD’s within TSD2 (oDb) and in the RASSP architecture tradeoff matrix (oRb).

4. Reflect refined test architecture into the executable specification for VP2 (oRb). In the system model portion, include DFT and BIST features where possible in the system functionality data. Include BIST/DFT features (e.g., boundary-scan, BIST, etc.) in behavioral models for all new designs. Any behavioral level simulations should verify BIST/DFT hardware/software codesign success. Incorporate degradation factors in system timing and performance data to accommodate any predicted on-line BIST impact. Include DFT/BIST requirements in physical constraint data. In the test bench portion (oDc), include proper stimulus and response data to verify BIST/DFT functionality and where possible, to validate

test requirements compliance during VP2 simulations. Concurrent performance simulations should verify that functional performance, degraded by any on-line BIST operations, still meets performance requirements. Back-annotations of the e-spec should be done, based on more detailed level simulations at later points. Simulation results for compliance verification, if any at this point, should be inserted into the Verification TSD, oDb, (and comparted to previous data) and in the RASSP architecture tradeoff matrix (oRd). Modify test architecture (oDb), if needed, based on e-spec simulation results.

5. Using iDe, refine test plans (ODd) and procedures (oDe), based on the Architecture Verification level test strategy, TSD2, and testability architecture, TA2.

6. Using iDg, refine DFT/BIST hardware/software allocation (oDg). Using iDf, continue with non-DFG and DFG-based BIST software development (oDf).

7. Conduct a review of the TSDs, test architectures, test plans, and test procedures for flowdown to the Detailed Design stage.

3.2.4.3 Outputs

3.2.4.3.1 Outputs to the Overall RASSP Processes

oRa.	Top level Test Architecture description and models embedded into functional architecture.	Step 7
oRb.	VP2 e-spec updated based on test architecture candidates.	Step 4
oRc.	Processor verification results, including BIST/DFT considerations	Step 4
oRd.	Updated Architecture Tradeoff Matrix	Step 4

3.2.4.3.2 Outputs to DFT Processes

oDa.	Completed fault model definitions.	Step 3
oDb.	Architecture-selection-level test strategy diagram, TSD2, and Testability Architecture, TA2, for selected processor system architecture.	Step 7
oDc.	Architecture component test bench definitions.	Step 4
oDd.	Test plans for Board, MCMs, and special chips (i.e., ASICs, FPGAs, bare dir or KGD).	Step 5
oDe.	Test procedures for system acceptance and subsystem acceptance.	Step 5
oDf.	Equivalent DFG’s for on-line BIST software	Step 6
oDg.	Completed BIST/DFT hardware/software allocation	Step 6

3.2.4.4 Handling of COTS

At this step, the "lead, follow, or get out of the way" strategy for COTS at the testability architecture level can be refined further. An example of using this COTS strategy is shown Figure 3-19. Controllability and observability analysis results can be used to determine what solution(s) could be examined for sections anticipated to employ COTS. Possibilities could be explored of absorbing some or all COTS functions (that are in non-testable devices) into ASICs or FPGA type devices that incorporate at least boundary-scan, if not BIST. A refinement of the COTS fault model may also take place in this stage.

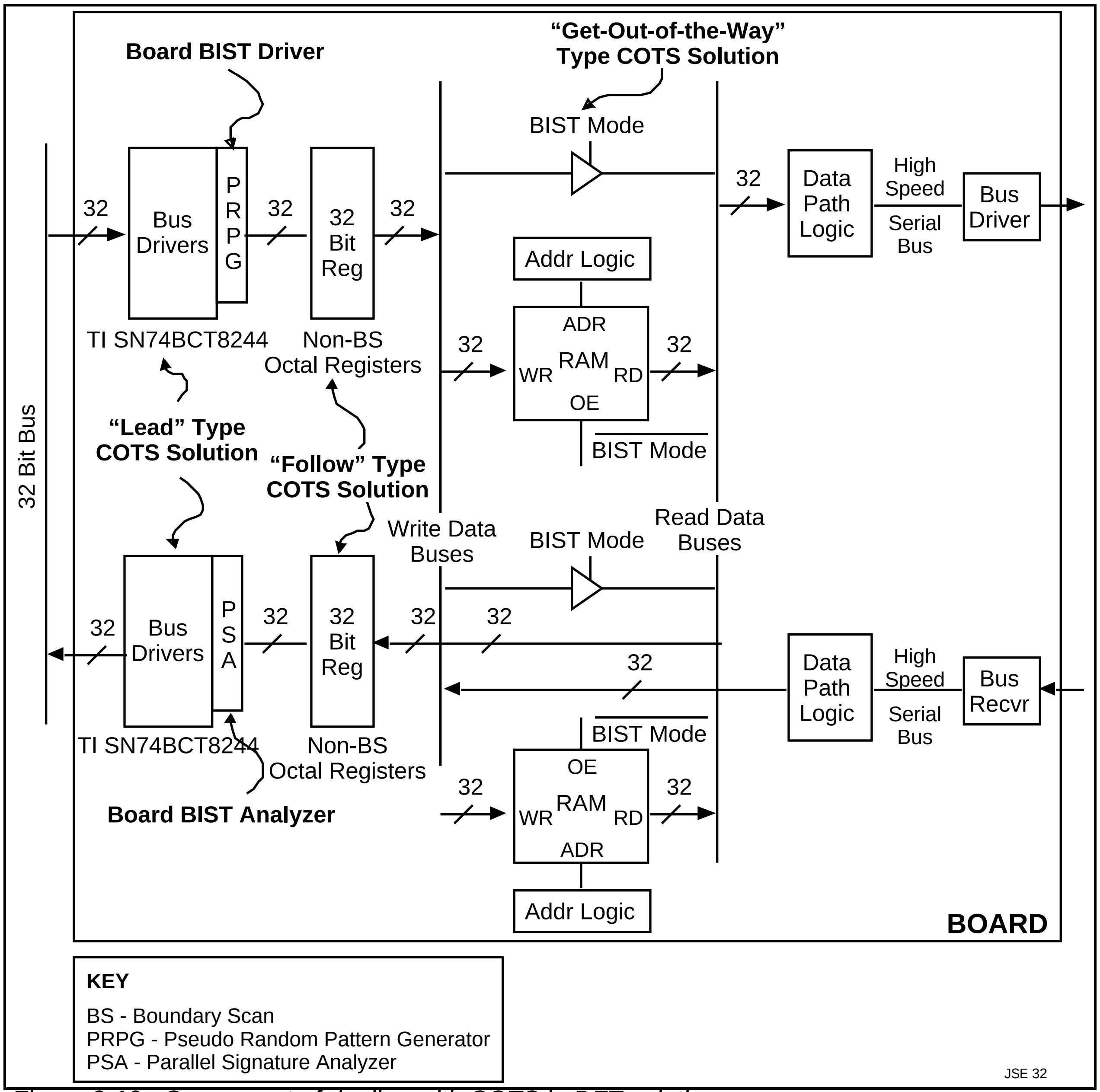


Figure 3-19. One aspect of dealing with COTS in DFT solutions.

3.2.4.5 Risk Management/Mini-Spirals

As before, this step could stimulate a mini-spiral (See RASSP methodology document.) to explore such things as a quickly emerging test or BIST/DFT technology that may become available in the project timeframe (e.g., front end BIST insertion).

3.2.4.6 Interface to Reuse Library

Any of the outputs of the steps in this stage could provide candidates for encapsulation in the reuse library.

3.2.4.7 Interface to RASSP Tools

(TBD)

3.2.5 Detailed Design

The focus of detailed design is synthesis and implementation of the selected architecture in hardware and software. From architecture verification, all boards have been broken down into behavioral blocks representing MCMs, ASICs, FPGAs or relatively small logic blocks (i.e., glue logic and/or functions to be implemented in PLDs). Major components such as processors, interconnects, and sensor interfaces have been selected, specified, and verified at least by simulation. Detailed design elaborates these designs into implementations by schematic capture, synthesis, place, route, autocode, and primitive optimization activities.

Physical prototypes are designed, fabricated, tested, and integrated with software per the program plan. The extent of the full system can range from a significant sub-assembly (i.e., a MCM or more typically a set of boards where each board type is present) to the complete system. Based upon these results the TSD's are updated with preliminary measurement data on fault population coverage. Design flaw data is collected based upon the extensive simulations of VP3 and the system prototype results. Preliminary results are collected on BIST coverage of manufacturing faults. If the program plan calls for a BIST demonstration, preliminary results are collected on BIST coverage of field support faults.

3.2.5.1 Inputs

3.2.5.1.1 Inputs from Overall RASSP Processes

iRa.	Reuse library	Step 1, 5-7
iRb.	VP2 Design description (detailed models, physical description)	Step 1-4
iRc.	VP2 Architecture and lower level component test benches	Step 1,3
iRd.	Physical prototype	Step 4-7

3.2.5.1.1 Inputs from DFT Processes

iDa.	TMAT Table	Step 1a, 1f
iDb.	Architecture-selection-level test strategy diagram, TSD2, and Testability Architecture, TA2, for selected processor system architecture.	Step 1b, 1c, 2, 4-7
iDc.	Architecture component test bench definitions.	Step 1e, 3
iDd.	Test plans for Board, MCMs, and special chips (i.e., ASICs, FPGAs, bare dir or KGD).	Step 1e, 2
iDe.	Test procedures for system acceptance and subsystem acceptance.	Step 1e, 2
iDf.	Equivalent DFG's for on-line BIST software	Step 1d

3.2.5.2 Process

1. For each design entity, perform the generic process shown in Figure 3-20.

   a. Using De, access the Test Metric/Tool Application Table (see Table A-1) and select required pair(s).

   b. Using Rd, capture the TSD and Test Architecture from the Architecture Definition stage. Also capture test requirements flowed down from the previous stage. Refine the TSD and test architecture at each level of packaging (check reuse library first).

   c. Using Df, in the hardware design path, incorporate DFT and BIST, per the testability architecture developed above (Check reuse). Incorporation should be done by synthesis at the highest VHDL level possible. Currently, synthesis is available only at the RTL level and only for chips. Ultimately, synthesis will involve higher packaging levels and earlier (behavioral) levels. Where synthesis is unavailable, the features must be inserted by designing them in concurrently with the functional design. VHDL models should be back-annotated to accurately reflect the DFT and BIST features.

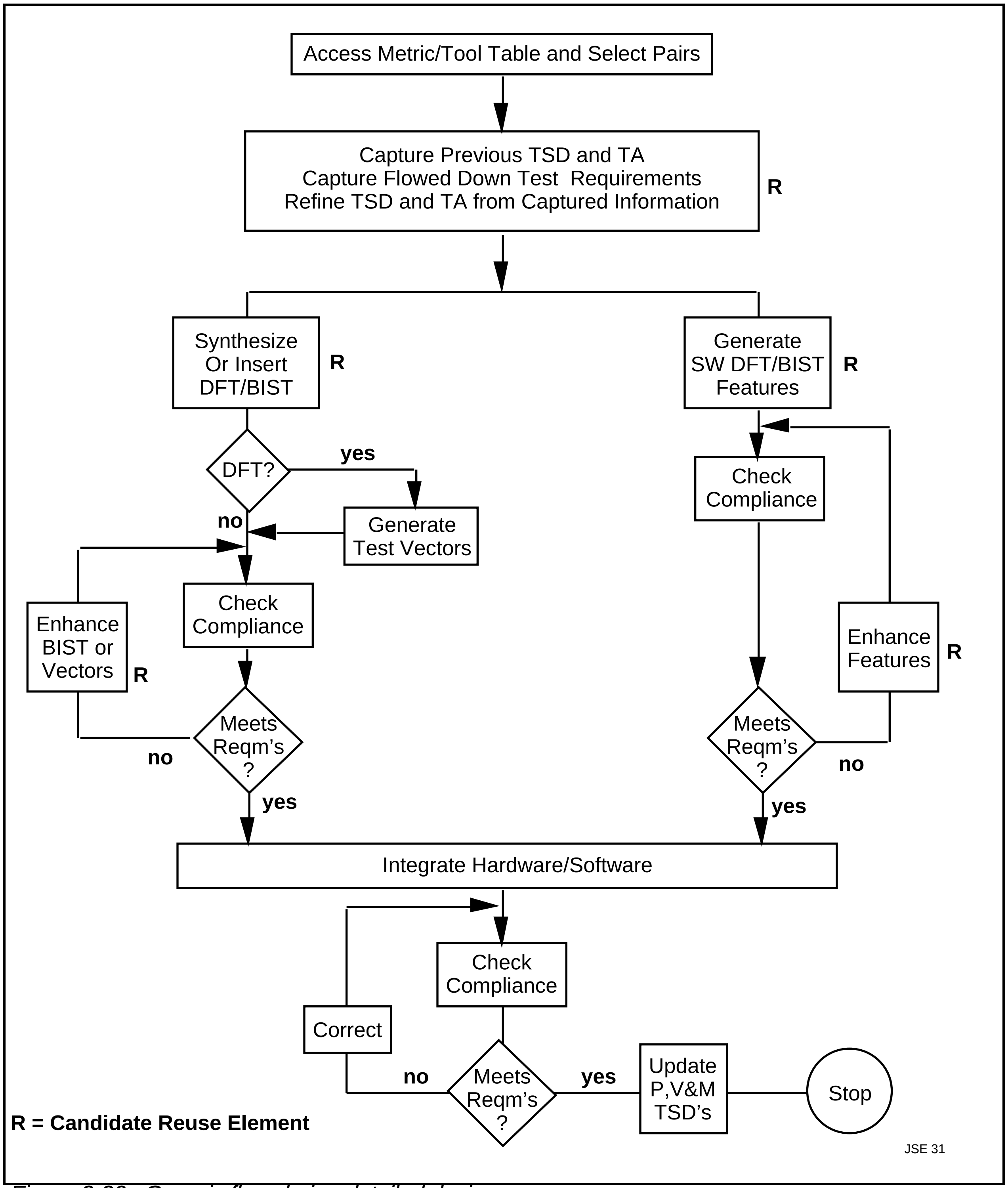

Access Metric/Tool Table and Select Pairs
Capture Previous TSD and TA
Capture Flowed Down Test Requirements
Refine TSD and TA from Captured Information
R
Synthesize Or Insert DFT/BIST
R
Generate SW DFT/BIST Features
R
DFT?
yes
no
Generate Test Vectors
Check Compliance
Check Compliance
Enhance BIST or Vectors
R
Enhance Features
R
Meets Reqm's ?
no
yes
Meets Reqm's ?
no
yes
Integrate Hardware/Software
Check Compliance
Correct
no
Meets Reqm's ?
yes
Update P,V&M TSD's
Stop
R = Candidate Reuse Element
JSE 31

Figure 3-20. Generic flow during detailed design.

Perform test domain analysis to stretch the reuse of all DFT/BIST structures. An example of this concept is shown in Figure 3-21, using the same hypothetical board. The source of BIST stimuli (8244) is analyzed to determine how much of the board (and eventually the rest of the system) can be tested using the PRPG and the PSA elements in the 8244s. A minimum of four domains (#1-4) are covered along with additional domains in the rest of the system. By adding loopback capabilities, fault isolation is improved; and the number of domains covered is doubled. Pseudorandom patterns emanate from the source 8244 and responses through the loopback are compressed in the parallel signature analyzer in the sink 8244. Test domain analysis can be used to stretch the limits of reuse across packaging levels, as well as across the same package level.

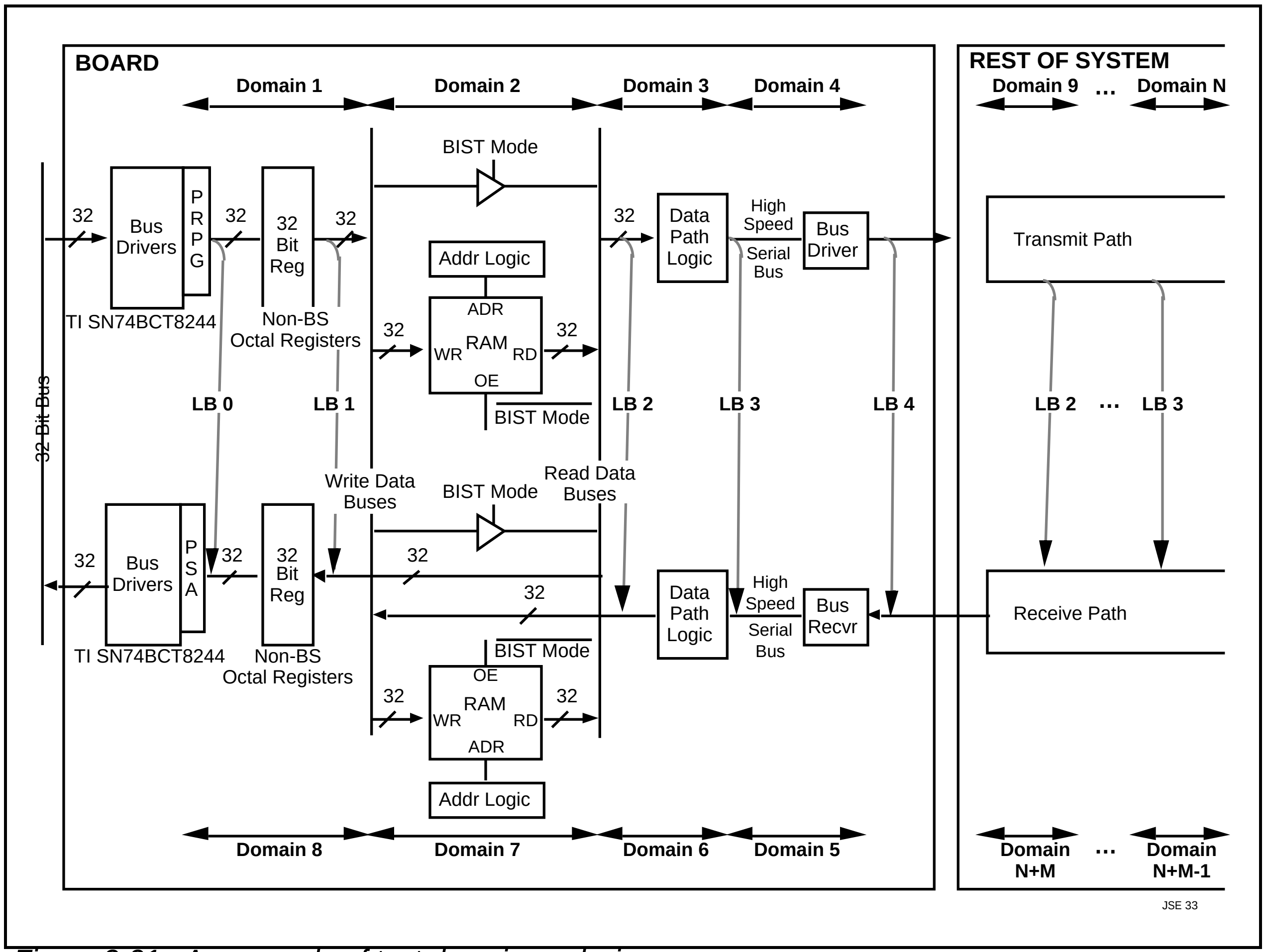


Figure 3-21. An example of test domain analysis.

d. Using Df, in the software path, DFT and BIST features are incorporated for both facilitating software testing, as well as for supporting hardware implemented DFT and BIST features (check reuse).

e. In the hardware path, test generation takes place next (Check reuse). There are several dimensions of test generation required in this step. Consider the life cycle phase of the test and the associated fault model (design flaws versus manufacturing defects versus field defects) and determine the nature of the patterns and their source:

1. Functional patterns used for verification of the functional (multiply-add, transfer data, etc.) correctness of the design. These come from VHDL test bench simulation efforts.

        2. Functional patterns used for verification of the test features (boundary-scan, internal scan, etc.) of the design. These also come from VHDL test bench simulation efforts.[3]
        3. Functional patterns used for design flaw testing or for at-speed, non-stuck-at fault testing. These may be derived from the VHDL test bench simulation efforts (and captured in WAVES).
        4. Test patterns use for structurally oriented testing (e.g., gate level stuck-at faults) which may require supplementary deterministic test pattern generation above the functional patterns developed above to meet fault coverage requirements.

    f. Compliance to requirements is checked by the appropriate metric/tool pair selected above for both hardware and software paths. Insufficient testability will result in an iteration in DFT enhancement.

    g. Integration of hardware/software, although in reality a continuous process through codesign, is shown as a separate block. The reason is that there is one more level of compliance checking to be performed and that is the joint DFT/BIST effects and capabilities of the final hardware/software codesigned elements.

    h. Prediction and verification TSDs are updated with the newly tracked values from the testbench simulation.

    i. The process described above does not cover the actual application of the tests, which is discussed below.

2. Using Db and Dd, refine and develop manufacturing and field test strategies, test architecture, test plans and procedures.

3. Using Rb, build upon existing VHDL test benches and structurally-oriented, physical prototype test-vectors to create production test-vector sets and field test-vector sets/TPSs. This becomes particularly productive and effective if a singular test strategy as espoused earlier was adopted.

4. Collect data on hardware and software design flaws and update the prediction and verification values for the design flaw section of the TSDs.

5. Collect test and BIST performance data and update the prediction and verification TSDs. Verify the effectiveness of DFT and BIST features encapsulated in the reuse library. Generate preliminary measurement TSD's as appropriate tests are conducted per the program plan.

6. Collect data on the impact of test on design time and cost. Update attributes in the TSDs. These data should be considered for encapsulation in the reuse library, to be used for default test time and cost values in initial TSDs for similar model years and systems.

7. Using Rd, evaluate test requirements, test strategies, test architectures, and vectors; data collection methods and forms; etc. for encapsulation in the reuse library.

[3] Functional patterns used for verification of the test features can only be used effectively at the structural VHDL model level, since the VHDL DID does not permit structure dependent signals to be modeled in behavioral bodies.

3.2.5.3 Outputs

3.2.5.3.1 Outputs to the Overall RASSP Processes

oRa.	Low level component test benches	Step 1, 3
oRb.	Verified functional hardware and software design, modified for DFT/BIST	Step 1

3.2.5.3.2 Outputs to DFT Processes

oDa.	Detailed, low level test requirements, test architecture and test strategy.	Step 1b,2, 7
oDb.	Compliance tracking data	Step 1f,1h,4-6
oDc.	Reuse elements	Step 7
oDd	Manufacturing and field test strategies, plans, procedures, test vector sets	Step 2
oDe.	Design flaw reports and statistics for TSD updates, manufacturing use, etc.	Step 4

3.2.5.4 Handling of COTS

Give priority to COTS selection as described in the plateau discussion earlier. In addition, use the "Lead, follow, or get out of the way approach" to dealing with COTS. This involves adding, or using existing, DFT features to the design to deal with COTS. As illustrated in Figure 3-19, COTS devices possessing BIST features, such as the TI SN74BCT8244 (having a PRPG and PSA mode), are given a "lead" role, since they can lead a test process. COTS devices, such as the four non-boundary-scannable octal flip-flops are given the "follow role," since, while they cannot lead, they at least do not impede the BIST test flow originating at the boundary-scan octals. Finally, the RAM chips, being very complex, and having no test features, coupled with an inability to pass pseudorandom patterns through during RAM tests, must be relegated to the "get out of the way" role which means bypassing them during BIST mode using a output tri-stating approach.

Checking for compliance for COTS sections will require defining the fault model, such that the coverage is measurable, using non-structural simulation models for the black box COTS elements or by using simplified functional fault models.

3.2.5.5 Risk management/Mini-Spirals

As before, this step could stimulate a mini-spiral to explore such things as a quickly emerging test or BIST/DFT technology that may become available in the project timeframe (e.g., board level BIST insertion, black box solutions.

3.2.5.6 Interface to Reuse Library

Any of the intermediate or final outputs of this stage could provide inputs to the reuse library.

3.2.5.7 Interface to RASSP Methodology and Tools

a. VHDL Test Benches
b. PRICE life cycle cost tool

3.2.6 Manufacturing Stage

3.2.6.1 Inputs

3.2.6.1.1 Inputs from Overall RASSP Processes

Ra. Reuse library

3.2.6.1.2 Inputs from DFT Processes

Da. Detailed, low level test requirements, tester architecture, test plans, test procedures, and test pattern sets (captured in WAVES) for testing integrated hardware and software at each level of packaging
Db. Compliance tracking data
Dc. Reuse elements
Dd Manufacturing and field test strategies, plans, procedures, test vector sets
De. Design flaw reports and statistics for TSD updates, manufacturing use, etc.
Df. Manufacturing defect reports
Dg. Reports from ESS testing
Dh. Reports from acceptance testing
Di. External Test and BIST performance data at all levels of packaging
Dj. Field test impact (cost and time) data
Dk. TMAT Table
Dl. Manufacturing test time and cost data

3.2.6.2 Process

a. Using Dr, access the Test Metrics/Tool Application Table (see Table A-1) and select required metric/tool pairs.

b. Using Da, set(s) of manufacturing test vectors, captured in WAVES, along with tester architecture, test plans and test procedures, are used for manufacturing test, along with BIST, at each level of packaging.

c. Using De, data are collected on latent hardware and software design flaws found in manufacturing and used to update the measurement values for the design flaw section of the TSDs.

d. Using Df, Dg, Dh, data are collected on manufacturing defects found and used to update the measurement values for the manufacturing section of the TSDs.

e. RMA data, if any, are collected and used to update the reliability model for failure rate weighting applications, and to measure compliance to the RMA requirements.

f. Using Di, test and BIST performance data are collected and used to update the verification and measurement TSDs, and to validate (or invalidate) the effectiveness of DFT and BIST features encapsulated in the reuse library.

g. Using Dl, impact data on manufacturing test time and cost are collected and used to update those attributes in the TSDs and are considered for encapsulation in the reuse library, to be used for default test time and cost values in initial TSDs for similar model years and systems.

h. Using Dg, feedback is provided on the value and effectiveness of the design phase ESS. (Did it help or hurt?)

i. Using Da and Dd, , manufacturing test strategies, test architectures, and vectors, data collection methods and forms, etc., are considered for encapsulation in the reuse library.

3.2.6.3 Outputs

3.2.6.3.1 Outputs to the Overall RASSP Processes

a. Updated reuse library

3.2.6.3.2 Outputs to DFT Processes

a. Updated verification and measurement TSDs
b. Design Flaw and Manufacturing Defect Data for TSD update and field use

3.2.6.4 Handling of COTS

Since the treatment of COTS in testing is a difficult problem when the COTS lacks testability features, the data collection discussed above should be particularly accurate for the COTS sections of the system, so that the method of handling COTS during the design phase can be validated or invalidated.

3.2.6.5 Risk Management/Mini-Spirals

Not applicable.

3.2.6.6 Interface to Overall RASSP Methodology and Tools

a. Preamp database/STEP information models

3.2.7 Field Support Phase

3.2.7.1 Inputs

3.2.7.1.1 Inputs from Overall RASSP Processes

Ra. Reuse library

3.2.7.1.2 Inputs from DFT Processes

Da. Detailed, low level test requirements, tester architecture, test plans, test procedures, and test pattern sets (captured in WAVES) for field testing of integrated hardware and software at each level of packaging
Db. Compliance tracking data
Dc. Reuse elements
Dd. Field test strategies, plans, procedures, test vector sets
De. Design flaw reports and statistics for TSD updates, field use, etc.
Df. Latent manufacturing defect reports
Dg. Field defect reports
Dh. External Test and BIST performance data at all levels of packaging
Di. Field test impact (cost and time) data
Dj. TMAT Table

3.2.7.2 Process

a. Using Dj, access the Test Metrics/Tool Application Table (see Table A-1) and select required metric/tool pairs.

b. Using Da, set(s) of field test vectors, captured in WAVES, along with tester architecture, test plans and test procedures, are used for field test, along with BIST, at each level of packaging.

c. Using De, data are collected on latent hardware and software design flaws found in manufacturing and used to update the measurement values for the design flaw section of the TSDs.

d. Using Df, data are collected on latent manufacturing defects found and used to update the measurement values for the manufacturing section of the TSDs.

e. Using Dg, data are collected on field defects found and used to update the measurement values for the field section of the TSDs.

e. RMA data, if any, are collected and used to update the reliability model for failure rate weighting applications, and to measure compliance to the RMA requirements.

f. Using Dh, test and BIST performance data are collected and used to update the verification and measurement TSDs, and to validate (or invalidate) the effectiveness of DFT and BIST features encapsulated in the reuse library.

g. Using Di, impact data on field test time and cost are collected and used to update those attributes in the TSDs and are considered for encapsulation in the reuse library, to be used for default test time and cost values in initial TSDs for similar model years and systems.

h. Using Dh and Di, feedback is provided on the value and effectiveness of the design phase ESS. (Did it help or hurt?)

i. Using Ra, , manufacturing test strategies, test architectures, and vectors, data collection methods and forms, etc., are considered for encapsulation in the reuse library.

3.2.7.3 Outputs

3.2.7.3.1 Outputs to the Overall RASSP Processes

a. Updated reuse library

3.2.7.3.2 Outputs to DFT Processes

a. Updated verification and measurement TSDs
b. Design Flaw, Manufacturing Defect, and Field Defect Data

3.2.7.4 Handling of COTS

Since the treatment of COTS in testing is a difficult problem when the COTS lacks testability features, the data collection discussed above should be particularly accurate for the COTS sections of the system, so that the method of handling COTS during the design phase can be validated or invalidated.

3.2.7.5 Risk Management/Mini-Spirals

Not applicable.

3.2.7.6 Interface to Overall RASSP Methodology and Tools

a. Reliability, maintainability, and availability modeling tools
b. Maintenance and logistics analysis tools
c. PRICE life cycle cost tool

3.3 Example of Application of the DFT Methodology

(TBD)

Figures 3-22a and -b show the hypothetical processing board and element. The board shown in Figure 3-22a could be packaged for commercial applications on a 6U VME board. Note, the architecture allows a variable number of nodes, memory array size and even memory types. Depending upon PE type, memory array size and number of PE's per node a family of multichip assemblies could be defined to accommodate various cost, size and weight tradeoffs. The processing element shown in Figure 3-22b could be implemented as a single chip, multichip assembly and/or a single chip plus memory.

3.4 Support for Upgradeability and Extensibility

Since the DFT Methodology is hierarchical, and its methods, such as the Test Strategy Diagrams and the test domain analysis, are independent of tools, test techniques, packaging approach, etc., the Methodology is very upgradeable and extensible.

3.5 Integration of DFT into RASSP Enterprise System

The RASSP program will deliver an integrated system called the RASSP system, which integrates the CAD tools used in the RASSP design process under a framework referred to as the enterprise framework. An *enterprise framework* provides the facilities and services necessary to integrate the automated processes of an enterprise.

In the RASSP system the enterprise framework provides support for work flow management, design data management, library management, computer-supported collaborative work and remote tool access. The *work flow management subsystem* of the RASSP enterprise system enables a RASSP system administrator to model and enforce a particular design methodology for a project. The *data management subsystem* of the enterprise framework provides facilities for configuration managing, and controlling access to design data files that may reside at various sites in a computer network. *Library management* in the RASSP system involves the release, cataloging, and searching of reusable design components.

Integration of DFT within the Enterprise System involves:

a. Capturing DFT process steps within the work flows and activity definitions.
b. Encapsulating DFT tools within the Enterprise Integration Framework.
c. Defining, selecting, developing (if required) and integrating initial items for the DFT reuse library elements.
d. Defining (and/or selecting), developing and integrating templates and standards for test related product data information for documentation and manufacturing release.

Each of these activities are described in the sections below.

3.5.1 Work Flow and Activity Definitions

Work flows are captured in IDEF-3X format. Each process step must have defined:

- Inputs (requirements, outputs of previous steps)
- Outputs (specifications for downstream steps, design items, results, candidate reuse library items)
- Controls (guidelines, reuse libraries)
- The state Mechanism (responsible individuals, tools)

The activity which takes place in each step is documented in Section 3 above and in the updated activity definitions which represent an instantiation of the more general steps in this document (i.e., specific tools selected as appropriate from the RASSP Design Environment).

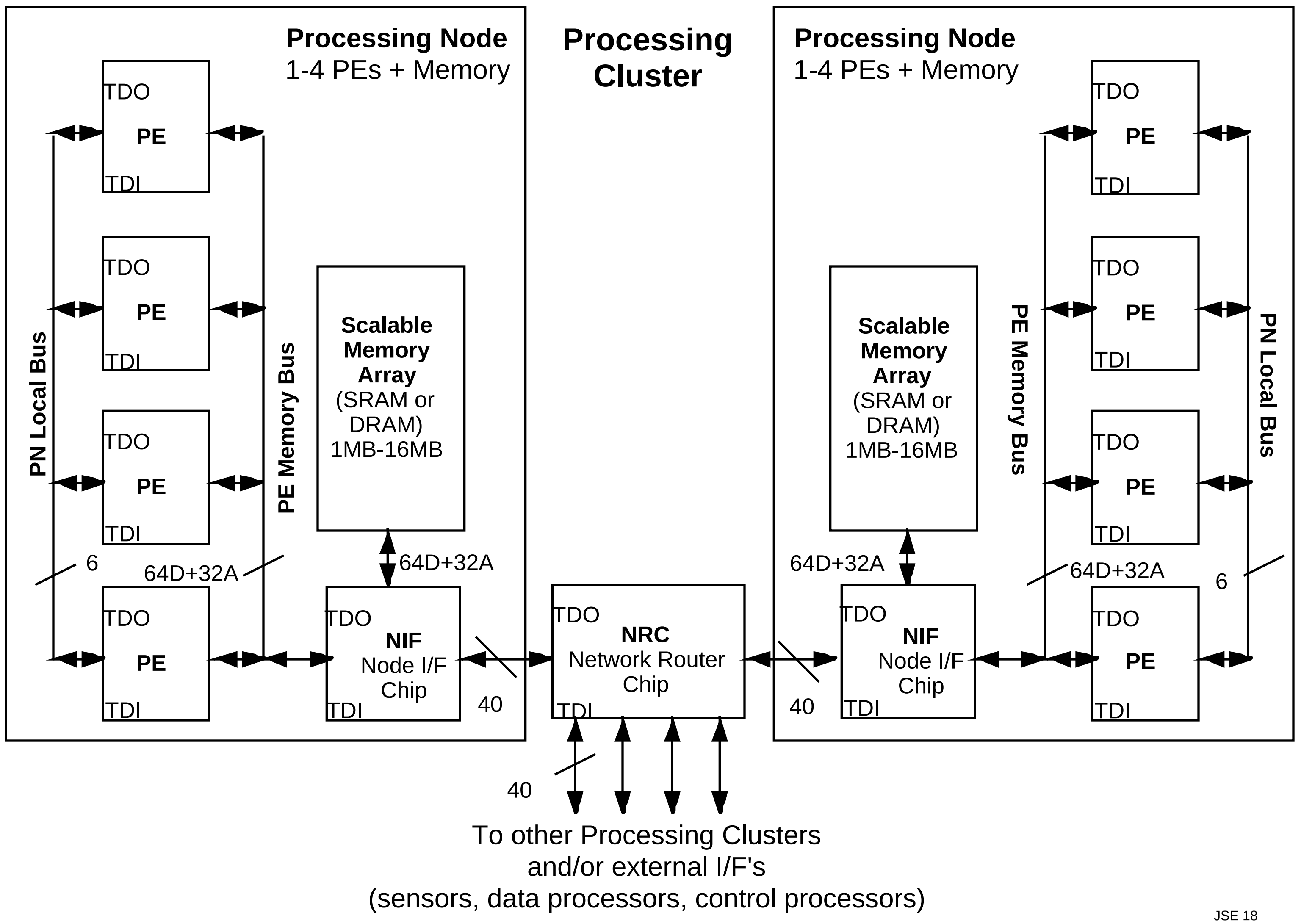


Figure 3-22a. Lockheed Martin ATL example processor board.

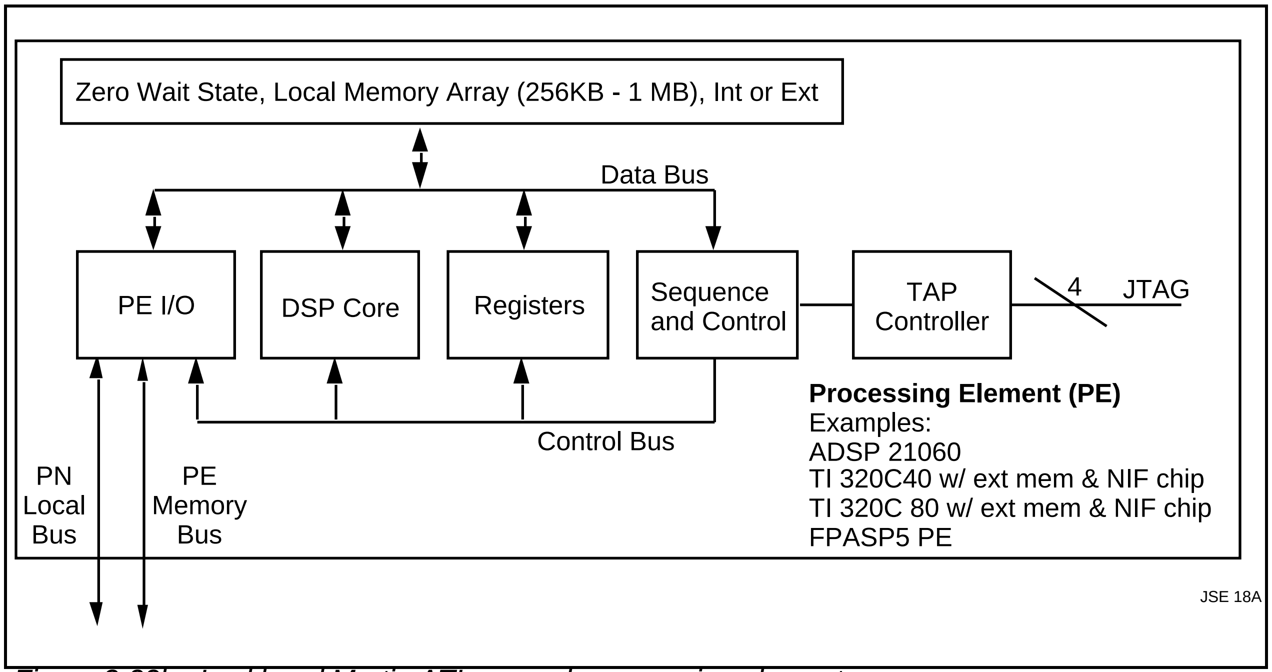


Figure 3-22b. Lockheed Martin ATL example processing element.

3.5.2 Tool Encapsulation

DFT specific tools and their data types are encapsulated within the Enterprise Framework. This integration allows the tools and data files for each process step (task) to be automatically "opened" upon initiation of the step. This ensures the designer has the tools and data required to perform the step and provides control and process monitoring for the project leader. Specifics of Enterprise Framework Integration are documented in "Enterprise Integration Framework System Requirements Specification" (9/94) and "Integration Procedures: Enterprise" (7/94).

3.5.3 DFT Reuse Libraries and Test Strategy Diagrams

Figure 3-23 shows the data flow diagram of the RASSP library management system, henceforth referred to as the RASSP Reuse Data Manager (RRDM). The RRDM stores descriptive data about all the reusable components released by the CAD tools. The user of a CAD tool invokes the RRDM from the CAD tool, and locates a reusable component by running a query on the descriptive data stored about the reusable components. The user may then view the reusable component using a standard viewer or a viewer specific to the tool that created the reusable component, and import the component into a design object.

The reusable components are stored in the format native to the tool that created it, and possibly in standard interchange formats (VHDL, ADA, EDIF, etc.). The design data about the reusable component is stored within the environment of the tool itself, while the descriptive data of the component is stored within the RRDM. The RASSP Enterprise Product Data Manager (EPDM) [Intergraph, 1994] manages and controls the access to the design data files of the reusable components. The descriptive data of the reusable components are modeled using the classes of an object-oriented class hierarchy, as shown in Figure 3-24.

One of the key ways in which the DFT Methodology contributes to the achievement of the RASSP goals is through the enforcement of reuse of DFT in four different dimensions. As described in Section 3.1.3.2 and pictured in Figure 3-2, the four dimensions of reuse are as follows:

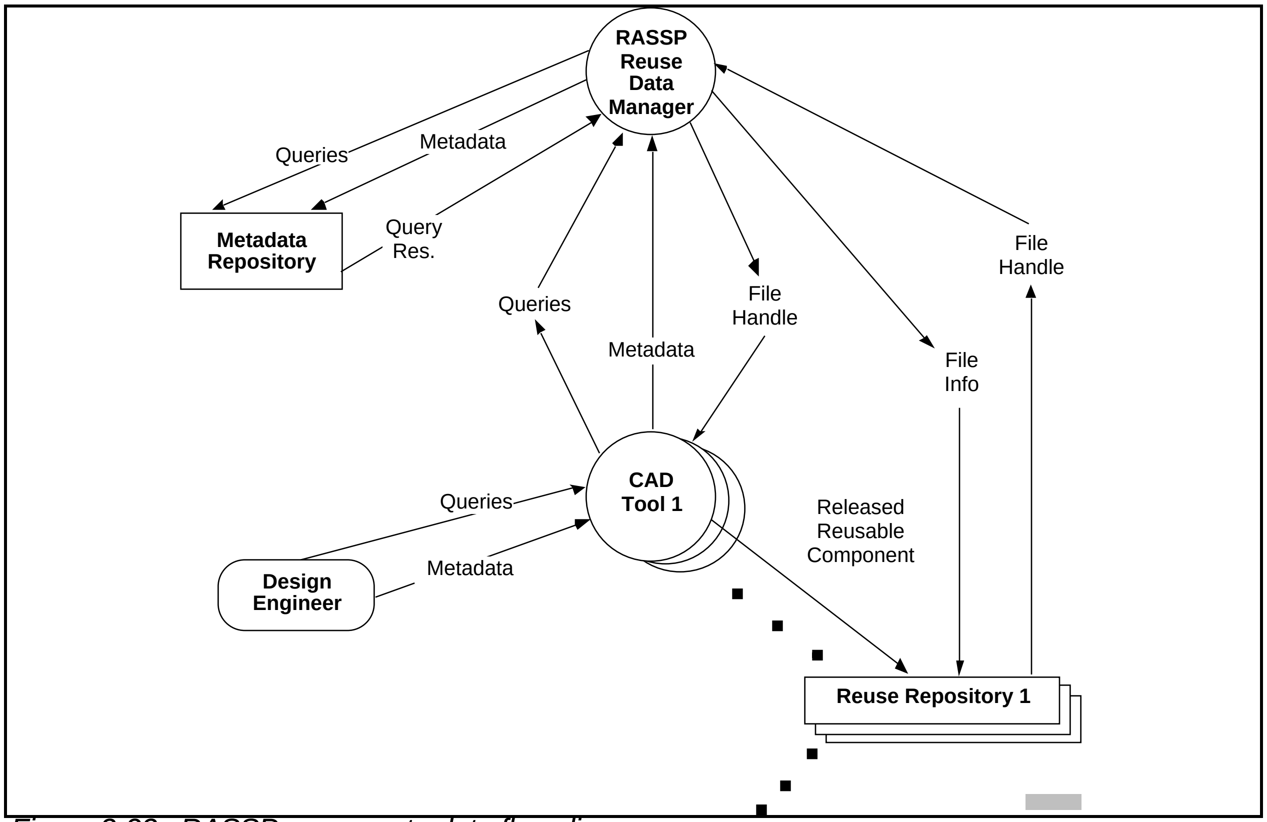


Figure 3-23. RASSP reuse meta data flow diagram.

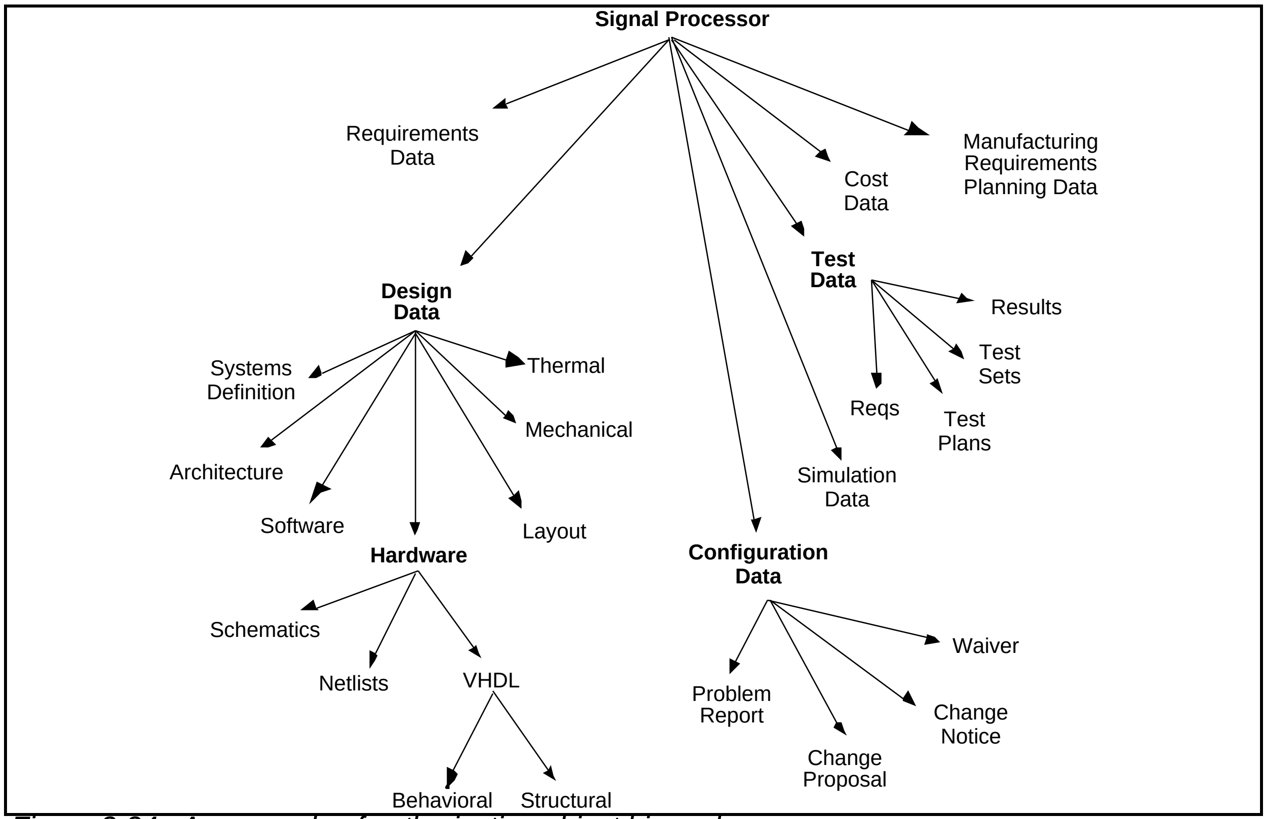


Figure 3-24. An example of authorization object hierarchy.

a. Across the life cycle phases within a given model year.
b. Across the packaging hierarchy within a given model year.
c. Across a single packaging level within a given model year.
d. As new model years unfold.

Not shown, but of key importance to design enterprise organizations is reuse across different product lines (which is similar to (d), as new model years unfold, for the purposes of the DFT methodology).

Reuse is used within the DFT methodology to provide the inputs of some steps in the process (rather than perform the step from scratch), and outputs of each process step become candidates for encapsulation in the reuse library for future model years or systems. Test related reuse items include, for example, test requirements; test strategies; DFT/BIST techniques for certain logic structures; testable chips, MCMs, etc.; BIST software modules; and test vector sets for certain library elements.

The use of the term reuse and reuse libraries must be clarified to understand how reuse in the DFT methodology relates to reuse within the Enterprise system (and RRDM).

Reuse - The process of utilizing selected elements that are outputs from any process step from earlier stages of the life cycle of a system, from other levels of the system hierarchy, from previous model years or from previous systems. Examples of re-use elements include requirements, application graphs or sub-graphs, design entities (i.e., VHDL modules), test strategies, BIST software, test vectors, techniques, documentation, etc.

Reuse Library - A set of encapsulated re-use elements plus documentation organized and/or cataloged for ease of selection and insertion. The reuse library must have a process to be executed by an organization in order to add reuse elements to the library systems. This process is to certify that things are worthy of being added to the reuse library and also to build the proper references in the reuse library system so that potential users of the reuse element are able to locate it (search based on attributes/characteristics, etc.).

Note that reuse is a process that can be controlled or not controlled. Under RASSP, entry of reuse elements into libraries is controlled via a structured process (see Figure 3-25) (reference: Kalathil, "Library Management Model for the RASSP System", 10/94). Libraries are needed for reuse across model years and product lines. DFT reuse library elements are managed within the enterprise system by RRDM. But the library paradigm as implemented in RRDM does not support the reuse proposed by the DFT methodology for:

a. Reuse across the life cycle phases within a given model year.
b. Reuse across the packaging hierarchy within a given model year.
c. Reuse across a single packaging level within a given model year.

Control and verification is required for these reuse elements just as it is required for library elements. This control and verification is implemented by the Test Strategy Diagrams described in this document. Details of the relationships between TSDs and RRDM are TBD. The baseline concept is to encapsulate TSDs under the test DOCH sub-hierarchy therefore providing a uniform view of reuse at the Enterprise level.

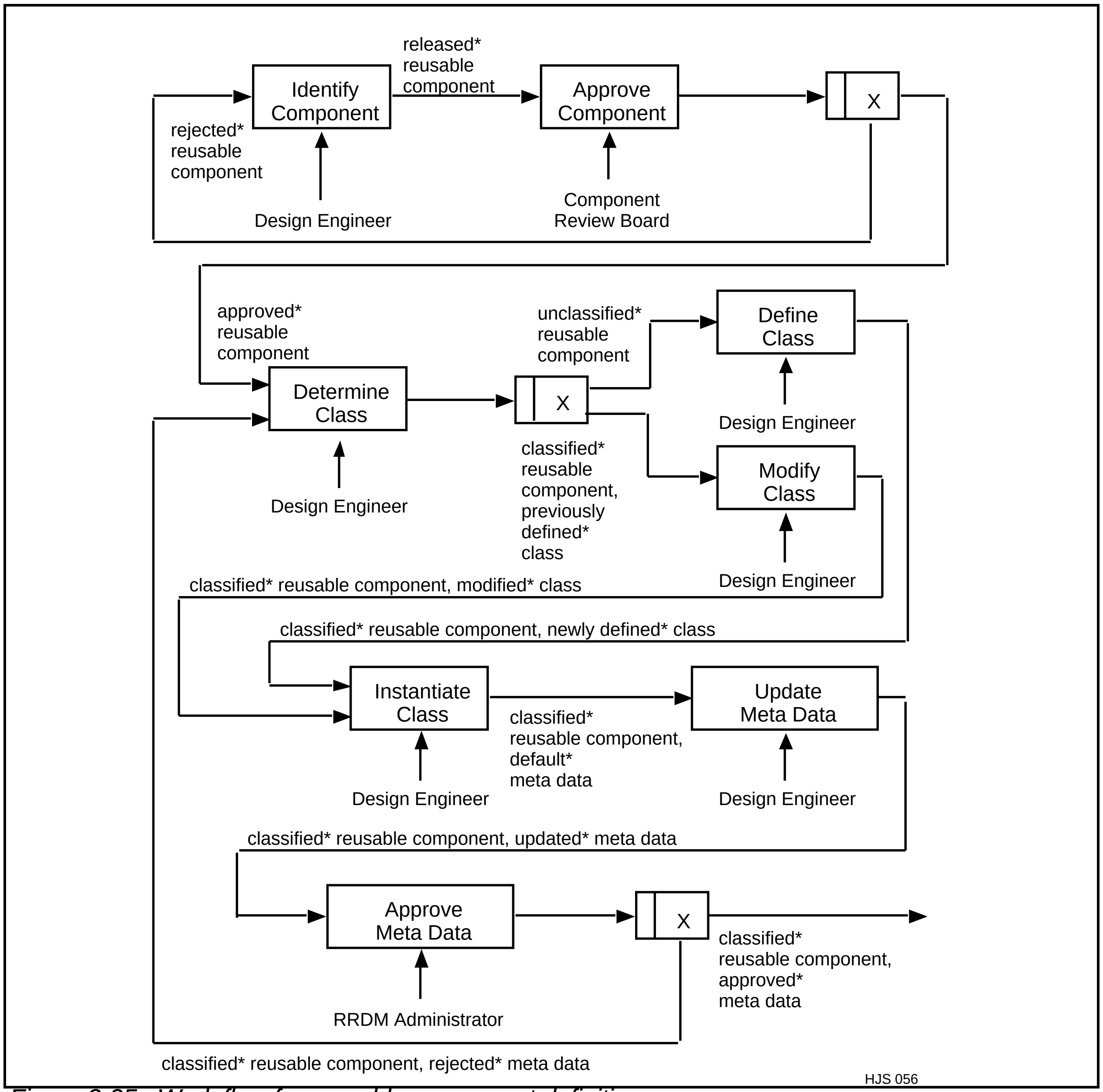
3.5.4 Test Related Product Data Management and Release

The DFT Methodology provides significant support for test related product data management. The traditional problems associated with 'walls' between design, production and field support are eliminated by unifying the test requirements for design verification, manufacturing and field test and then managing the design-for- testability process with the Test Strategy Diagrams. This provides a common framework for test requirements and solutions to be understood, traded off and leveraged between different items and organizations. The suggested Testability Architecture also provides significant support when followed. The use of BIST and hierarchical Test and Maintenance buses and controllers provides the embedding of 'test knowledge' within the product itself at the appropriate levels making diagnostics more effective, simpler to understand and easier to resolve.

Key items to be resolved yet include specific standards for managing and representing test information. Some of these standards are in the process of being developed under programs such as PAP-E. One key standard which is endorsed and used within the methodology is IEEE WAVES for the development and documentation of product test vectors.

Related Enterprise documents:

"PCA Manufacturing Interface: Requirements" (10/94)
"PC Manufacturing Interface Definition" (12/94)
"Parts Taxonomy for Manufacturing Library Information" 12/94



released* reusable component
Identify Component
Approve Component
X
rejected* reusable component
Design Engineer
Component Review Board
approved* reusable component
Determine Class
X
unclassified* reusable component
Define Class
Design Engineer
classified* reusable component, previously defined* class
Modify Class
Design Engineer
Design Engineer
classified* reusable component, modified* class
classified* reusable component, newly defined* class
Instantiate Class
Update Meta Data
classified* reusable component, default* meta data
Design Engineer
Design Engineer
classified* reusable component, updated* meta data
Approve Meta Data
X
classified* reusable component, approved* meta data
RRDM Administrator
classified* reusable component, rejected* meta data
HJS 056

Figure 3-25. Work flow for reusable component definition.

3.6 Contribution of the DFT Methodology to RASSP Goals

The DFT Methodology contributes to achievement of the overall RASSP goals in two ways:

First, adoption of DFT practices such as being developed and practiced within industry results in reduced cycle time, reduced cost, improved quality, predictable schedules (including integration and test) or in other words improved time to market (or more importantly time to profit). For example, companies have seen 4-5X reduction in board test time by using boundary scan based testing. (For further information on the benefits of DFT see references in Table 3-1 below).

Table 3-1. A sample of the numerous articles/literature which documents the benefits of DFT.

Ambler, et al	"The Economics of DFT", 3 part series in Evaluation Engineering, Sept., Oct., and Nov. 1994.	Discusses barriers to DFT, solutions for overcoming them and gives case histories of the impact of full scan and boundary scan
Bennets	"Progress in DFT: A personal View", IEEE Design and Test of Computers, Spring, 1994.	Defines Quif - Quality Improvement Factor
Milo	"Success with Boundary Scan", Evaluation Engineering, Feb. 1995.	Matsushita computed a 245% return on investment
Schlumberger Technologies	Boundary Scan - The Real Benefits of Test	Free disk & electronic document with case histories
Texas Instruments	IEEE 1149.1 Testability - Primer	Free Literature Book. Good overview of structured DFT and boundary scan

Secondly, the structured DFT methodology provides improvement of the DFT process itself compared to current industry practice. This is achieved by the introduction of proven system engineering practices such as the consolidation of test requirements and by leveraging the top down development f the overall RASSP methodology to flow-down the test strategies and architecture from the system to chip packaging levels and across life cycle phases of the product.

Specific contributions of the DFT Methodology to meeting the RASSP goals discussed in Section 2.2 is as follows:

a. Promotes concurrent engineering by providing a specific methodology for integrating test with design activities. The methodology integrates tightly with the RASSP methodology and leverages top down design, virtual prototyping and hardware/software co-design.
b. Enhances the probability of first pass success by providing specific steps to ensure requirements are consistent, valid and realizable and by providing a structured process for flowing requirements, strategies and architecture down to lower levels. The impact of errors on schedule and cost are minimized by incorporation of integrated diagnostics which detect, isolate and correct as appropriate and/or required.
c. Promotes a singular test strategy to reduce test development time and cost across the product life cycle (example, PC based boundary-scan test for design and manufacturing and then reuse of boundary scan test in the field via embedded boundary scan controllers).
d. Leverages reuse of any output of any DFT methodology step. Reuse for subsequent model years and other products is supported via the RASSP Reuse Data Manager within the RASSP System. In addition, a structured process and mechanism is provided for ensuring reuse of test resources within a model year:
   - Across the life cycle phases
   - Across the packaging hierarchy
   - Across a single packaging level

   Control and verification is required for these reuse elements just as it is required for library elements. This control and verification is implemented by the Test Strategy Diagrams described in this document. The test strategy diagram method enforces reuse analysis and knits all of the DFT Methodology steps together.
e. Provides a framework for codesign of DFT/BIST with functional hardware/software and integrates tightly with the RASSP methodology. The framework facilitates automation and high level synthesis.

4.0 SUMMARY

The RASSP DFT Methodology has been presented. It is viewed as an evolving methodology, which can be upgraded and extended effectively. The DFT methodology presented in this document bridges the gap between the overall design methodology and the specific work flows and tools being implemented in the RASSP System. It is tightly integrated with the overall methodology and the RASSP System. It directly supports the overall RASSP goal of 4X reduction in cycle time and cost.

The definition of testing was expanded to encompass all phases of the life cycle and to include as fundamental activities: detection, isolation and correction.

The methodology depends upon a commitment by the team (including management) to include DFT to enhance product quality and to reduce time to profit.

Process steps are included to check requirements for consistency, validity and reliability.

Test requirements are consolidated for design verification, manufacturing acceptance and field support to ensure a singular test solution.

Process steps are included to predict, verify and measure solutions to identify problems early and/or to provide feedback for subsequent model years.

Test strategies and architecture are shared and flowed down to ensure consistency and to minimize cost. A structured testability architecture based upon boundary scan and BIST is promoted.

The RASSP System (i.e., RASSP Reuse Data Manager) is used to control and verify test reuse library elements. Test Strategy Diagrams are used to control, enforce and verify reuse of test resources across and within levels of the system hierarchy. The TSDs also provide a basis for management (predict, verify and measure) of the requirements as they are flowed down.

5.0 GLOSSARY

5.1 Acronyms and Abbreviations

A/D	Analog to Digital
ABBET	A Broad Based Environment for Test
ASIC	Application-Specific Integrated Circuit
ATE	Automatic Test Equipment
ATPG	Automatic Test Pattern Generation
ATS	Automatic Test Sets
BFM	Bus Functional Model
BIST	Built-In-Self-Test
BIT	Built-In-Test
BITE	Built-In Test Equipment
CAD	Computer-Aided Design
CAE	Computer-Aided Engineering
CALS	Computer-Aided Logistics Support
CAM	Computer-Aided Manufacturing
CAPE	Computer-Aided Parametric Estimating
CASE	Computer-Aided Software Engineering
CAT	Computer-Aided Test
CDMS	Computer-Aided Design Data Management System
CDR	Critical Design Review
CE	Concurrent Engineering
CFG	Control Flow Graph
CFI	Computer-Aided Design Framework Initiative
CLMS	Component and Library Management System
CND	Cannot Duplicate
COTS	Commercial Off The Shelf
CPU	Central Processing Unit
CSCI	Computer Software Configuration Items
D/I/C	Detect/Isolate/Correct
DB	Database
DEMVAL	Demonstration/Validation
DFD	Data Flow Diagram
DFG	Data Flow Graph
DFT	Design-For-Testability
DMA	Direct Memory Access
DMM	Design Methodology Manager
DoD	Department of Defense
DOM	CFI Standard for Design Object Management
DR	CFI Standard for Design Representation
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
DT&E	Development Test and Evaluation
E-Specs	Electronic Specifications (of electronic equipment)
EDA	Electronic Design Automation
EDIF	Electronic Data Interchange Format
EDM	Enterprise Desktop Manager
EPDM	Enterprise Product Data Management
EPI	Engineering Process Improvement
ESS	Environmental Stress Screening
EXPRESS-G	Information Modeling Language - Graphical Representation
FMEA	Failure Modes and Effects Analysis
FMECA	Failure Modes and Effects Criticality Analysis
FPGA	Field-Programmable Gate Array
FSED	Full-Scale Engineering Development
GFE	Government-Furnished Equipment

GFI	Government-Furnished Information
GFLOPS	Billion Floating-Point Operations per Second
GOTS	Government Off the Shelf
GUI	Graphical User Interface
HCI	Human-Computer Interface
HDI	High-Density Interconnect
HDL	Hardware Description Language
HOL	High-Order Language
HTML	Hyper Text Markup Language
HW	Hardware
I/NFM	Intergraph Network File Manager
I/O	Inputs/Outputs
IC	Integrated Circuit
ICD	Interface Control Document
IDDq	Quiescent Source to Drain Current Test
IDEF3	Standard workflow graphical representation format
IEEE	Institute of Electronics and Electrical Engineers
ILS	Integrated Logistics Support
IPDT	Integrated Product Development Team
IPPD	Integrated Product/Process Development
IISA	Instruction Set Architecture
ISO	International Standards Organization
ITC	CFI Standard for Intertool Communication
JCALS	Joint Computer-Aided Logistics Support
JIAWG	Joint Integrated Avionics Working Group
JTAG	Joint Test Action Group (IEEE 1149)
LCC	Life Cycle Cost
LMS	Library Management System
LOCST	LSSD On-chip Self Test
LRU	Line Replaceable Unit
LSA	Logistic Support Analysis
M-data	Measurement Results Data
MANTECH	Manufacturing Technology
MCM	Multi-Chip Assemblies
MCM	Multi-Chip Modules
MFLOPS	Million Floating-Point Operations per Second
MMC	Martin Marietta Corporation
MOE	Measure of Effectiveness
MTBF	Mean Time Between Failures
OO	Object-Oriented
OS	Operating System
OSI	Open Systems Interconnect
P-data	Prediction Results Data
PCA	Printed Circuit Assembly
PCB	Printed Circuit Board
PDCM	Product Data Control Module
PDES	Product Data Exchange using STEP
PDM	Product Data Manager
PDR	Preliminary Design Review
PDT	Product Development Team
PE	Processing Element
PGM/DFL	Parallel Graph Method/Data Flow Language
PGSE	Programming Graph Simulation Environment
PLD	Programmable Logic Device
PMB	Performance Measurement Baseline
PMO	Program Management Office
PreAMP	Pre-Competitive Advanced Manufacturing Process

PRICE	Parametric Review of Information for Costing and Evaluation
PRR	Production Readiness Review
PWA	Printed Wiring Assembly
QA	Quality Assurance
R&M	Reliability and Maintainability
RAM/ILS	Reliability, Availability, Maintainability/Integrated Logistics Support
RASSP	Rapid Prototyping of Application-Specific Signal Processors
RDBMS	Relational Database Management System
RDD-100	System Design CAD tool - Ascent Logic
RMWG	RASSP Methodology Working Group
RTL	Register Transfer Level
RTM	Requirements Traceability Matrix
RTOK	Retest OK
SDR	System Design Review
SEM	Standard Electronic Module
SEMP	System Engineering Management Plan
SEMS	System Engineering Management Schedule
SMT	Surface Mount Technology
SOW	Statement of Work
SP	Signal Processor
SQL	Standard Query Language
SRAM	Static Random Access Memory
SRR	System Requirements Review
SRS	System Requirements Specification
STEP	Standard for The Exchange of Product Data
STUMPS	Self-Test Using MISR and Parallel SRSG
SW	Software
SWAP	Size, Weight, and Power
T&E	Test and Evaluation
T&M Bus	Test and Maintenance Bus
TAP	Test Access Port
TBD	To Be Determined
TBR	To Be Resolved
TES	CFI Standard for Tool Encapsulation
TPS	Test Program Set
TRD	Test Requirements Document
TSD	Test Strategy Diagram
UUT	Unit Under Test
V-data	Verification Results Data
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuits
VLSI	Very-Large-Scale Integration
VTEST	Virtual Test
WAVES	IEEE Standard for Waveform & Vector Exchange
WBS	Work Breakdown Structure

5.2 Terms

Anomaly	A departure from the required behavior or design to be detected, isolated and corrected. Refers to the aggregate of physical faults and design flaws. The hierarchy for physical anomalies is defects, which cause failures, which are modeled as faults, and may manifest themselves as errors in system operation. These apply to both the manufacturing and field environments. They (from defect on) may be solid, transient, or intermittent. The design anomaly hierarchy is simply flaw and error.

Architectural Body	A VHDL construct that is used to define the semantics of a design entity. The semantics are defined in terms of mechanisms that change the values of the signals attached to the output ports of the design entity interface. There may be more than one architecture body for a design entity. All architecture bodies for a design entity share the same interface.
Architecture Selection	Architecture selection is the heart of the RASSP HW/SW codesign which utilizes a library based, DFG driven approach to software development combined with an iterative performance trade-off analysis to support rapid selection/analysis of candidate architectures.
Architecture Verification	Architecture verification is an iterative, hierarchical process whose role is to verify the functionality and detailed performance of a candidate architecture using a combination of testbed hardware, simulator(s), and or emulator(s) prior to detailed hardware implementation.
Back Annotation	The process of assigning values to model attributes as the result of the use of an external assessment tool; especially in assigning precise timing values within higher abstract models from more detailed lower level model simulations.
Behavioral Model	An abstract, high-level VHDL description which expresses the function and timing characteristics of the corresponding physical unit independent of any particular implementation, especially devoid of specific internal structure.
BIST (Built-In-Self-Test)	The capability for an item to test itself, with minimal or no external test equipment. BIST may be implemented in hardware or software at any level of packaging. In this document, BIST is considered to be synonymous with "BIT" and "self-test." Furthermore, BIST is considered to encompass "diagnostics" when it includes a fault isolation capability. Thus, BIST may provide detection, isolation (diagnostics), and possibly correction (with fault tolerance).
Bus Functional Model	Used to define the operation of a component with respect to its surrounding environment. The interface between the component and its environment are modeled in detail, even though all of the functions internal to the component do not have to be modeled, particularly not at the same level of detail.
Co-Simulation	The term "co-simulation" is used in two contexts: In the context of hardware/software co-simulation, the term refers to the act of simulating the execution of software on target hardware. This is accomplished through a hardware simulation of the target hardware interpreting software instructions. In the context of other domains, besides HW/SW, the term refers to the act of cooperatively running multiple distinct simulators concurrently with inter-process communication between them. Each simulator is simulating a distinct section or aspect of the target system. This can apply to simulations within the same domain, such as Verilog and Quicksim, or to simulators in divergent domains, such as Spice, VHDL, and SPW.
Command Program (CP)	The part of a software program which sequences and controls data flow graphs (DFG's).

Component	A component is any logically separable hardware unit. They can be combined to form a higher level component by being interconnected. Thus components are directly related to the nodes in the design hierarchy. The VHDL DID requires that VHDL model components correspond to physical components.
Correction	The process of removing the cause (maintenance) or effects (fault tolerance) of a design flaw, manufacturing defect or physical fault.
Data Flow Graph (DFG)	A directed graph that depicts information flow between signal-processing primitive operations as "arcs" and the transforms or operations that are applied on the data as "nodes".
Debugging	A form of testing associated with the detection, isolation, and correction of design flaws only.
Defect	A physical breakdown of an interconnection, such as a foil trace, connector, or cable, or a physical breakdown of a device, such as a transistor, resistor, capacitor, etc.
Design Entity	An entity interface together with an associated architecture body defines a design entity. Different design entities may share the same entity interface but employ different architecture bodies.
Design Flaw	A mistake in the design or implementation of a circuit, assembly or software routine which may result in an error in system operation.
Design-For-Testability	The incorporation of a capability in any embodiment of any part of a system that will facilitate the processes of detecting, isolating, and correcting anomalies (design flaws, manufacturing defects, and field defects
Detection	The process of determining the presence of an anomaly, including design flaws, manufacturing defects, and field defects.
Diagnosing	The phase of testing associated with locating the source of the anomaly.
Diagnostics	The fault isolation capability of DFT and BIST. Diagnostics, when implemented, is considered to be an integral part of DFT and BIST and not a separate capability.
DSP (Digital Signal Processor)	A processor system specialized for the computation of signal processing algorithms. It usually consists of many programmable processor elements interconnected via networks to each other and to memory, sensors, displays and other external devices. It is often distinguished from general purpose- or data- processors in that it must operate in real-time, it often has a much higher data input rate, and it usually must perform a higher percentage of mathematical, often floating-point, operations.
Embodiment	One form or instantiation of a system or its parts.
Error	Incorrect behavior of a system, sub-assembly or logic circuit, due to the effects of a design flaw or the propagation of a physical fault through at least one level of gating.
Executable Specification (ESpec)	A description of a component or system that can be executed in a computer simulation to reflect the precise behavior of the intended device. Currently, E-Specs may often be restricted to describing only specific aspects of the component or system such as timing, performance, or function.

Failure	Incorrect transistor level behavior of a logic circuit, due to the presence of a defect.
Fault	Incorrect gate level behavior, due to the presence of a failure.
Fault Ambiguity Analysis	An analysis of a system which identifies fault ambiguity groups (groups of components or modules for which a fault cannot be isolated within the group and, hence, repair would cause all items to be replaced).
Functional Analysis	The process of decomposing system requirements into functional blocks which define the behavior of the system.
Functional Model	A model of a hardware system that describes the response of the system to stimuli in a way that is independent of any implementation, and does not provide any information about the timing characteristics of the system being modeled.
Hardware/Software Codesign	The joint development and verification of both hardware and software via simulation/emulation from the hardware/software partitioning of functionality through design release. HW/SW codesign should result in the development of a virtual prototype (see definition for virtual prototype).
Instruction Set Architecture (ISA)	Describes the externally visible state of a programmable processor and the functions that the processor can perform. An ISA level model of a processor will execute any machine program for that processor and give the same results as the physical machine, as long as all input stimuli are sent to the ISA level model simulation on the same simulated clock cycle as they arrive at the real processor.
Interoperable	Two VHDL models of the same module are interoperable if one model can be substituted for the other without introducing errors into the system. Two VHDL models are also interoperable if they can be connected together as components without introducing errors into the system.
Isolation	The process of determining the location of a design flaw, manufacturing defect or field fault in a unit under test.
Leaf Module	A design entity that has no associated structural architecture body. Examples of possible leaf modules for a structural VHDL model include power supplies, analog circuit blocks, and digital logic gates.
Measurement	The process of determining conformance to a requirement has been met by measurements, tests and/or data collection.
Methodology	The body of rules employed by a [engineering] discipline; a particular procedure or set of procedures.
Performance Model	A model which exhibits the measures of quality of a design that relate to the timeliness of the system in reacting to stimuli. Measures associated with performance include response time, throughput, and utilization.
Prediction	The process of determining conformance to a requirement has been met by analysis and/or comparison with similar library elements.
Primitive	A software routine (or set of routines) that completes a function within a data flow graph (DFG).

Process - See Methodology	No distinction is made between methodology and process.
Processor Element	A programmable device that is one of many that operate cooperatively through a network in a processor system.
Prototype Library Element	A hardware model or software primitive(application or OS service) which is a candidate for inclusion in the reuse library. It has not been fully tested, validated and documented. It is however available for use in the early stages of HW/SW codesign for high level architecture tradeoffs.
Register Transfer Level (RTL) Model	Describes a system in terms of registers, combinational circuitry, low level buses, and control circuits, usually implemented as finite state machines.
Re-Use	The process of utilizing selected elements that are outputs from any process step from earlier stages of the life cycle of a system, from other levels of the system hierarchy, from previous model years or from previous systems. Examples of re-use elements include requirements, application graphs or sub-graphs, design entities (i.e., VHDL modules), test strategies, BIST software, test vectors, techniques, documentation, etc.
Re-Use Library	A set of encapsulated re-use elements plus documentation organized and/or cataloged for ease of selection and insertion.
Structural Model	Represents a system or component in terms of the interconnection topology of the set of internal components.
Subsystem	A major component of a system. For RASSP, the signal processor is considered as a subsystem.
Synthesis	The process of creating a representation of a system at a lower level of abstraction from a higher level of abstraction. The synthesized representation should have the same function as the higher level representation.
System	Depending upon one’s perspective a system could represent a platform, sensor system, signal processor or processing board. For RASSP, a system represents a sensor system such as a radar, sonar or infrared sensor system.
System Configuration	The system configuration consists of the major subsystems which makeup the system. The major components of a RASSP system include the exciter, transmitter, antenna, receiver, signal processor and data processor.
System Definition	The process of analyzing customer requirements, performing functional analysis and system synthesis, and performing system level tradeoffs to determine the functional and performance specifications for each subsystem.
System Requirements Analysis	The process of analyzing and interpreting system requirements with the customer to refine the purpose and manner in which the user will operate the system.
System Synthesis	The process of performing top level system tradeoffs to allocate the functional requirements into performance and physical specifications for each subsystem.
Target Hardware	The hardware that is the result of the design process, as distinguished from hardware used in the design process.

Test And Maintenance Buses	A hierarchy of standardized buses used for communication of test information between test and maintenance controllers. Examples are the IEEE 1149.1 and 1149.5 buses.
Test And Maintenance Controllers	A hierarchy of functions used to control system test and maintenance activities. The functions communicate through a hierarchy of standardized test and maintenance buses.
Test Architecture	A suite consisting of the UUT and any test equipment required, based on the developed test strategy.
Testability Architecture	Specification of the DFT and BIST features in the UUT.
Tester Architecture	Specification of the configuration of test equipment required for any externally based testing.
Test Bench	A VHDL test bench is a collection of VHDL modules which apply stimuli to a module under test (MUT), compare the MUT's response with an expected output, and report any differences observed and expected responses during simulation.
Test Means	A vehicle used to detect, isolate, and possible correct an item under test. The nature of the vehicle and item under test depends on the life cycle step and level of system hierarchy at which the test is applied.
Test Strategy	A 2D matrix of ordered test means for detecting, correcting and isolating a total population of items. Examples of items to be D/I/C include, design flaws, manufacturing defects and field defects.
Test Strategy Diagram	The test strategy diagram (TSD) is a technique which is used in the DFT Methodology to "knit" all processes together and to provide a means of carrying information between and within process steps.
Testability	An attribute of a design at any level of abstraction that reflects the ease with which the item can be tested. Testability is considered poor if any characteristic of the item under test makes it difficult to generate, evaluate, or apply tests.
Testing	The process of detecting, isolating, and correcting an anomaly arising from any phase of the system's life cycle.
Validated Library Element	A prototype library element which has been designed for reuse, tested, validated and documented according to the standards defined for the reuse library.
Verification	The process of determining conformance to a requirement has been met by prototyping, simulation and/or detailed analysis.
Virtual Prototype	The set of simulation models that comprises a prototype processor. When exercised, the virtual prototype should behave (function and performance) as closely as possible to its physical counterpart.

6.0 REFERENCES

6.1 RASSP Documents

a. RASSP Methodology, Version 1.0, December 1994.
b. RASSP Model Year Architecture Working Document, Version 1.0, October 28, 1994.
c. Testability Architecture Description, Version x.x, in-process.
d. CAD System Description, Baseline 1.0, CDRL A007, Ver. 2.0, 3/8/95 (in-progress draft).

6.2 Non-RASSP Documents

The Documentation of Military Electronic Computers with the VHDL Handbook, Preliminary Draft Manuscript, Section VIII, Modeling Testability with VHDL Models, April 16,1993.

Flynn, et al., "Using WAVES in a Top-Down Design Methodology", Proc. of VIUF Fall 1994 Conf., Component Modeling, paper 12.1.

Sanchez, "Concurrent Engineering with DFT in the Digital System: A Parallel Process", Proc. of ITC 94, paper 36.1.

TABLE A-1. TEST METRICS/TOOL APPLICATION TABLE (TMAT)
5/4/95

PROCE SS STEP	STEP NAME	METRIC	TOOL (T) OR METHOD (M) EXAMPLES
3.2.1.2	SYSTEM DEFINITION	REQM'S CHK (REALIZABLE, CONSISTENT, VALID) MANAGEMENT COMMITMENT	RTI TOOL OR MANUAL ANALYSIS MANUAL ASSESSMENT OF TOOLS, RESOURCES, SCHEDU
3.2.2.2	FUNCTIONAL DESIGN	TEST MEANS EFFECTIVENESS FOR SELECTION	HISTORICAL DATA OR REUSE DATA
3.2.3.2	ARCHITECTURE SELECTION (ANOMALY STATISTICS)	DESIGN FLAWS	DESIGN PROBLEM REPORTIN (M, T)
3.2.3.2	ARCHITECTURE SELECTION (DFT/BIST IMPACT)	ANALYSIS DATA FOR BIST IMPACT ON PERFORMANCE	STATISTICAL DATA BASED ON SELECTED TESTABILITY ARCHITECTURE
3.2.3.2	ARCHITECTURE SELECTION (TESTABILITY ANALYSIS DATA)	TESTABILITY ANALYSIS DATA AT THE ARCHITECT. LEVEL	TDM ANALYSIS AT THE ARCHITECT. LEVEL
		AMBIGUITY GROUP ANALYSIS	TDM ANALYSIS AT THE ARCHITECT. LEVEL
3.2.3.2	ARCHITECTURE SELECTION (SIMULATION PERFORMANCE STATISTICS)	FLAW DETECTION COVERAGE OF DESIGN FLAWS	SIMULATION LOG; MANUAL L
		FLAW ISOLATION COVERAGE OF DESIGN FLAWS	SIMULATION LOG; MANUAL L
		FLAW CORRECTION COVERAGE OF DESIGN FLAWS	MANUAL LOG
3.2.3.2	ARCHITECTURE SELECTION (COST DATA)	COST OF TDM ANALYSIS	TDM ANALYSIS LOG
		COST OF SIMULATION	SIMULATION LOG
3.2.4.2	ARCHITECTURE VERIFICATION (ANOMALY STATISTICS)	DESIGN FLAWS	DESIGN PROBLEM REPORTIN (M, T)
3.2.4.2	ARCHITECTURE VERIFICATION (DFT/BIST IMPACT)	ANALYSIS DATA FOR BIST IMPACT ON PERFORMANCE	STATISTICAL DATA BASED ON SELECTED TESTABILITY ARCHITECTURE

PROCE SS STEP	**STEP NAME**	**METRIC**	**TOOL (T) OR METHOD (M) EXAMPLES**
3.2.4.2	ARCHITECTURE VERIFICATION (TESTABILITY ANALYSIS DATA)	TESTABILITY ANALYSIS DATA AT THE ARCHITECT. LEVEL	TDM ANALYSIS AT THE ARCHITECT. LEVEL
		AMBIGUITY GROUP ANALYSIS	TDM ANALYSIS AT THE ARCHITECT. LEVEL
3.2.4.2	ARCHITECTURE VERIFICATION (SIMULATION PERFORMANCE STATISTICS)	FLAW DETECTION COVERAGE OF DESIGN FLAWS	SIMULATION LOG; MANUAL L
		FLAW ISOLATION COVERAGE OF DESIGN FLAWS	SIMULATION LOG; MANUAL L
		FLAW CORRECTION COVERAGE OF DESIGN FLAWS	MANUAL LOG
3.2.4.2	ARCHITECTURE VERIFICATION (COST DATA)	COST OF TDM ANALYSIS	TDM ANALYSIS LOG
		COST OF SIMULATION	SIMULATION LOG
3.2.5.2	DETAILED DESIGN (ANOMALY STATISTICS)	DESIGN FLAWS BY SYSTEM AND POPULATION OF SYSTEMS	DESIGN PROBLEM REPORTIN (M, T)
		PROTOTYPE ASSEMBLY FAULTS	PROTOTYPE PROBLEM REPORTING (M, T)
3.2.5.2	DETAILED DESIGN (TESTABILITY ANALYSIS DATA)	TESTABILITY ANALYSIS DATA AT THE CHIP, BOARD LEVEL	C/O ANALYSIS OR ATPG AND FAULT SIM OR PROB FAULT GRADING AT THE CHIP LEVEL C/O ANALYSIS (E.G., TEST ACCESS ANALYZER) OR ATPG AND FAULT SIM OR PROB FAU GRADING AT THE BOARD LEV TDM ANALYSIS AT THE SYSTE LEVEL
3.2.5.2	DETAILED DESIGN (BIST PERFORMANCE STATISTICS)	FAULT DETECTION COVERAGE OF PROTOTYPE DEFECTS	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT ISOLATION COVERAGE OF PROTOTYPE DEFECTS	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT CORRECTION COVERAGE OF PROTOTYPE DEFECTS	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FALSE ALARM RATE	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT DETECTION TIME FOR PROTOTYPE DEFECTS	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)

PROCESS STEP	**STEP NAME**	**METRIC**	**TOOL (T) OR METHOD (M) EXAMPLES**
		FAULT ISOLATION TIME FOR MFG. DEFECTS	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT CORRECTION TIME FOR PROTOTYPE DEFECTS	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		AVERAGE AND MAXIMUM AMBIGUITY GROUP SIZE	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
3.2.5.2	DETAILED DESIGN (BIST PERFORMANCE PREDICTION)	FAULT DETECTION COVERAGE AT CHIP, BOARD, AND SYSTEM LEVELS	FSIM OR PROB FAULT GRADI AT CHIP AND BOARD LEVEL; INTEGRATION OF LOW LEVEL VALUES, PLUS BACKPLANE & CABLE COVERAGE AT SYSTEM LEVEL; ALSO, BS BASED FAUL INSERTION AT BOARD AND SYSTEM LEVEL
		FAULT ISOLATION COVERAGE	FSIM OR PROB FAULT GRADI AT CHIP AND BOARD LEVEL; INTEGRATION OF LOW LEVEL VALUES, PLUS BACKPLANE & CABLE COVERAGE AT SYSTEM LEVEL; ALSO, BS BASED FAULT INSERTION AT BOARD AND SYSTEM LEVEL
		FAULT CORRECTION COVERAGE	FSIM OR PROB FAULT GRADI AT CHIP AND BOARD LEVEL; INTEGRATION OF LOW LEVEL VALUES, PLUS BACKPLANE & CABLE COVERAGE AT SYSTEM LEVEL; ALSO, BS BASED FAUL INSERTION AT BOARD AND SYSTEM LEVEL
3.2.5.2	DETAILED DESIGN (MFG AND FIELD TEST PERFORMANCE STATISTICS)	FAULT DETECTION COVERAGE AT CHIP, BOARD, AND SYSTEM LEVELS	ATPG AND FSIM OR PROB FAU GRADING AT CHIP AND BOAR LEVEL; INTEGRATION OF LOW LEVEL FC VALUES, PLUS BACKPLANE & CABLE COVERAGE AT SYSTEM LEVEL ALSO, BS BASED FAULT INSERTION AT BOARD AND SYSTEM LEVEL
		FAULT ISOLATION COVERAGE OF PROTOTYPE DEFECTS	ATPG AND FSIM OR PROB FAU GRADING AT CHIP AND BOAR LEVEL; INTEGRATION OF LOW LEVEL FC VALUES, PLUS BACKPLANE & CABLE COVERAGE AT SYSTEM LEVEL ALSO, BS BASED FAULT INSERTION AT BOARD AND SYSTEM LEVEL

PROCE SS STEP	STEP NAME	METRIC	TOOL (T) OR METHOD (M) EXAMPLES
		FAULT CORRECTION COVERAGE OF PROTOTYPE DEFECTS	ATPG AND FSIM OR PROB FAU GRADING AT CHIP AND BOAR LEVEL; INTEGRATION OF LOW LEVEL FC VALUES, PLUS BACKPLANE & CABLE COVERAGE AT SYSTEM LEVEL ALSO, BS BASED FAULT INSERTION AT BOARD AND SYSTEM LEVEL
3.2.5.2	DETAILED DESIGN (PROTOTYPE TEST PERFORMANCE STATISTICS)	FAULT DETECTION COVERAGE OF PROTOTYPE DEFECTS	TEST EQUIPMENT LOGS FRACAS (M, T)
		FAULT ISOLATION COVERAGE OF PROTOTYPE DEFECTS	TEST EQUIPMENT LOGS FRACAS (M, T)
		FALSE ALARM RATE	TEST EQUIPMENT LOG; FRACA (M, T)
		FAULT DETECTION TIME FOR PROTOTYPE DEFECTS	TEST EQUIPMENT LOG; FRACA (M, T)
		FAULT ISOLATION TIME FOR MFG. DEFECTS	TEST EQUIPMENT LOG; FRACA (M, T)
3.2.5.2	DETAILED DESIGN (TIME AND COST STATISTICS)	LOGIC SIMULATION TIME AND COST	SIMULATION LOG (M, T)
		TEST GENERATION TIME AND COST	ATPG LOG (M, T)
		FAULT SIMULATION TIME AND COST	SIMULATION LOG (M, T)
		FAULT INSERTION TIME AND COST	FAULT INSERTION LOG (M, T)
3.2.6.2	MANUFACT. (ANOMALY STATISTICS)	DESIGN FLAWS BY SYSTEM AND POPULATION OF SYSTEMS	DESIGN PROBLEM REPORTIN (M, T)
		MANUF. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS	ATE-BASED DATA COLLECTIO (T); AUTOMATIC FAULT HISTO LOG (T); FRACAS (M, T)
3.2.6.2	MANUFACT. (BIST PERFORMANCE STATISTICS)	FAULT DETECTION COVERAGE OF MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O, I, D LEVELS)	ATE-BASED DATA COLLECTIO (T); AUTOMATIC FAULT HISTO LOG; FRACAS (M, T)
		FAULT ISOLATION COVERAGE OF MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS	ATE-BASED DATA COLLECTIO (T); AUTOMATIC FAULT HISTO LOG; FRACAS (M, T)

PROCESS STEP	STEP NAME	METRIC	TOOL (T) OR METHOD (M) EXAMPLES
		FAULT CORRECTION COVERAGE OF MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS	ATE-BASED DATA COLLECTIO (T); AUTOMATIC FAULT HISTO LOG; FRACAS (M, T)
		FALSE ALARM RATE	ATE-BASED DATA COLLECTIO (T); AUTOMATIC FAULT HISTO LOG; FRACAS (M, T)
		FAULT DETECTION TIME FOR MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O, I, D LEVELS)	ATE-BASED DATA COLLECTIO (T); AUTOMATIC FAULT HISTO LOG; FRACAS (M, T)
		FAULT ISOLATION TIME FOR MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O, I, D LEVELS)	ATE-BASED DATA COLLECTIO (T); AUTOMATIC FAULT HISTO LOG; FRACAS (M, T)
		FAULT CORRECTION TIME FOR MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O, I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
3.2.6.2	MANUFACT. (ATE PERFORMANCE STATISTICS)	FAULT DETECTION COVERAGE OF MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS	ATE-BASED DATA COLLECTIO (T); AUTOMATIC FAULT HISTO LOG; FRACAS (M, T)
		FAULT ISOLATION COVERAGE OF MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS	ATE-BASED DATA COLLECTIO (T); AUTOMATIC FAULT HISTO LOG; FRACAS (M, T)
		FAULT ISOLATION COVERAGE OF FIELD DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS	ATE-BASED DATA COLLECTIO (T); AUTOMATIC FAULT HISTO LOG; FRACAS (M, T)
		FALSE ALARM RATE	ATE-BASED DATA COLLECTIO (T); AUTOMATIC FAULT HISTO LOG; FRACAS (M, T)
		FAULT DETECTION TIME FOR MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS	ATE-BASED DATA COLLECTIO (T); AUTOMATIC FAULT HISTO LOG; FRACAS (M, T)
		FAULT ISOLATION TIME FOR MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS	ATE-BASED DATA COLLECTIO (T); AUTOMATIC FAULT HISTO LOG; FRACAS (M, T)

PROCESS STEP	STEP NAME	METRIC	TOOL (T) OR METHOD (M) EXAMPLES
		AVERAGE AND MAXIMUM AMBIGUITY GROUP SIZE	ATE-BASED DATA COLLECTIO (T); AUTOMATIC FAULT HISTO LOG; FRACAS (M, T)
3.2.6.2	MANUFACT. (RMA PERFORMANCE STATISTICS)	RELIABILITY (MTBF) BY SYSTEM AND POPULATION OF SYSTEMS	ATE-BASED DATA COLLECTIO (T); AUTOMATIC FAULT HISTO LOG; FRACAS (M, T)
		MAINTAINABILITY (MTTR) BY SYSTEM AND POPULATION OF SYSTEMS	ATE-BASED DATA COLLECTIO (T); AUTOMATIC FAULT HISTO LOG; FRACAS (M, T)
		AVAILABILITY BY SYSTEM AND POPULATION OF SYSTEMS	ATE-BASED DATA COLLECTIO (T); AUTOMATIC FAULT HISTO LOG; FRACAS (M, T)
3.2.6.2	MANUFACT. (TEST COST STATISTICS)	COST OF TEST APPLICATION BY SYSTEM AND POPULATION OF SYSTEMS	ATE-BASED DATA COLLECTIO (T);
3.2.7.2	FIELD (ANOMALY STATISTICS)	DESIGN FLAWS BY SYSTEM AND POPULATION OF SYSTEMS	DESIGN PROBLEM REPORTIN (M, T)
		MANUF. DEFECTS BY SYSTEM AN POPULATION OF SYSTEMS	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FIELD DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
3.2.7.2	FIELD SUPPORT (BIST PERFORMANCE STATISTICS)	FAULT DETECTION COVERAGE OF MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O, I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT DETECTION COVERAGE OF FIELD DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O. I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT ISOLATION COVERAGE OF MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O, I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT ISOLATION COVERAGE OF FIELD DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O. I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)

PROCE SS STEP	STEP NAME	METRIC	TOOL (T) OR METHOD (M) EXAMPLES
		FAULT CORRECTION COVERAGE OF MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O, I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT CORRECTION COVERAGE OF FIELD DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O. I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FALSE ALARM RATE	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT DETECTION TIME FOR MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O, I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT DETECTION TIME FOR FIELD DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O. I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT ISOLATION TIME FOR MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O, I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT ISOLATION TIME FOR FIELD DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O. I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT CORRECTION TIME FOR MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O, I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT CORRECTION TIME FOR FIELD DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O. I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
3.2.7.2	FIELD SUPPORT (ATE PERFORMANCE STATISTICS)	FAULT DETECTION COVERAGE OF MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O, I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)

PROCESS STEP	STEP NAME	METRIC	TOOL (T) OR METHOD (M) EXAMPLES
		FAULT DETECTION COVERAGE OF FIELD DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O. I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT ISOLATION COVERAGE OF MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O, I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT ISOLATION COVERAGE OF FIELD DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O. I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FALSE ALARM RATE	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT DETECTION TIME FOR MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O, I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT DETECTION TIME FOR FIELD DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O. I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT ISOLATION TIME FOR MFG. DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O, I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		FAULT ISOLATION TIME FOR FIELD DEFECTS BY SYSTEM AND POPULATION OF SYSTEMS (O. I, D LEVELS)	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		AVERAGE AND MAXIMUM AMBIGUITY GROUP SIZE	ATE-BASED DATA COLLECTIO (T); AUTOMATIC FAULT HISTO LOG; FRACAS (M, T)
3.2.7.2	FIELD SUPPORT (RMA PERFORMANCE STATISTICS)	RELIABILITY (MTBF) BY SYSTEM AND POPULATION OF SYSTEMS	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
		MAINTAINABILITY (MTTR) BY SYSTEM AND POPULATION OF SYSTEMS	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)

PROCESS STEP	STEP NAME	METRIC	TOOL (T) OR METHOD (M) EXAMPLES
		AVAILABILITY BY SYSTEM AND POPULATION OF SYSTEMS	AUTOMATIC FAULT HISTORY LOG; FRACAS (M, T)
3.2.7.2	FIELD SUPPORT (TEST COST STATISTICS)	COST OF TEST APPLICATION BY SYSTEM AND POPULATION OF SYSTEMS	ATE-BASED DATA COLLECTIO (T); MANUAL FIELD DATA COLLECTION

Footnotes:

1. Statistical data is data obtained through simulations, using BIST impact incorporated in the functional model. Goodness is based on the quality of the simulation.
2. TDM is a topological dependency model, such as STAMP, STAT, etc.
3. Flaw detection affects architecture selection because it is another indicator of the sufficiency of test access in the architecture.
4. Cost data for all DFT processes are collected to ascertain cost effectiveness for future use.
5. Metrics are the same for arch. selection and verification since one is a refinement of the other.
6. Prototype assembly faults are determined to some extent in design phase through some limited testing and again more thoroughly in manufacturing.
7. Fault detection, isolation, correction coverage is determined in prototype testing through measurement techniques, such as logging and FRACAS.
8. Min and max ambiguity group size is collected also, to determine the average.
9. BIST Performance statistics are measured values, while prediction is predicted values
10. FD Coverage numbers will vary, depending on the fault model.