

Design for testability in hardware-software systems

Citation for published version (APA):

Vranken, H. P. E., Witteman, M. F., & Wuijtswinkel, van, R. C. (1996). Design for testability in hardware-software systems. *IEEE Design and Test of Computers*, 13(3), 79-87.

Document status and date:

Published: 01/01/1996

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Design for Testability in Hardware-Software Systems

CREATING TESTABLE designs is key to developing complex hardware and/or software systems that function reliably throughout their operational life. Without testability, design flaws may escape detection until a product is in the hands of users; equally, operational failures may prove difficult to detect and diagnose.

Increased system complexity makes thorough assessment of system integrity by testing external black-box behavior almost impossible. System complexity also complicates test equipment and procedures. Design for testability should increase a system's testability, resulting in improved quality while reducing time to market and test costs.

The term *system* means many things to different people. We consider a system to be an integrated set of hardware and/or software modules. Each module is an identifiable part of the system with strictly defined functionality. Hardware, software, or a mix of both, can implement each module. Our view of DFT therefore relates to testing at this abstracted system level, and

Clearly, in today's complex systems, hardware and software approaches to DFT must work together to achieve a successful overall solution. The authors investigate existing and new concepts that may lead to a single design for test strategy in the future.

is a function of the combined testability of all system modules.

Traditionally, hardware designers and test engineers have focused on proving the correct manufacture of a design and on locating and repairing field failures. They have developed several highly structured and effective so-

lutions to this problem, including scan design and self test. Design verification has been a less formal task, based on the designer's skills. However, designers have found that structured design-for-test features aiding manufacture and repair can significantly simplify design verification. These features reduce verification cycles from weeks to days in some cases.

In contrast, software designers and test engineers have targeted design validation and verification. (Unlike hardware, software does not break during field use. Design errors, rather than incorrect replication or wear out, cause operational bugs.) Efforts have focused on improving specifications and programming styles rather than on adding explicit test facilities. For example, modular design, structured programming, formal specification, and object orientation have all proven effective in simplifying test.

Although these different approaches are effective when we can cleanly separate a design's hardware and software parts, problems arise when boundaries blur. For example, in the early design

HARALD P.E. VRANKEN

Eindhoven University of
Technology

MARC F. WITTEMAN

RONALD C. VAN

WUIJTSWINKEL

KPN Research

stages of a complex system, we must define system level test strategies. Yet, we may not have decided which parts to implement in hardware and which in software. In other cases, software running on general-purpose hardware may initially deliver certain functions that we subsequently move to firmware or hardware to improve performance. Designers must ensure a testable, finished design regardless of implementation decisions. Supporting hardware-software codesign^{1,2} requires "cotesting" techniques, which draw hardware and software test techniques together into a cohesive whole. (For a review of current DFT techniques, see the box.)

The access problem

Limited access to individual modules often limits a complex system's testability. Design and implementation of a system comprising multiple modules is very attractive, because we can subdivide the system complexity into comprehensible parts. Nevertheless, after assembly, the complete system's behavior turns into one black box with the multiplied complexity of all its components.

For instance, we can model a module's behavior using a state machine, expressing behavior in terms of states, transitions, and conditions. Verifying all state transitions can test a module's valid behavior. In general, if a module has N states (state space N), there will be at least N state transitions, requiring at least N different tests.

In a complex system of several modules, the number of states increases rapidly. The system's black box behavior consists of all the modules' state spaces. If the system contains K modules with N states each, the composite system has state space N^K . We call this exponential growth state space explosion.

Clearly, the testability of modular systems improves considerably if we can test the modules separately. Hence, modular hardware-software designs should incorporate access paths for test-

ing to enable the testing of separate modules. Researchers have widely applied this divide-and-conquer approach to testing complex, modular, digital circuits.

Design for system level testability

We base design for system level testability³ on a clear separation between implementation-independent system specification and the actual hardware-software system implementation. In the design process, we first create a specification of the system's functional behavior. Such a behavioral specification leads to a clear, thorough understanding of the system to be developed—one not blurred by implementation details. This specification provides a solid basis for partitioning the system into hardware and software, and choosing an appropriate architecture.

To achieve system level design for testability, we must add system level test requirements to the specification. This aims at improving the controllability and observability of system-embedded modules. Next, we must transform the implementation-independent test requirements into actual hardware and/or software requirements. Placing test requirements in the specification can have a serious impact on the actual system implementation. A design may implement test requirements as existing test facilities like boundary scan paths. On the other hand, test requirements may also demand new hardware and software test facilities.

Separating specification from the actual implementation is a basic principle of modern design methods. Such methods include structured⁴ and object-oriented⁵ analysis and design, as well as hardware-software codesign.^{1,2} Therefore, design for system level testability fits well within these modern design methods.

System level testability in the specification

Our basic principle is that we can

tackle system test complexity by partitioning the system into modules. Inserting test functionality into the system allows us to perform system testing by first testing the individual modules and then the interactions between modules. Hardware testing (for example, macro testing⁶) has adopted the same principle.

During system specification, the strategy toward design for system level testability has two parts.

- Partitioning the system—Structured, modular design methods automatically lead to improved testability. However, we can further improve testability by making it a major criterion for system partitioning.
- Adding test functionality—This allows us to control and observe individual modules and the interactions among them for testing purposes. We first state test functionality in the system specification, without concern for implementation details. In the next step of the design process, we incorporate test functionality in the actual hardware-software system implementation.

Partitioning. There are many heuristics and rules of thumb for system partitioning. Minimizing dependencies between modules and minimizing parallelism inside a module are important to improving testability.

The partitioning criterion of minimum dependence between modules means that we should partition the system into independent modules. We achieve this by minimizing the interactions and communication between modules. During testing, we now can isolate a module from its environment with relative ease.

Minimizing parallelism inside a module offers the advantage of producing well-testable modules. A module's par-

Current practice

Designers generally employ quite different DFT techniques for hardware and software.

Hardware DFT

Recently, designers have developed hardware DFT techniques and test standards for built-in test in ICs, printed circuit boards (PCBs), and hardware systems.⁷ The enormous increase in integrated-circuit complexity, the introduction of ASIC technologies, and the advent of multichip modules have made DFT an inevitable necessity for IC testing. Nowadays, IC design commonly incorporates built-in self-test (BIST) techniques.^{8,9}

The increasing numbers and density of complex ICs on PCBs and the rise of surface-mounted technology called for new PCB-testing techniques. Traditional hierarchical testing techniques and in-circuit testing with a bed of nails became impractical. IEEE Std 1149.1 (boundary scan) provides a board level solution.¹⁰

Today's digital system consists of a hierarchy of boards and chips. We mount ICs on a PCB, a rack of PCBs forms a subsystem, and several subsystems build the system. Because of the enormous complexity, it has been very difficult to access individual components during system test. Recent initiatives have sought to extend component and board level test to higher, subsystem and system level tests. The aim is to improve observation and control of system components and their interconnections.

For instance, Whetsel¹¹ proposes a board-to-backplane connection method to hierarchically access and test IEEE 1149.1-based boards and ICs in a system environment. This technique tests tree-configured systems via a test master located at the root of the tree. The test master can access ICs at the tree leaves by addressing the tree nodes. This ap-

proach offers advantages such as a uniform way of accessing all system levels, support for parallel testing of system components (which accelerates testing time), and easier system expansion. IEEE Std 1149.5 (module test and maintenance bus) is a new board-to-backplane connection method.¹² We may even extend the hierarchy of ICs, boards, subsystems, and systems to networks of communicating systems. ISO IS 10164 defines an implementation-independent framework for managing, testing, and diagnosing systems in communication networks (see the next section also). In addition to the development of hardware DFT techniques and standards, Sheppard and Simpson have performed significant work on system level testability analysis techniques.^{9,10} They provide solutions for problems encountered during system testing and diagnosis.

Software DFT

Testing software components is a very wide area, and several techniques are available.¹³ Software DFT generally focuses on improving specification, programming, and testing techniques. Designers rarely practice software DFT in the sense of adding test facilities for observation and control of software modules. (Communications software and telecommunication protocols in particular are rare fields applying this kind of software DFT.) Rather, software designers must work around problems such as limited access to software modules during test.

Telecommunication protocols are becoming more and more complex and, at the same time, testing processes grow in complexity. In 1991, ISO 9646 standardized a framework for conformance testing to overcome many basic testing problems.¹⁶ Among other things, the framework proposes to specify and ap-

ply conformance tests at each protocol layer. In this way, the standard subdivides communication capabilities into modules to be tested separately.

The ISO expected such a structured approach to improve test quality. Unfortunately, this approach is often infeasible in practice because of the limited test access to the system. Hence, practical test applications tend to follow a kind of hybrid module and system test model. Although tests focus on single aspects within the modules (protocol layers), they incorporate (and require) the correct behavior of all other system parts.

For instance, an ISDN (integrated services digital network) protocol stack contains several software modules, that is, protocol layers. Although we may have developed a separate test suite for each layer, testing requires the complete functionality to be available. A tester connected to the ISDN device must emulate the complete behavior of all modules, even if it is to test just one layer. We can often not completely test an embedded module under test because of our inability to trigger all stimuli or observe all signals. This test approach is therefore expensive, inefficient, and incomplete.

An approach overcoming these problems is already available for communication networks. ISO IS 10164 defines an implementation-independent framework for managing, testing, and diagnosing systems in telecommunication networks. Part 12 of that standard¹⁷ defines a framework for remotely manageable diagnostics using Open Systems Interconnection management (McRee¹⁸ provides a tutorial). A draft standard for part 14 defines a system management function, consisting of generic definitions, services, and functional units for test and maintenance¹⁹ (see also the Solutions in standardization section).

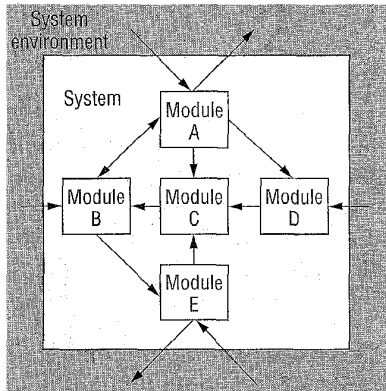


Figure 1. Model of a partitioned system.

allelism is an important complexity measure. As indicated in the previous section, the number of possible states increases exponentially if a module consists of interacting, parallel, finite-state machines. Since this state space explosion dramatically increases the number of required test scenarios, minimizing parallelism also reduces this number.

Ideally, we can model a system as a set of communicating processes. The minimum-dependence criterion aims to minimize process interactions. Minimum parallelism allows each module to correspond to a single sequential process.

Figure 1 shows an example of a partitioned system with five modules. We have applied the minimum-dependence criterion to partition the system into a limited number of modules. We next applied the minimum-parallelism criterion to split complex modules into smaller ones. For instance, a complex module initially contained both modules B and C, but we decomposed this module to limit test complexity.

Adding test functionality. To test individual modules and their interactions, we offer test stimuli to the modules and observe the responses at their boundaries. This requires us to control the module boundaries and observe them directly in the system environment. In general, however, this is not

possible; we require paths via other modules to offer test stimuli and observe the responses of a module under test. In Figure 1, we can neither control module C's boundaries, nor can we directly observe them in the system environment. Testing thus requires test paths through other modules.

These limited control and observation capabilities seriously reduce testability for several reasons.

- We must set up and maintain test paths to and from the module under test. This may be infeasible or require significant effort.
- When the test detects an error, we do not know whether the error occurred in the module under test or in the paths.
- In real-time systems, the order and timing of events is critical. Thus, during test, we should be able to control the timing of incoming events and observe the timing of outgoing events. This is difficult to achieve without direct access to the module under test.

To improve testability, we add test functionality to the system specification and use three kinds of test functions.

Transparent test mode (TTM). We can eliminate the accessibility problem if the modules that constitute the path to the module under test are transparent. They are transparent in the sense that they convey signals without change. We achieve this by extending the module's behavior with an additional transparent-test operating mode. Whenever a module switches into this TTM, it passes incoming events directly to outgoing events in a predefined way, providing a transparent path from module inputs to module outputs.

Additionally, if it is not possible or desirable to make a test path from the tester to every access point of the module under test, we can include a test re-

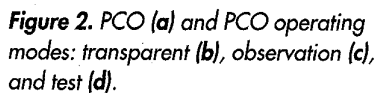
sponder. This test responder more or less inverts the test path: It returns controllable signals from the module under test to the tester.

A disadvantage of these functions is that we often design them specifically for testing one module, which hardly fits with a generic testability approach. Nonetheless, the TTM concept can be useful.²⁰

Built-in self-test (BIST). We can equip a module with self-test, which reduces the required controllability and observability from the system environment. The module's BIST functionality offers test stimuli to the module and observes and evaluates the responses. We start and control the BIST in the module from the system environment. When the test ends, it returns a go/no-go response or diagnostic information to the system environment.

Point of control and observation (PCO). At the module boundaries, we can insert points allowing us to control and observe interconnections between modules directly in the system environment. We insert a PCO in an interconnection between two modules. As shown in Figure 2, a PCO has three operation modes, selected by the mode input. Table 1 lists how we use these different modes. (Readers can contrast this abstracted representation with the analog test access technique proposed for standardization as IEEE Std 1149.4.²¹)

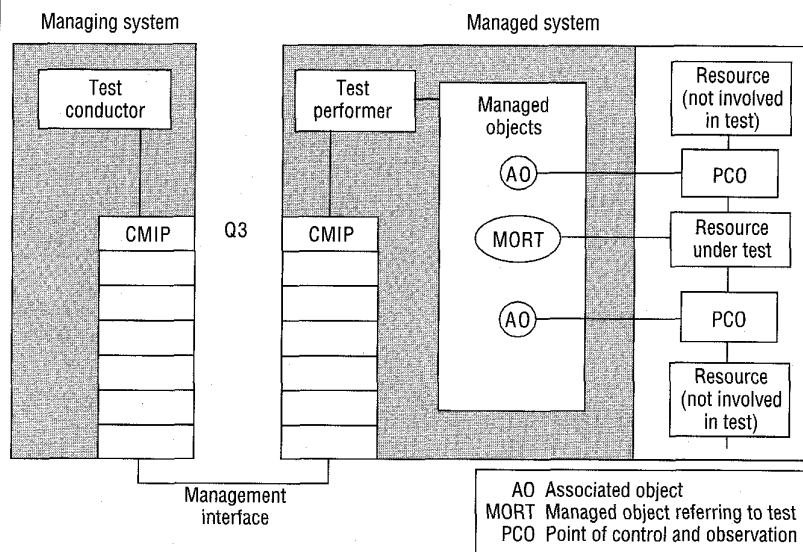
A complete PCO implementation supports all three modes. Of course, it is also possible to implement only two modes. For instance, a point of observation (PO) supports only the transparent and observation modes. Besides using PCOs for observing and controlling interconnections, we can also equip data stores with them. In observation mode, we can use the PCO to monitor the contents of a data store. In test mode, we can use the PCO to read



TTM and PCO functionalities offer paths between the system environment and embedded modules. In addition to test information, these paths can also transport system management information, such as programming updates and data.

Table 1. PCO operating modes.

Mode	Function
Transparent	We use this mode during normal operation when we require no observation or control. The PCO input passes directly to the PCO output, so the PCO is completely transparent. The control input and observation output are of no concern.
Observation	This mode monitors the system during normal operation. The PCO input passes to both the PCO and observation outputs. The observation output monitors data passing through the PCO in the system environment. The control input is also of no concern in this mode.
Test	This mode tests modules in isolation from their environment. The PCO input passes to the observation output. The control input passes to the PCO output. We use the PCO to observe the sending module at the PCO input and to control the receiving module at the PCO output. Both control and observation actions proceed independently, so we can simultaneously test both modules.



Standardization) and ITU (International Telecommunications Union, formerly CCITT) are standardizing the application of telecommunication management principles to test purposes in draft ISO 10164-14.¹⁹ They call this technique resource boundary testing²² (see Figure 3). The basis of this concept originated

Experiments. The TRIBUNE project (Testing, Ratification and Interoperability of the Broadband User Network Interface) implemented an example use of PCOs in a practical situation. This

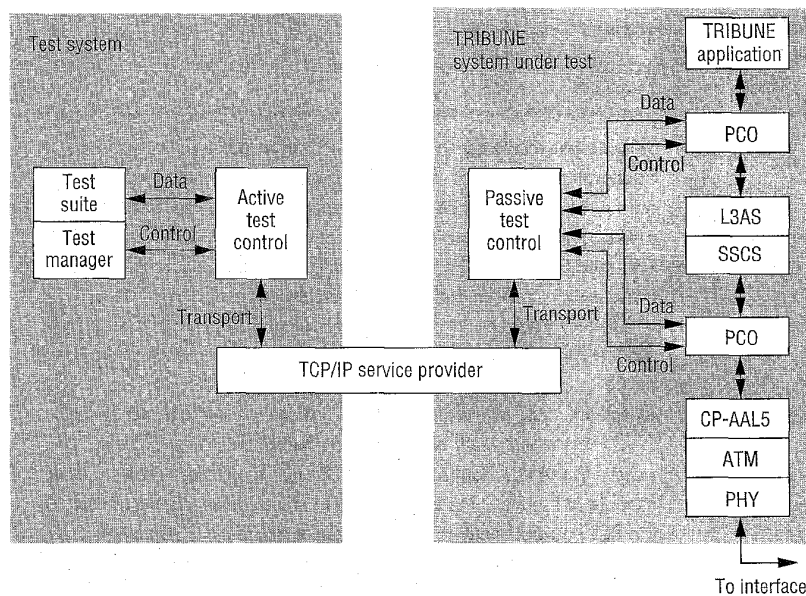


Figure 4. ATM test configuration.

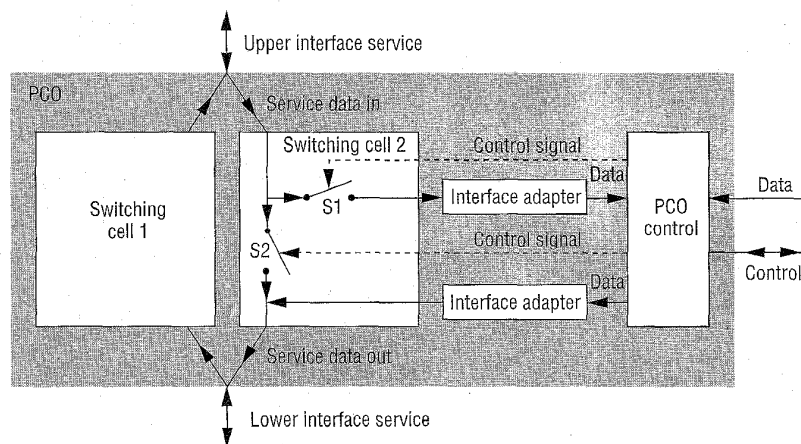


Figure 5. PCO schematic view.

is project 2081 of Race (Research of Advanced Communications Technologies in Europe).

Within TRIBUNE, 12 European companies are developing a test environment for broadband ISDN systems based on asynchronous transfer mode technology. Project participants designed, implemented, and tested communication systems based on pre-

liminary ITU broadband protocol specifications. The TRIBUNE systems' signaling plane contains five protocol layers or sublayers (L3AS, SSCS, CP-AAL5, ATM, and PHY). A detailed description of this experiment is available in Witteman and van Wuijtswinkel.²³

Test architecture. In this experiment, a special test interface connects all sys-

tems. The test interface has functions similar to the standardized test access described earlier. However we do not use the test interface for purposes other than testing. Figure 4 shows the TRIBUNE test configuration. The test system ships its testing data over a TCP/IP (Transmission Control Protocol/Internet Protocol) channel to the system under test. The TCP/IP channel uses a simple test management protocol to perform multiplexing, flow control, and identification of data and control signals. In the system under test, it demultiplexes and directs the signals to the appropriate PCOs.

TRIBUNE systems implement a limited number of PCOs, each placed between two communicating entities, that is, at a protocol interface. The PCOs have functions to observe and control the behavior of the protocol layers under test (implementation under test). Figure 5 shows a PCO with two identical switching cells to control information streams in both directions at the interface. The basis for this technique is the IEEE standard boundary scan concept¹⁰ developed for hardware testing. In Figure 4, two PCOs are necessary to completely test the implementation under test (L3AS and SSCS). We could easily add more PCOs, although TRIBUNE purposes did not require them.

PCO design. The PCO control part uses separate signals to exchange data and control information (see Figure 5). The control signals can be transferred directly to the cells. We may, however, have to adapt the data to the coding of service primitives at the upper and lower interfaces of the implementation under test. These interface adapters depend on the implementation. Two symbolic switches control the information stream within a switching cell. Switch S1 controls whether or not the test system can observe the incoming information stream. A View (V) or Blind (B) control signal can set S1. Switch S2

controls whether or not the incoming information stream should transfer transparently to the adjacent layer. A Connect (C) or Disconnect (D) control signal sets S2. Four different switch combinations yield four possible modes (or states) for each switching cell.

Experiment evaluation. The TRIBUNE test architecture allows various test configurations not possible with conventional test techniques. The architecture not only allows direct testing of internal modules, it also facilitates interoperability testing of the communication stacks without using the application software. The test architecture (Figure 4) shows how we can test the combination of layers L3AS and SSCS using a multilayer, local test method. The test system can control both the upper (to L3AS) and lower (to SSCS) tester, which implies a substantial increase in the number of traversed states. In this way, we achieve a more balanced test coverage. Comparing this technique with conventional test methods, we estimate that it is possible to double the number of implementable and executable test purposes. Moreover, the actual tests can be shorter, due to significant reduction of the synchronizing sequences' lengths.

The addition of explicit test functions in the system under test does take extra money and time. We represent these costs as lines of programming code. Since we specified the TRIBUNE systems formally in the Specification and Description Language, we measure the code in units of SDL lines.

An average protocol layer takes an estimated 5,000 lines of SDL code. A single PCO contains about 800 lines, while the passive test control function needs 500 lines. Any additional PCO may require 80 new lines of code. Thus, a system under test containing a protocol stack with five layers and five PCOs will need 26,620 lines of code, of which test functions consume 6%.

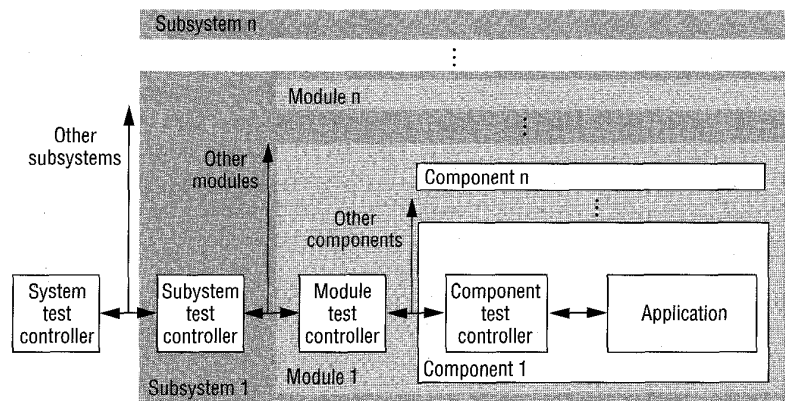


Figure 6. Hierarchical test architecture.

Implementation aspects

We next discuss the step from specification to hardware-software implementation, focusing on how to implement the specified test functionality in hardware and/or software.

When constructing the system's hardware-software architecture, we can incorporate a recursive test hierarchy into the system (Figure 6). For full test and diagnostic control, this test architecture should at least offer the functions in the following list. Preferably, these functions should be available at every test hierarchy level:

- initialization of system, subsystems, modules, or components (mode setting, reset)
- access to and control of system components at lower levels in the hierarchy
- transportation of test stimuli
- control of built-in test facilities
- collection of test results
- identification of components

In general, two opposing strategies exist for incorporating a hierarchical test architecture: centralized and distributed.^{7,14} In a centralized strategy, a single test control module at the top level accesses and controls all lower levels in the system. The distributed strategy dis-

tributes test control over the individual levels as much as possible. Although both strategies have advantages and disadvantages, we prefer the distributed strategy for the reasons discussed next.

The centralized strategy does not require us to equip every test level with module-specific test and maintenance knowledge, providing a simple and low-cost test architecture. However, because the central test module contains the implementation knowledge, we may not interchange modules that are functionally equal but implemented in different technologies. Hence, the centralized strategy is rather inflexible. Furthermore, it introduces significant communication overhead for transporting test data between hierarchy levels.


The distributed test strategy is more flexible, because it locates test knowledge at the individual system levels. This facilitates concurrent testing and thus reduces test time. Distribution of test functions also reduces the system test control module's complexity. Furthermore, locating implementation knowledge at the individual test levels stimulates implementation independence, because the system test controller can operate at an implementation-independent level. This also eliminates the need for complex and application-specific test interfaces. By providing standardized, general-

purpose test interfaces, the distributed strategy facilitates use of commercially available products that are fully interchangeable. In this way, we can produce highly testable systems with little effort. These advantages show that the distributed approach is highly suitable for testing complex systems.

The centralized and distributed test strategies represent extremes. In practice, designers may often use a mixed strategy that incorporates features of both.

HARDWARE AND SOFTWARE designers have developed different DFT techniques. Hardware DFT focuses on implementation, while software DFT focuses on specification.

The application of these techniques alone does not produce satisfactory system level testability. The power of the individual DFT techniques may well need to come together in one overall DFT approach.

Modular systems, composed of multiple parts, offer no advantages to testers, unless we can test the system parts separately. Modular design should imply modular testing. The next challenge will be developing a complete design process. Such a process should provide implementation-independent testability requirements in the specification, like transparent test mode, built-in self-test, and points of control and observation. We must incorporate these testability requirements in the hardware-software system implementation. This implies translating high-level testability requirements onto existing or new test function implementations. 

Acknowledgments

We gratefully acknowledge Ilja van Rhee for his considerable contribution to this article. We also thank René Segers and Mario Stevens of the Eindhoven University of Technology for their support. Finally, we thank Colin Maunder for his review and valuable suggestions.

References

1. W.H. Wolf, "Hardware-Software Co-Design of Embedded Systems," *Proc. IEEE*, Vol. 82, No. 7, IEEE, Piscataway, N.J., 1994, pp. 967-989.
2. D.D. Gajski and F. Vahid, "Specification and Design of Embedded Hardware-Software Systems," *IEEE Design & Test of Computers*, Vol. 12, No. 1, Spring 1995, pp. 53-67.
3. H.P.E. Vranken et al., "System-Level Testability of Hardware/Software Systems," *Proc. Int'l Test Conf.*, IEEE CS Press, 1994, pp. 134-142.
4. P.T. Ward and S.J. Mellor, *Structured Development for Real-Time Systems*, Prentice Hall, Englewood Cliffs, N.J., 1985.
5. I. Jacobson et al., *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, Reading, Mass., 1992.
6. F.P.M. Beenker, R.G. Bennetts, A.P. Thijssen, *Testability Concepts for Digital ICs: The Macro Test Approach*, Kluwer, 1995.
7. C. Maunder, "A Universal Framework for Managed Built-In Test," *Proc. Int'l Test Conf.*, IEEE Computer Society Press, Los Alamitos, Calif., 1993, pp. 21-29.
8. V.D. Agrawal, C.R. Kime, and K.K. Saluja, "A Tutorial on Built-In Self-Test, Part 1: Principles," *IEEE Design & Test of Computers*, Vol. 10, No. 1, Mar. 1993, pp. 73-82.
9. V.D. Agrawal, C.R. Kime, and K.K. Saluja, "A Tutorial on Built-In Self-Test, Part 2: Applications," *IEEE Design & Test of Computers*, Vol. 10, No. 2, June 1993, pp. 69-77.
10. *IEEE Std 1149.1, Test Access Port and Boundary-Scan Architecture*, IEEE, 1990.
11. L. Whetsel, "Hierarchically Accessing 1149.1 Applications in a System Environment," *Proc. Int'l Test Conf.*, IEEE CS Press, 1993, pp. 517-526.
12. *IEEE Std 1149.5, Standard Module Test and Maintenance Bus*, IEEE, 1995.
13. J.W. Sheppard and W.R. Simpson, series of 6 articles on system test and diagnosis, *IEEE Design & Test of Computers*, Sept. 1991 to June 1993.
14. W.R. Simpson and J.W. Sheppard, *System Test and Diagnosis*, Kluwer, Boston, 1994.
15. B. Beizer, *Software Testing Techniques*, Van Nostrand Reinhold, New York, 1990.
16. *ISO/IEC 9646-1, Information Technology—OSI Conformance Testing Methodology and Framework; Part 1: General Concepts*, Int'l Organization for Standardization, Geneva, Apr. 1994.
17. *ISO/IEC IS 10164-12 Information Technology—Open Systems Interconnection—Systems Management; Part 12: Test Management Function*, (also Recommendation ITU X.745), Int'l Organization for Standardization, 1992.
18. R. McRee, "OSI Test Management: An Introduction," *IEEE Design & Test of Computers*, Vol. 12, No. 4, Winter 1995, pp. 68-80.
19. *ISO/IEC IS 10164-14 (Draft) Information Technology—Open Systems Interconnection—Systems Management, Part 14: Confidence and Diagnostic Test Categories* (ITU Recommendation X.737), Int'l Organization for Standardization, June 1994.
20. M.S. Abadir and M.A. Breuer, "A Knowledge-Based System for Designing Testable VLSI Chips," *IEEE Design & Test of Computers*, Vol. 2, No. 4, Aug. 1985, pp. 56-68.
21. K.P. Parker, J.E. McDermid, and S. Oresjo, "Structure and Metrology for an Analog Testability Bus," *Proc. Int'l Test Conf.*, IEEE CS Press, 1993, pp. 309-322.
22. R.C. van Wuijtswinkel and M.F. Witteman, "Testing Using Telecommunications Management," *Protocol Test Systems VII*, Chapman & Hall, London, 1995.
23. M.F. Witteman and R.C. van Wuijtswinkel, "ATM Broadband Testing Us-

ing the Ferry Principle," *Protocol Test Systems VI*, North-Holland, Amsterdam, 1994, pp. 125-138.



Harald P.E. Vranken is currently a PhD student in the Department of Electrical Engineering at the Eindhoven University of Technology. His research interests include design for test and testing at the system level, focusing on embedded, real-time systems that incorporate both hardware and embedded software. Vranken holds the MS degree from the Department of Electrical Engineering at the Eindhoven University of Technology, the Netherlands, and also completed the postmaster's education program in information and communication technology at the Stan Ackermans Institute in Eindhoven.



Marc F. Witteman is a project leader for testing communication systems at KPN Research, where he started work in conformance testing. He has participated in several test projects for GSM, intelligent network, and ATM protocols. His favorite subjects are the investigation of testability and the design of test architectures. Currently, his main concern is the validation of IN services and chip cards. Witteman holds an MSc from the Department of Electrotechnical Engineering at the University of Delft, the Netherlands.



Ronald C. van Wuijtswinkel also works for KPN Research, the research laboratory of Royal PTT Nederland N.V. He has participated in several test projects for GSM, IN, and ATM protocols and for PSTN, ISDN and IN services. These experiences contributed to new developments that improve communication systems testability. He currently develops and plans joint test activities between Unisource partners. Van Wuijtswinkel graduated with an MSc from the Department of Electrical Engineering at the Eindhoven University of Technology.

Address questions or comments about this article to Marc Witteman, KPN Research, PO Box 421, 2260 AK Leidschendam, the Netherlands; M.F.Witteman@research.kpn.com.

Special Issue on Economics of Design and Test

Issue date: Fall 1997

Submissions due: November 4, 1996

For this special issue of *IEEE Design & Test of Computers*, guest editors Tony Ambler and Magdy Abadir seek significant contributions that address the current and future design, test, and manufacturing trends and how they are driven by the economics of delivering ever-increasingly complex microelectronic systems. Areas of interest include, but are not limited to

- Cost of ATE, DFT (scan versus BIST)
- Relationship between design, test, quality, and cost
- Quality and reliability effects on maintenance costs
- Cost effectiveness of design for test
- Test cost modeling (micro and macro models)
- CAE tools for analyzing design and test cost
- Design/test standards
- Models for design, test, and manufacturing process flow
- High-level trade-off analysis tools
- MCMs and WSI
- Design automation tools
- INDUSTRIAL CASE STUDIES WELCOMED!

Interested authors may submit four copies of a 35-page, double-spaced paper, in English, to the guest editor(s) no later than **November 4, 1996**. Each copy must contain contact information (contact name, postal address, telephone number, and e-mail address) and a 100-word abstract. Papers must not be under consideration elsewhere or published in a similar form. *IEEE Design & Test* publishes articles of near-term interest to the professional engineering community.

Guest Editors:

Prof. Tony Ambler
Brunel University
Uxbridge, Middlesex UB8 3PH, UK
Phone: 44 (1895) 20-33-80
Fax: 44 (1895) 25-87-28
tony.ambler@brunel.ac.uk
(UT Austin from September 1st)

Dr. Magdy Abadir
Motorola
9737 Great Hills Trail, Austin, TX 78759
Phone: (512) 795-7192
Fax: (512) 795-7510
abadir@ibmoto.com