IEEE Std 1149.1™-2001 (R2008)
(Revision of IEEE Std 1149.1-1990)

IEEE Standard Test Access Port and Boundary-Scan Architecture

Sponsor

Test Technology Standards Committee
of the
IEEE Computer Society

Reaffirmed 27 March 2008

Approved 14 June 2001

IEEE-SA Standards Board

Abstract: Circuitry that may be built into an integrated circuit to assist in the test, maintenance, and support of assembled printed circuit boards is defined. The circuitry includes a standard interface through which instructions and test data are communicated. A set of test features is defined, including a boundary-scan register, such that the component is able to respond to a minimum set of instructions designed to assist with testing of assembled printed circuit boards. Also, a language is defined that allows rigorous description of the component-specific aspects of such testability features.
Keywords: boundary scan, boundary-scan architecture, Boundary-Scan Description Language, boundary-scan register, BSDL, circuit boards, circuitry, integrated circuit, printed circuit boards, TAP, test, test access port, VHDL, VHSIC Hardware Description Language

---

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Print: ISBN 0-7381-2944-5 SH94949
PDF: ISBN 0-7381-2945-3 SS94949

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied "**AS IS**."

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

IEEE is the sole entity that may authorize the use of certification marks, trademarks, or other designations to indicate compliance with the materials set forth herein.

Introduction

(This introduction is not part of IEEE Std 1149.1-2001, Standard Test Access Port and Boundary-Scan Architecture.)

This standard defines a test access port and boundary-scan architecture for digital integrated circuits and for the digital portions of mixed analog/digital integrated circuits. The facilities defined by the standard seek to provide a solution to the problem of testing assembled printed circuit boards and other products based on highly complex digital integrated circuits and high-density surface-mounting assembly techniques. They also provide a means of accessing and controlling design-for-test features built into the digital integrated circuits themselves. Such features might, for example, include internal scan paths and self-test functions as well as other features intended to support service applications in the assembled product.

History of the development of this standard

The process of developing this standard began in 1985, when the Joint European Test Action Group (JETAG) was formed in Europe. During 1986, this group expanded to include members from both Europe and North America and, as a result, was renamed the Joint Test Action Group (JTAG). Between 1986 and 1988, the JTAG Technical Subcommittee developed and published a series of proposals for a standardized form of boundary scan. In 1988, the last of these proposals—JTAG Version 2.0—was offered to the IEEE Testability Bus Standards Committee (P1149) for inclusion in the standard then under development. The Testability Bus Standards Committee accepted this approach. It decided that the JTAG proposal should become the basis of a standard within the Testability Bus family, with the result that the P1149.1 project was initiated. Following these decisions, the JTAG Technical Subcommittee became the core of the IEEE Working Group that developed this standard.

After the initial approval of this standard in February 1990 and its subsequent publication, the Working Group immediately began efforts to develop a supplement for the purpose of correction, clarification, and enhancement. This effort, spurred and guided by interaction between developers and users of the original standard, culminated in IEEE Std 1149.1a-1993, which was approved in June 1993.

The major changes to this standard introduced by IEEE Std 1149.1a-1993 were

— The addition of two optional instructions, *CLAMP* and *HIGHZ*, which standardized the names and specifications of features often implemented as design-specific features
— The addition of an optional facility to switch a component from a mode in which it complies to this standard into one in which it supports another design-for-test approach

Further, starting with a proposal made by Kenneth P. Parker and Stig Oresjo in 1990, an effort was undertaken to develop a language to describe components that conform to this standard. This effort concluded in the approval of IEEE Std 1149.1b-1994 in September 1994.

The major change introduced to this standard by IEEE Std 1149.1b-1994 was the addition of Annex B, which defines the Boundary-Scan Description Language. All other changes were minor and were strictly for clarification.

Changes introduced by this revision

This revision is primarily a housekeeping update, designed to consolidate learning from the first 10 years of the standard’s use into the standard document.

The principal changes introduced are

— To reduce the risk of accidental entry into test mode, the requirement that a binary code for the *EXTEST* instruction be {000...0} has been removed and use of this binary code for other instructions that result in entry to test mode has been deprecated [see 8.1.1 e), 8.8.1 h), B.8.11.3 d), and B.8.11.3 e)].

— To increase the flexibility with which instructions may be implemented and merged, the implicitly merged *SAMPLE/PRELOAD* instruction has been redefined as two separate instructions: *SAMPLE* and *PRELOAD*. *These instructions can continue to share a single binary code*, effectively resulting in a merged *SAMPLE/PRELOAD* instruction, but alternatively, they may now share binary codes with other instructions, provided that no rules for any of the merged instructions are violated. [see 8.1.1 g), 8.2.1 b), 8.6, 8.7, B.5.1.2, B.6.3 b), B.8.11.3 f), B.8.11.3 g), B.8.11.3 h), B.8.11.3 i), B.8.11.3 j), and B.8.13.3 a)].

— To enable more efficient implementation of boundary-scan register cells provided at system logic outputs, the source of data to be captured in such cells in response to the *SAMPLE* instruction is now allowed to be at the connected system pin [see 8.6.1 d), 11.6.1 a), 11.6.1 b), B.8.14.4 l), and B.10.2.4.1 b)]. Additionally, three new cell types based on this implementation (**BC_8, BC_9,** and **BC_10**) have been added to the Standard VHDL Package (see B.5.1.2, B.9, and Table B.3).

— To permit more flexible boundary-scan register cell implementations, sharing of circuitry between the boundary-scan register and other elements of the test and/or system logic has been allowed in limited cases [see 11.2.1 j) and 11.2.1 k)].

— To support more complete description of IC pin drivers with bus keeper circuits, a new value for <disable result> has been defined (**KEEPER**, see B.5.1.2, B.8.14.1, and B.8.14.3.7).

— To track the widespread acceptance of BSDL, the language has been made a normative part of the standard and its use for documentation has been mandated (see 13.3.1 c) and Annex B).

Additionally, a number of minor changes were made to correct and clarify the language of the standard [of special note, see 4.8.1 c), 11.7.1 b), B.8.14.4 k), and B.8.14.4 n)].

Participants

At the time of the issue of IEEE Std 1149.1-2001, the members of the working group were:

Christopher J. Clark, *Chair*

Adam W. Ley, *Editor*

John Andrews
Bill Aronson
Carl F. Barnhart
Dilip K. Bhavsar
Terry Borroz
John Braden
Bill Bruce
Tapan J. Chakraborty
Sung Chung
Jim Coleman
Frans De Jong
Bulent Dervisoglu
Ted Eaton
Bill Eklow
Dean Geerdes
Grady L. Giles
Peter Hansen
Neil Jacobson
Najmi Jarwala
London Jin
Wuudiann Ke
Arthur Khu
Ramesh Krishnamurthy
Tom Langford
Larry Lee
Colin Maunder
Benoit Nadeau-Dostie
Jay Nejedlo
Kenneth P. Parker
Michael Ricchetti
Gordon D. Robinson
Robert J. Russell
Adam Sheppard
Steve Stark
Ron Walther
Lee Whetsel
Thomas W. Williams

The following members of the balloting group voted on this standard. Balloters may have voted for approval, disapproval, or abstention. :

John Andrews
Carl Barnhart
Roger Bennetts
Terry Borroz
John Braden
Keith Chow
Sung S. Chung
Chris J. Clark
Frans de Jong
Ted Eaton
William Eklow
Dean Geerdes
Grady Giles
Peter Hansen
Lee F. Horney
Mitsuaki Ishikawa
Neil Jacobson
Jake Karrfalt
Thomas L. Langford
Adam W. Ley
Gregory A. Maston
Colin Maunder
Patrick McHugh
Earl J. Meiers
Yinghua Min
James A. Monzel
Benoit Nadeau-Dostie
Jay Nejedlo
Bruce E. Peterson
Mike Ricchetti
Gordon Robinson
Jaideep Roy
Robert J. Russell
Scott A. Yalcourt
T. W. Williams

When the IEEE-SA Standards Board approved this standard on 14 June 2001, it had the following membership:

Donald N. Heirman, *Chair*
James T. Carlo, *Vice Chair*
Judith Gorman, *Secretary*

Satish K. Aggarwal
Mark D. Bowman
Gary R. Engmann
Harold E. Epstein
H. Landis Floyd
Jay Forster*
Howard M. Frazier
Ruben D. Garzon
James H. Gurney
Richard J. Holleman
Lowell G. Johnson
Robert J. Kennelly
Joseph L. Koepfinger*
Peter H. Lips
L. Bruce McClung
Daleep C. Mohla
James W. Moore
Robert F. Munzner
Ronald C. Petersen
Gerald H. Peterson
John B. Posey
Gary S. Robinson
Akio Tojo
Donald W. Zipse

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaison:

Alan Cookson, *NIST Representative*
Donald R. Volzka, *TAB Representative*

Don Messina
IEEE Standards Project Editor

Contents

1. Overview 1

1.1 Scope 1
1.2 Purpose 1
1.3 Document outline 5
1.4 Conventions 6

2. References 6

3. Definitions and Acronyms 6

3.1 Definitions 6
3.2 Acronyms 9

4. Test access port (TAP) 9

4.1 Connections that form the test access port (TAP) 9
4.2 Test clock input (TCK) 9
4.3 Test mode select input (TMS) 11
4.4 Test data input (TDI) 11
4.5 Test data output (TDO) 12
4.6 Test reset input (TRST*) 12
4.7 Interconnection of components compatible with this standard 13
4.8 Subordination of this standard within a higher-level test strategy 15

5. Test logic architecture 16

5.1 Test logic design 17
5.2 Test logic realization 17

6. The TAP controller 18

6.1 TAP controller state diagram 18
6.2 TAP controller operation 24
6.3 TAP controller initialization 30

7. The instruction register 33

7.1 Design and construction of the instruction register 33
7.2 Instruction register operation 34

8. Instructions 36

8.1 Response of the test logic to instructions 36
8.2 Public instructions 37
8.3 Private instructions 38
8.4 The *BYPASS* instruction 38
8.5 Boundary-scan register instructions 39
8.6 The *SAMPLE* instruction 41
8.7 The *PRELOAD* instruction 43
8.8 The *EXTEST* instruction 45

8.9 The *INTEST* instruction 48
8.10 The *RUNBIST* instruction 53
8.11 The *CLAMP* instruction 56
8.12 Device identification register instructions 57
8.13 The *IDCODE* instruction 58
8.14 The *USERCODE* instruction 58
8.15 The *HIGHZ* instruction 59

9. Test data registers 61

9.1 Provision of test data registers 61
9.2 Design and construction of test data registers 63
9.3 Test data register operation 64

10. The bypass register 66

10.1 Design and operation of the bypass register 66

11. The boundary-scan register 67

11.1 Introduction 68
11.2 Register design 72
11.3 Register operation 74
11.4 General rules regarding cell provision 76
11.5 Provision and operation of cells at system logic inputs 79
11.6 Provision and operation of cells at system logic outputs 87
11.7 Provision and operation of cells at bidirectional system logic pins 102
11.8 Redundant cells 108
11.9 Special cases 109

12. The device identification register 111

12.1 Design and operation of the device identification register 112
12.2 Manufacturer identity code 114
12.3 Part-number code 115
12.4 Version code 115

13. Conformance and documentation requirements 115

13.1 Claiming conformance to this standard 115
13.2 Prime and second source components 117
13.3 Documentation requirements 117
Annex A (informative) An example implementation using level-sensitive design techniques 120
Annex B (normative) Boundary-scan description language 129

IEEE Standard Test Access Port and Boundary-Scan Architecture

1. Overview

1.1 Scope

This standard defines test logic that can be included in an integrated circuit to provide standardized approaches to

- testing the interconnections between integrated circuits once they have been assembled onto a printed circuit board or other substrate;
- testing the integrated circuit itself; and
- observing or modifying circuit activity during the component's normal operation.

The test logic consists of a boundary-scan register and other building blocks and is accessed through a Test Access Port (TAP).

1.2 Purpose

1.2.1 An overview of the operation of IEEE Std 1149.1

This subclause provides a general overview of the operation of a component compatible with this standard and provides a background to the detailed discussion in later subclauses.

The circuitry defined by this standard allows test instructions and associated test data to be fed into a component and, subsequently, allows the results of execution of such instructions to be read out. All information (instructions, test data, and test results) is communicated in a serial format.

The sequence of operations would be controlled by a bus master, which could be either an automatic test equipment (ATE) or a component that interfaces to a higher-level test bus as a part of a complete system maintenance architecture. Control is achieved through signals applied to the Test Mode Select (TMS) and Test Clock (TCK) inputs of the various components connected to the bus master. Starting from an initial state in which the test circuitry defined by this standard is inactive, a typical sequence of operations would be as follows.

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

Once the instruction has been loaded, the selected test circuitry is configured to respond. In some cases, however, it is necessary to load data into the selected test circuitry before a meaningful response can be made. Such data is loaded into the component serially in a manner analogous to the process used previously to load the instruction. Note that the movement of test data has no effect on the instruction present in the test circuitry.

After execution of the test instruction, based where necessary on supplied data, the results of the test can be examined by shifting data out of the component to or through the bus master.

Note that in cases where the same test operation is to be repeated but with different data, new test data can be shifted into the component while the test results are shifted out. There is no need for the instruction to be reloaded.

Operation of the test circuitry may proceed by loading and executing several further instructions in a manner similar to that described and would conclude by returning the test circuitry and, where required, on-chip system circuitry to its initial state.

1.2.2 The use of IEEE Std 1149.1 to test an assembled product

This subclause outlines the use of the boundary-scan circuitry defined by this standard during the process of testing an assembled product such as a printed circuit board.

The test problem for any product constructed from a collection of components can be decomposed into three goals:

a) To confirm that each component performs its required function;
b) To confirm that the components are interconnected in the correct manner; and
c) To confirm that the components in the product interact correctly and that the product performs its intended function.

This approach can be applied to a board constructed from integrated circuits, to a system constructed from printed circuit boards, or to a complex integrated circuit constructed from a set of simpler functional modules. To simplify the discussion, this description henceforth will concentrate on the case of an assembled printed circuit board constructed from a collection of digital integrated circuits.

At the board level, goal a) and goal b) typically are achieved by using in-circuit test techniques; for goal c), a functional test is required. However, in-circuit test techniques have significant limitations when viewed against evolving surface-mount interconnection technology, for example, the difficulty of making reliable contact to miniaturized features of the printed circuit board using a bed-of-nails fixture. How, then, might the above three test goals be achieved if test access becomes limited to the normal circuit connections, plus a relatively small number of special-purpose test connections?

Considering goal a), it is clear that the vendor of an integrated circuit used in the board-level design will have an established test methodology for that component. The components could be tested on a proprietary ATE system or by using a self-test procedure embedded in the design. Information on the test methodology adopted is typically not available to the component purchaser. Even where self-test modes of operation are known to exist, they may not be documented and therefore are not available to the component user. Alternative sources of test data for the board test engineer may be the component test libraries supplied with

in-circuit test systems or the test programs developed by component users for incoming inspection of delivered devices.

Wherever the test data for a component originates, the next step is to use it once the component has been assembled onto the printed circuit board. If access is limited to the normal connections of the assembled circuit, this task may be far from simple. This is particularly true if the surrounding components are complex or if the board designer has tied some of the components' connections to fixed logic levels or has left component pins unconnected. Normally, it will not be possible to test the component in the same way that it was tested in isolation unless an in-circuit test is achievable.

To ensure that built-in test facilities can be used or that preexisting test patterns can be applied, a framework is needed that can be used to convey test data to or from the boundaries of individual components so that they can be tested as if they were freestanding. This framework will also allow access to and control of built-in test facilities of components. Boundary scan coupled with a test access bus provides such a framework.

The objective of this standard is to define a boundary-scan architecture that can be adopted as a standard feature of integrated circuit designs, thus allowing the required test framework to be created on assembled printed circuit boards and other products.

1.2.3 What is boundary scan?

The boundary-scan technique involves the inclusion of a shift-register stage (contained in a boundary-scan register cell) adjacent to each component pin so that signals at component boundaries can be controlled and observed using scan testing principles.

Figure 1-1 illustrates an example implementation for a boundary-scan register cell that could be used for an input or output connection to an integrated circuit. Dependent on the control signals applied to the multiplexers, data can be either loaded into the scan register from the Signal-in port (e.g., the input pin) or driven from the register through the Signal-out port of the cell (e.g., into the core of the component design). As will be discussed in detail in Clause 11, the second flip-flop (controlled by input Clock B) is provided to ensure that the signals driven out of the cell in the latter case are held while new data is shifted into the cell using input Clock A. This flip-flop is not required in all cases but is included in Figure 1-1 to simplify the discussion.

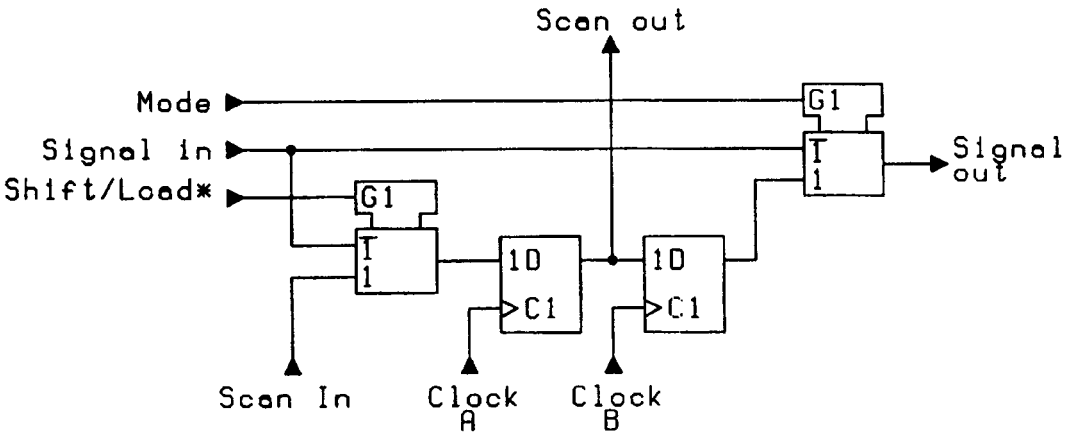


Figure 1-1—A boundary-scan register cell

The boundary-scan register cells for the pins of a component are interconnected to form a shift-register chain around the border of the design, and this path is provided with serial input and output connections and

appropriate clock and control signals. Within a product assembled from several integrated circuits the boundary-scan registers for the individual components could be connected in series to form a single path through the complete design, as illustrated in Figure 1-2. Alternatively, a board design could contain several independent boundary-scan paths.

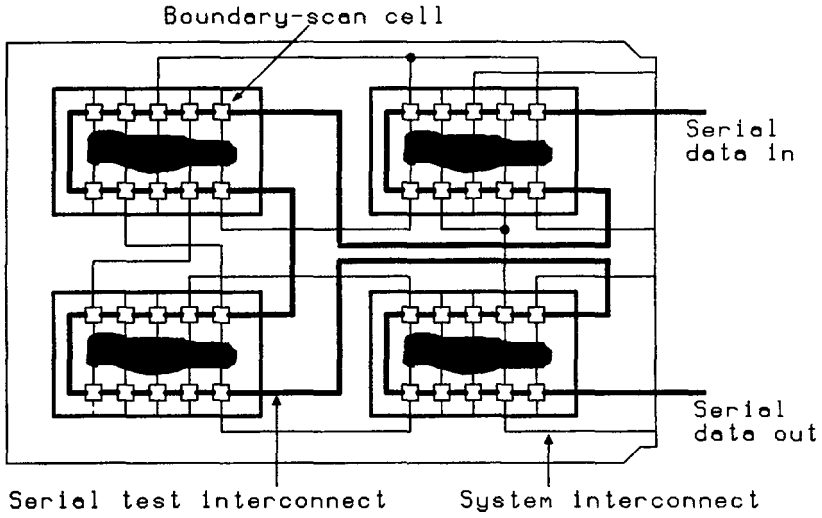


Figure 1-2—A boundary-scannable board design

If all the components used to construct a circuit have a boundary-scan register, the resulting serial path through the complete design can be used in two ways:

a) To allow the interconnections between the various components to be tested, test data can be shifted into all the boundary-scan register cells associated with component output pins and loaded in parallel through the component interconnections into those cells associated with input pins; and
b) To allow the components on the board to be tested, the boundary-scan register can be used as a means of isolating on-chip system logic from stimuli received from surrounding components while an internal self-test is performed. Alternatively, if the boundary-scan register is suitably designed, it can permit a limited slow-speed static test of the on-chip system logic since it allows delivery of test data to the component and examination of the test results.

These tests allow the first two goals discussed earlier to be achieved through the use of the boundary-scan register. In effect, tests applied using the register can detect many of the faults that in-circuit testers currently address, but without the need for extensive bed-of-nails access. The third goal—to functionally test the operation of the complete product—remains and can be achieved either by using a functional (through the pins) ATE system or by using a system-level self-test, for example.

Note also that by parallel loading the cells at both the inputs and outputs of a component and shifting out the results, the boundary-scan register provides a means of “sampling” the data flowing through a component without interfering with the behavior of the component or the assembled board. This mode of operation is valuable for design debugging and fault diagnosis since it permits examination of connections not normally accessible to the test system.

1.2.4 The use of IEEE Std 1149.1 to achieve other test goals

In addition to its application in testing printed circuit assemblies and other products containing multiple components, the test logic defined by this standard can be used to provide access to a wide range of design-

for-test features built into the components themselves. Such features might include internal scan paths, self-test functions [e.g., using built-in logic block observer (BILBO) elements], or other support functions.

Design-for-test features such as these can be accessed and controlled using the data path between the serial test data pins of the TAP defined by this standard. Instructions that cause internal reconfiguration of the component's system logic such that the test operation is enabled may be shifted into the component through the TAP.

1.3 Document outline

Circuit designs such as that defined by this standard are more easily understood if their specifications are accompanied by general descriptive material that places the details of the various parts of the design in perspective and provides examples of implementation. Clause 1 therefore contains an overview of the application of this standard to the testing of the digital portions of an electronic product consisting of many integrated circuits.

Subsequent clauses of the document contain the specifications for particular features of this standard. Two classes of material are contained in these clauses:

1.3.1 Specifications

Subclauses entitled Specifications contain the rules, recommendations, and permissions that define this standard:

a) *Rules* specify the mandatory aspects of this standard. Subclauses that are rules contain the word *shall*.
b) *Recommendations* indicate preferred practice for designs that seek to conform to this standard. Subclauses that are recommendations contain the word *should*.
c) *Permissions* show how optional features may be introduced into a design that seeks to conform to this standard. These features will extend the application of the test circuitry defined by this standard. Subclauses that are permissions contain the word *may.*

1.3.2 Descriptions

Material not contained in subclauses entitled Specifications is descriptive material that illustrates the need for the features being specified or their application. This material includes schematics that illustrate a possible implementation of the specifications in this standard. Annex A to this standard contains an alternative implementation example. The descriptive material also discusses design decisions made during the development of this standard.

CAUTION: The descriptive material contained in this standard is for illustrative purposes only and does not define a preferred implementation. Examples are provided throughout this standard to illustrate possible circuit implementations. Where discrepancies between examples and specifications may occur, the specifications always take precedence. Readers should exercise caution when using these examples to ensure full compliance in their specific applications. In particular, it is emphasized that the examples are designed to effectively communicate the meaning of this standard. As such, they are logically correct; however, as always, a particular implementation may not operate properly with respect to timing and other parametric characteristics. One example of this concern is that the sample TAP controller implementation depicted in Figure 6-5 reasonably assumes a significantly greater delay in the flip-flop sourcing A and A* than in the inverter sourcing TCK*. It is possible to design a circuit where this assumption is violated, causing a critical race to occur that would invalidate the behavior of the TAP controller. Therefore, it is recommended that implementations be fully verified to ensure compliance under required operating conditions.

1.4 Conventions

The following conventions are used in this standard:

a) The rules, recommendations, and permissions in each Specifications subclause are contained in a single alphabetically indexed list. References to each rule, recommendation, or permission are shown in the form:

```
              15.1.1 c) 2)
                  |  |  |
 Subclause number   |  |
                Index   |
          Option (if any)
```

b) Instruction and state names defined in this standard are shown in *italic* type in the text of this standard.
c) Names of states and signals that pertain to the test data registers defined by this standard contain the characters DR, while those that pertain to the instruction register contain the characters IR.
d) Names for signals that are active in their low state have an asterisk as the final character, e.g., TRST*.
e) A positive logic convention is used; i.e., a logic 1 signal is conveyed as the more positive of the two voltages used for logic signals.

2. References

This standard shall be used in conjunction with the following standards. When the following standards are superseded by an approved revision, the revision shall apply.

EIA/JEP106, JEDEC Publication 106, Standard Manufacturer's Identification Code[1]

IEEE 100, *The Authoritative Dictionary of IEEE Standards Terms*, Seventh Edition[2]

IEEE Std 1076-1993, IEEE Standard VHDL Language Reference Manual

IEEE Std 1149.4-1999, IEEE Standard for a Mixed Signal Test Bus

3. Definitions and Acronyms

For the purposes of this standard, the following terms and definitions apply. IEEE 100, *The Authoritative Dictionary of IEEE Standards Terms*, Seventh Edition should be referenced for terms not defined in this clause.

3.1 Definitions

3.1.1 active: When associated with a logic level (e.g., active-low), this term identifies the logic level to which a signal shall be set to cause the defined action to occur. When referring to an output driver (e.g., an active drive), this term describes the state in which the driver is capable of determining the voltage of the network to which it is connected.

[1]Copies can be obtained from EIA/JEDEC, 2500 Wilson Blvd., Arlington, VA 22201-3834, USA (www.jedec.org).
[2]IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA (http://standards.ieee.org/).

3.1.2 bidirectional pin: A component pin that can either drive or receive signals from external connections.

3.1.3 blind interrogation: Access to a facility (e.g., the device identification register) without prior knowledge of the test logic operation of the specific component being accessed.

3.1.4 built-in logic block observer (BILBO): A shift-register based structure used in some forms of self-testing circuit design.

3.1.5 capture: Load a value into a data register or the instruction register as a consequence of entry into the *Capture-DR* or *Capture-IR* controller state, respectively.

3.1.6 chip-on-board testing: A test of a component after it has been assembled onto a printed circuit board or other substrate.

3.1.7 clock: A signal where transitions between the low and high logic levels (or vice versa) are used to indicate when a stored-state device, such as a flip-flop or latch, may perform an operation.

3.1.8 falling edge: A transition from a high to a low logic level. In positive logic, a change from logic 1 to logic 0. Events that are specified to occur on the rising (falling) edge of a signal should be completed within a fixed (frequency-independent) delay specified by the component supplier.

3.1.9 high: The higher of the two voltages used to convey a single bit of information. For positive logic, a logic 1.

3.1.10 inactive: When referring to an output driver (e.g., an inactive drive), this term describes the state in which the driver is not capable of determining the voltage of the network to which it is connected.

3.1.11 input pin: A component pin that receives signals from an external connection.

3.1.12 instruction: A binary data word shifted serially into the test logic in order to define its subsequent operation.

3.1.13 least significant bit (LSB): The digit in a binary number representing the lowest numerical value. For shift-registers, the bit located nearest to the serial output, or the first bit to be shifted out. The least significant bit of a binary word or shift-register is numbered 0.

3.1.14 level-sensitive scan design (LSSD): A variant of the scan design technique that results in race-free, testable digital electronic circuits.

3.1.15 low: The lower of the two voltages used to convey a single bit of information. For positive logic, a logic 0.

3.1.16 most significant bit (MSB): The digit in a binary number representing the greatest numerical value. For shift-registers, the bit farthest from the serial output, or the last bit to be shifted out. Logic values expressed in binary form are shown with the most significant bit on the left.

3.1.17 nonclock: A signal where the transitions between the low and high logic levels do not themselves cause operation of stored-state devices. The logic level is important only at the time of a transition on a clock signal.

3.1.18 output pin: A component pin that drives signals onto external connections.

3.1.19 pin: The point at which connection is made between the integrated circuit and the substrate on which it is mounted (e.g., the printed circuit board). For packaged components, this would be the package pin; for components mounted directly on the substrate, this would be the bonding pad.

3.1.20 prime source: In the event that several vendors offer pin-for-pin compatible components, the prime source is the vendor that introduced the component type.

3.1.21 private: A design feature intended solely for use by the component manufacturer.

3.1.22 public: A design feature, documented in the component data sheet, that may be used by purchasers of the component.

3.1.23 reset: The establishment of an initial logic condition that can be either logic 0 or logic 1, as determined by the context.

3.1.24 rising edge: A transition from a low to a high logic level. In positive logic, a change from logic 0 to logic 1. Events that are specified to occur on the rising (falling) edge of a signal should be completed within a fixed (frequency-independent) delay specified by the component supplier.

3.1.25 scan design: A design technique that introduces shift-register paths into digital electronic circuits and thereby improves their testability.

3.1.26 scan path: The shift-register path through a circuit designed using the scan design technique.

3.1.27 second source: In the event that several vendors offer pin-for-pin compatible components, second-source suppliers are vendors of the component other than the prime source.

3.1.28 selected test data register: A test data register is selected when it is required to operate by an instruction supplied to the test logic.

3.1.29 signature analysis: A technique for compressing a sequence of logic values output from a circuit under test into a small number of bits of data (signature) that, when compared to stored data, will indicate the presence or absence of faults in the circuit.

3.1.30 stand-alone testing: A test of a component performed before it is assembled onto a board or other substrate, for example, using Automatic Test Equipment (ATE).

3.1.31 stuck-at fault: A failure in a logic circuit that causes a signal connection to be fixed at 0 or 1 regardless of the operation of the circuitry that drives it.

3.1.32 system: Pertaining to the nontest function of the circuit.

3.1.33 system logic: Any item of logic that is dedicated to realizing the nontest function of the component or is at the time of interest configured to achieve some aspect of the nontest function.

3.1.34 system pin: A component pin that feeds, or is fed from, the on-chip system logic.

3.1.35 test logic: Any item of logic that is a dedicated part of the test logic architecture or is at the time of interest configured as a part of the test logic architecture.

3.1.36 update: Transfer of a logic value from the shift-register stage of a data register cell or an instruction register cell into the latched parallel output stage of the cell as a consequence of the falling edge of the test clock input in the *Update-DR* or *Update-IR* controller state, respectively.

3.1.37 3-state pin: A component output pin where the drive may be either active or inactive (for example, at high impedance).

3.2 Acronyms

ATE	automatic test equipment
BILBO	built-in logic block observer
LSB	least significant bit
LSSD	level-sensitive scan design
MSB	most significant bit
TAP	test access port (see Clause 4)
TCK	test clock input (see 4.2)
TDI	test data input (see 4.4)
TDO	test data outputs (see 4.5)
TMS	test mode select (see 4.3)
TRST*	test reset (see 4.6)
TTL	transistor transistor logic

4. Test access port (TAP)

The TAP is a general-purpose port that can provide access to many test support functions built into a component, including the test logic defined by this standard. It is composed as a minimum of the three input connections and one output connection required by the test logic defined by this standard. An optional fourth input connection provides for asynchronous initialization of the test logic defined by this standard.

4.1 Connections that form the test access port (TAP)

4.1.1 Specifications

Rules

a) The TAP shall include the following connections (defined in 4.2, 4.3, 4.4, and 4.5): TCK, TMS, TDI, and TDO.
b) Where the TAP controller is not reset at power-up as a result of features built into the test logic, a TRST* input shall be provided as defined in 4.6 (see also 6.3).
c) All TAP inputs and outputs shall be dedicated connections to the component (i.e., the pins used shall not be used for any other purpose).

4.1.2 Description

Dedicated TAP connections are required to allow access to the full range of mandatory features of this standard.

4.2 Test clock input (TCK)

The test clock input (TCK) provides the clock for the test logic defined by this standard.

4.2.1 Specifications

Rules

a) Stored-state devices contained in the test logic shall retain their state indefinitely when the signal applied to TCK is stopped at 0.

Recommendations

b) Since TCK inputs for many components may be controlled from a single driver, care should be taken to ensure that the load presented by TCK is as small as possible.

Permissions

c) Stored-state devices contained in the test logic may retain their state indefinitely when the signal applied to TCK is stopped at 1.

4.2.2 Description

The dedicated TCK input is included so that the serial test data path between components can be used independently of component-specific system clocks, which may vary significantly in frequency from one component to the next. It also permits shifting of test data concurrently with normal system operation of the component. The latter facility is required to support the use of the TAP and test data registers in a design for on-line system monitoring. The provision of an independent clock ensures that test data can be moved to or from a chip without changing the state of the on-chip system logic. The independent clock is also essential if boundary-scan registers are to be usable for board interconnect testing in all circumstances—including cases where system clock signals are derived in one component for use in others.

While TCK will in many cases be driven by a free-running clock with a nominal 50% duty cycle, there may be situations where the clock needs to stop for a period. One example is when an ATE needs to fetch test data from backup memory (e.g., disc), since some test systems are unable to keep the clock running during such an operation. This standard requires that TCK can be stopped at 0 indefinitely without causing any change to the state of the test logic. While the TCK signal is stopped at 0, stored-state devices are required to retain their state so that the test logic may continue its operation when clock operation restarts. Optionally, a component also may allow TCK to be stopped at 1 for an indefinite period.

Many parts of the test logic perform operations in response to the rising or falling edge of TCK, as indicated by the use of the phrase “on the rising (falling) edge of TCK.” These operations have to be completed within a fixed (frequency-independent) delay after the occurrence of the relevant change at TCK, and this delay has to be specified by the component supplier. Therefore, the phrase “on the rising (falling) edge of TCK” should be interpreted as “within a specified delay after the rising (falling) edge of TCK.”

NOTE—In many applications, the TCK signal applied to components that conform to this standard will have a duty cycle close to 50% (i.e., the periods that the clock spends at 0 and 1 will be equal) at a given frequency. It is expected that all propagation delays will be such that correct operation is achieved under these circumstances (50% duty cycle at a given TCK frequency), particularly when data is being transferred between TDO of one chip and TDI of another.

4.3 Test mode select input (TMS)

The signal received at TMS is decoded by the TAP controller to control test operations.

4.3.1 Specifications

Rules

a) The signal presented at TMS shall be sampled by the test logic on the rising edge of TCK.
b) The design of the circuitry fed from TMS shall be such that an undriven input produces a logical response identical to the application of a logic 1.

Recommendations

c) Since the TMS inputs for many components may be controlled from a single driver, care should be taken to ensure that the load presented by TMS is as small as possible.

4.3.2 Description

Rule 4.3.1 b) is included so that the TAP controller is forced into the *Test-Logic-Reset* controller state in the case of an undriven TMS pin. This ensures that normal operation of the complete design can continue without interference from the test logic (see 6.3). For TTL-compatible designs, the rule may be met by including a pull-up resistor in the component's TMS input circuitry.

Signal values presented at TMS are sampled by the test logic on the rising edge of TCK. It is expected that the bus master (ATE, bus controller, etc.) will change the signal driven to the TMS inputs of connected components on the falling edge of TCK. The waveforms shown elsewhere in this standard reflect this expectation.

4.4 Test data input (TDI)

Serial test instructions and data are received by the test logic at TDI.

4.4.1 Specifications

Rules

a) The signal presented at TDI shall be sampled into the test logic on the rising edge of TCK.
b) The design of the circuitry fed from TDI shall be such that an undriven input produces a logical response identical to the application of a logic 1.
c) When data is being shifted from TDI toward TDO, test data received at TDI shall appear without inversion at TDO after a number of rising and falling edges of TCK determined by the length of the instruction or test data register selected.

4.4.2 Description

The data pins (TDI and TDO) provide for serial movement of test data through the circuit. The requirement for data to be propagated from TDI to TDO without inversion is included to simplify the operation of components compatible with this standard linked on a printed circuit board.

Values presented at TDI are clocked into the selected register (instruction or test data) on a rising edge of TCK. It is expected that the bus master (ATE, bus controller, etc.) will change the signal driven to the TDI input of the first component on a serial board-level path on the falling edge of TCK. The waveforms shown elsewhere in this standard reflect this expectation.

Rule 4.4.1 b) is included so that open-circuit faults in the board-level serial test data path cause a defined logic value to be shifted into the test logic. Note that when this constant value is shifted into the instruction register, the bypass register will be selected (as will be discussed further in 8.4). For TTL-compatible designs, this rule may be met by inclusion of a pull-up resistor in the component's TDI input circuitry.

4.5 Test data output (TDO)

TDO is the serial output for test instructions and data from the test logic defined in this standard.

4.5.1 Specifications

Rules

a) Changes in the state of the signal driven through TDO shall occur only on the falling edge of TCK.
b) The TDO driver shall be set to its inactive drive state except when the scanning of data is in progress (see 6.2).

4.5.2 Description

To ensure race-free operation, changes on TAP inputs (TMS and TDI) are clocked into the test logic defined by this standard on the rising edge of TCK while changes at the TAP output (TDO) occur on the falling edge of TCK. Similarly, for test logic able to drive or receive signals from system pins (e.g., the boundary-scan register), signals driven out of the component from the test logic change state on the falling edge of TCK, while those entering the test logic are clocked in on the rising edge (as will be discussed in 9.3).

The contents of the selected register (instruction or data) are shifted out of TDO on the falling edge of TCK. In the illustrations given in this document, edge-operated circuit designs generally are used. For an edge-operated implementation, note that the TDO output changes shall be delayed until the falling edge of TCK, which can be achieved by including a flip-flop clocked by the falling edge of TCK in the TDO output buffer. Where the registers are constructed from master and slave latches controlled by nonoverlapping clocks, the retiming required by Rule 4.5.1 a) is an inherent feature of the design.

The ability of TDO to switch between active and inactive drive is required to allow parallel, rather than serial, connection of board-level test data paths in cases where this is required. In TTL or CMOS technologies, for example, this requirement may be met through use of a 3-state output buffer.

4.6 Test reset input (TRST*)

The optional TRST* input provides for asynchronous initialization of the TAP controller (see 6.3).

4.6.1 Specifications

Rules

a) If TRST* is included in the TAP, the TAP controller shall be asynchronously reset to the *Test-Logic-Reset* controller state when a logic 0 is applied to TRST* (see 6.3).

NOTE—As a result of this event, all other test logic in the component is asynchronously reset to the state required in the *Test-Logic-Reset* controller state.

b) If TRST* is included in the TAP, the design of the circuitry fed from that input shall be such that an undriven input produces a logical response identical to the application of a logic 1.
c) TRST* shall not be used to initialize any system logic within the component.

Recommendations

d) To ensure deterministic operation of the test logic, TMS should be held at 1 while the signal applied at TRST* changes from 0 to 1.

4.6.2 Description

Initialization of the TAP controller in turn causes asynchronous initialization of other test logic included in the design, as discussed in the subsequent clauses of this standard.

Rule 4.6.1 b) is included to ensure that in the case of an unterminated TRST* input, test logic operation can proceed under control of signals applied at the TMS and TCK inputs. For TTL-compatible designs, this rule may be met by inclusion of a pull-up resistor in the TRST* input circuitry of the component.

Rule 4.6.1 c) ensures that the test logic can be reset independently of the on-chip system logic. This allows the test logic to be disabled by hard-wiring TRST* to logic 0.

Recommendation 4.6.1 d) is included to ensure that the test logic responds predictably when the signal applied to TRST* changes from 0 to 1. If rising edges occur simultaneously at TRST* and TCK when a logic 0 is applied to TMS, a race will occur, and the TAP controller may either remain in the *Test-Logic-Reset* controller state or enter the *Run-Test/Idle* controller state.

4.7 Interconnection of components compatible with this standard

4.7.1 Specifications

Permissions

a) The TAP input and output connections may be interconnected at the board level in a manner appropriate to the assembled product.

4.7.2 Description

Figure 4-1, Figure 4-2, and Figure 4-3 illustrate three alternative board-level interconnections of components conforming to this standard.

In each example, the test bus may be controlled either by an ATE system or by a component that provides an interface to a test bus at the next level of product assembly (for example, at the board/backplane interface). In this standard, the device that controls the board-level test bus is referred to as the bus master.

Note that the minimum configuration (shown in Figure 4-1) contains

— Two broadcast signals (TMS and TCK) fed from the testability bus master to all slaves in parallel; and
— A serial path formed by a daisy-chain connection of the serial test data pins (TDI and TDO).

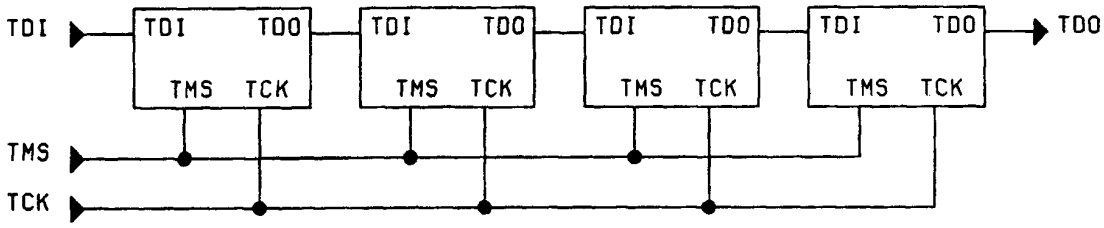


Figure 4-1—Serial connection using one TMS signal

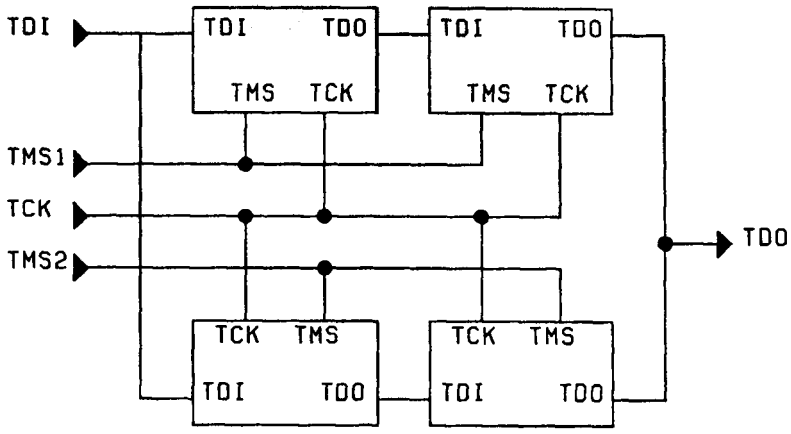


Figure 4-2—Connection in two paralleled serial chains

The hybrid serial/parallel connection shown in Figure 4-2 uses a pair of coordinated TMS signals (TMS1 and TMS2) to ensure that only one serial path is scanning data at a given time. This configuration makes use of the 3-state feature of the TDO output pin, which ensures that only the components that are scanning data have TDO in the active drive state.

Figure 4-3 shows the four components connected to give four separate serial paths through the complete board design. These paths have separate TDI and TDO signals but can be controlled from common TCK and TMS signals.

When choosing a configuration for the board-level interconnection of components conforming to this standard, it is necessary to consider the capability of test equipment and test pattern generators. It is fully expected that any test equipment and/or test pattern generators that intend to support a test methodology based on the boundary-scan architecture defined by this standard would be able to test the board-level configuration of Figure 4-1, since the degenerate form of this configuration is a single conformant component. On the other hand, some test equipment and/or test pattern generators may not be able to test the board-level configurations of Figure 4-2 and Figure 4-3.

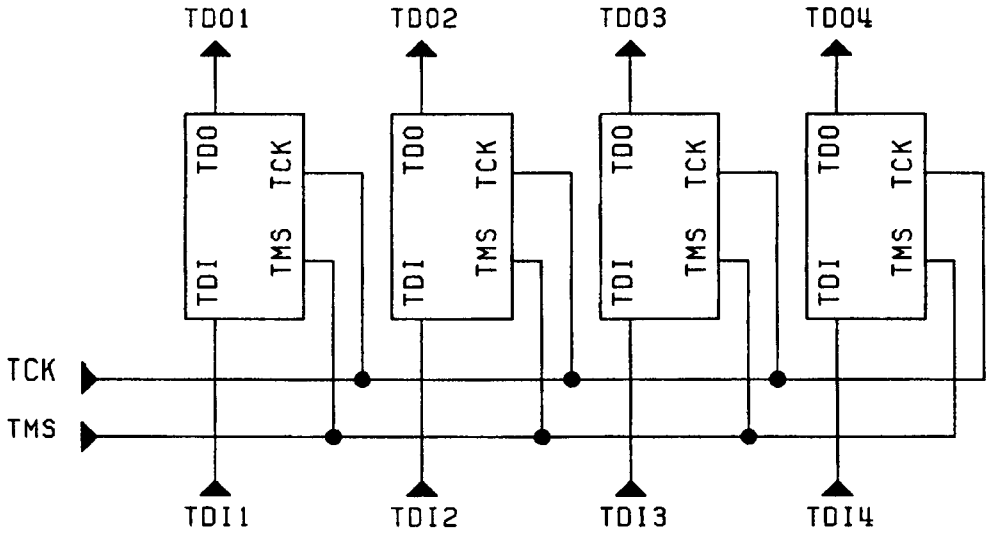


Figure 4-3—Multiple independent paths with common TMS and TCK signals

4.8 Subordination of this standard within a higher-level test strategy

While the test logic specified by this standard has been designed to be extensible to meet the particular needs of individual designers or companies (for example, by the flexibility of the instruction register), occasions may arise when it will be desirable to terminate compliance with this standard by a component temporarily and enable complementary test functionality. An example (illustrated in Annex A) involves a Level-Sensitive Scan Design (LSSD) infrastructure required for use during “stand-alone" component testing, which cannot be simultaneously operated with the test functionality defined by this standard (which is required to support testing of boards onto which the components implementing the two testing techniques will be assembled).

This subclause defines how compliance with this standard may be “switched on” or “switched off.” The rules require the change of test functionality to be under the control of signals applied at one or more component pins. Compliance has to be effected by a single logic pattern applied at these pins, not by a sequence of such patterns.

4.8.1 Specifications

Rules

a) If a component is to be designed having both
   1) Test functionality compliant with this standard; and
   2) Other test functionality that is not to be controlled via the test circuitry and the means of control defined in this standard,

compliance with this standard shall be enabled/disabled by one or more steady-state logic patterns (called “compliance-enable” patterns) applied at a fixed set of component inputs, to be called “compliance-enable inputs.”

NOTE—The steady-state combinational logic pattern may be chosen from a set of such “compliance-enable” patterns, all of which have equivalent effect [see Permission 4.8.1 h)].

b) Any one of the compliance-enable patterns, when applied to the compliance-enable inputs without regard to preceding patterns on these inputs, shall cause the component to be fully compliant with this standard.
c) Once compliance with this standard is established by the application of a compliance-enable pattern at the compliance-enable inputs, compliance to this standard shall be maintained continuously until the logic pattern applied at the compliance-enable inputs ceases to be a compliance-enable pattern.

NOTES

1—This rule implies that transition between compliance-enable patterns must produce no untoward effects on compliance. Limiting the number of compliance-enable patterns is one way to prevent problems from arising.

2—The rules in other subclauses of this standard apply only when compliance is enabled. Therefore, where compliance-enable inputs are provided, each rule should be considered to be prefaced by "When compliance to this standard is enabled, ..." For example, Rule 4.1.1 c) should be read as stipulating that the TAP pins are dedicated connections and may not be used for any other purpose while compliance to this standard is enabled. When compliance is disabled, the TAP connections may be reused, for example, to provide controls for an alternative test mode of component operation.

3—The event of enabling compliance with this standard by changing the logic pattern applied at the compliance-enable inputs of a component need not have an effect on the component equivalent to that of power-up of the component.

d) Compliance-enable inputs shall be dedicated inputs to the component and shall not be used for any other purpose.

Recommendations

e) The number of compliance-enable inputs provided on a component should be minimized.

Permissions

f) A component may have zero, one, or more compliance-enable inputs.
g) If a component with compliance-enable input(s) has a TRST* line included in its TAP implementation, the design of the component may require that the TRST* input be driven low at the time of application of a compliance-enable pattern in order to achieve reset of the relevant test logic concurrent with the operation of that test logic.
h) A component may have several compliance-enable patterns, all of which have an equivalent effect.

4.8.2 Description

If compliance-enable inputs are provided, there shall exist at least one logic pattern that, when applied at the compliance-enable inputs, will result in the component becoming fully compliant with this standard.

5. Test logic architecture

This clause defines the top-level design of the test logic accessed through the TAP. Detailed design requirements for the various blocks contained within the test logic design are contained in the subsequent clauses of this standard.

5.1 Test logic design

5.1.1 Specifications

Rules

a) The following elements shall be contained in the test logic architecture:
   1) A TAP (see Clause 4);
   2) A TAP controller (see Clause 6);
   3) An instruction register (see Clause 7); and
   4) A group of test data registers (see Clause 9).
b) The instruction and test data registers shall be separate shift-register based paths that are connected in parallel and have a common serial data input and a common serial data output connected to the TAP TDI and TDO signals, respectively.
c) The selection between the alternative instruction and test data register paths between TDI and TDO shall be made under the control of the TAP controller, as defined in 6.2.

5.1.2 Description

A conceptual view of the top-level design of the test logic architecture defined by this standard is shown in Figure 5-1. This figure and the others included in the descriptive material contained in this standard are examples intended only to illustrate a possible embodiment of this standard. *These figures do not indicate a preferred implementation*.

Key features of the design are as follows:

a) The TAP controller. This receives TCK and interprets the signals on TMS. The TAP controller generates clock or control signals or both as required for the instruction and test data registers and for other parts of the architecture. The specification for the TAP controller is contained in Clause 6.
b) The instruction register. This allows the instruction to be shifted into the design. The instruction is used to select the test to be performed or the test data register to be accessed or both. The specification for the instruction register is contained in Clause 7.
c) The group of test data registers. The group of test data registers shall include a bypass and a boundary-scan register. It also may include an optional device identification register and further optional test data registers. Further information on the structure of the group of test data registers is contained in Clause 9.

Note that, depending on the style of implementation of the test logic defined by this standard, circuitry may be required, in the output stage shown in Figure 5-1, to retime the signal passing through it to occur on the falling edge of TCK.

5.2 Test logic realization

5.2.1 Specifications

Rules

a) The TAP controller, the instruction register, and the associated circuitry necessary for control of the instruction and test data registers shall be dedicated test logic (i.e., these test logic blocks shall not perform any system function).
b) If test access is required to a test data register without causing any interference to the operation of the on-chip system logic, the circuitry used to construct that test data register shall be dedicated test logic.

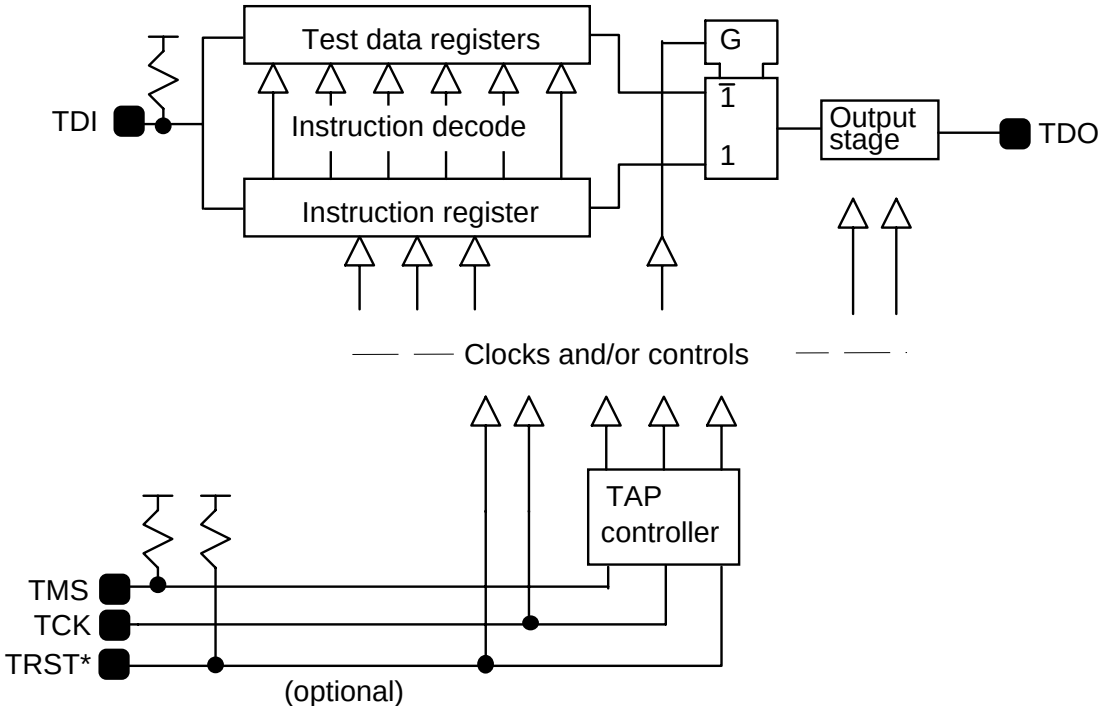


Figure 5-1—A conceptual schematic of the test logic

5.2.2 Description

While the example implementations contained in this standard show the various test data registers to be separate physical entities, circuitry may be shared between the test data registers provided that the rules contained in this standard are met. For example, this would allow the device identification register and the boundary-scan register to share shift-register stages, in which case the requirements of this standard would be met by operating the common circuitry in two different modes—the device identification register mode and the boundary-scan register mode.

6. The TAP controller

The TAP controller is a synchronous finite state machine that responds to changes at the TMS and TCK signals of the TAP and controls the sequence of operations of the circuitry defined by this standard.

6.1 TAP controller state diagram

6.1.1 Specifications

Rules

a) The state diagram for the TAP controller shall be as shown in Figure 6-1.

NOTE—The value shown adjacent to each state transition in this figure represents the signal present at TMS at the time of a rising edge at TCK.

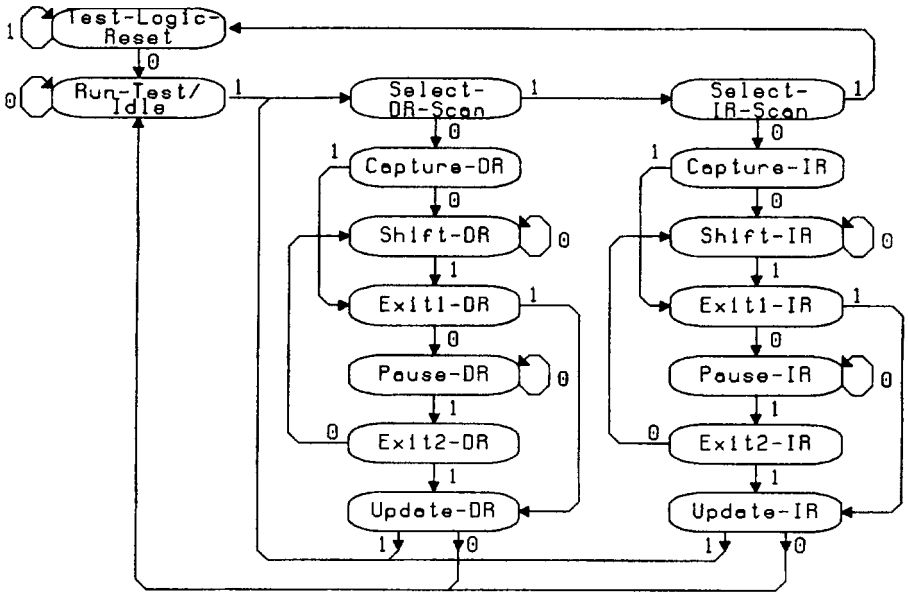


Figure 6-1—TAP controller state diagram

b) All state transitions of the TAP controller shall occur based on the value of TMS at the time of a rising edge of TCK.
c) Actions of the test logic (instruction register, test data registers, etc.) shall occur on either the rising or the falling edge of TCK in each controller state, as shown in Figure 6-2.

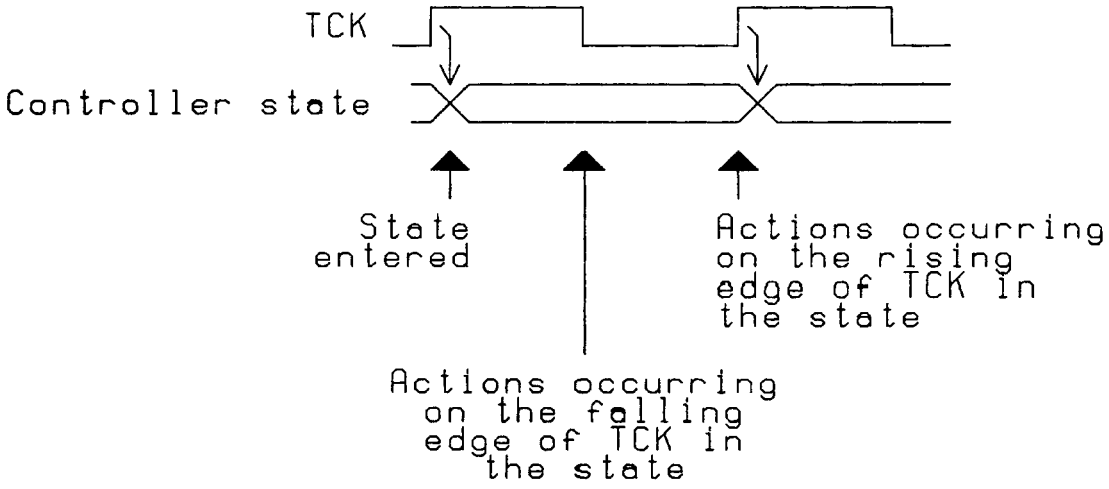


Figure 6-2—Timing of actions in a controller state

6.1.2 Description

The behavior of the TAP controller and other test logic in each of the controller states is briefly described as follows. Rules governing the behavior of the test logic defined by this standard in each controller state are contained in later clauses of this standard.

Test-Logic-Reset

The test logic is disabled so that normal operation of the on-chip system logic (i.e., in response to stimuli received through the system pins only) can continue unhindered. This is achieved by initializing the instruction register to contain the *IDCODE* instruction or, if the optional device identification register is not provided, the *BYPASS* instruction (see 7.2). No matter what the original state of the controller, it will enter *Test-Logic-Reset* when TMS is held high for at least five rising edges of TCK. The controller remains in this state while TMS is high.

If the controller should leave the *Test-Logic-Reset* controller state as a result of an erroneous low signal on the TMS line at the time of a rising edge on TCK (for example, a glitch due to external interference), it will return to the *Test-Logic-Reset* state after three rising edges of TCK with the TMS line at the intended high logic level. The operation of the test logic is such that no disturbance is caused to on-chip system logic operation as the result of such an error. On leaving the *Test-Logic-Reset* controller state, the controller moves into the *Run-Test/Idle* controller state, where no action will occur because the current instruction has been set to select operation of the device identification or bypass register (see 7.2). The test logic is also inactive in the *Select-DR-Scan* and *Select-IR-Scan* controller states.

Note that the TAP controller will also be forced to the *Test-Logic-Reset* controller state by applying a low logic level at TRST*, if such is provided, or at power-up (see 6.3).

Run-Test/Idle

A controller state between scan operations. Once entered, the controller will remain in the *Run-Test/Idle* state as long as TMS is held low. When TMS is high and a rising edge is applied at TCK, the controller moves to the *Select-DR-Scan* state.

In the *Run-Test/Idle* controller state, activity in selected test logic occurs only when certain instructions are present. For example, the *RUNBIST* instruction causes a self-test of the on-chip system logic to execute in this state (see 8.10). Self-tests selected by instructions other than *RUNBIST* also may be designed to execute while the controller is in this state.

For instructions that do not cause functions to execute in the *Run-Test/Idle* controller state, all test data registers selected by the current instruction shall retain their previous state (i.e., Idle).

The instruction does not change while the TAP controller is in this state.

Select-DR-Scan

This is a temporary controller state in which all test data registers selected by the current instruction retain their previous state.

If TMS is held low and a rising edge is applied to TCK when the controller is in this state, the controller moves into the *Capture-DR* state and a scan sequence for the selected test data register is initiated. If TMS is held high and a rising edge is applied to TCK, the controller moves on to the *Select-IR-Scan* state.

The instruction does not change while the TAP controller is in this state.

Select-IR-Scan

This is a temporary controller state in which all test data registers selected by the current instruction retain their previous state.

If TMS is held low and a rising edge is applied to TCK when the controller is in this state, then the controller moves into the *Capture-IR* state and a scan sequence for the instruction register is initiated. If TMS is held high and a rising edge is applied to TCK, the controller returns to the *Test-Logic-Reset* state.

The instruction does not change while the TAP controller is in this state.

Capture-DR

In this controller state data may be parallel-loaded into test data registers selected by the current instruction on the rising edge of TCK. If a test data register selected by the current instruction does not have a parallel input, or if capturing is not required for the selected test, the register retains its previous state unchanged.

The instruction does not change while the TAP controller is in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters either the *Exit1-DR* state if TMS is held at 1 or the *Shift-DR* state if TMS is held at 0.

Shift-DR

In this controller state, the test data register connected between TDI and TDO as a result of the current instruction shifts data one stage toward its serial output on each rising edge of TCK. Test data registers that are selected by the current instruction but are not placed in the serial path retain their previous state unchanged.

The instruction does not change while the TAP controller is in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller either enters the *Exit1-DR* state if TMS is held at 1 or remains in the *Shift-DR* state if TMS is held at 0.

Exit1-DR

This is a temporary controller state. If TMS is held high, a rising edge applied to TCK while in this state causes the controller to enter the *Update-DR* state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the *Pause-DR* state.

All test data registers selected by the current instruction retain their previous state unchanged.

The instruction does not change while the TAP controller is in this state.

Pause-DR

This controller state allows shifting of the test data register in the serial path between TDI and TDO to be temporarily halted. All test data registers selected by the current instruction retain their previous state unchanged.

The controller remains in this state while TMS is low. When TMS goes high and a rising edge is applied to TCK, the controller moves on to the *Exit2-DR* state.

The instruction does not change while the TAP controller is in this state.

Exit2-DR

This is a temporary controller state. If TMS is held high and a rising edge is applied to TCK while in this state, the scanning process terminates and the TAP controller enters the *Update-DR* controller state. If TMS is held low and a rising edge is applied to TCK, the controller enters the *Shift-DR* state.

All test data registers selected by the current instruction retain their previous state unchanged.

The instruction does not change while the TAP controller is in this state.

Update-DR

Some test data registers may be provided with a latched parallel output to prevent changes at the parallel output while data is shifted in the associated shift-register path in response to certain instructions (e.g., *EXTEST*, *INTEST*, and *RUNBIST*). Data is latched onto the parallel output of these test data registers from the shift-register path on the falling edge of TCK in the *Update-DR* controller state. The data held at the latched parallel output should not change other than in this controller state unless operation during the execution of a self-test is required (e.g., during the *Run-Test/Idle* controller state in response to a design-specific public instruction).

All shift-register stages in test data registers selected by the current instruction retain their previous state unchanged.

The instruction does not change while the TAP controller is in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters either the *Select-DR-Scan* state if TMS is held at 1 or the *Run-Test/Idle* state if TMS is held at 0.

Capture-IR

In this controller state the shift-register contained in the instruction register loads a pattern of fixed logic values on the rising edge of TCK. In addition, design-specific data may be loaded into shift-register stages that are not required to be set to fixed values (see Clause 7).

Test data registers selected by the current instruction retain their previous state. The instruction does not change while the TAP controller is in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters either the *Exit1-IR* state if TMS is held at 1 or the *Shift-IR* state if TMS is held at 0.

Shift-IR

In this controller state the shift-register contained in the instruction register is connected between TDI and TDO and shifts data one stage toward its serial output on each rising edge of TCK.

Test data registers selected by the current instruction retain their previous state. The instruction does not change while the TAP controller is in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller either enters the *Exit1-IR* state if TMS is held at 1 or remains in the *Shift-IR* state if TMS is held at 0.

Exit1-IR

This is a temporary controller state. If TMS is held high, a rising edge applied to TCK while in this state causes the controller to enter the *Update-IR* state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Pause-IR state.

Test data registers selected by the current instruction retain their previous state. The instruction does not change while the TAP controller is in this state and the instruction register retains its state.

Pause-IR

This controller state allows shifting of the instruction register to be halted temporarily.

Test data registers selected by the current instruction retain their previous state. The instruction does not change while the TAP controller is in this state and the instruction register retains its state.

The controller remains in this state while TMS is low. When TMS goes high and a rising edge is applied to TCK, the controller moves on to the *Exit2-IR* state.

Exit2-IR

This is a temporary controller state. If TMS is held high and a rising edge is applied to TCK while in this state, termination of the scanning process results, and the TAP controller enters the *Update-IR* controller state. If TMS is held low and a rising edge is applied to TCK, the controller enters the *Shift-IR* state.

Test data registers selected by the current instruction retain their previous state. The instruction does not change while the TAP controller is in this state and the instruction register retains its state.

Update-IR

The instruction shifted into the instruction register is latched onto the parallel output from the shift-register path on the falling edge of TCK in this controller state. Once the new instruction has been latched, it becomes the current instruction.

Test data registers selected by the current instruction retain their previous state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the *Select-DR-Scan* state if TMS is held at 1 or the *Run-Test/Idle* state if TMS is held at 0.

The *Pause-DR* and *Pause-IR* controller states are included so that shifting of data through the test data or instruction register can be halted temporarily. For example, this might be necessary in order to allow an ATE system to reload its pin memory from disc during application of a long test sequence. Boundary-scan test sequences are likely to extend to the order of 10^7 test patterns for complex board designs.

The TAP controller states include the three basic actions required for testing: stimulus application (*Update-DR*), execution (*Run-Test/Idle*), and response capture (*Capture-DR*). However, not all these actions are required for every type of test. Table 6-1 lists the actions required for key types of test supported by this standard.

For scan testing, the stimulus is made available for use at the end of shifting or, if a parallel output latch is included, by updating the parallel output in the *Update-DR* controller state. The results of the test are captured into the test data register during the *Capture-DR* controller state.

Table 6-1—Use of controller states for different test types

Test type	Action required in this controller state		
	Update-DR	***Run-Test/Idle***	***Capture-DR***
Boundary-scan external test (e.g., *EXTEST*)	Yes	No	Yes
Boundary-scan internal scan test (e.g., *INTEST*)	Maybe	Maybe	Yes
Boundary-scan *PRELOAD*	Yes	No	Maybe
Boundary-scan *SAMPLE*	Maybe	No	Yes
Internally controlled built-in tests (e.g., *RUNBIST*)	No	Yes	No
Internal scan test (i.e., a specific user-defined PUBLIC instruction)	Maybe	No	Yes

For internally controlled self-testing circuit designs, the starting values of the registers are available at the end of shifting: there is no parallel output latch to update. The registers should operate under control of the internal test logic during *Run-Test/Idle*. Since the result is already contained in a test data register, no action is required during the *Capture-DR* controller state.

For an internal scan test, the target register consists of a serial concatenation of storage elements that support the normal system operation of the component. The construction of such a register is beyond the scope of this standard, and parallel output latches may or may not be present in a given implementation.

6.2 TAP controller operation

6.2.1 Specifications

Rules

a) The TAP controller shall change state only in response to the following events:
   1) A rising edge of TCK;
   2) A transition to logic 0 at the TRST* input (if provided); or
   3) Power-up.

b) The TAP controller shall generate signals to control the operation of the test data registers, instruction registers, and associated circuitry as defined in this standard (Figure 6-3 and Figure 6-4).

NOTE—In these figures, the assumption is made that the signals applied to TMS and TDI change state on the falling edge of TCK. The time at which these signals change state is not defined by this standard, but it should be such that the set-up and hold requirements of TMS and TDI are met. It is further assumed that the design includes the optional device identification register. Therefore, the figures show the *IDCODE* instruction being set onto the output of the instruction register in the *Test-Logic-Reset* controller state. If the device identification register is not included in the design, the output of the instruction register will be set to the *BYPASS* instruction in the *Test-Logic-Reset* controller state.

c) The TDO output buffer and the circuitry that selects the register output fed to TDO shall be controlled as shown in Table 6-2.

d) Changes at TDO defined in Table 6-2 shall occur on the falling edge of TCK after entry into the state.

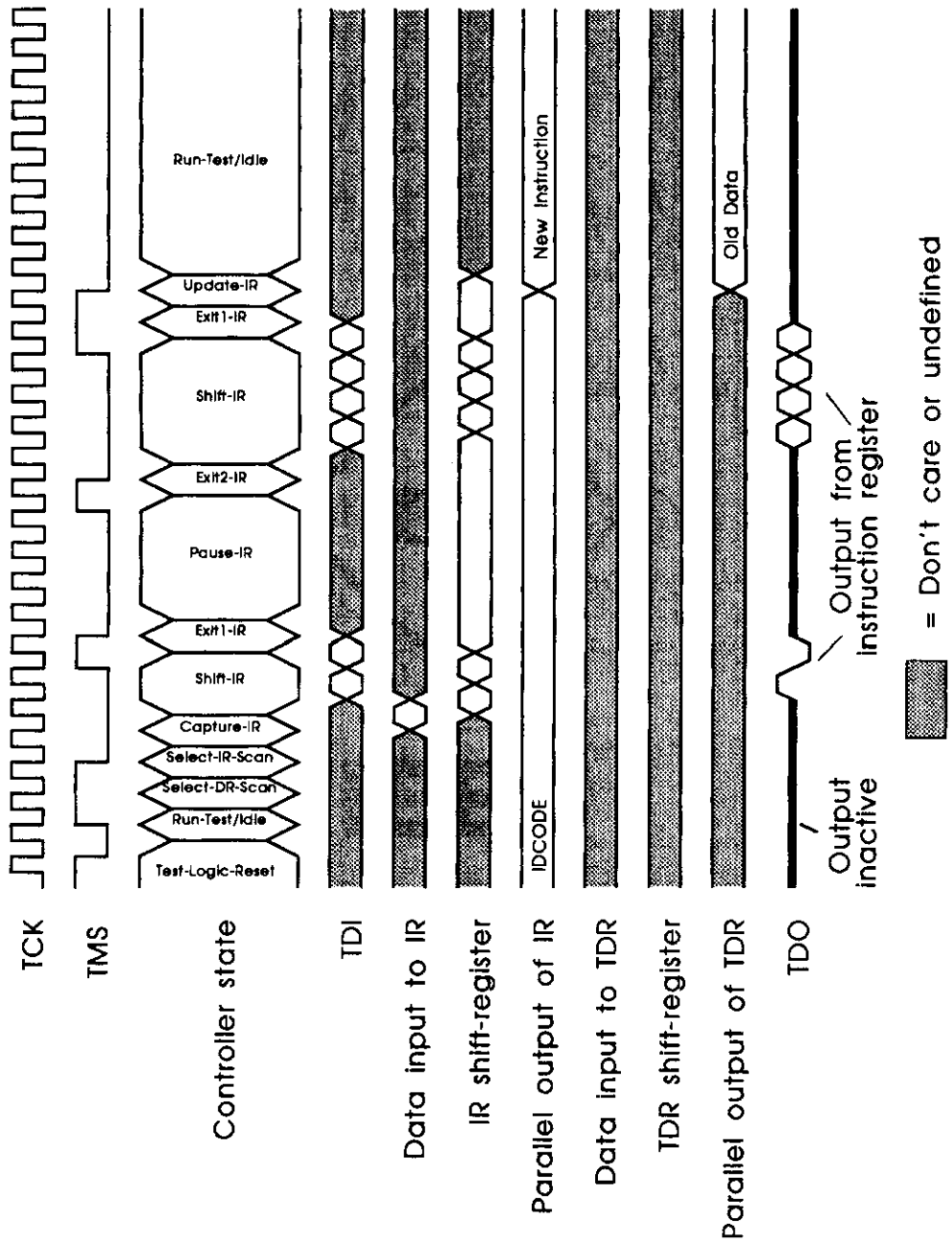


Figure 6-3—Test logic operation: instruction scan

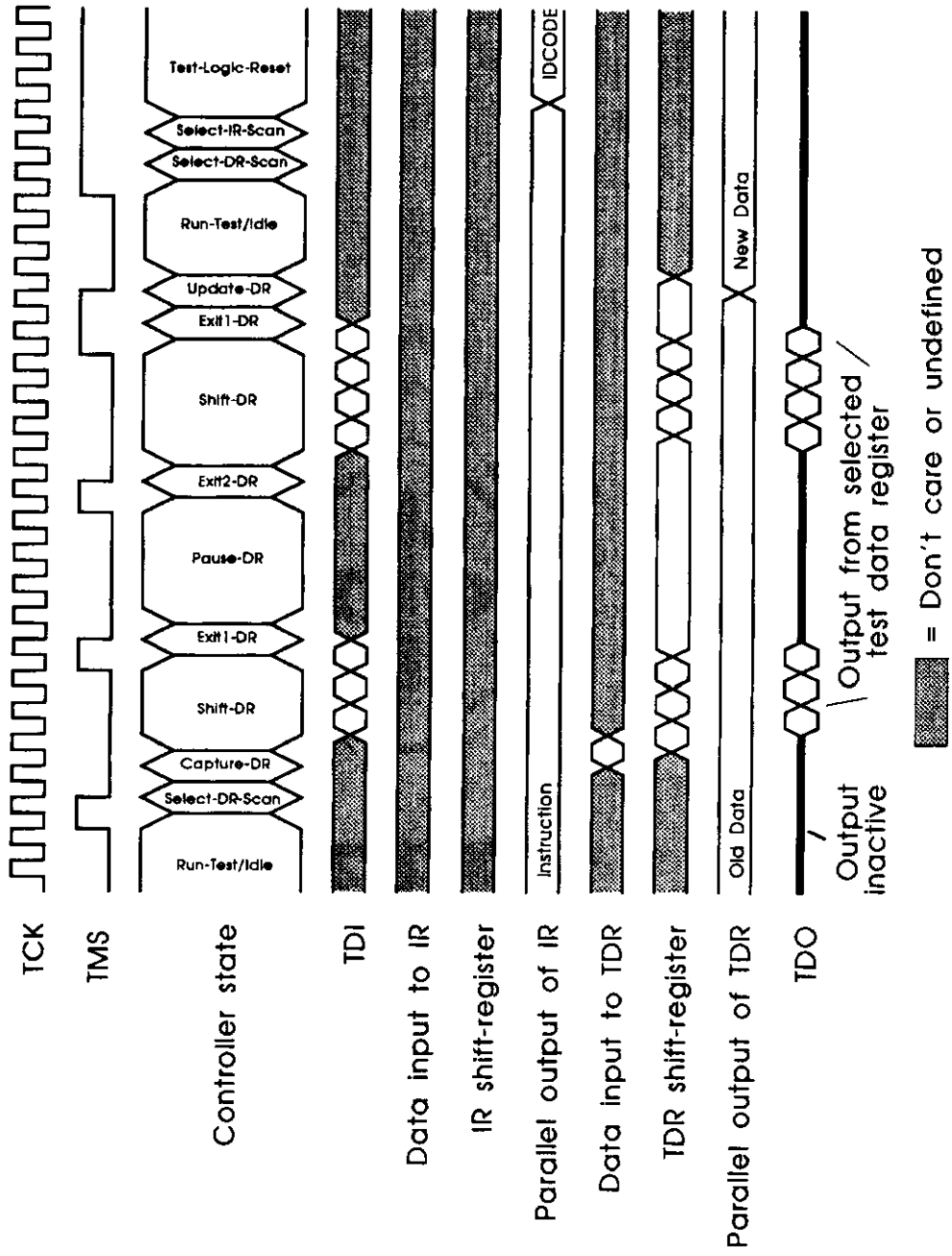


Figure 6-4—Test logic operation: data scan

Table 6-2—Test logic operation in each controller state

Controller state	Register selected to drive TDO	TDO driver
Test-Logic-Reset	Undefined	Inactive
Run-Test/Idle	Undefined	Inactive
Select-DR-Scan	Undefined	Inactive
Select-IR-Scan	Undefined	Inactive
Capture-IR	Undefined	Inactive
Shift-IR	Instruction	Active
Exit1-IR	Undefined	Inactive
Pause-IR	Undefined	Inactive
Exit2-IR	Undefined	Inactive
Update-IR	Undefined	Inactive
Capture-DR	Undefined	Inactive
Shift-DR	Test data	Active
Exit1-DR	Undefined	Inactive
Pause-DR	Undefined	Inactive
Exit2-DR	Undefined	Inactive
Update-DR	Undefined	Inactive

NOTE—Some components designed before publication of this standard may conform in every respect except that they have TDO active in the *Capture-IR*, *Pause-IR*, *Exit1-IR*, *Exit2-IR*, *Capture-DR*, *Pause-DR*, *Exit1-DR*, and *Exit2-DR* controller states, in addition to the *Shift-IR* and *Shift-DR* controller states. The functionality of these components is indistinguishable from that of components that fully conform to this standard except where the TDO output of such a component is connected to the TDO output of another (e.g., as shown in Figure 4-2).

6.2.2 Description

An example of a circuit that meets the requirements of 6.2.1 is shown in Figure 6-5 and Figure 6-6. This circuit generates a range of clock and control signals required not only to control the selection between the alternative instruction and test data register paths and the activity of TDO (as defined in Table 6-2) but also to control the example implementations of other items of test logic that are contained in this standard.

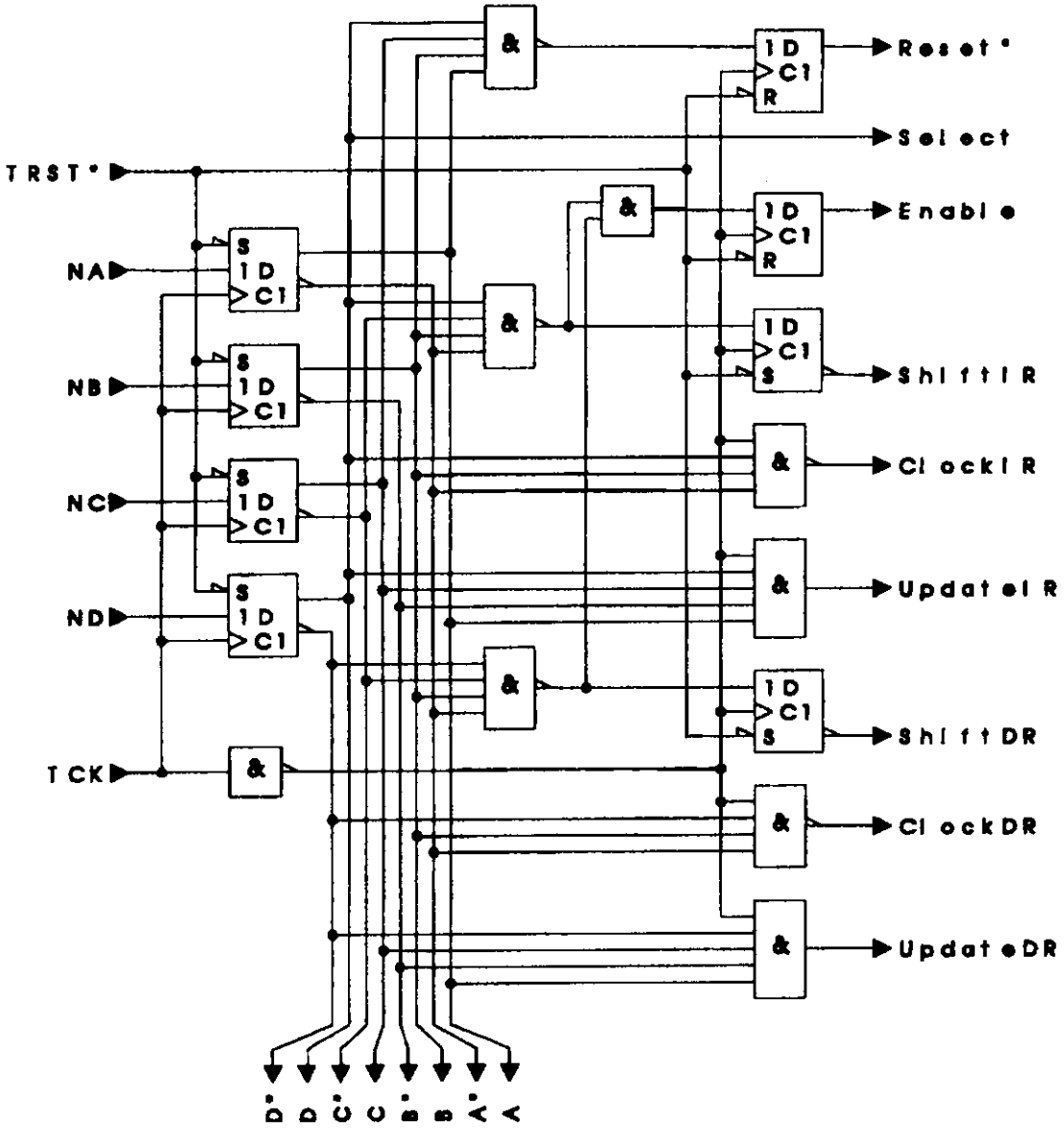
The assignment of controller states in the example implementation is given in Table 6-3.

The Boolean equations for the next state logic in Figure 6-5 and Figure 6-6 are as follows:

```
ND := DC* + DB + T*CB* + D*CB*A*
NC := CB* + CA + TB*
NB := T*BA* + T*C* + T*D*B + T*D*A* + TCB* + TDCA
NA := T*C*A + TB* + TA* + TDC
```

where

```
T = value present at TMS
```



NOTE—The circuit in Figure 6-5 generates the various control signals used by the example circuits illustrated elsewhere in this standard. Note that the Select signal would be used to control the multiplexer shown in Figure 5-1 and the Enable signal would be used for 3-state control of the TDO output. Note also that while the ShiftDR and ClockDR signals may be broadcast to all test data registers, distribution of the UpdateDR control signal will be controlled according to the instruction held in the instruction register such that the signal is fed only to the test data register that is selected as the serial path between TDI and TDO.

Figure 6-5—A TAP controller implementation—state registers and output logic

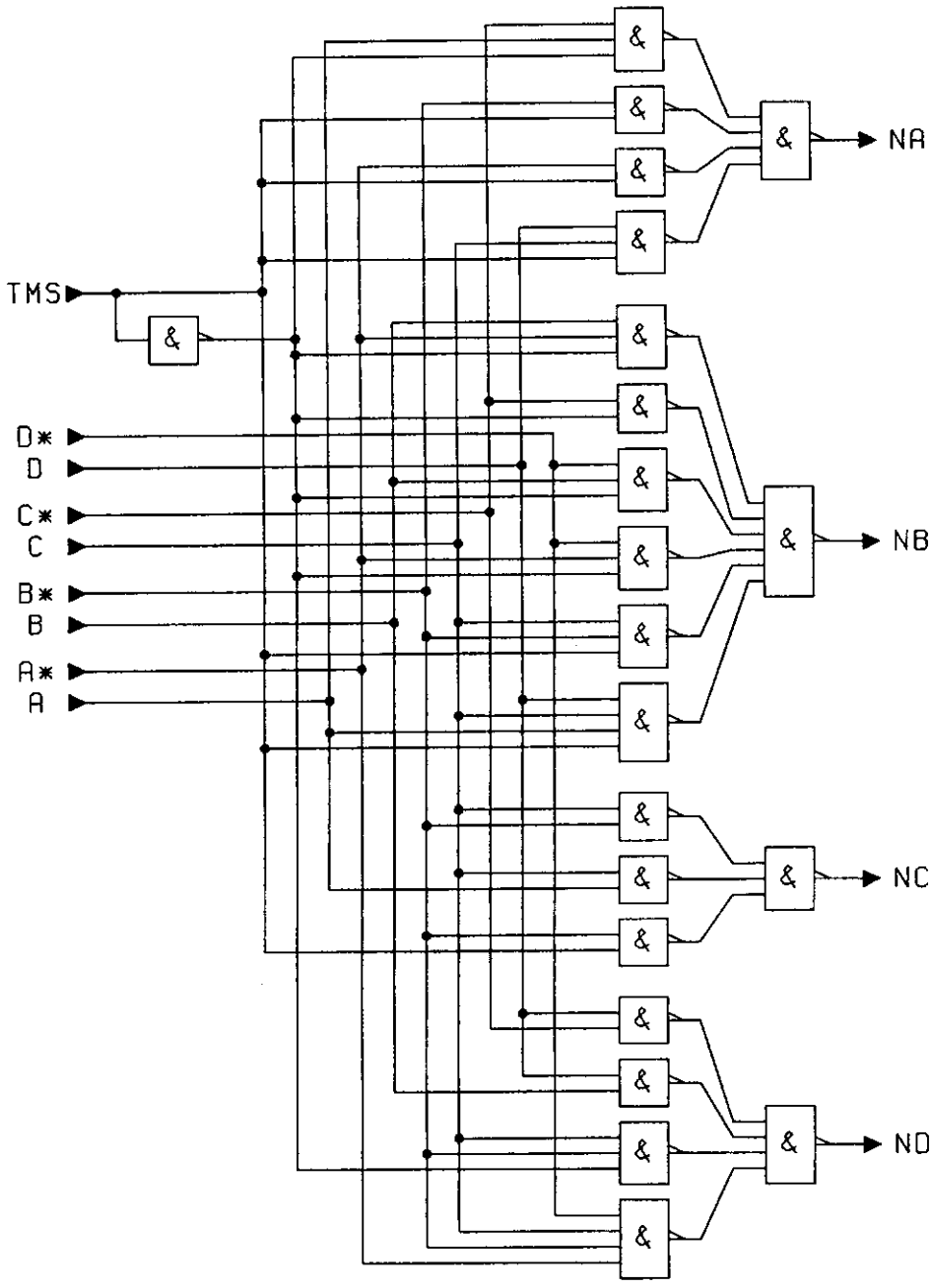


Figure 6-6—A TAP controller implementation—next state logic

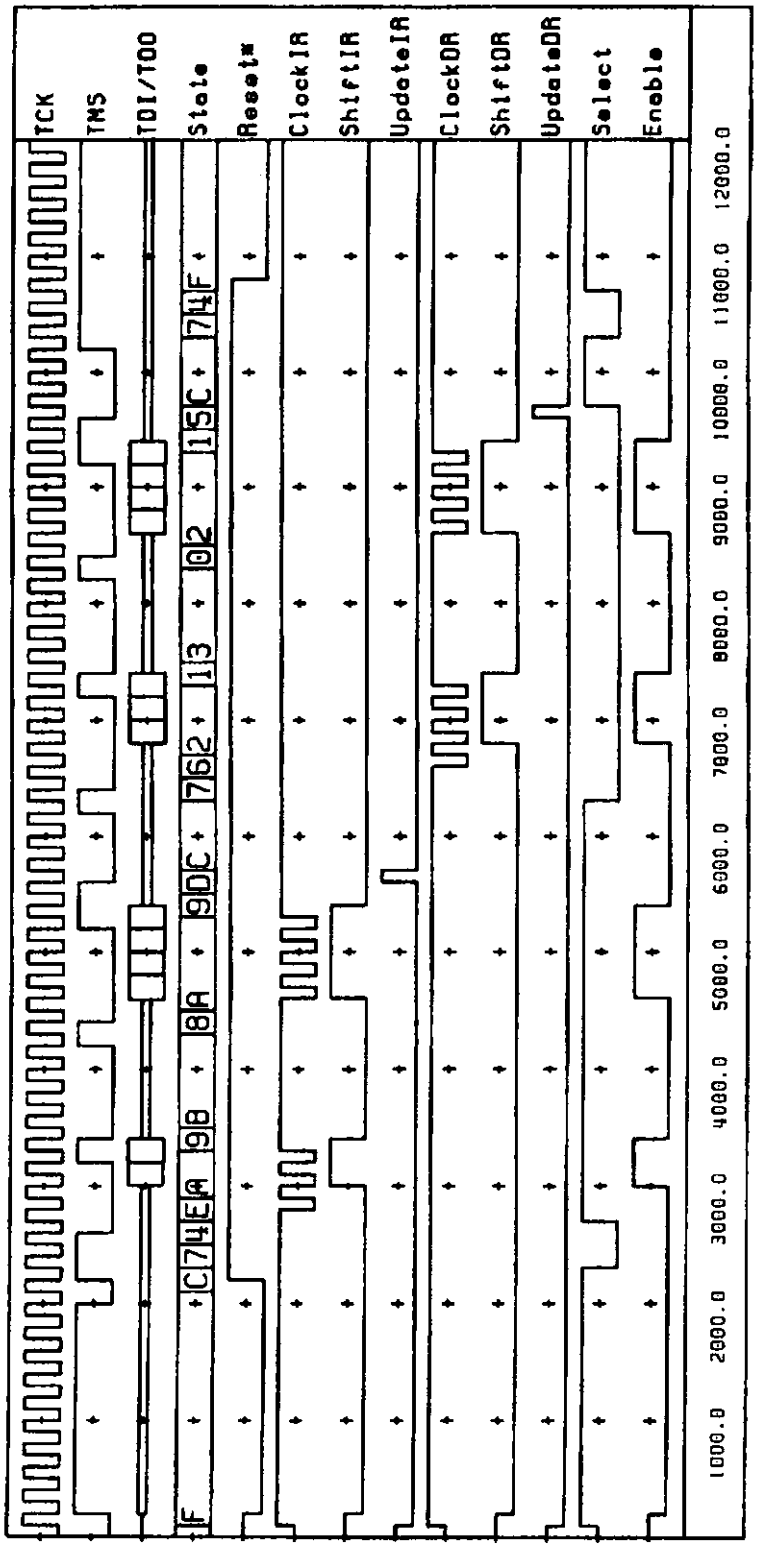


Figure 6-7—Operation of the example TAP controller

Figure 6-7 shows the operation of this controller implementation through instruction and test data register scan cycles.

Table 6-3—State assignments for example TAP controller

Controller State	DCBA (hex)
Exit2-DR	0
Exit1-DR	1
Shift-DR	2
Pause-DR	3
Select-IR-Scan	4
Update-DR	5
Capture-DR	6
Select-DR-Scan	7
Exit2-IR	8
Exit1-IR	9
Shift-IR	A
Pause-IR	B
Run-Test/Idle	C
Update-IR	D
Capture-IR	E
Test-Logic-Reset	F

6.3 TAP controller initialization

6.3.1 Specifications

Rules

a) The TAP controller shall be forced into the *Test-Logic-Reset* controller state at power-up either by use of the TRST* signal or as a result of circuitry built into the test logic.

NOTE—If the TAP controller is to be reset at power-up using TRST*, the design of the assembled system has to ensure that a logic 0 is applied to TRST* when power is applied. Similarly, where the TAP controller is to be reset using TRST* after enabling of compliance to this standard as described in 4.8, the design of the assembled system has to ensure that a logic 0 is applied to TRST* when compliance is enabled.

b) The TAP controller shall not be initialized by the operation of any system input, such as a system reset.

c) Where a dedicated reset pin (TRST*) is provided to allow initialization of the TAP controller, initialization shall occur asynchronously when the TRST* input changes to the low logic level.

d) Where the TAP controller is initialized at power-up by operation of circuitry built into the test logic, the result shall be equivalent to that which would be achieved by application of a logic 0 to a TRST* input.

6.3.2 Description

In a board design that contains wired junctions or buses, provision shall be made to ensure that at power-up any period of contention between drivers on the bus is kept within limits that ensure that no damage occurs to the components on the board.

When boundary-scan circuitry is inserted between the on-chip system logic and package pins, it becomes essential to ensure that shortly after power-up this circuitry enters a state where buses and wired junctions are controlled by the system circuitry, i.e., the *Test-Logic-Reset* controller state.

NOTE—Clause 11 contains rules that ensure that boundary-scan circuitry at system pins does not interfere with normal system operation when the *Test-Logic-Reset* controller state is selected.

While the TAP controller will synchronously enter the *Test-Logic-Reset* controller state after five rising edges at TCK (provided that TMS is held high), the worst-case time taken to reach this state may exceed that at which damage could occur. Further, it cannot be guaranteed that the clock will be running at the time at which power is applied to the board. Therefore, the “reset at power-up” requirement is included.

The requirement can be met in a variety of ways, for example, by inclusion of a power-up reset within the integrated circuit or by asymmetric design of the latches or registers used to construct the TAP controller. It also could be met by inclusion of a dedicated TRST* pin for the TAP controller. However, a system reset cannot also be used to initialize the TAP controller, since this would compromise the ability to test system interconnections at the board level using the boundary-scan circuitry. In some systems it also may be possible to use the independence of the system and test resets to allow sampling and examination of data after a system crash. This would require that the test logic be reset before reinitialization of the on-chip system logic.

Where a power-up reset facility is provided within the component, this can be used to initialize both the system and the test logic, for example, as shown in Figure 6-8.

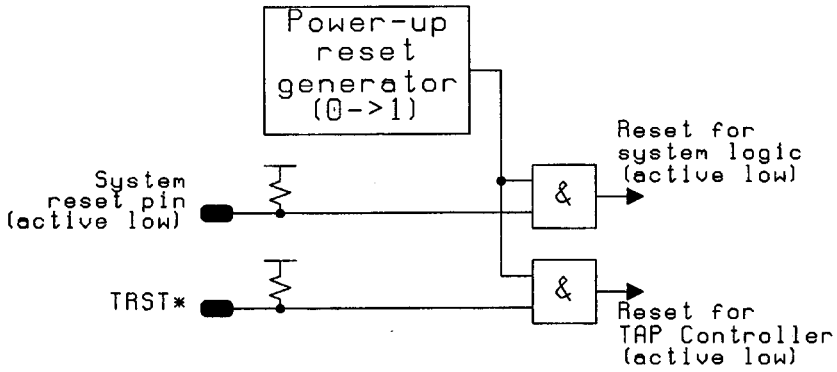


Figure 6-8—Use of power-up reset for system and test logic

7. The instruction register

The instruction register allows an instruction to be shifted into the design. The instruction is used to select the test to be performed or the test data register to be accessed or both. As will be discussed in Clause 8, a number of mandatory and optional instructions are defined by this standard. Further design-specific instructions can be added to allow the functionality of the test logic built into a component to be extended.

Optionally, the instruction register allows examination of design-specific information generated within the component.

This clause contains the design requirements for the instruction register.

7.1 Design and construction of the instruction register

The instruction register is a shift-register-based design that has an optional parallel input for register cells other than the two nearest to the serial output. The instruction shifted into the register is latched at the completion of the shifting process.

7.1.1 Specifications

Rules

a) The instruction register shall include at least two shift-register-based cells capable of holding instruction data.
b) The instruction shifted into the instruction register shall be latched such that changes in the effect of an instruction occur only in the *Update-IR* and *Test-Logic-Reset* controller states (see 7.2).
c) There shall be no inversion of data between the serial input and the serial output of the instruction register.
d) The two least significant instruction register cells (i.e., those nearest the serial output) shall load a fixed binary “01” pattern (the 1 into the least significant bit location) in the *Capture-IR* controller state (see 7.2).

Recommendations

e) Where the parallel inputs of instruction register cells are not required to load design-specific information, then these cells should be designed to load fixed logic values (0 or 1) in the *Capture-IR* controller state.

Permissions

f) Parallel inputs may be provided to instruction register cells (other than the two least significant cells) to permit capture of design-specific information in the *Capture-IR* controller state.

7.1.2 Description

The parallel output from the instruction register is latched to ensure that the test logic is protected from the transient data patterns that will occur in its shift-register stages as new instruction data is entered. The latched parallel output is controlled such that it can change state only in the *Update-IR* and *Test-Logic-Reset* controller states. The timing and nature of these changes are discussed in detail in 7.2.

The minimum size (two instruction register cells) is necessary to meet rules stated elsewhere in this standard:

a) The instruction register shall allow selection of the bypass register.

b) The instruction register shall allow access to the boundary-scan register in at least three configurations (*EXTEST*, *PRELOAD*, and *SAMPLE*—see 8.2).

It is permissible [see Permission 8.1.1 g)] for instructions to share binary codes provided that none of the rules that specify the merged instructions is violated. In earlier editions of this standard, SAMPLE and PRELOAD were specified as a merged instruction—*SAMPLE/PRELOAD*—such that the four instructions mandated by this standard—*BYPASS*, *EXTEST*, *PRELOAD*, and *SAMPLE*—could be implemented using three binary codes, leaving the fourth achievable with a minimum 2-bit instruction register available for implementation of an optional instruction.

In addition, fault isolation of the board-level serial test data path shall be supported. This is achieved by loading a constant binary “01” pattern into the least significant bits of the instruction register at the start of the instruction-scan cycle.

The inclusion of the optional design-specific data inputs to the instruction register allows key data signals within the device to be examined at the start of testing, with future test actions potentially depending on the design-specific information gathered. Where the parallel inputs to instruction register cells are not used for design-specific information, it is recommended that these cells be designed to load a fixed logic value (0 or 1) during the *Capture-IR* controller state.

7.2 Instruction register operation

7.2.1 Specifications

Rules

a) The behavior of the instruction register in each TAP controller state shall be as defined in Table 7-1.
b) All actions resulting from an instruction shall terminate when a different instruction is transferred to the parallel output of the instruction register (i.e., in the *Update-IR* or *Test-Logic-Reset* controller states).
c) All operations of shift-register stages shall occur on the rising edge of TCK after entry into a controller state.

Table 7-1—Instruction register operation in each controller state

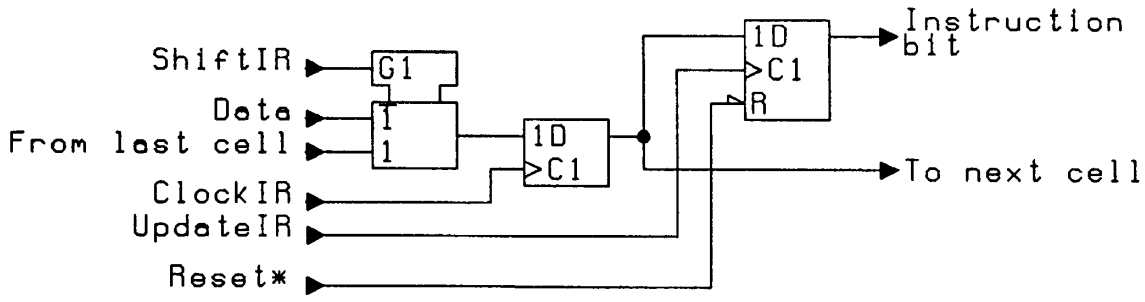
Controller state	Shift-register stage	Parallel output
Test-Logic-Reset	Undefined	Set to give the *IDCODE* (or *BYPASS*) instruction
Capture-IR	Load 01 into LSBs and, optionally, design-specific data or fixed values into MSBs	Retain last state
Shift-IR	Shift toward serial output	Retain last state
Exit1-IR *Exit2-IR* *Pause-IR*	Retain last state	Retain last state
Update-IR	Retain last state	Load from shift-register
All other states	Undefined	Retain last state

d) The data present at the parallel output of the instruction register shall be latched from the shift-register stage on the falling edge of TCK in the *Update-IR* controller state.
e) After entry into the *Test-Logic-Reset* controller state as a result of the clocked operation of the TAP controller, the *IDCODE* instruction (or, if there is no device identification register, the *BYPASS* instruction) shall be latched onto the instruction register output on the falling edge of TCK.
f) If the TRST* input is provided, when a low signal is applied to the input, the latched instruction shall change asynchronously to *IDCODE* (or, if no device identification register is provided, to *BYPASS*).

7.2.2 Description

Figure 7-1 shows an implementation of an instruction register cell that satisfies these requirements and operates in response to the signals generated by the example TAP controller design contained in 6.2:

a) The parallel output (labeled Instruction bit) is updated at the end of the instruction-scan cycle during the *Update-IR* controller state. This shall occur on the falling edge of TCK because a change in the latched instruction can result in a change at system output pins due to the operation of the boundary-scan register. Such changes shall occur on the falling edge of TCK as defined in Clause 11. Note that in Figure 7-1 an edge-triggered flip-flop is provided adjacent to the shift-register stage to meet this requirement. Alternative implementations, for example, where a level-operated latch is used or the storage element follows (rather than precedes) the instruction decoding logic, are permissible.



NOTE—The parallel output flip-flop in this figure is provided with a reset input. To meet Rules 7.2.1 e) and 7.2.1 f) , some or all instruction register cells will require use of a set, rather than a reset, input.

Figure 7-1—An instruction register cell

b) The clock input to the register in the serial path is applied only during the *Capture-IR* and *Shift-IR* controller states.
c) The parallel output is reset in the *Test-Logic-Reset* controller state as a result of a logic 0 received at the Reset* input of the cell. Referring to Figure 6-5 and Figure 6-6, notice that a low Reset* signal will be generated on the falling edge of TCK after entry into the *Test-Logic-Reset* controller state under control of TMS and TCK (TRST* held at 1). The parallel output of the instruction register will change on the falling edge of TCK, as is the case in the *Update-IR* controller state. In contrast, when a logic 0 is applied to TRST*, Reset* consequently is asserted (low) (see Figure 6-5) and the change at the parallel output occurs immediately, irrespective of the state of TMS or TCK. Note that some cells will need to be designed such that the parallel output is set high during this controller state so that the value of the *IDCODE* (or *BYPASS*) instruction is loaded onto the complete register's outputs as required by Rule 7.2.1 e).

d) Application of a 0 at TRST* causes the parallel output to be asynchronously set low. Again, some cells may need to be designed to be set high by TRST* such that the value of the *IDCODE* (or *BYPASS*) instruction is forced onto the register's outputs.

Note that the parallel data inputs to the two least significant stages (instruction register stages 0 and 1) shall be tied to fixed logic levels (1 for the least significant bit, 0 for the next-least significant bit).

Rule 7.2.1 b) ensures that the operation of the test data registers, etc., is determined only by the current instruction and that there is no possibility that actions resulting from any instruction (e.g., execution of an internal self-test) can continue once the instruction is removed. The circuit under test may not be in a known state if a new instruction is loaded before the previous one has run to completion.

8. Instructions

The instruction register allows instructions to be entered serially into the test logic during an instruction-register scan cycle. This clause defines the minimum range of instructions that shall be supplied and the operations that occur in response to those instructions. Optional instructions and the resulting operation of the test logic are also defined, together with the requirements for extensions to the instruction set defined in this standard.

8.1 Response of the test logic to instructions

8.1.1 Specifications

Rules

a) Each instruction shall completely define the set of test data register(s) that may operate and (where required) interact with the on-chip system logic while the instruction is current.
b) Test data registers that are not selected by the current instruction shall be controlled such that they do not interfere with the operation of the on-chip system logic or the selected test data registers.
c) Each instruction shall cause a single serial test data register path to be enabled to shift data between TDI and TDO in the *Shift-DR* controller state (as defined in Table 6-2).
d) Instruction binary codes that are not otherwise required to provide control of test logic shall be equivalent to the *BYPASS* instruction (see 8.4).

Recommendations

e) Use of the binary code {000...0} should be avoided for instructions that disrupt normal (i.e., nontest) operation of the component.

NOTE—Earlier editions of this standard mandated that a binary code for the *EXTEST* instruction be {000...0} (i.e., a logic 0 is loaded into every instruction register cell). While use of this binary code is no longer mandated, nor is it prohibited, it should be noted that such use may be detrimental to the implementation of high-reliability systems, as an apparent stuck-at-zero fault condition at a component's TDI pin could result in unexpected selection of *EXTEST* and consequent removal of the component from normal service.

Permissions

f) The mode of operation of a test data register may be defined by a combination of the current instruction and further control information contained in test data registers.
g) Two or more instructions may share a single binary code provided that all the rules for the separate instructions are met.

8.1.2 Description

The instructions loaded into the instruction register are decoded in order to achieve two key functions.

First, each instruction defines the set of test data registers that may operate while the instruction is current. Other test data registers should be controlled such that they cannot interfere with the operation of the on-chip system logic or with the operation of the selected test data registers. Several registers may be set into test modes simultaneously (for an example, see 9.2).

Second, an instruction defines the serial test data register path that is used to shift data between TDI and TDO during data register scanning. Note that a particular instruction may result in a single test data register being connected between TDI and TDO or in several test data registers being serially interconnected between TDI and TDO (for an example, see 9.2).

Rule 8.1.1 d) ensures that every pattern of 1s and 0s that can be fed into the instruction register produces a defined response and, in particular, that a test data register is connected between TDI and TDO for every possible instruction binary code.

Permission 8.1.1 g) allows for the merging of instructions to operate under a single binary code when their respective behaviors are not mutually exclusive. A prime example of such merging would be a sharing of a single binary code between *SAMPLE* and *PRELOAD*. The resulting merged behavior, which could be called *SAMPLE/PRELOAD*, would be fully equivalent to that mandated in earlier editions of this standard.

8.2 Public instructions

8.2.1 Specifications

Rules

a) Public instructions shall be available for use by purchasers of a component.
b) The following public instructions shall be provided in all components claiming conformance to this standard: *BYPASS*, *SAMPLE*, *PRELOAD*, and *EXTEST* (see 8.4, 8.6, 8.7, and 8.8, respectively).
c) If the optional device identification register is included in a component, the *IDCODE* instruction shall be provided.
d) If the optional device identification register is included in a user-programmable component that does not allow the programming via the test logic defined by this standard, then the *USERCODE* instruction shall be provided.
e) The binary codes for the *BYPASS* instruction shall be as defined in 8.4.

Recommendations

f) It is recommended that products support either the *INTEST* or the *RUNBIST* instruction or both (see 8.9 and 8.10).

Permissions

g) A design may offer public instructions in addition to those defined in this standard to give the device purchaser access to design-specific features.
h) Where binary codes for public instructions are not defined by this standard, they may be assigned as required for the particular design.

8.2.2 Description

Public instructions provide the component purchaser with access to test features that help in test tasks, e.g., go/no-go testing of the component via its self-test and board interconnect test via the boundary-scan register. The purchaser expects that the results of such tests will be independent of the variant of the component installed in a particular board, of the source of the component, etc. An exception, of course, is when the test results are intended to distinguish the variant, etc., as would be the case if the *IDCODE* instruction were used (see 8.13).

The binary code of an instruction is the sequence of data bits shifted serially into the instruction register from TDI during the *Shift-IR* controller state.

8.3 Private instructions

8.3.1 Specifications

Permissions

a) The public instructions may be supplemented with private instructions intended solely for the use of the component manufacturer.
b) The operation of private instructions need not be documented.
c) If private instructions are utilized in a component, the vendor shall clearly identify any instruction binary codes that, if selected, would cause hazardous operation of the component.

8.3.2 Description

Private instructions allow the component manufacturer to use the TAP and test logic to gain access to test features embedded in the design for design verification, production testing, or fault diagnosis. The component manufacturer may require tests performed using these features to give results that differ between variants of the component, for example, that would render documentation and use by component purchasers difficult.

Note that some private instructions may cause a component to operate in a manner that could be hazardous. For example, if a private instruction causes component inputs to become outputs for test data, etc., then damage may result if the instruction is selected while the component is surrounded by other components on an assembled board. The vendor shall therefore clearly identify any instruction binary codes that may cause hazardous operation if used by the component purchaser.

8.4 The *BYPASS* instruction

The bypass register contains a single shift-register stage and is used to provide a minimum-length serial path between the TDI and the TDO pins of a component when no test operation of that component is required. This allows more rapid movement of test data to and from other components on a board that are required to perform test operations.

8.4.1 Specifications

Rules

a) Each component shall provide a *BYPASS* instruction.
b) A binary code for the *BYPASS* instruction shall be {111...1} (i.e., a logic 1 entered into every instruction register cell).

c) The *BYPASS* instruction shall select the bypass register to be connected for serial access between TDI and TDO in the *Shift-DR* controller state.
d) When the *BYPASS* instruction is selected, all test data registers that can operate in either system or test modes shall perform their system function.
e) When the *BYPASS* instruction is selected, the operation of the test logic shall have no effect on the operation of the on-chip system logic.

Permissions

f) The *BYPASS* instruction may have binary codes in addition to that defined in Rule 8.4.1 b).

8.4.2 Description

The *BYPASS* instruction can be entered by holding TDI at a constant high value and completing an instruction-scan cycle. The demands on the host test system consequently are reduced in cases where access is required, say, to only chip 57 on a 100-chip board. In this case, the overall instruction pattern that shall be shifted into the design consists of a background of 1s with a small field of specific instruction data.

Note also that since the TDI input is designed such that when it is not terminated it behaves as though a high signal were being applied, an open circuit fault in the serial board-level test data path will cause the bypass register to be selected after an instruction-scan cycle. Therefore, no unwanted interference with the operation of the on-chip system logic can occur.

Where no device identification register is provided in a component, the *BYPASS* instruction is forced into the latches at the parallel outputs of the instruction register during the *Test-Logic-Reset* controller state. This ensures that a complete serial path through either bypass or device identification registers is established.

A consideration for usage is that if *BYPASS* is operated in some components while test-mode instructions (e.g., *EXTEST*) are operated in others, the normal system-logic operation of those components that are operating *BYPASS* may conflict with the test operation of the others. Therefore, careful analysis of interactions is necessary.

8.5 Boundary-scan register instructions

As discussed in Clause 1, the boundary-scan register is composed of cells connected between the on-chip system logic and the component's system input and output pins. This subclause is included to provide an overview of the structure and operation of the boundary-scan register that will assist the reader in understanding the specifications for the mandatory and optional instructions that make use of the boundary-scan register.

Design requirements for the boundary-scan register instructions are contained in 8.6 to 8.11. Requirements for the design of boundary-scan register cells are contained in Clause 11.

8.5.1 An overview of the operation of the boundary-scan register

The boundary-scan register is a shift-register-based structure that includes a variety of different cell designs matched onto the requirements of the particular component. Different cell designs are used according to the type of system pin concerned (input, output, 3-state, bidirectional) and according to the set of boundary-scan instructions supported.

A simplified view of a boundary-scan register is shown in Figure 8-1.

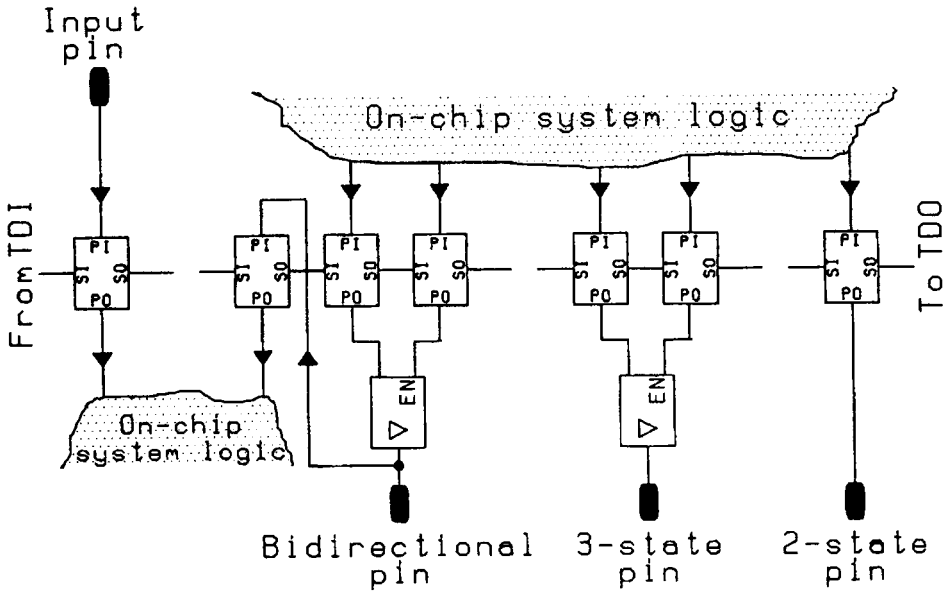


Figure 8-1—A simplified view of the boundary-scan register

An example implementation for a cell that could be used in each of the locations shown in Figure 8-1 is given in Figure 8-2.

The connections labeled PI, PO, SI, and SO in Figure 8-2 are connected to adjacent cells, the on-chip system logic, and the system pins, as shown in Figure 8-1. Like all the cells shown in this standard, that shown in Figure 8-2 is designed to respond to the ClockDR, ShiftDR, and UpdateDR signals generated by the example TAP controller implementation shown in Figure 6-5 and Figure 6-6. The Mode input shall be controlled according to the type of pin connected to the cell (input, output, etc.) and the specific instruction selected.

Use of this cell design, with appropriate signals supplied to the Mode input of each cell, will result in a component that supports the *SAMPLE*, *PRELOAD*, *EXTEST,* and *INTEST* instructions. As will be discussed in Clause 11, other cell designs are possible that meet the requirements of this standard for different sets of instructions. For example:

a) R2 may be either a flip-flop (as shown) or a latch.
b) R2 is optional for cells that feed data from a system pin to the on-chip system logic, e.g., the cells at system input pins. The lower input to M2 would in such cases be fed directly from the output of R1.
c) If the INTEST instruction were not supported, R2 and M2 could be omitted from cells that feed data from a system pin to the on-chip system logic. The input labeled PI then would be connected directly to the output labeled PO.

8.5.2 Specifications for boundary-scan register instructions

The specifications for boundary-scan instructions given in the following subclauses of this clause define

a) Whether the instruction is mandatory or optional;
b) Which test data registers can be connected in the serial path between TDI and TDO;
c) The restrictions (if any) on the choice of binary codes for each instruction (i.e., the patterns of 1s and 0s that, when shifted into the instruction register, cause the instruction to be selected); and
d) The flow of data between the component's system pins, the boundary-scan register cells, and the on-chip system logic.

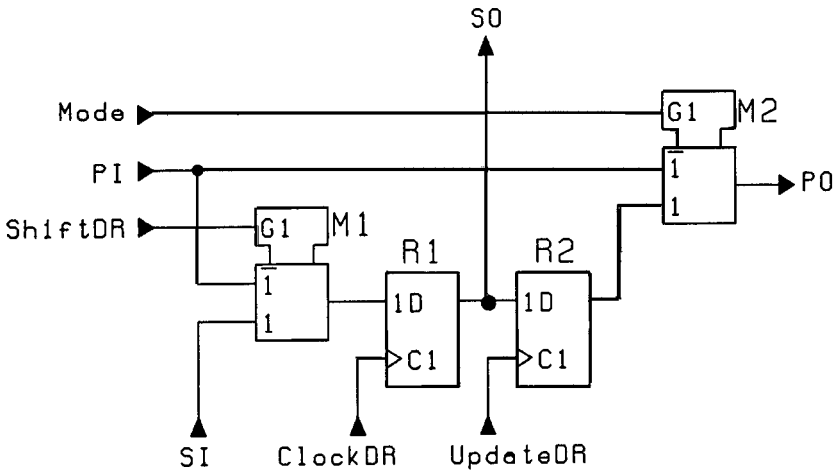


Figure 8-2—An example boundary-scan register cell design

The specifications are supported by descriptive text that includes a version of Figure 8-3 that shows one input and one output for a component. The solid bold lines in later copies of this figure show the mandatory data flows for each instruction.

8.6 The *SAMPLE* instruction

The mandatory *SAMPLE* instruction allows a snapshot of the normal operation of the component to be taken and examined.

8.6.1 Specifications

Rules

a) Each component shall provide a *SAMPLE* instruction.
b) The *SAMPLE* instruction shall select *only* the boundary-scan register to be connected for serial access between TDI and TDO in the *Shift-DR* controller state (i.e., no other test data register may be connected in series with the boundary-scan register).
c) When the *SAMPLE* instruction is selected, the operation of the test logic shall have no effect on the operation of the on-chip system logic or on the flow of signals between the system pins and the on-chip system logic.
d) When the *SAMPLE* instruction is selected, the states of all signals flowing from the on-chip system logic or through system pins (input or output) shall be loaded into the boundary-scan register on the rising edge of TCK in the *Capture-DR* controller state.

NOTE—The intent of this rule is to specify when the loading action should occur. Detailed specifications for the choices of signal values to be loaded are provided in Rules 11.5.1 f) and 11.6.1 h), respectively, for system logic inputs and system logic outputs.

Recommendations

e) Where each of *SAMPLE* and *PRELOAD* implements the functionality of the other, they should share a common binary value(s).

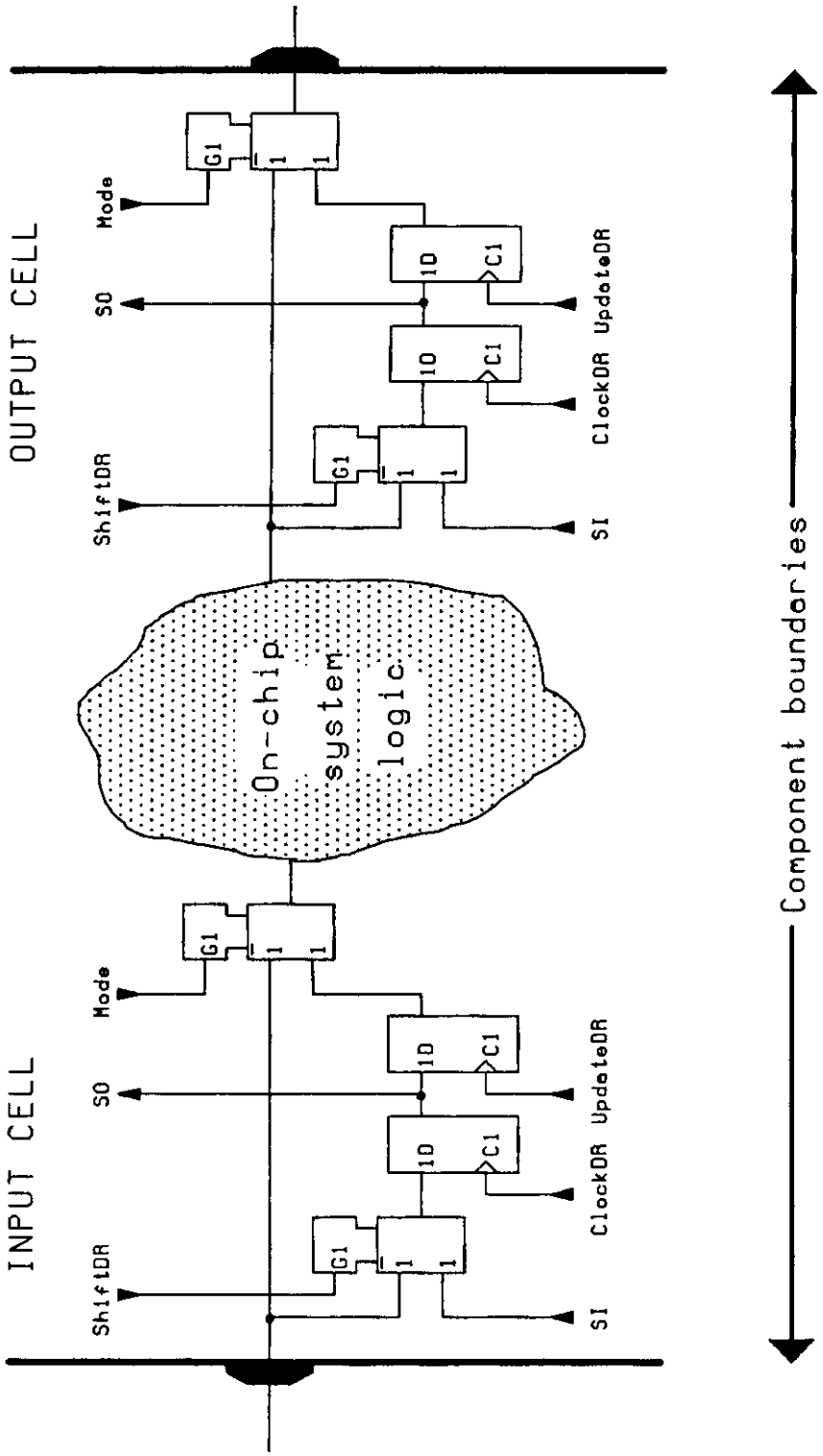


Figure 8-3—Circuit used to illustrate boundary-scan instructions

Permissions

f) When the *SAMPLE* instruction is selected, parallel output registers/latches included in boundary-scan register cells may load the data held in the associated shift-register stage on the falling edge of TCK in the *Update-DR* controller state.
g) The binary value(s) for the *SAMPLE* instruction may be selected by the component designer.

8.6.2 Description

The *SAMPLE* instruction allows a snapshot to be taken of the states of the component's input and output signals without interfering with the normal operation of the assembled board. The snapshot is taken on the rising edge of TCK in the *Capture-DR* controller state, and the data can then be viewed by shifting through the component's TDO output. Example applications of the *SAMPLE* capability are

a) To provide an analog to the guided-probing process performed on an assembled board during functional test diagnosis, but without the need for physical contact; and
b) Verification of the interaction between components during normal functional operation.

The flow of data for the *SAMPLE* instruction is shown in Figure 8-4. As can be seen, *SAMPLE* can be used without causing interference to the normal operation of the on-chip system logic. Data received at system input pins is supplied without modification to the on-chip system logic, data from the on-chip system logic is driven without modification through the system output pins, etc. For the example boundary-scan register cell design given in Figure 8-2, this is achieved by holding the Mode input at 0 when the *SAMPLE* instruction is selected.

NOTES

1—At output pins, the signal samples may be either that output from the component or that output from the on-chip system logic.

2—A component may be designed such that the *SAMPLE* and *PRELOAD* instructions are combined by assigning the same binary code to both. While *SAMPLE* captures data into the boundary-scan register and allows it to be shifted out through TDO for examination, a specific use of data shifted in at TDI is not mandated. In contrast, *PRELOAD* shifts data into the boundary-scan register through TDI such that it can be loaded into the register's parallel output registers/latches in advance of selecting an instruction (such as *EXTEST*) that supplies the data held in these registers/latches to the component's output pins. The data captured into the boundary-scan register before shifting is not defined. The mutual exclusivity of these behaviors permits the instructions to be merged where desired [see Permissions 8.1.1 g), 8.6.1 f), and 8.7.1 f)]. Further, where *SAMPLE* and *PRELOAD* instructions are merged in this fashion, by moving the TAP controller through the state sequence *Capture-DR* → *Exit1-DR* → *Update-DR* while the merged *SAMPLE/PRELOAD* instruction is selected, the state of the signals flowing into and out of the on-chip system logic at the time of sampling can be loaded onto the latched parallel output of the boundary-scan shift register.

8.7 The *PRELOAD* instruction

The mandatory *PRELOAD* instruction allows data values to be loaded onto the latched parallel outputs of the boundary-scan shift register before selection of the other boundary-scan test instructions.

8.7.1 Specifications

Rules

a) Each component shall provide a *PRELOAD* instruction.
b) The *PRELOAD* instruction shall select *only* the boundary-scan register to be connected for serial access between TDI and TDO in the *Shift-DR* controller state (i.e., no other test data register may be connected in series with the boundary-scan register).

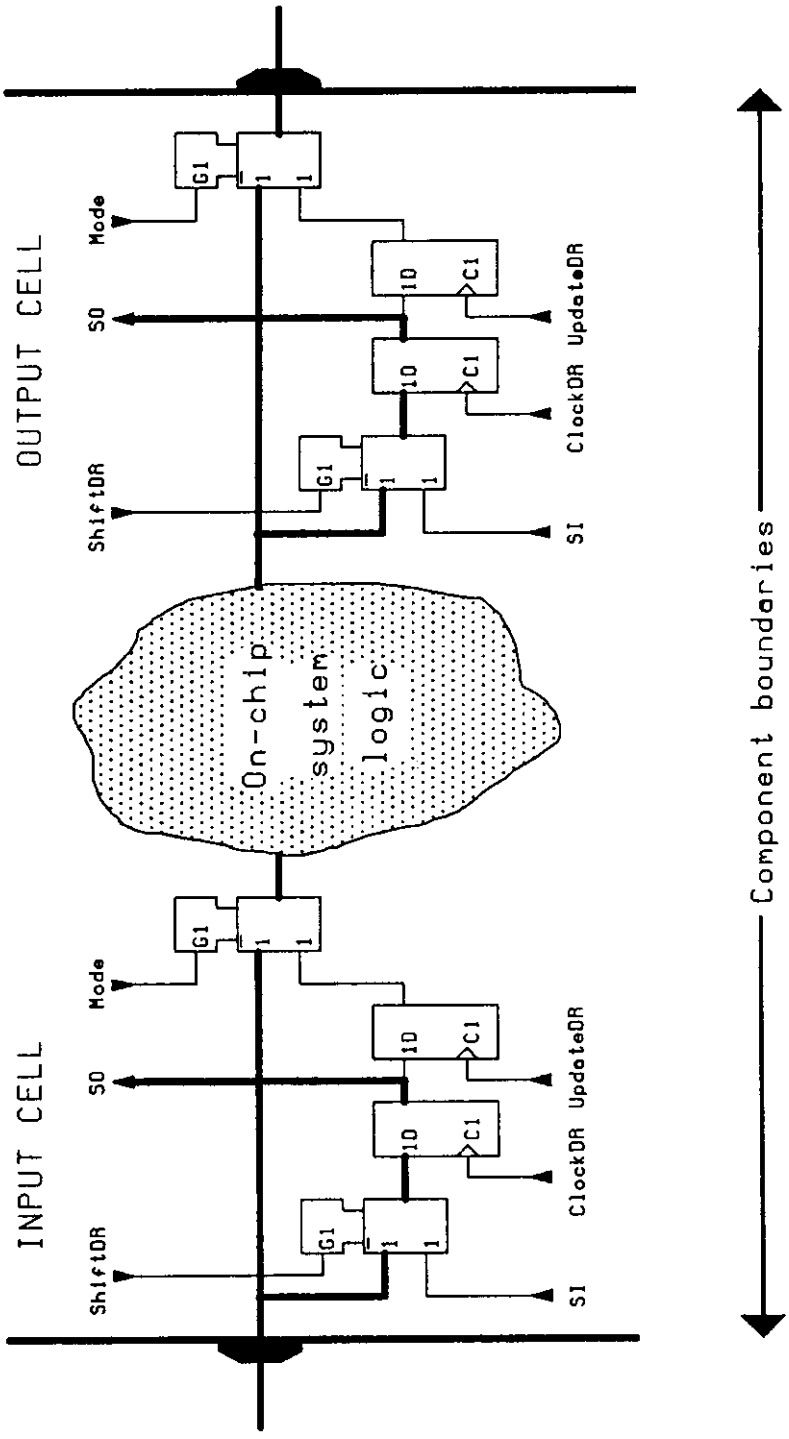


Figure 8-4—Data flow for the *SAMPLE* instruction

c) When the *PRELOAD* instruction is selected, the operation of the test logic shall have no effect on the operation of the on-chip system logic or on the flow of signals between the system pins and the on-chip system logic.
d) When the *PRELOAD* instruction is selected, parallel output registers/latches included in boundary-scan register cells shall load the data held in the associated shift-register stage on the falling edge of TCK in the *Update-DR* controller state.

Recommendations

e) Where each of *SAMPLE* and *PRELOAD* implements the functionality of the other, they should share a common binary value(s).

Permissions

f) When the *PRELOAD* instruction is selected, the states of all signals flowing through system pins (input or output) may be loaded into the boundary-scan register on the rising edge of TCK in the *Capture-DR* controller state.
g) The binary value(s) for the *PRELOAD* instruction may be selected by the component designer.

8.7.2 Description

The *PRELOAD* instruction allows scanning of the boundary-scan register without causing interference to the normal operation of the on-chip system logic. It thus allows an initial data pattern to be placed at the latched parallel outputs of boundary-scan register cells (e.g., as provided in the cells connected to system output pins) before the selection of another boundary-scan test operation. For example, before the selection of the *EXTEST* instruction, data can be loaded onto the latched parallel outputs using *PRELOAD*. As soon as the *EXTEST* instruction has been transferred to the parallel output of the instruction register, the preloaded data is driven through the system output pins. This ensures that known data, consistent at the board level, is driven immediately when the *EXTEST* instruction is entered; without *PRELOAD*, indeterminate data would be driven until the first scan sequence had been completed.

The flow of data for the *PRELOAD* instruction is shown in Figure 8-5. Data received at system input pins is supplied without modification to the on-chip system logic, data from the on-chip system logic is driven without modification through the system output pins, etc. For the example boundary-scan register cell design given in Figure 8-2, this is achieved by holding the Mode input at 0 when the *PRELOAD* instruction is selected.

NOTE—A component may be designed such that the *SAMPLE* and *PRELOAD* instructions are combined by assigning the same binary code to both. While *SAMPLE* captures data into the boundary-scan register and allows it to be shifted out through TDO for examination, a specific use of data shifted in at TDI is not mandated. In contrast, *PRELOAD* shifts data into the boundary-scan register through TDI such that it can be loaded into the register's parallel output registers/latches in advance of selecting an instruction (such as *EXTEST*) that supplies the data held in these registers/latches to the component's output pins. The data captured into the boundary-scan register before shifting is not defined. The mutual exclusivity of these behaviors permits the instructions to be merged where desired [see Permissions 8.1.1 g), 8.6.1 f), and 8.7.1 f)]. Further, where *SAMPLE* and *PRELOAD* instructions are merged in this fashion, by moving the TAP controller through the state sequence *Capture-DR* → *Exit1-DR* → *Update-DR* while the merged *SAMPLE/PRELOAD* instruction is selected, the state of the signals flowing into and out of the on-chip system logic at the time of sampling can be loaded onto the latched parallel output of the boundary-scan shift register.

8.8 The *EXTEST* instruction

The mandatory *EXTEST* instruction allows testing of off-chip circuitry and board level interconnections. Data typically would be loaded onto the latched parallel outputs of boundary-scan shift-register stages by using the *PRELOAD* instruction before selection of the *EXTEST* instruction.

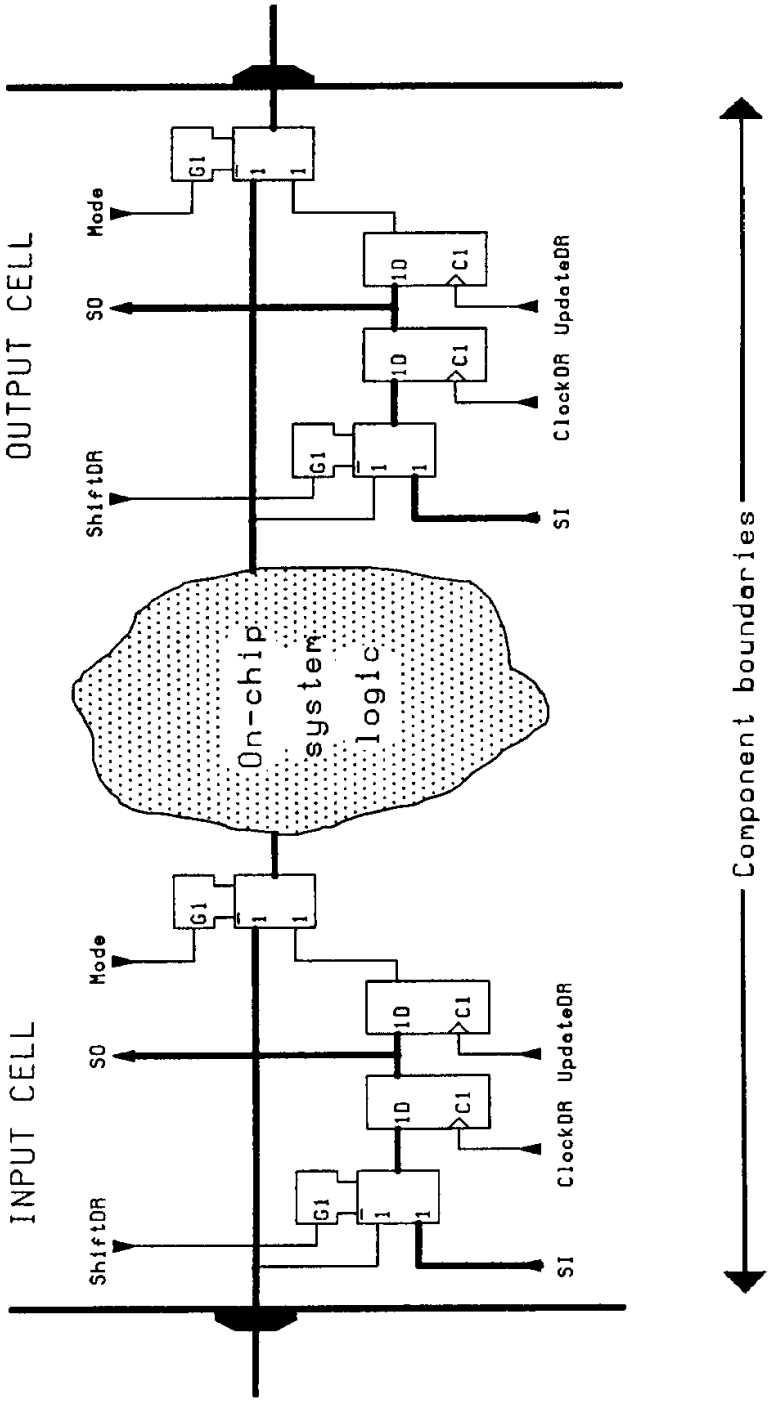


Figure 8-5—Data flow for the *PRELOAD* instruction

NOTE—After use of the *EXTEST* instruction, the on-chip system logic may be in an indeterminate state that will persist until a system reset is applied. Therefore, the on-chip system logic may need to be reset on return to normal (i.e., nontest) operation.

8.8.1 Specifications

Rules

a) Each component shall provide an *EXTEST* instruction.
b) The *EXTEST* instruction shall select only the boundary-scan register to be connected for serial access between TDI and TDO in the *Shift-DR* controller state (i.e., no other test data register may be connected in series with the boundary-scan register).
c) While the *EXTEST* instruction is selected, the on-chip system logic shall be controlled such that it cannot be damaged as a result of signals received at the system input or system clock input pins.

NOTE—This might be achieved by placing the on-chip system logic in a reset or "hold" state while the *EXTEST* instruction is selected.

d) When the *EXTEST* instruction is selected, the state of all signals driven from system output pins shall be completely defined by the data held in the boundary-scan register and change only on the falling edge of TCK in the *Update-DR* controller state.
e) When the *EXTEST* instruction is selected, the state of all signals received at system input pins shall be loaded into the boundary-scan register on the rising edge of TCK in the *Capture-DR* controller state.

Recommendations

f) The data loaded into boundary-scan register cells located at system output pins (2-state, 3-state, or bidirectional) in the *Capture-DR* controller state when the *EXTEST* instruction is selected should be independent of the operation of the on-chip system logic.
g) A value should be defined for each boundary-scan register cell that, when the *EXTEST* instruction is selected, will permit all component outputs to be overdriven simultaneously for an indefinite period without risk of damage to the component.

NOTE—This is easily achieved if all outputs can be set to an inactive drive state by previous use of the *PRELOAD* instruction.

Permissions

h) The binary value(s) for the *EXTEST* instruction may be selected by the component designer.

8.8.2 Description

The *EXTEST* instruction allows circuitry external to the component package—typically the board interconnect—to be tested. Boundary-scan register cells at output pins are used to apply test stimuli, while those at input pins capture test results. This instruction also allows testing of blocks of components that do not themselves incorporate boundary-scan registers. The flow of data through the boundary-scan register cells in this configuration is shown in Figure 8-6. For example, at input pins, data is first captured into the shift-register path and then shifted out of the component for examination; at output pins, data shifted into the component is applied to the external interconnection.

Typically, the first test stimulus to be applied using the *EXTEST* instruction will be shifted into the boundary-scan register using the *PRELOAD* instruction. Thus, when the change to the *EXTEST* instruction takes place in the *Update-IR* controller state, known data will be driven immediately from the component onto its external connections. Where a total of *N* tests are to be applied using the *EXTEST* instruction, stimuli

for tests 2 to *N* will be shifted in while the results from tests 1 to *N*-1 are shifted out. Note that while the results from the final test—test *N*—are shifted out, a determinate set of data shall be shifted in that will leave the board in a consistent state at the end of the shifting process. This can be achieved by shifting the stimuli for test *N* (or indeed any other test) into the boundary-scan register again.

The *EXTEST* instruction also allows component outputs to be set to a state that minimizes the risk of damage when overdriven during in-circuit testing [see Recommendation 8.8.1 g)]. Such testing may be used where not all components on an assembled board are testable via boundary scan.

Note that the boundary-scan register cells located at input pins may optionally be designed to allow signals to be driven into the on-chip system logic when the *EXTEST* instruction is selected. This allows user-defined values to be established at the system logic inputs, preventing misoperation in response to noise signals arriving from the board-level interconnect. The values driven may be constant for the duration that *EXTEST* is selected (e.g., by including a blocking gate at the input to the system logic) or may be loaded serially through the boundary-scan register, as shown in Figure 8-6.

Recommendation 8.8.1 f), where followed, ensures that data shifted out of the component in response to the *EXTEST* instruction is not altered by the presence of faults in the on-chip system logic. This simplifies diagnosis since any errors in the output bit stream can be caused only by faults in off-chip circuitry, in board-level interconnections, or in the boundary-scan registers used to apply the test.

While the *EXTEST* instruction is selected, the on-chip system logic may receive input signals that differ significantly from those expected during normal (nontest) operation. Rule 8.8.1 c) places the responsibility for correct handling of this situation on the component designer. If the on-chip system logic can tolerate any permutation of input signals that is received, no specific design changes are required to meet this rule. (An example here would be the case where the on-chip system logic is entirely combinational.) However, for some components there may be input sequences that could place the on-chip system logic in a state where damage may result. In these cases, it is the responsibility of the designer to prevent the on-chip system logic from processing the “illegal” inputs while the *EXTEST* instruction is selected. As noted, this may be achieved by placing the on-chip system logic into a reset or “hold” state.

Alternatively, the data held in the boundary-scan register may be presented to the on-chip system logic while the *EXTEST* instruction is selected. Note that where this is the case, Rule 11.3.1 e) prohibits the imposition of any restriction on the logic values that may be driven to the on-chip system logic.

Note that while earlier editions of this standard mandated that a binary code for *EXTEST* be {000...0}, the use of this binary code for *EXTEST* and all other test-mode instructions has been deprecated [see Recommendation 8.1.1 e)] and the associated note).

8.9 The *INTEST* instruction

The optional *INTEST* instruction is one of two instructions defined by this standard that allow testing of the on-chip system logic while the component is assembled on the board. Using the *INTEST* instruction, test stimuli are shifted in one at a time and applied to the on-chip system logic. The test results are captured into the boundary-scan register and are examined by subsequent shifting. Data typically would be loaded onto the latched parallel outputs of boundary-scan shift-register stages using the *PRELOAD* instruction before selection of the *INTEST* instruction.

The following rules apply where the *INTEST* instruction is provided.

NOTE—After use of the *INTEST* instruction, the on-chip system logic may be in an indeterminate state that will persist until a system reset is applied. Therefore, the on-chip system logic may need to be reset on return to normal (i.e., nontest) operation.

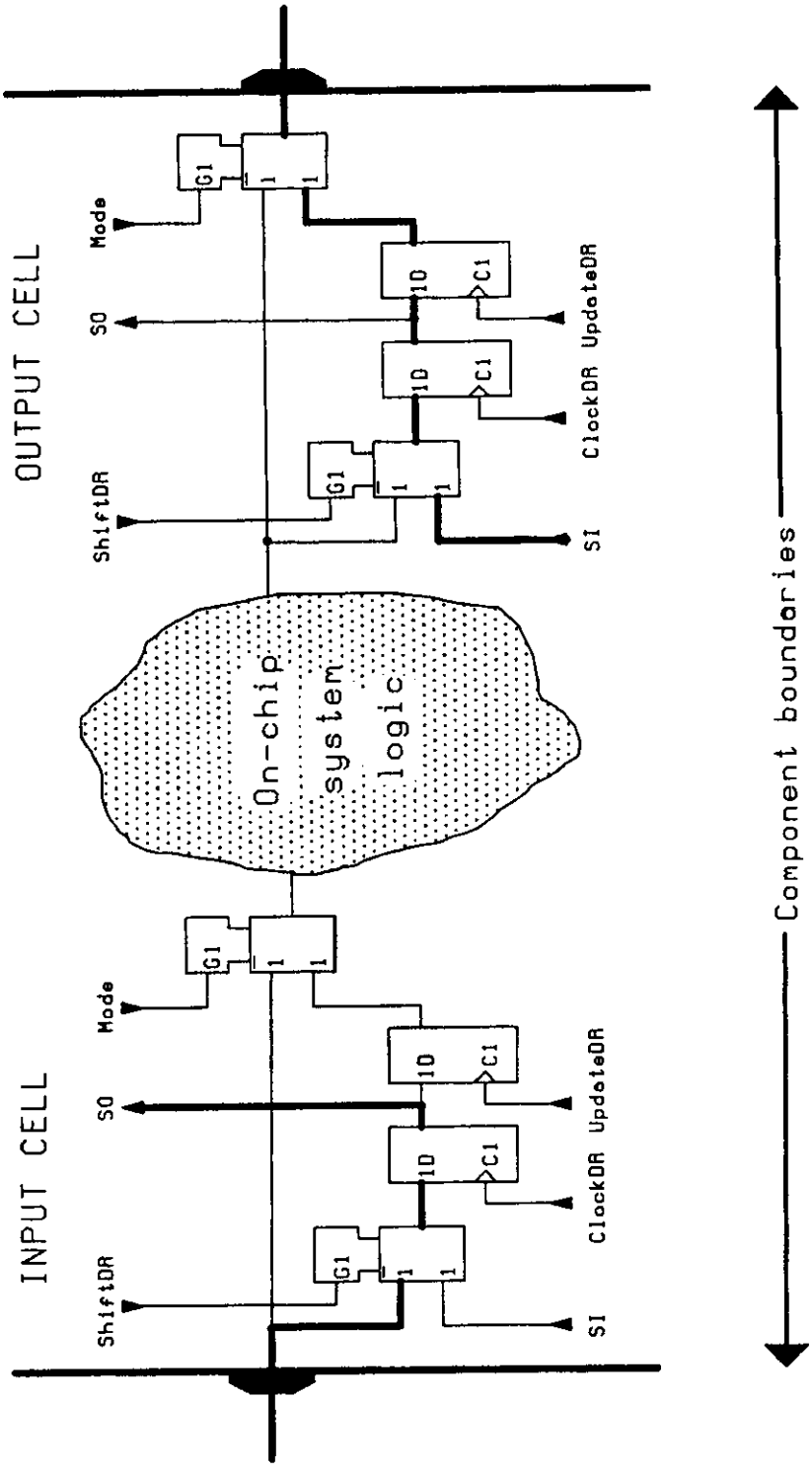


Figure 8-6—Test data flow while the *EXTEST* instruction is selected

8.9.1 Specifications

Rules

a) The *INTEST* instruction shall select only the boundary-scan register to be connected for serial access between TDI and TDO in the *Shift-DR* controller state (i.e., no other test data register may be connected in series with the boundary-scan register).
b) The on-chip system logic shall be capable of single-step operation while the *INTEST* instruction is selected.
c) When the *INTEST* instruction is selected, all system outputs from the component shall be defined as follows:
    1) All signals driven out of the component shall be defined by data held in the boundary-scan register and shall change only on the falling edge of TCK in the *Update-DR* controller state or on selection of the *INTEST* instruction; or
    2) All outputs from the component (including those that are 2-state nontest signals) shall be placed in an inactive drive state (e.g., high-impedance) on selection of the *INTEST* instruction.
d) When the *INTEST* instruction is selected, the state of all nonclock signals driven into the system logic from the boundary-scan register shall be completely defined by the data held in the register.
e) When the *INTEST* instruction is selected, the state of all signals output from the system logic to the boundary-scan register shall be loaded into the register on the rising edge of TCK in the *Capture-DR* controller state.

Recommendations

f) For boundary-scan register cells located at system input pins (clock or nonclock) or at bidirectional pins configured as inputs, the data loaded in the *Capture-DR* controller state when the *INTEST* instruction is selected should be independent of the operation of off-chip circuitry or board-level interconnections.

Permissions

g) The binary value(s) for the *INTEST* instruction may be selected by the component designer.

8.9.2 Description

The *INTEST* instruction allows static (slow-speed) testing of the on-chip system logic, with each test pattern and response being shifted through the boundary-scan register. The *INTEST* instruction requires that the on-chip system logic can be operated in a single-step mode, where the circuitry moves one step forward in its operation each time shifting of the boundary-scan register is completed.

The flow of data through the boundary-scan register cells while the instruction is selected is shown by the bold paths in Figure 8-7. The topmost bold path through the cell at the output pin is that taken by the results of the test of the on-chip system logic; the lowermost path is that taken by the data to be held at the pin while the test is applied. Note that for each test, the latched parallel output of the boundary-scan register cell at the system output pin is updated from data shifted in *before* the state of the shift-register is overwritten with the test response.

While the *INTEST* instruction is selected, the boundary-scan register assumes the role of the ATE system used for stand-alone component testing. Cells at nonclock system input pins are used to apply the test stimulus, while those at system output pins capture the response. Stimuli and responses are moved into and out of the circuit by shifting the boundary-scan register. Note that this requires that the boundary-scan register cells located at system input pins be able to drive signals into the on-chip system logic.

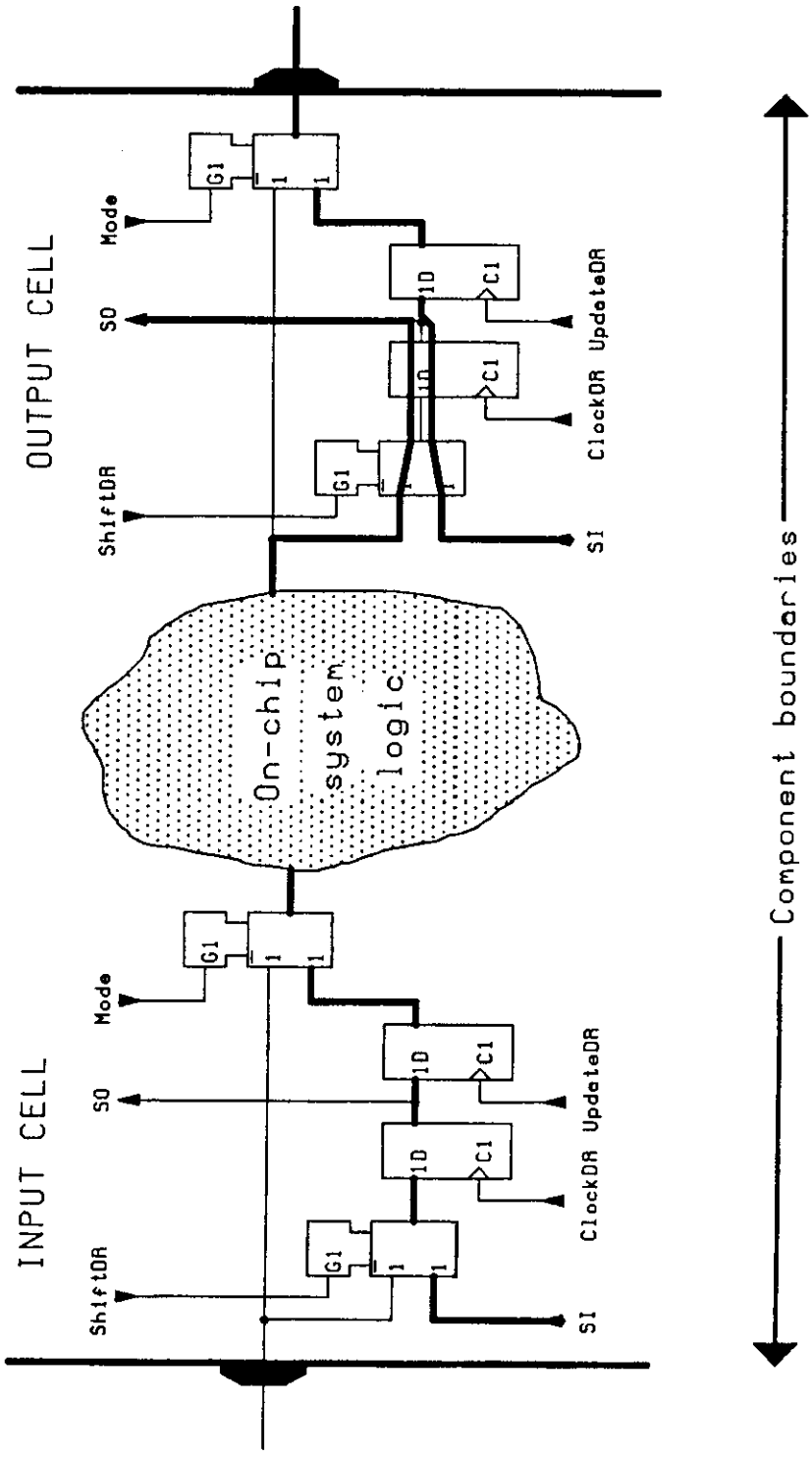


Figure 8-7—Test data flow while the *INTEST* instruction is selected

Typically, the on-chip system logic will receive a sequence of clock events between application of the stimulus and capture of the response such that single-step operation is achieved. The specification of boundary-scan register cells for system clock input pins allows the clocks for the on-chip system logic to be obtained in several ways while the *INTEST* instruction is selected. The following are offered as examples:

a) The signals received at system clock pins can be fed directly to the on-chip system logic as during normal operation of the component. Where this option is selected, the component design shall guarantee that precisely one single step of operation of the on-chip system logic occurs while, at least, a specified minimum number of TCK cycles are applied during the *Run-Test/Idle* controller state. The component shall be designed so that only one single step of operation is performed whether or not more than the specified minimum number of TCK cycles is applied while the TAP controller is in the *Run-Test/Idle* controller state. This may, for example, require that clock signals incoming to the component be gated before application to the on-chip system logic. In this way, operation of the on-chip system logic can be inhibited while test data is shifted through the boundary-scan register. Figure 8-8 illustrates how the system clock applied to the component should be controlled during testing of the on-chip system logic using the *INTEST* instruction.

   While Figure 8-8 illustrates a situation in which the system clock is a single positive-going pulse, Rule 8.9.1 b) can be generalized to apply to components that employ multiple clock cycles for each step of operation or that have several clock input pins at which multiphase clock signals are received. Note that while Figure 8-8 shows entry into the *Run-Test/Idle* controller state from the *Update-DR* controller state, clock pulses also would be applied to the on-chip system logic if the *Run-Test/Idle* controller state were entered from the *Update-IR* controller state.

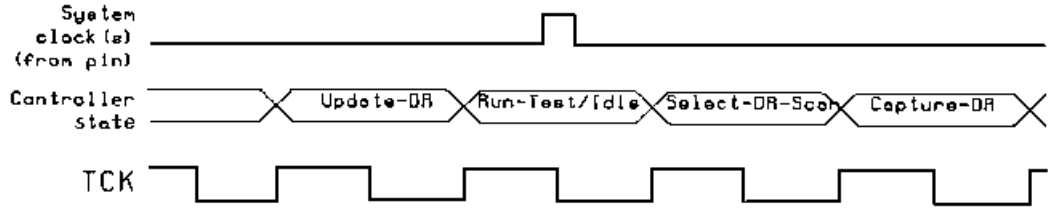


Figure 8-8—Control of applied system clock during *INTEST*

b) The on-chip system logic can be supplied with clock signals derived from TCK in the *Run-Test/Idle* controller state. In all other controller states, the clocks should not change state. Figure 8-9 shows a derived clock signal where the on-chip system logic responds to rising clock edges, for example.

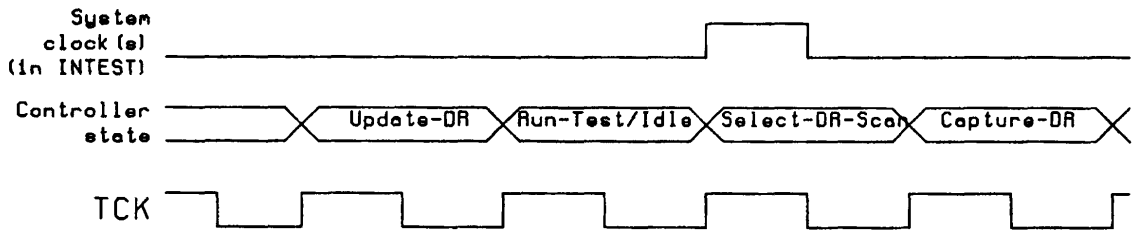


Figure 8-9—Use of TCK as clock for on-chip system logic during *INTEST*

c) Circuitry may be built into the component that, on entry into the *Run-Test/Idle* controller state, allows the on-chip system logic to complete one step of operation. For example, if the component were a microprocessor, it would be permitted to complete a single processing cycle, for example, by internal generation of a pulse on the hold signal. In this case, the clock(s) applied at the system clock pin(s) during the test could be free-running.
d) Clock signals can be shifted in via the boundary-scan path in the same manner in which nonclock signals for the on-chip system logic are supplied. Note that this will require the boundary-scan register to be shifted for each distinct clock signal state (e.g., twice for a single-phase clock).

NOTE—This may be a hazard-prone operation for certain circuit designs.

While the *INTEST* instruction is selected, the state of all system output pins is determined by the test logic. There are two options. First, the pin state may be determined by the data held in the boundary-scan register, shifted onto the latched parallel outputs of the register during each pass through the scan sequence for the register. Second, every system output pin may be forced to an inactive drive state (e.g., high-impedance). This ensures that surrounding components on an assembled board are supplied known signal levels while the on-chip system logic test is in progress. Typically, a consistent set of data values would be shifted into the appropriate stages of the boundary-scan register using the *PRELOAD* instruction before the selection of the *INTEST* instruction. This data pattern is then reloaded each time a new *INTEST* test pattern is shifted into the boundary-scan register.

Recommendation 8.9.1 f), where followed, ensures that data shifted out of the component in response to the *INTEST* instruction is not altered by the presence of faults in off-chip system logic, board-level interconnections, etc. This simplifies diagnosis, since any errors in the output bit stream can be caused only by faults in the on-chip system logic or in the boundary-scan register.

8.10 The *RUNBIST* instruction

The optional *RUNBIST* instruction causes execution of a self-contained self-test of the component. Use of the instruction allows a component user to determine the health of the component without the need to load complex data patterns and without the need for single-step operation (as required for the *INTEST* instruction). While the *RUNBIST* instruction is selected, the state of all system output pins is determined by the test logic. There are two options. First, the pin state may be determined by the data held in the boundary-scan register, shifted onto the latched parallel outputs of the register during each pass through the scan sequence for the register. Second, every system output pin may be forced to an inactive drive state (e.g., high-impedance).

The following rules apply where the *RUNBIST* instruction is provided.

NOTE—After use of the *RUNBIST* instruction, the on-chip system logic may be in an indeterminate state that will persist until a system reset is applied. Therefore, the on-chip system logic may need to be reset on return to normal (i.e., nontest) operation.

8.10.1 Specifications

Rules

a) When the *RUNBIST* instruction is selected, the test data register into which the results of the self-test(s) will be loaded shall be connected for serial access between TDI and TDO in the *Shift-DR* controller state.
b) Self-test mode(s) of operation accessed through the *RUNBIST* instruction shall execute only in the *Run-Test/Idle* controller state.

c) Where a test data register is required to be initialized before execution of the self-test, this shall occur at the start of the self-test without any requirement to shift data into the component (i.e., there shall be no requirement to enter seed values into any test data register).

NOTE—As per Rule 8.10.1 k) 1), the boundary-scan register may (optionally) need to be initialized to define the state of signals driven from system output pins. However, this value should not be used as a seed for the self-test operation, since it may be board-dependent.

d) A duration shall be specified for the test executed in response to the *RUNBIST* instruction (e.g., a number of rising edges of TCK or the system clock).
e) The result of the self-test(s) executed in response to the *RUNBIST* instruction shall be loaded into the test data register connected between TDI and TDO no later than the rising edge of TCK in the *Capture-DR* controller state.
f) After the specified minimum duration, the test result observed by loading and shifting of the test data register selected by the *RUNBIST* instruction shall be constant regardless of when the *Capture-DR* controller state is entered.
g) Use of the *RUNBIST* instruction shall give the same result in all versions of a component.
h) Data shifted out of a component after completion of execution of a self-test accessed using the *RUNBIST* instruction shall be independent of the operation of off-chip circuitry or board-level interconnections.
i) All stages of the test data register selected by the *RUNBIST* instruction shall be set to determinate logic states (0 or 1) no later than the rising edge of TCK in the *Capture-DR* controller state.
j) The design of the component shall ensure that results of self-tests executed in response to the *RUNBIST* instruction are not affected by signals received at nonclock system input pins.
k) When the *RUNBIST* instruction is selected, all system outputs from the component shall be defined as follows:
    1) All signals driven out of the component shall be defined by data held in the boundary-scan register and shall change only on the falling edge of TCK in the *Update-DR* controller state or on selection of the *RUNBIST* instruction; or
    2) All outputs from the component (including those that are 2-state nontest signals) shall be placed in an inactive drive state (e.g., high-impedance) on selection of the *RUNBIST* instruction.
l) The states of the parallel output registers or latches in boundary-scan register cells located at system output pins (2-state, 3-state, or bidirectional) shall not change while the *RUNBIST* instruction is selected unless the associated pin has been placed in an inactive drive state (e.g., high-impedance) as defined in Rule 8.10.1 k) 2).

Recommendations

m) Where possible, components compatible with this standard should support the *RUNBIST* instruction.

Permissions

n) The binary value(s) for the *RUNBIST* instruction may be selected by the component designer.
o) Where a component includes multiple self-test functions, these functions may be executed either concurrently or in a sequence determined by the component manufacturer in response to the *RUNBIST* instruction. In the latter case, all sequencing should be taken care of within the component itself without requiring the alteration of the instruction register contents.
p) Additional public instructions may be provided to give user access to individual self-test functions within a component.
q) The test data register connected between TDI and TDO when the *RUNBIST* instruction is selected may be the boundary-scan register.
r) While the *RUNBIST* instruction is selected, the boundary-scan register may act as a pattern generator or signature compactor in the *Run-Test/Idle* controller state provided that Rule 8.10.1 l) is met.

8.10.2 Description

The *RUNBIST* instruction provides the component purchaser with a means of running a user-accessible self-test function within the component as a result of a single instruction. This permits all components on a board that offer the *RUNBIST* instruction to execute their self-tests concurrently, providing a rapid health check for the assembled board. Note, however, that the component manufacturer can include further private or public instructions to give access to individual self-test functions one at a time or to self-test functions that are not invoked by the *RUNBIST* instruction.

The sequence of steps required for completion of the execution of *RUNBIST* can be defined as

a) (Optional) initialization of the boundary-scan register (for example, via *PRELOAD*). This is required if the pin state during BIST is to be determined by the data in the latched parallel outputs of the register.
b) Initiate BIST: scan the *RUNBIST* instruction into the instruction register.
c) Execute BIST: cause the TAP controller to remain in its *Run-Test/Idle* controller state for the duration required for completion of the execution of BIST.
d) Evaluate BIST results: bring the TAP controller to the *Shift-DR* controller state and scan out the test results (e.g., a signature) from the register connected to TDI and TDO by the *RUNBIST* instruction.

While the test is proceeding, the test logic defines the outputs from the component. As for the *INTEST* instruction, two options are available:

— The pin state may be determined by the data held in the boundary-scan register.
— Every system output pin may be forced to an inactive drive state (e.g., high-impedance).

Where the former option is selected, the data values driven through the system output pins are fixed at the time the *RUNBIST* instruction is selected, based on data held in the boundary-scan register at that time. (This data may have been preloaded using the *PRELOAD* instruction.) The boundary-scan register is controlled such that the data held in the latched parallel outputs of cells that feed system output pins does not change while the *RUNBIST* instruction is selected. Referring to Figure 8-2, this might, for example, be achieved by holding the UpdateDR signal at 0 while the *RUNBIST* instruction is selected. The Mode signal would be held at 1.

Boundary-scan register cells also may be used to hold programmed signal values at inputs to the on-chip system logic while the self-test is executing (again, as shown in Figure 8-7). Alternatively, boundary-scan register cells located at nonclock system logic inputs can be designed to act as a source of self-test data for the on-chip system logic. Similarly, boundary-scan register cells located at system logic outputs can act as compactors for the results of the self-test.

The specification of boundary-scan register cells for system clock input pins allows the clocks for the on-chip system logic to be obtained in one of two ways while the *RUNBIST* instruction is selected:

a) The signals received at system clock pins can be fed directly to the on-chip system logic as during normal operation of the component. Where this is done, the design of the component shall ensure that the self-test executes *only* in the *Run-Test/Idle* controller state. The clock may, however, be active in other controller states.
b) The on-chip system logic can be supplied with clock signals derived from TCK in the *Run-Test/Idle* controller state. In all other controller states, the clocks should not change state.

The rules relating to the duration of a self-test executed in response to the *RUNBIST* instruction [Rules 8.10.1 d) and 8.10.1 f)] ensure that sufficient clock edges can be applied to allow completion of self-tests executed concurrently in different components on an assembled board. Thus, in a product containing components with self-test lengths of 1000, 5000, 10 000, and 50 000 rising clock edges on TCK, the

complete board shall be left in the *Run-Test/Idle* controller state for at least 50 000 rising clock edges to ensure that all tests complete satisfactorily. Tests that complete before 50 000 clock edges have been applied will hold their results until they are accessed.

Rule 8.10.1 g) is included to ensure that the test for an assembled board is independent of the versions of the components mounted on it. This is an important consideration when working in a maintenance or repair environment, where the versions of the components used on a board may not be known. The rule can be met by forming the exclusive-OR of the result from execution of the *RUNBIST* instruction with a fixed (version-dependent) pattern. The output from this function would become the result loaded into the boundary-scan register or the other test data register connected between TDI and TDO.

Rule 8.10.1 h) ensures that data shifted out of the component in response to the *RUNBIST* instruction is not altered by the presence of faults in off-chip system logic, board-level interconnections, etc. This simplifies diagnosis, since any errors in the output bit stream can be caused only by faults in the on-chip system logic or in the test data register connected in the path between TDI and TDO.

8.11 The *CLAMP* instruction

The optional *CLAMP* instruction allows the state of the signals driven from component pins to be determined from the boundary-scan register while the bypass register is selected as the serial path between TDI and TDO. The signals driven from the component pins will not change while the *CLAMP* instruction is selected.

The following rules apply where the *CLAMP* instruction is provided.

NOTE—After use of the *CLAMP* instruction, the on-chip system logic may be in an indeterminate state that will persist until a system reset is applied. Therefore, the on-chip system logic may need to be reset on return to normal (i.e., nontest) operation.

8.11.1 Specifications

Rules

a) The CLAMP instruction shall select the bypass register to be connected for serial access between TDI and TDO in the Shift-DR controller state.

NOTE—The bypass register will behave fully as defined in Clause 10 while the *CLAMP* instruction is selected. Therefore, it will load a logic 0 during the *Capture-DR* controller state and shift data during the *Shift-DR* controller state.

b) When the *CLAMP* instruction is selected, the state of all signals driven from system output pins shall be completely defined by the data held in the boundary-scan register. (For example, this data may be shifted into the boundary-scan register by previous use of the *PRELOAD* instruction.)
c) The states of the parallel output registers or latches in boundary-scan register cells located at system output pins (2-state, 3-state, or bidirectional) shall not change while the *CLAMP* instruction is selected.
d) When the *CLAMP* instruction is selected, the on-chip system logic shall be controlled such that it cannot be damaged as a result of signals received at the system input or system clock input pins.

NOTE—This might be achieved by placing the on-chip system logic in a reset or “hold” state while the *CLAMP* instruction is selected.

Permissions

e) The binary value(s) for the *CLAMP* instruction may be selected by the component designer.

8.11.2 Description

During testing of a particular IC or a cluster of ICs on a loaded printed circuit board, it may be necessary to place static “guarding” values on signals that control operation of logic not involved in the test, for example, to place it in a state where it cannot respond to signals received from the logic under test. Such instances will undoubtedly occur during the change from in-circuit testing to testing that is based extensively on boundary scan. In such cases, the “guarding” signal values would be maintained during application of the test.

The *EXTEST* instruction could be used for this purpose. This instruction would be loaded serially into the ICs that drive the signals on which “guarding” values are required. The required signal values would be loaded as a part of the complete serial data stream shifted into the board-level path both at the start of the test and each time a new test pattern is entered. A limitation of this approach is that the length of the data pattern to be shifted for each test is increased by inclusion of the boundary-scan registers in the ICs involved in the “guarding” process. As a result, the test application rate is reduced.

The optional *CLAMP* instruction allows “guarding” values to be applied using the boundary-scan registers of the appropriate ICs but does not retain these registers in the serial path during test application. In a case in which the *CLAMP* instruction was used to create “guarding,” the following process would be used:

NOTE—It is presumed in the following description that every component implements the optional *CLAMP* instruction.

a) Before the test, the *PRELOAD* instruction would be loaded into all ICs that will provide “guarding” signals during the upcoming test. Call this group of ICs *G.* If test set-up data is required in ICs not in *G* (i.e., in those ICs that will participate actively in the upcoming test), the *PRELOAD* instruction also may be loaded into these ICs at this time.
b) Shift the “guarding” pattern into all relevant boundary-scan register cells of the ICs in *G.* Any test set-up data required for the ICs to be tested also is loaded.
c) From this point on, until the test is concluded, every time instructions are to be scanned into devices on the board, enter the *CLAMP* instruction into the ICs in *G.* As long as the *CLAMP* instruction is maintained as the active instruction in the ICs of *G*, the output signal values of these ICs will be determined by the “guarding” data in their boundary-scan registers. Also, as a consequence of the use of the *CLAMP* instruction, the ICs in *G* all have their bypass registers selected throughout the test; thus, they contribute very little to the overall test time.

8.12 Device identification register instructions

Use of the optional device identification register allows a code to be serially read from the component that shows

a) The manufacturer’s identity;
b) The part number; and
c) The version number for the part.

Two instructions are defined by this standard that use the device identification register: *IDCODE* and *USERCODE*. These instructions are defined in 8.13 and 8.14. Use of the *IDCODE* instruction will provide information on the base component, while use of the *USERCODE* instruction will provide information on the particular programming of an off-board programmable component (e.g., a fuse-programmable logic device).

8.13 The *IDCODE* instruction

8.13.1 Specifications

Rules

a) Where a device identification register is included in the design, the component shall provide an *IDCODE* instruction.
b) The *IDCODE* instruction shall select *only* the device identification register to be connected for serial access between TDI and TDO in the *Shift-DR* controller state (i.e., no other test data register may be connected in series with the device identification register).
c) When the *IDCODE* instruction is selected, the vendor identification code shall be loaded into the device identification register on the rising edge of TCK after entry into the *Capture-DR* controller state.
d) When the *IDCODE* instruction is selected, all test data registers that can operate in either system or test modes shall perform their system function.
e) When the *IDCODE* instruction is selected, the operation of the test logic shall have no effect on the operation of the on-chip system logic.

Permissions

f) The binary value(s) for the *IDCODE* instruction may be selected by the component designer.

8.13.2 Description

Where a device identification register is included in a component design, the *IDCODE* instruction is forced into the instruction register's parallel output latches during the *Test-Logic-Reset* controller state. This allows the device identification register to be selected by manipulation of the broadcast TMS and TCK signals, as well as by a conventional instruction register scan operation.

The importance of this means of selecting access to the device identification register is that it permits blind interrogation of the components assembled onto a printed circuit board, etc. Thus, in circumstances where the component population may vary (e.g., due to different programming of programmable parts), it is possible to determine which components exist in a product.

8.14 The *USERCODE* instruction

8.14.1 Specifications

Rules

a) Where a device identification register is included in the design and the component is user-programmable such that the programming cannot otherwise be determined by use of public instructions (see 8.2), the component shall provide a *USERCODE* instruction.
b) The *USERCODE* instruction shall select the device identification register to be connected for serial access between TDI and TDO in the *Shift-DR* controller state (i.e., no other test data register may be connected in series with the device identification register).
c) When the *USERCODE* instruction is selected, the 32-bit user-programmable identification code shall be loaded into the device identification register on the rising edge of TCK after entry into the *Capture-DR* controller state.
d) When the *USERCODE* instruction is selected, all test data registers that can operate in either system or test modes shall perform their system function.

e) When the *USERCODE* instruction is selected, the operation of the test logic shall have no effect on the operation of the on-chip system logic.

Permissions

f) The binary value(s) for the *USERCODE* instruction may be selected by the component designer.

8.14.2 Description

The *USERCODE* instruction allows a user-programmable identification code to be loaded and shifted out for examination. This instruction is required only for programmable components in which the programming cannot be determined through use of the test logic. The instruction allows the programmed function of the component to be determined.

8.15 The *HIGHZ* instruction

Use of the optional *HIGHZ* instruction places the component in a state in which *all* of its system logic outputs are placed in an inactive drive state (e.g., high impedance). In this state, an in-circuit test system may drive signals onto the connections normally driven by a component output without incurring the risk of damage to the component.

The following rules apply where the *HIGHZ* instruction is provided.

NOTE—After use of the *HIGHZ* instruction, the on-chip system logic may be in an indeterminate state that will persist until a system reset is applied. Therefore, the on-chip system logic may need to be reset on return to normal (i.e., nontest) operation.

8.15.1 Specifications

Rules

a) The *HIGHZ* instruction shall select the bypass register to be connected for serial access between TDI and TDO in the *Shift-DR* controller state.

NOTE—The bypass register will behave fully as defined in Clause 10 while the *HIGHZ* instruction is selected. Therefore, it will load a logic 0 during the *Capture-DR* controller state and shift data during the *Shift-DR* controller state.

b) When the *HIGHZ* instruction is selected, all system logic outputs (including 2-state and 3-state outputs and bidirectional pins) of the component shall immediately be placed in an inactive-drive state.

NOTES

1—If present, pull-up, pull-down, and keeper circuits are not required to be disabled.

2—According to the rules in 11.2.1, there shall be no consequential change in the states of the parallel output registers or latches in boundary-scan register cells. For example, on leaving the *HIGHZ* instruction and selecting the *EXTEST* instruction, the data held in the boundary-scan register before selection of the *HIGHZ* instruction should be applied to the system output pins.

c) When the *HIGHZ* instruction is selected, the on-chip system logic shall be controlled such that it cannot be damaged as a result of signals received at the system input or system clock input pins.

NOTE—This might be achieved by placing the on-chip system logic in a reset or “hold” state while the *HIGHZ* instruction is selected.

Permissions

d) The binary value(s) for the *HIGHZ* instruction may be selected by the component designer.

8.15.2 Description

On boards where not all the components are compatible with this standard, a need will continue to exist to use in-circuit test techniques in which test signals from an ATE system are driven into internal connections of the assembled board. To allow this to be done without risk of damage to the components that normally would control these connections, components should be designed such that their system logic output pins can be placed in an inactive-drive state while in-circuit testing proceeds. On a component compatible with this standard, provision of the *HIGHZ* instruction allows such a state to be entered by use of the TAP. (On components that do not comply with this standard, this typically would be achieved by using a dedicated test-control pin.)

A further use of the *HIGHZ* instruction is to allow a source of test data to be connected to one or more signals internal to a loaded board in place of the normal driver(s). An example application is shown in Figure 8-10.

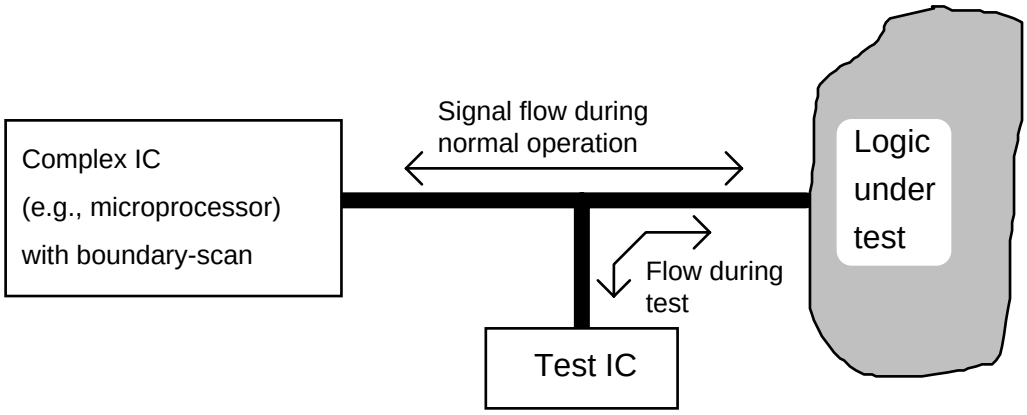


Figure 8-10—Use of the *HIGHZ* instruction

During normal operation, the outputs of the test chip would be in an inactive drive state (e.g., high-impedance), while the outputs of the processor would be active. During testing, the *HIGHZ* instruction is entered into the processor chip with the result that its outputs enter the inactive drive state. The test chip then can be enabled to drive the connections into the logic under test (which might, for example, be an array of memory chips).

Note that where the system requirement is for a 2-state output pin and both logic states are actively driven, a 3-state buffer will have to be provided purely to allow entry into the inactive state when the *HIGHZ* instruction is selected. The enable input to this buffer will be supplied directly from the instruction decoder, as illustrated in Figure 8-11. (In Figure 8-11, the signal from the instruction decoder would be logic 1 other than when the *HIGHZ* instruction is selected.) No boundary-scan register cell is required in this signal path.

For a 2-state pin where only one state is driven actively, the output should be forced into the inactively driven state when the *HIGHZ* instruction is selected. For example, the output pull-down transistor for an open-collector output would be forced off.

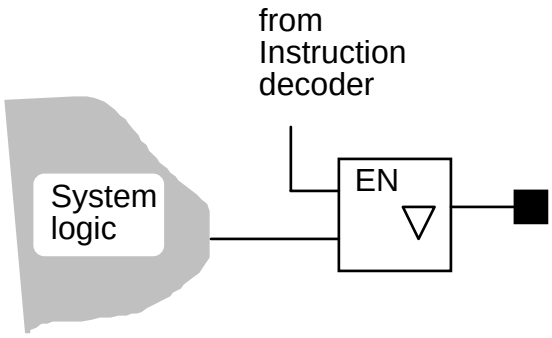


Figure 8-11—Provision of *HIGHZ* at a 2-state pin

9. Test data registers

The test logic architecture contains a minimum of two test data registers—the bypass and boundary-scan registers. In addition, the design of a third, optional, test data register is defined—the device identification register.

The architecture is extensible beyond the minimum requirements specified in this standard to allow access to any test-support features embedded in the design. These features might include scan-test, self-test registers, or access to key registers in the design (for example, via scannable shadow registers). Additional test data registers need not be intended for public access and use.

Each named test data register has a fixed length and can be accessed by using one or more instructions. The registers can, where appropriate, share circuitry and can be concatenated to form further registers, provided that each distinct combination is given a new name (thus allowing it to meet the fixed length requirement).

This clause defines the common design requirements for all test data registers incorporated in the test logic architecture defined by this standard. Specific design requirements for the bypass, boundary-scan, and device identification registers are contained in Clauses 10, 11, and 12, respectively.

9.1 Provision of test data registers

9.1.1 Specifications

Rules

a) The group of test data registers shall include, as a minimum, a bypass register and a boundary-scan register designed according to the requirements contained in this clause and in Clause 10 and Clause 11, respectively.
b) Where a device identification register is included in the group of scannable test data registers, it shall be designed according to the requirements contained in this clause and in Clause 12.
c) All test data registers shall be designed according to the requirements contained in this clause.

Permissions

d) Design-specific test data registers may be provided within the group of test data registers to give access to design-specific testability features.
e) Design-specific test data registers may (but need not) be publicly accessible.

9.1.2 Description

Figure 9-1 shows the bypass, boundary-scan, and optional test data registers realized as a set of shift-register based elements connected in parallel between a common serial input and a common serial output. Selection of the register that forms the serial path at a given time is controlled from the instruction register. In Figure 9-1, this is shown to be achieved using a multiplexer; however, other implementations are possible.

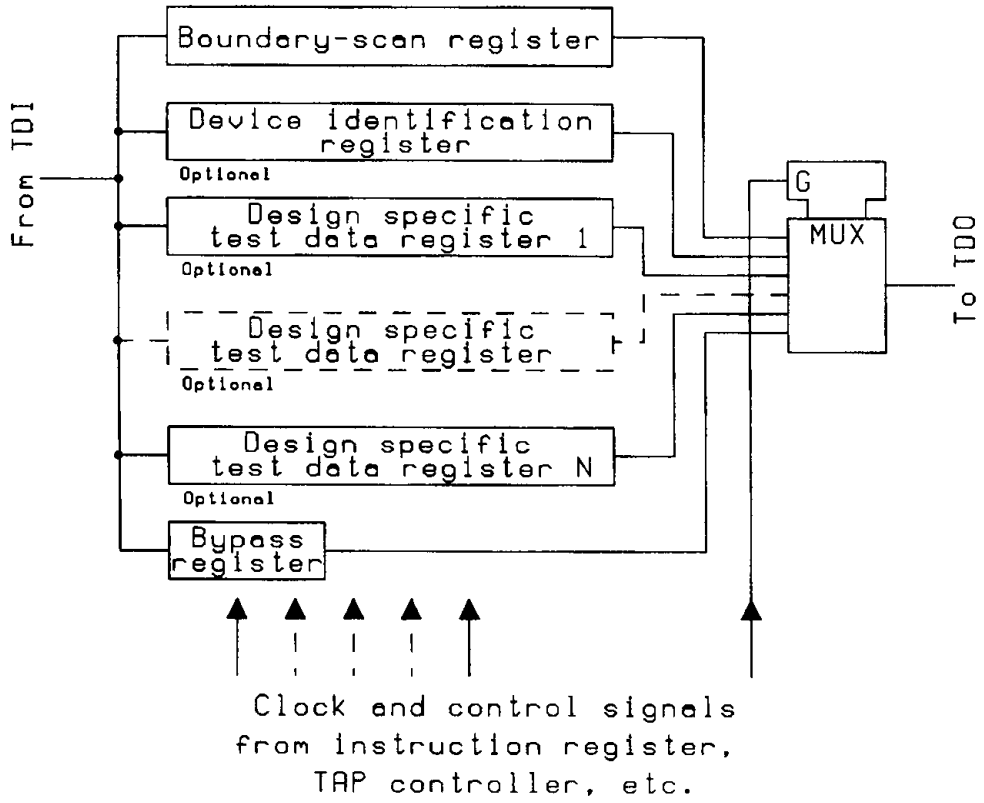


Figure 9-1—An implementation of the group of test data registers

The registers shown in Figure 9-1 are briefly described as follows:

The bypass register

This provides a single-bit serial connection through the circuit when none of the other test data registers is selected. This register can, for example, be used to allow test data to flow through a particular device to other components in a product without affecting the normal operation of the particular component. The specification for the bypass register is contained in Clause 10.

The boundary-scan register

This allows testing of board interconnections, detecting typical production defects such as opens, shorts, etc. It also allows access to the inputs and outputs of components when testing their system logic or sampling of signals flowing through the system inputs and outputs. The specification for the boundary-scan register is contained in Clause 11.

The device identification register

This is an optional test data register that allows the manufacturer, part number, and variant of a component to be determined. If this register is included, it should conform to the specification contained in Clause 12.

The design-specific test data registers

These optional registers may be provided to allow access to design-specific test support features in the integrated circuit, such as self-tests, scan paths, etc. They need not be intended for public use and access but may be made so if the component designer wishes.

9.2 Design and construction of test data registers

9.2.1 Specifications

Rules

a) Each test data register shall be given a unique name.
b) The design of each test data register shall be such that when data is shifted through it, data applied to TDI appears without inversion at TDO after an appropriate number of TCK transitions when the TAP controller is in the *Shift-DR* state.
c) The length of each test data register shall be fixed, independent of the instruction by which it is accessed.
d) For programmable components, the length of each test data register shall be independent of the way the component is programmed.

Permissions

e) A test data register may be constructed from segments or circuitry also used in one or more other registers provided that the resulting design complies fully with the rules in this standard.

NOTE—The resulting combination shall be given a name distinct from those of the registers from which it is constructed.

f) Circuitry (including the shift-register paths) in the various test data registers included in a design may be shared between test data registers provided that the rules contained in this standard are met.
g) Unless specifically prohibited by this standard, circuitry contained in test data registers may be used to perform system functions when test operation is not required.

9.2.2 Description

While the example implementations contained in this standard show the various test data registers to be separate physical entities, circuitry may be shared between the test data registers provided that the rules contained in this standard are met. For example, this would allow the device identification register and the boundary-scan register to share shift-register stages, in which case the requirements of this standard would be met by operating the common circuitry in two different modes—the device identification register mode and the boundary-scan register mode. Except where identified specifically, the test data registers also may perform system functions, and thus be a part of the on-chip system logic, when they are not required to perform test functions.

Rule 9.2.1 c) requires that the length of any test data register be fixed; i.e., a test data register named FRED shall always contain, say, 20 stages no matter how or when it is accessed. Note that virtual registers are allowed (i.e., a named test data register may be built from circuitry shared with other test data registers or

with the system logic). Therefore, requirements may exist for different segments of a single physical register to be accessed for different tests. In these cases, Rule 9.2.1 c) can be met by assigning a unique name to each distinguishable register configuration (see Figure 9-2 and Table 9-1). Rule 9.2.1 d) requires that the length of a test data register also be fixed independently of how a component is programmed. These restrictions are necessary to avoid unnecessary complication of the software used to generate and apply tests.

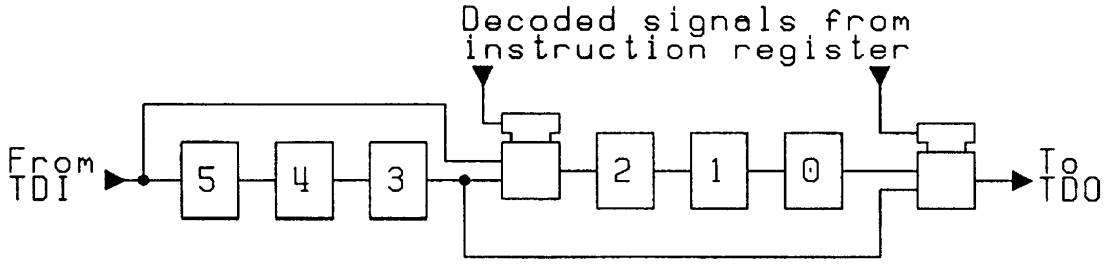


Figure 9-2—Construction of test data registers from shared circuitry

Table 9-1—Naming of test data registers that share circuitry

Test data register name	Stages that form the register
WHOLE_REG	5, 4, 3, 2, 1, 0
FRONT_REG	2, 1, 0
BACK_REG	5, 4, 3

9.3 Test data register operation

9.3.1 Specifications

Rules

a) Each instruction shall identify a test data register that will be serially connected between TDI and TDO.
b) The test data register connected between TDI and TDO shall shift data one stage toward TDO after each rising edge of TCK in the *Shift-DR* controller state.
c) In the *Test-Logic-Reset* controller state, all test data registers shall be set so that either they perform their system function (if one exists) or they do not interfere with the operation of the on-chip system logic.
d) Where a test data register is required to load data from a parallel input in response to the current instruction, this data shall be loaded on the rising edge of TCK after entry into the *Capture-DR* controller state.
e) Test data registers enabled to drive data off-chip shall be designed such that component outputs change only
    1) On the falling edge of TCK after entry to the *Update-DR*, *Update-IR*, *Run-Test/Idle*, or *Test-Logic-Reset* controller state as a result of signals applied at TCK and TMS; or
    2) Immediately on entry into the *Test-Logic-Reset* controller state as a result of a logic 0 being applied at TRST*.

NOTE—This may require that the register be provided with a latched parallel output.

f) Where a test data register is required to operate in response to the *RUNBIST* instruction, the required operation shall occur in the *Run-Test/Idle* controller state.
g) Where no operation of a selected test data register is required in a given controller state in response to the current instruction, the register shall retain its last state unchanged.
h) Test data registers that are not selected by the current instruction shall be set so that either they perform their system function (if one exists) or they do not interfere with the operation of the on-chip system logic.

Permissions

i) In addition to the test data register enabled for shifting between TDI and TDO, an instruction may select further test data registers.

NOTE—These registers should retain their last state in the *Shift-DR* controller state but otherwise meet the rules set out above.

9.3.2 Description

These requirements ensure that test data registers operate correctly in conjunction with the TAP and TAP controller.

Note that while an instruction may select test operation of more than one test data register, there can be only one test data register between TDI and TDO. All other selected test data registers retain their state in the *Shift-DR* controller state. One use of this capability is as follows.

Consider the case where several test data registers need to be accessed in sequence in order to establish starting conditions for a test or to examine test results. As an example, Table 9-2 shows the sequence of events (starting from the *Test-Logic-Reset* controller state) that would be required if two design-specific test data registers and the boundary-scan register needed to be accessed in order to execute an instruction. Figure 9-3 shows the design of the group of test data registers for this example.

Table 9-2—Sequential access to test data registers

Step	Action
0	Test logic inactive in the *Test-Logic-Reset* controller state.
1	Enter instruction that selects register A for connection between TDI and TDO.
2	Scan required initial values into register A.
3	Enter instruction that selects register B for connection between TDI and TDO and also keeps register A in its test mode of operation.
4	Scan required initial values into register B. Register A retains its state.
5	Enter the test instruction that selects test operation of registers A and B and connects the boundary-scan register between TDI and TDO.
6	Scan required values for the component inputs and outputs into the boundary-scan register. Registers A and B retain their states.
7	Execute the instruction by entering the *Run-Test/Idle* controller state.
8	Enter instruction that selects register B for connection between TDI and TDO and also keeps register A in its test mode of operation.
9	Scan test results out of register B. Register A retains its state.
10	Enter instruction that selects register A for connection between TDI and TDO.
11	Scan test results out of register A.

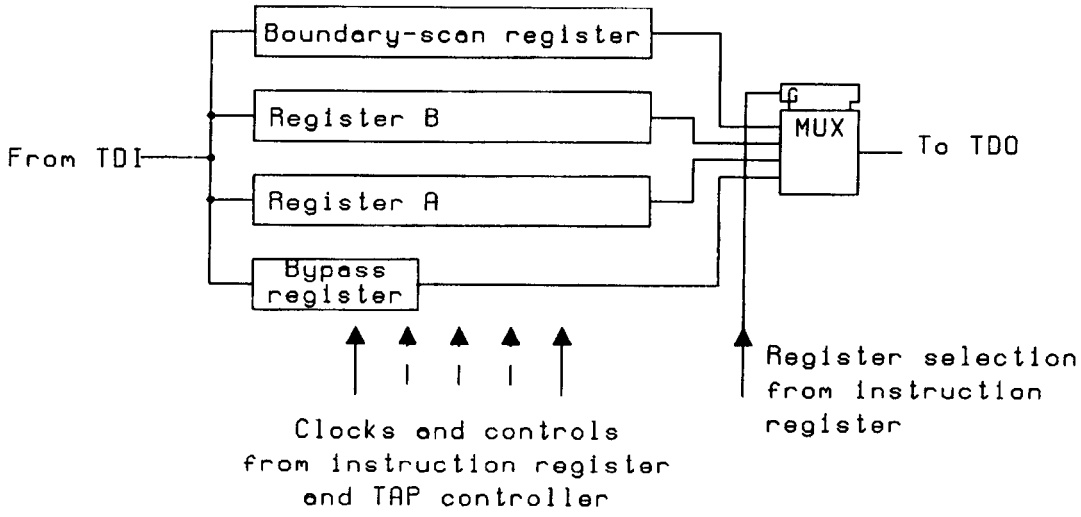


Figure 9-3—Example design containing two optional test data registers

In step 4, serial access is required to register B in order to set its initial condition as required for execution of the test. However, register A was set to its required initial condition in step 2, and so it is necessary to design the test logic such that register A can retain its state between steps 2 and 7 (when the self-test is executed) while register B and the boundary-scan register are accessed. Similarly, the test logic shall be designed such that register B retains the initial condition set during step 4 until step 7 and such that the reverse sequence of events can occur after completion of execution of the self-test.

Note that the design of the test logic may be such that test data registers shall be accessed in a fixed order in order to achieve the desired result. For example, the test logic may not allow register B to retain its state while register A is scanned.

10. The bypass register

The bypass register provides a minimum length serial path for the movement of test data between TDI and TDO. This path can be selected when no other test data register needs to be accessed during a board-level test operation. Use of the bypass register in a component speeds access to test data registers in other components on a board-level test data path.

10.1 Design and operation of the bypass register

10.1.1 Specifications

Rules

a) The bypass register shall consist of a single shift-register stage.
b) When the bypass register is selected for inclusion in the serial path between TDI and TDO by the current instruction, the shift-register stage shall be set to a logic zero on the rising edge of TCK after entry into the *Capture-DR* controller state.
c) The circuitry used to implement the shift-register stage in the bypass register shall not be used to perform any system function (i.e., it shall be a dedicated part of the test logic).

d) The operation of the bypass register shall have no effect on the operation of the on-chip system logic.

10.1.2 Description

The bypass register may be implemented as shown in Figure 10-1.

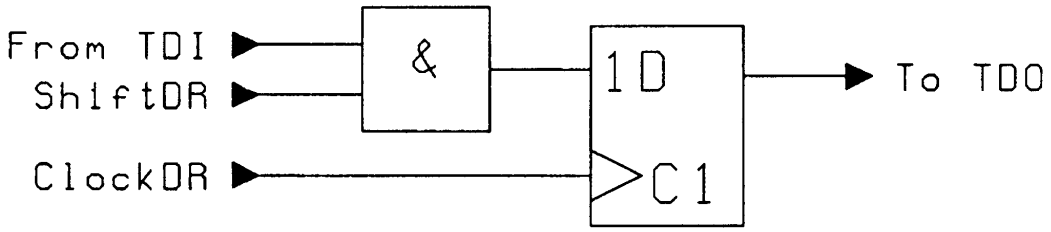


Figure 10-1—A bypass register implementation

The provision of this register allows bypassing of segments of the board-level serial test data register that are not required for a specific test. Test access times to the segments of interest are reduced.

As an example, consider a circuit board containing 100 integrated circuits, each of which has 100 bits in its boundary-scan register. The boundary-scan path on the assembled board would include 10 000 shift-register stages if all the segments were connected in series simultaneously. This would give protracted test times, for example, when accessing just one of the integrated circuits on the path.

The ability to bypass segments of the shift-register path under control of the appropriate instruction register allows considerable shortening of the overall path in such circumstances. Continuing the example, 99 of the components could be set to shift only through their bypass register, with the integrated circuit under test having its full boundary-scan register in circuit. This would give a total serial path length of 199 stages—a considerable reduction compared to 10 000.

Rule 10.1.1 b) is included so that the presence or absence of a device identification register in the test logic can be determined by examination of the serial output data. The bypass register (which is selected in the absence of a device identification register) loads a logic 0 at the start of a scan cycle, whereas a device identification register loads a constant logic 1 into its LSB. When the *IDCODE* instruction is loaded into the instruction register, a subsequent data register scan cycle will allow the first bit of data shifted out of each component to be examined—a logic 1 showing that a device identification register is present. This allows blind interrogation of device identification registers by setting the *IDCODE* instruction as outlined in 12.1.

A requirement of the *BYPASS* instruction is that, when it is selected, the on-chip system logic shall continue its normal operation undisturbed. Rule 10.1.1 c) is included so that this requirement can be met. Note, however, provided that Rule 10.1.1 d) is met, the shift-register stage may be a shared resource used by several of the registers defined by this standard and also by any design-specific test data register.

11. The boundary-scan register

The boundary-scan register allows testing of circuitry external to a component, for example, board interconnect or external components that do not conform to this standard. The register also permits the system signals flowing into and out of the system logic to be sampled and examined without causing

interference with the normal (nontest) operation of the on-chip system logic. Optionally, additional test functions may be supported—for example, testing of the on-chip system logic.

11.1 Introduction

This clause specifies the design of the boundary-scan register in a component and the operation of the register in response to the various instructions defined by this standard.

Among the registers required by this standard, the boundary-scan register is the most complex. Its complexity lies neither in its shifting function nor in its architectural placement in parallel with the other required test data registers, both of which conform to the rules set out in Clause 9. The complexity lies instead in the manner in which the register is connected around the on-chip system logic and in its operation in response to the instructions defined in Clause 8. Design requirements for both connectivity and functional operation vary from cell to cell and are determined both by the type of signal (input to, or output from, the on-chip system logic) and by the set of instructions to be supported.

The design specifications are presented in three groups:

a) *Register design and operation (11.2 and 11.3).* The structure of the boundary-scan register is specified. Also presented are specifications for the operation of the shift-register at the heart of the boundary-scan register and for the parallel output latches or flip-flops that are required for some shift-register stages.

b) *Cell provision and operation (11.4 through 11.8).* Rules are presented that specify where boundary-scan register cells must be provided and how they are to be connected between the system pins of a component and the system inputs and outputs of the on-chip system logic. Further rules are presented that define how signals are to be routed through boundary-scan register cells in response to selection of the various instructions defined by this standard.

c) *Cell merging (11.9).* Finally, permissible ways in which boundary-scan register cells required by the cell provisioning rules can be merged are specified. Application of these rules will reduce the cost of implementation of a boundary-scan register in certain special cases.

To simplify the presentation of the rules, this clause uses the terminology and approach described in the following subclauses.

11.1.1 Approach

To simplify the presentation of the rules in this clause, the boundary-scan register will be described as though it were being added to an already finished design—one that does not conform to this standard. Such a design may be thought of as shown in Figure 11-1.

Note the following features in Figure 11-1:

a) *The system logic.* This is the circuitry that realizes the nontest, digital function of the component.

b) *The system pins.* The term “system pin” is used throughout this standard to refer to any system (i.e., nontest) terminal to which an external connection may be made. For packaged components, external connections are made to package pins, typically by means of a soldering process. However, in some cases an integrated circuit chip will be assembled directly onto a substrate without prior packaging. In such cases, the term “system pin” should be interpreted as the point to which the external connection is made, i.e., the bonding pad.

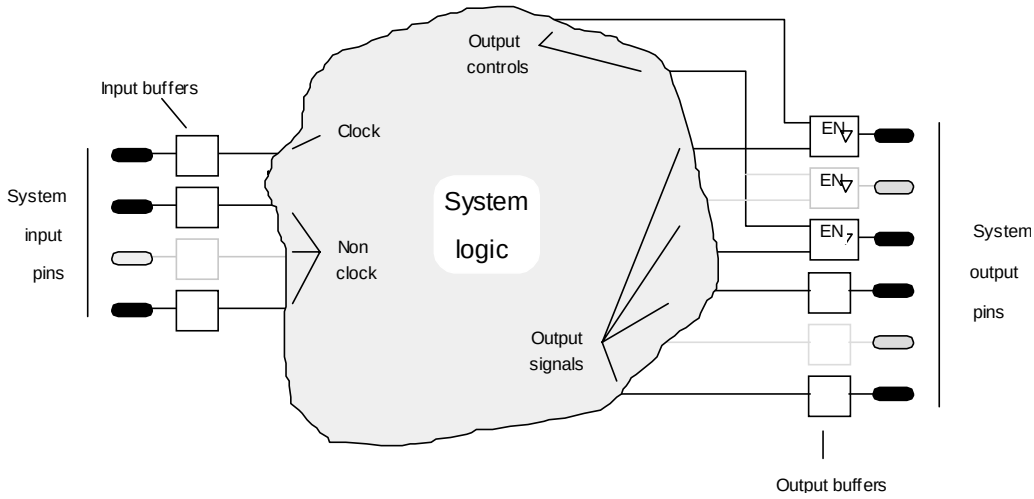


Figure 11-1—A component without boundary scan

c) *The input/output buffers*. The buffers are connected between the system pins and the system logic. Note that output buffers may have control as well as data inputs from the system logic. The signals received at the control inputs determine the manner in which the output buffer operates. For example, Figure 11-1 shows several 3-state output buffers at which the enable input (EN) is used to determine whether the output is driven. Other types of output buffer at which control signals may be used to determine different characteristics of the signal driven off-chip are possible.

d) *The inputs and outputs of the system logic*.

Two types of input to the on-chip system logic should be distinguished, as follows:

1) *Clock inputs*. Transitions at these inputs, from the low to the high logic level (or vice versa), are used to indicate when a stored-state device, such as a flip-flop or latch, may perform an operation. In an edge-triggered design, the edges (logic level transitions) received at clock inputs are used to trigger operation of all or part of the on-chip system logic, and steady-state logic values received at these signals have no significance. In a level-sensitive design, clock inputs are used to enable storage devices in the on-chip system logic to load data values. Note that the values loaded into stored-state devices are not determined by the values of clock inputs.

2) *Nonclock inputs*. This group includes all other inputs to the on-chip system logic. Typically, signals applied at these inputs are used to supply data or select an operation to be performed.

Outputs from the on-chip system logic drive output buffers (or, as will be discussed later, inputs to mixed analog/digital circuit blocks external to the system logic). It is necessary to distinguish between two types of signal that may be driven to an output buffer, as follows:

3) *Output control signals*. In a component without boundary scan, such as that shown in Figure 11-1, these signals would directly drive “enable” inputs of output buffers and hence determine whether they actively affect the state of the respective connected system pins.

4) *Data signals*. In a component without boundary scan, these signals would drive data inputs to output buffers (or, as will be discussed later, inputs to mixed analog/digital circuit blocks external to the system logic).

NOTE—A single output from the on-chip system logic may drive an output control signal to one output buffer and a data signal to another.

11.1.2 Signal paths to the on-chip system logic

Each signal path into the on-chip system logic is considered to be a fanout tree with one or more branches. Signals enter the fanout tree at the trunk (e.g., from an input buffer) and leave through the branches (e.g., at the inputs of the on-chip system logic). For example, Figure 11-2 shows signal paths between one system input pin and several inputs to the on-chip system logic. (In many cases, the fanout tree will be regarded as being contained within the system logic. In these cases, only a single point-to-point connection is considered to be present between the system input pin and the system logic, that is, a connection with only one fanout branch.)

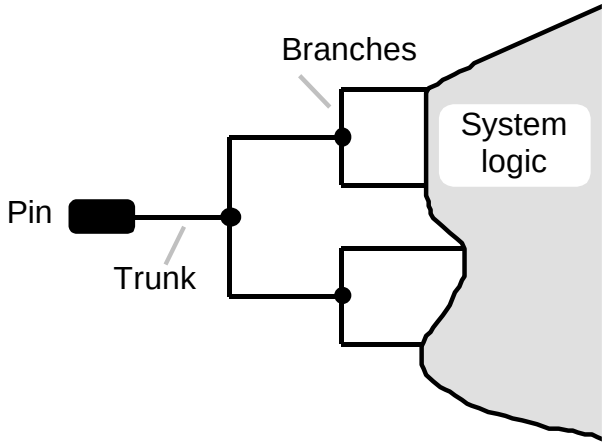


Figure 11-2—Input connection

11.1.3 Boundary-scan register cell

NOTE—The rules contained in this clause describe the boundary-scan register as a logical device, not a particular physical implementation. Further, for clarity of presentation, the example boundary-scan register cell designs presented in this clause show the circuitry to be separate from that used to construct the various other features defined in this standard. However, be aware that the rules of this standard permit parts of the circuitry used to construct boundary-scan register cells—notably the shift-register stages—to be used in the implementation of other features defined by this standard, such as the bypass and device identification registers. Where circuitry is shared between the boundary-scan register and other features defined by this standard, boundary-scan register cells may appear that are more complex than those described here.

Every boundary-scan register cell is considered to have a number of data terminals (at least two) and a number of clock and control inputs appropriate to the style of implementation. Contained within each cell is a single shift-register stage that often is provided with a parallel input and a parallel output (which may be latched). This shift-register stage uses two of the data connections of the cell as a serial input and a serial output. By way of these connections, the cell is linked to the cells before and after it in the boundary-scan register.

Cells that have three data terminals allow signals entering or leaving the on-chip system logic to be observed but not controlled. For such a cell, the third data terminal functions as a parallel input to a parallel-in, serial-out shift-register stage. When the boundary-scan register is selected as the serial path between TDI and TDO by an instruction, the data present at this terminal is loaded into the shift-register stage on the rising edge of TCK in the *Capture-DR* controller state (i.e., as required by the rules of Clause 9). Cells of this type may be

called “observe-only” cells. They will be connected to a signal path entering or leaving the on-chip system logic, as shown in Figure 11-3.

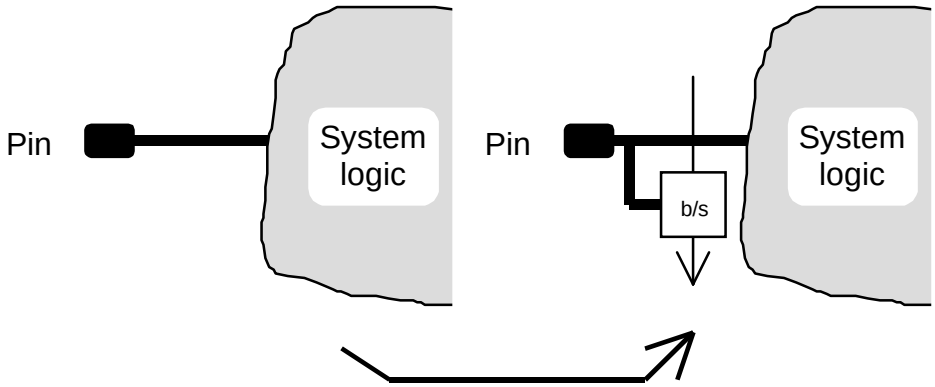


Figure 11-3—Connection of an observe-only boundary-scan register cell

Cells with four or more data terminals are inserted into signal paths entering or leaving the on-chip system logic, as shown for the case of an input in Figure 11-4. Such cells may be called “control-and-observe” cells. The shift-register stage in a control-and-observe cell can load the value of the signal path into which they are inserted and hence allow observation of that signal. Also, when required, the value held in the shift-register stage can be driven to the wire in place of the normal (nontest) source. In some cases, a constant signal value also may be driven to the wire in place of the normal (nontest) source.

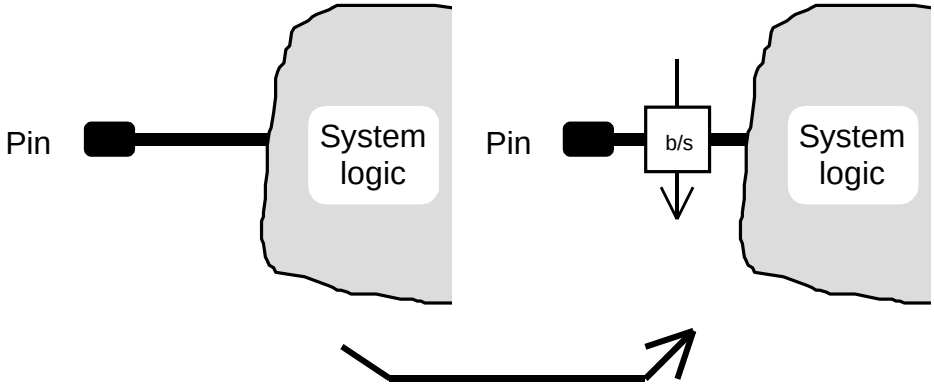


Figure 11-4—Insertion of a control-and-observe boundary-scan register cell

Cells that have only two data terminals are redundant in the sense that they could have been omitted from the design without jeopardizing compliance of the component to this standard. Such cells may be simple shift-register stages. Typically, they will exist in a component design as a result of programming or customization of boundary-scan register cells (see 11.8).

The rules presented later in this clause define the manner of cell insertion or addition required at each type of connection into or out of the system logic.

A conceptual model of a control-and-observe boundary-scan register cell is shown in Figure 11-5. Such cells contain a parallel-in, parallel-out shift-register stage. Signals flow into the cell through one terminal of the

cell (termed the parallel input) and out through another (termed the parallel output). Logic (typically, one or more multiplexers) within a control-and-observe cell determines routing of the signal on the parallel input of the cell to the parallel output of the cell. The signal may be routed through the "parallel" terminals of the parallel-in, parallel-out shift-register stage, or, in normal (nontest) operation of the component, the routing logic is set such that the signal is driven directly from the parallel input of the cell to the parallel output without any change in signal value.

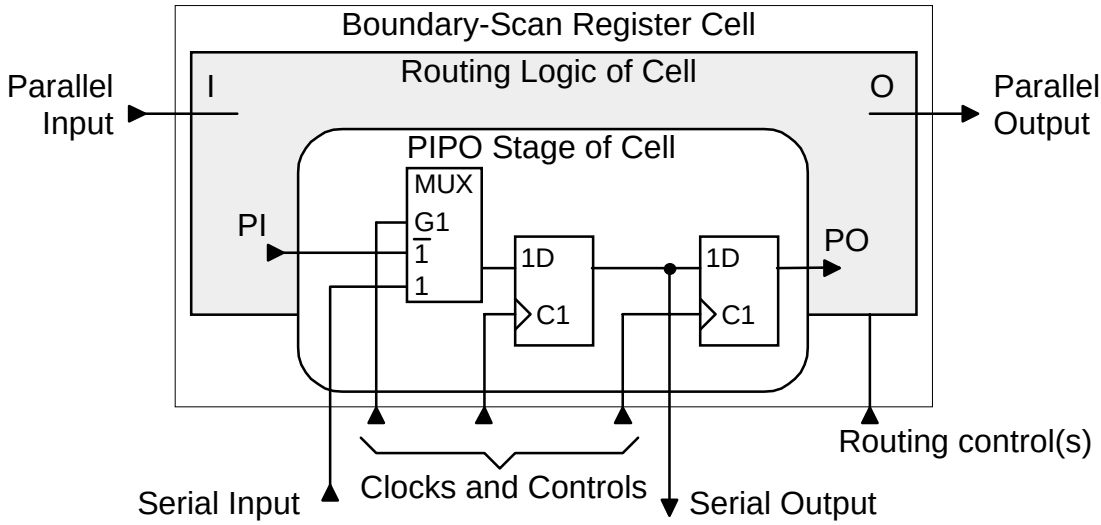


Figure 11-5—Conceptual view of a control-and-observe boundary-scan register cell

11.2 Register design

11.2.1 Specifications

Rules

a) The boundary-scan register shall consist only of boundary-scan register cells as defined in this clause.
b) Sufficient boundary-scan register cells shall be provided to meet the requirements for each system connection into or out of the on-chip system logic fully, as defined later in this clause.
c) Each boundary-scan register cell shall contain a single shift-register stage and shall have a serial input terminal and a serial output terminal by means of which the cell is linked to the cells before and after it in the boundary-scan register or, in the case of the cells at each end of the register, to the remainder of the test logic defined by this standard.
d) A boundary-scan register cell shall have two or more data terminals [including the serial terminals required by Rule 11.2.1 c)].

NOTE—Each cell will also have several clock and control inputs, the number of which will be determined by the style of implementation.

e) Boundary-scan register cells that have three data terminals shall be designed such that one data terminal is connected by routing logic to a parallel input to the shift-register stage in the cell.

NOTE—Such cells are named "observe-only" cells.

f) For boundary-scan register cells that have four or more data terminals, data terminals other than those used for serial input and output shall be parallel inputs and parallel outputs of the cell and be connected by routing logic
   1) To each other; and
   2) To the parallel input and parallel output of the shift-register stage.

NOTES

1—Such cells are named “control-and-observe” cells.

2—Often, but not always, shift-register stages will require latched parallel outputs.

g) For a given component, the ordering of cells in the boundary-scan register shall be fixed. In particular, the ordering shall not vary as a result of any operation of the on-chip system logic.
h) For a given component, the length of the boundary-scan register shall be fixed. In particular, the length shall not vary as a result of any operation of the on-chip system logic.
i) In the event that no boundary-scan register cells are required for a component, a register consisting of a single shift-register stage shall be provided.

NOTE—This situation will arise where a component contains only test logic as defined or permitted by this standard. Such a component could be described as being dedicated to testing; it will not contribute to the system function of an assembled board.

j) Operation of boundary-scan register cells shall not be interfered with or interrupted by the normal operation of any system function (see 11.3).

NOTE—Subject to conformance with Rule 11.2.1 k), circuitry may be shared between the boundary-scan register and another part of the test logic. For example, the shift-register stages also may be used by another test data register.

k) Where a boundary-scan register cell has a latched parallel output, if a system or another test function alters the value of the latched parallel output (i.e., the latched parallel output may be shared), the value that would have existed if the latched parallel output were dedicated shall be restored upon entry into the *EXTEST*, *INTEST*, *CLAMP*, or other instruction requiring the boundary-scan register (see Rules 11.3.1 b) and 11.3.1 c) and the description in 11.3.2).

NOTE—Subject to conformance with Rules 11.2.1 j) and 11.2.1 k), circuitry may be shared between the boundary-scan register and the normal system logic.

Permissions

l) Rule 11.2.1 i) may be met by selecting the bypass register whenever this standard requires selection of the boundary-scan register as the path between TDI and TDO.
m) Boundary-scan register cells may be connected in any order.

NOTE—See Rule 11.2.1 g).

11.2.2 Description

The boundary-scan register is a mandatory feature of this standard and shall be included in each component that claims conformance to this standard. It consists of a number of cells equal to the number of shift-register stages contained in the register. These cells are positioned around the on-chip system logic of a component, as specified later in this clause. They are connected to form a single shift-register-based path that is connected between TDI and TDO in the *Shift-DR* controller state when an appropriate instruction is selected. (With regard to the instructions defined in this standard, the boundary-scan register is defined to be the serial path between TDI and TDO in the *Shift-DR* controller state for the *SAMPLE*, *PRELOAD*, *EXTEST*, *INTEST*, and, optionally, *RUNBIST* instructions.)

Figure 11-6 illustrates the design of the shift-register portion of the boundary-scan register. Note that not all stages are provided with latched parallel outputs.

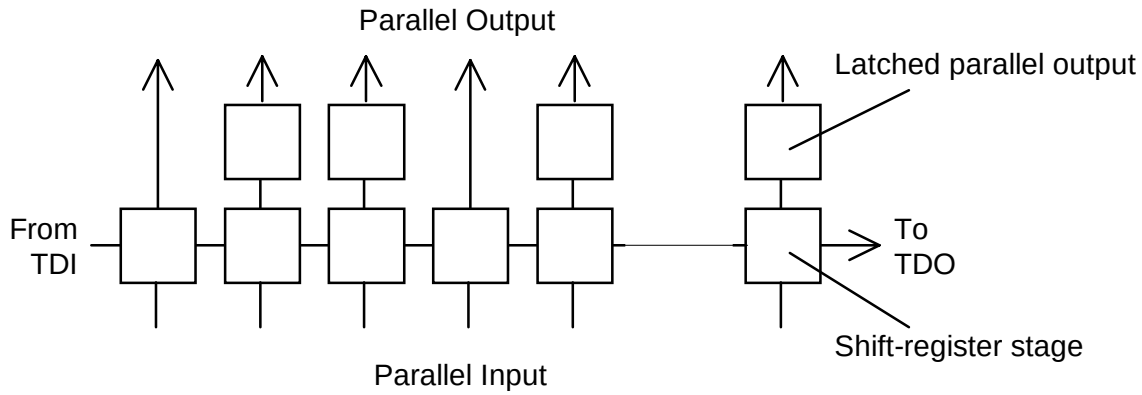


Figure 11-6—Boundary-scan shift-register design

For a given component, the minimum permissible length of the boundary-scan register is a function of the number and type of system (nontest) connections into and out of the on-chip system logic. (Typically, these will be off-chip connections.) The rules that determine the minimum set of cells that has to be provided are presented later in this clause. Note, however, that every component claiming to be compliant to this standard shall have a boundary-scan register that contains at least one shift-register stage.

11.3 Register operation

11.3.1 Specifications

Rules

a) When the boundary-scan register is selected as the serial path between TDI and TDO in the *Shift-DR* controller state, data entered at TDI shall appear without inversion at TDO after the application of a number of paired rising and falling edges at TCK equivalent to the length of the boundary-scan register.
b) When the *EXTEST*, *INTEST*, or *PRELOAD* instruction is selected, latched parallel outputs of the boundary-scan shift-register shall change state only on the falling edge of TCK in the *Update-DR* controller state, at which time each shall be set to the state of its corresponding shift-register stage.
c) When the *BYPASS*, *CLAMP*, *HIGHZ*, *IDCODE*, *RUNBIST*, or *USERCODE* instruction is selected, all latched parallel outputs shall retain their state at least until
    1) The *Test-Logic-Reset* controller state is entered as a result of the application of a logic 0 at TRST*; or
    2) The first falling edge of TCK occurs in the *Test-Logic-Reset* controller state when that state is entered as a result of signals applied at TCK and TMS.

NOTE—Permission 11.3.1 h) allows the boundary-scan register to be reset after entry into the *Test-Logic-Reset* controller state, and so in a case in which such implementation details are not known, it should be assumed that the states of the latched parallel outputs of the boundary-scan register are unknown once this state has been entered.

d) No limit shall be imposed on the number of system output pins that may change state in a single *Update-DR*, *Update-IR*, or *Test-Logic-Reset* controller state as a result of the operation of the test logic; neither shall restrictions be placed on the data patterns that may be driven (e.g., a maximum limit on the number of system logic outputs that may drive a logic 1).

NOTE—Designers should consider that in test mode, a boundary-scan test may manipulate several buses in ways that could never occur during normal system operation. This may cause transient or steady-state power consumption to exceed that expected for normal system operation. Be especially aware of this when breadboarding, socketing, or fixturing components compatible with this standard because in these environments, power distribution may be suboptimal.

e) No limit shall be placed on the combinations of logic values that may be shifted into the boundary-scan register.

NOTE—During test operations, combinations of signals may be driven either to the on-chip system logic or off-chip that will not arise during normal operation of the component. This is particularly the case where multiple boundary-scan register cells are used to drive signals that normally are driven from a single source [see Rules 11.5.1 c), 11.5.1 d), and 11.6.1 e)].

Permissions

f) When the *SAMPLE* instruction is selected, latched parallel outputs of the boundary-scan shift-register may change state only on the falling edge of TCK in the *Update-DR* controller state, at which time each shall be set to the state of its corresponding shift-register stage.
g) The delay between the falling edge of TCK and consequent changes at system output pins may be deliberately skewed between system outputs, e.g., because of a need to avoid simultaneous switching at several or all system outputs.

NOTE—In the case of the example implementations shown in this standard, this skew could be added by injecting small delays in the UpdateDR clock path and *Mode-n* control signals to each boundary-scan register cell. Such skew may be required both for boundary-scan register cells that feed data signals and for those that feed output control signals. In the latter case, the added skew will prevent excessive current demand due to simultaneous changes from “disable” to “active” or vice versa.

h) Where boundary-scan register cells are provided with shift-register stages with latched parallel outputs, these outputs may be reset to either logic state (0 or 1)
    1) When the *Test-Logic-Reset* controller state is entered as a result of application of a logic 0 at TRST*; or
    2) On the first falling edge of TCK in the *Test-Logic-Reset* controller state when that state is entered as a result of signals applied at TCK and TMS.

11.3.2 Description

To meet the requirements of the *SAMPLE* and *PRELOAD* instructions, it has to be possible to move data through the boundary-scan register without interfering with the normal system operation of the component. This may be achieved by making the shift-register stages used by the boundary-scan register a dedicated part of the test logic; that is, they do not perform any system function. Alternatively, the shift-register stages may be a shared resource used by several of the registers defined by this standard and by any design-specific test data register.

In contrast, for the public instructions defined by this standard, the latched parallel output required at some boundary-scan register stages is controlled such that it retains its last state whenever the boundary-scan register is not selected by the current instruction. This latter feature allows a set of data values to be shifted into the register and placed onto the various latched parallel outputs by use of the *PRELOAD* instruction. These values then will be retained until they are modified at the end of a subsequent shifting process (i.e., in the *Update-DR* controller state). Thus, when an instruction is entered that selects the data held in the boundary-scan register to be driven out of the component (e.g., the *EXTEST* instruction), the previously loaded data is immediately available for use. (See the discussion in 8.7.)

11.4 General rules regarding cell provision

11.4.1 Specification

Rules

a) One or more boundary-scan register cells shall be provided at each system input or output of the on-chip system logic, as detailed in 11.5 and 11.6.
b) For components that contain analog circuitry external to the on-chip system logic (i.e., analog circuitry having off-chip connections as shown in Figure 11-7), the connections between the on-chip analog circuits and the on-chip system logic shall be considered to be off-chip connections.

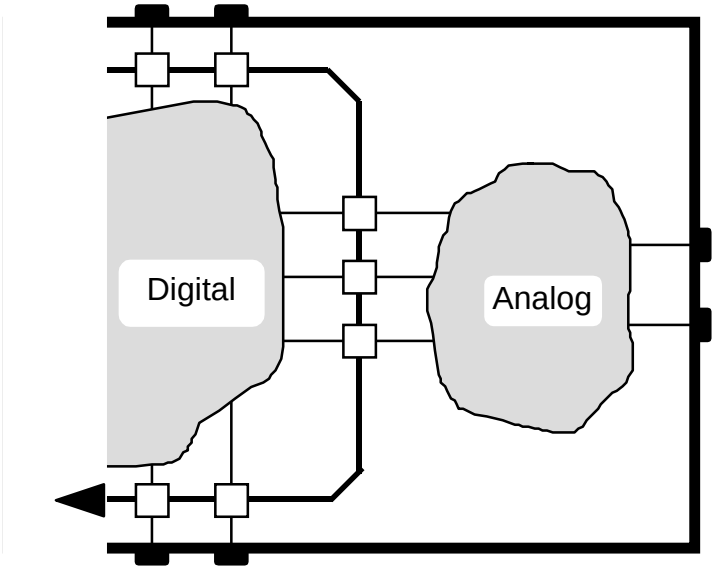


Figure 11-7—A component that contains analog circuitry

c) Boundary-scan register cells shall not be provided at
   1) TAP pins (TCK, TDI, TDO, TMS, and TRST*);
   2) Pins that enable compliance to this standard (see 4.8); or
   3) Nondigital pins (e.g., power and analog pins).

NOTE—Differential drivers and receivers that operate through detection of the direction of current flow are considered "analog" circuits. Therefore, in such cases, this rule requires a single boundary-scan register cell to be placed between each differential driver or receiver and the on-chip system logic, as illustrated in Figure 11-9. See 11.4.2 for information on the application of these rules to other types of paired inputs and outputs, e.g., differential signals that operate using conventional logic voltages.

d) The connection of boundary-scan register cells of the control-and-observe type shall be such that if each cell were replaced by a short-circuit connection between its parallel input and parallel output, the normal (nontest) logical operation of the component would not be altered.

NOTE—There may, however, be changes in performance.

e) There shall be no logic between any boundary-scan register cell and the system pin to which that cell is connected.

NOTE—"Transparent" devices such as buffers and I/O buffers are not considered to be "logic" and may exist outside the boundary-scan register. Inverters also may exist outside the boundary-scan register subject to conformance to Rules 11.5.1 i), 11.5.1 j), 11.6.1 l), 11.6.1 m), 11.6.1 n), and 11.6.1 o). Devices that perform a logic operation (such as

gates, flip-flops, and latches) are considered to be logic devices and shall not be placed outside the boundary-scan register.

Permissions

f) Where some inputs or outputs are not connected to package pins in a particular packaged configuration of a component, boundary-scan register cells may be provided for the unconnected signals.

11.4.2 Description

Figure 11-8 illustrates the placement of boundary-scan register cells for basic input, output, and bidirectional pin types.

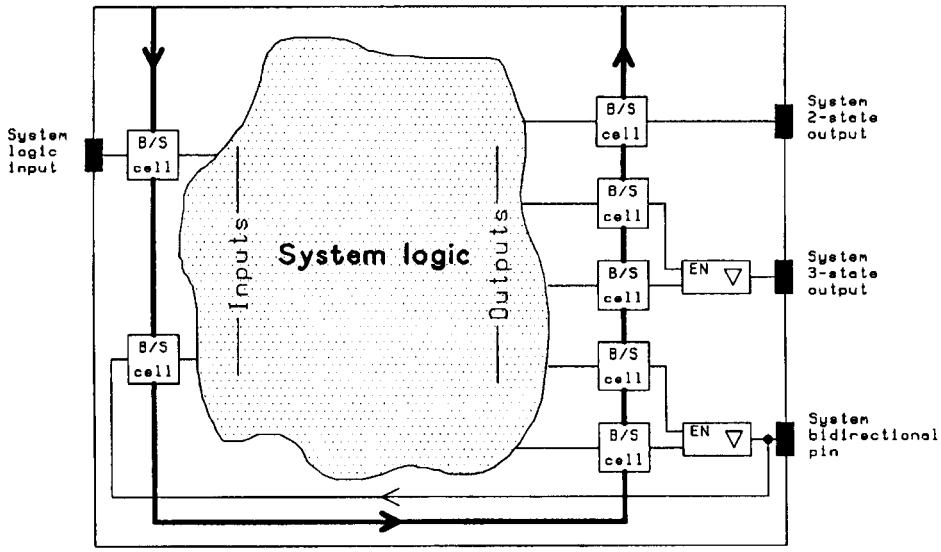


Figure 11-8—Placement of boundary-scan register cells

Boundary-scan register cells are placed such that the state of each digital system pin (including clock pins) can be controlled and/or observed using the boundary-scan register. These cells also may allow the state of the system logic inputs and outputs to be controlled and observed, respectively.

Extension of the design of the boundary-scan register to cover cases in which analog circuit blocks are located external to the on-chip system logic, between the logic and the pins, is straightforward. In such components, the signals that form the interface between the purely digital circuitry and the mixed analog-digital circuit block(s) are considered to be equivalent to system pins. Therefore, boundary-scan register cells are provided for connections that flow to or from the mixed analog-digital block(s) (see Rule 11.4.1 b) and Figure 11-7).

The specification of the boundary-scan register also addresses cases in which logic signals are communicated between components by nondigital or nonelectronic means. Examples would be using optical interconnect or capacitive coupling. In these cases, drivers and receivers are considered to be analog circuit blocks and are placed outside the boundary-scan register cells for the relevant logic signals.

Alternatively, readers of this standard may wish to reference the architecture and capabilities defined by IEEE Std 1149.4-1999, IEEE Standard for a Mixed Signal Test Bus. IEEE Std 1149.4-1999 describes circuits to deliver and measure both noncontinuous (dc) and continuous (ac) signals to and from analog component pins and internal logic via a standardized test interface that is a superset of, and fully compatible

with, the test access port defined by this standard. Of further note, IEEE Std 1149.4-1999 also makes specific provisions for testing of differential-signaling component pins.

The case in which a pair of system pins is used to carry a single logic signal into or out of a component (e.g., at a differential input or output; see Figure 11-9) is slightly more complex and merits further discussion.

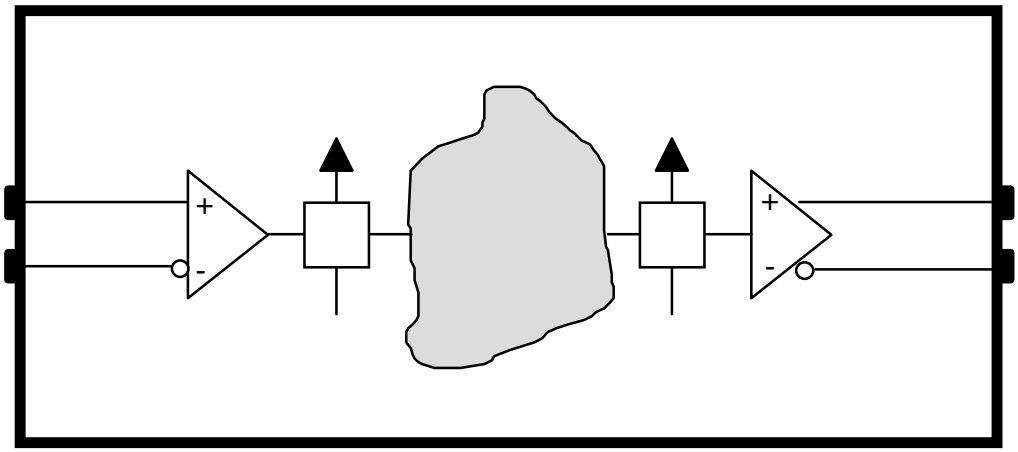


Figure 11-9—A component with differential inputs and outputs

Typically, “paired” I/O will be provided to enhance the performance of connections between components, for example, to enable reliable communication between components in a noisy environment or to reduce skew in high-performance systems. The characteristics that differentiate paired I/O from conventional digital signals are as follows:

a) The signals flowing through the pair of pins typically are driven from a single buffer and are always received by a single buffer.
b) The signals always should be connected in pairs if the “enhanced” behavior (e.g., noise rejection) of the chip-to-chip connection is to be maintained.

Two types of paired I/O are commonplace:

— Those that work by altering or sensing the direction of current flow around the loop formed by the two connections
— Those that appear in many respects to be “conventional” digital signals, for example, because they use TTL-compatible voltage levels

For current-flow paired I/O signals, the differential input or output buffer should be considered an analog circuit. Therefore, one boundary-scan register cell shall be placed between each buffer and the on-chip system logic (Figure 11-9).

For paired conventional signals, the location of the boundary-scan register cells at output pins will depend on the precise characteristics of the link. The following is suggested:

— Where it is possible for the output signals to be used independently (losing the enhanced characteristics of the connection but retaining the ability to convey the logic signal), two cells should be provided for each driver, one for each output pin. This ensures testability of board-level interconnections where one of the output signals from a pair is used to drive a “conventional” input pin on another component.
— Where the signals that constitute the pair cannot be used independently, a single boundary-scan register cell should be provided between the system logic and the output buffer.

For a paired “conventional” input, a single boundary-scan register cell has to be placed between the buffer and the on-chip system logic, allowing observation of the logic signal transmitted across the pair.

11.5 Provision and operation of cells at system logic inputs

This subclause provides rules for the provision and operation of boundary-scan register cells at inputs to the on-chip system logic. These inputs may be driven by buffers at system input pins or at bidirectional system pins when they operate as system inputs. Alternatively, such inputs may be driven by mixed analog/digital circuit blocks located external to the on-chip system logic.

As discussed in 11.1.1, system logic inputs may be either clock or nonclock. Many of the rules presented in 11.5.1 are common to both types of input. However, some rules apply only to one input type. Where this is the case, the rule, recommendation, or permission is labeled “*For clock (nonclock) inputs only:*.”

NOTE—This subclause addresses provision of boundary-scan register cells at inputs to the on-chip system logic in cases in which these inputs are driven by system input pins during normal (nontest) operation. The case in which an input to the on-chip system logic is driven by a system bidirectional pin during normal (nontest) operation is discussed in 11.7.

11.5.1 Specifications

Rules

a) Each signal received at a system logic input (e.g., from an input buffer) shall be capable of being observed by at least one boundary-scan register cell (Figure 11-10).

NOTE—The cells may be observe-only or control-and-observe. Where more than one control-and-observe cell is provided to observe a single signal received at a system logic input, Rule 11.5.1 e) applies. Note that such additional cells are redundant in the sense that they could be omitted from the design without jeopardizing compliance to this standard (see 11.8).

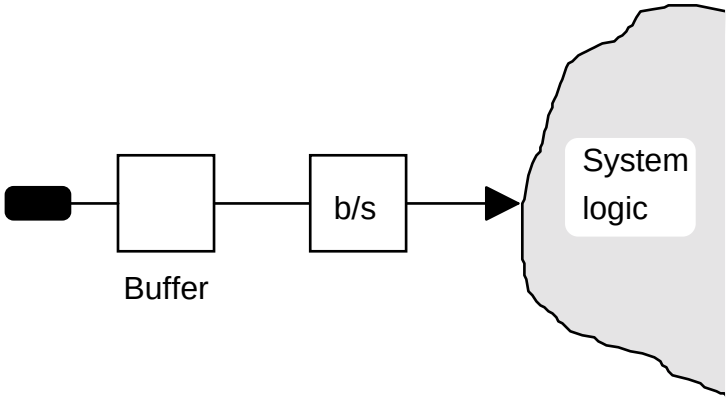


Figure 11-10—Provision of a boundary-scan register cell at a system input

b) Each boundary-scan register cell that can observe a signal received at a system logic input shall observe precisely one such signal.
c) *For nonclock inputs only:* If the *INTEST* instruction is supported, each nonclock input *I* to the on-chip system logic shall be driven from precisely one boundary-scan register cell, and this cell shall be one of those that observe the system input signal that drives *I* during normal (nontest) operation.

NOTES

1—For clock inputs, provision of a control-and-observe cell is optional [see Rule 11.5.1 d) and Permission 11.5.1 l)].

2—It is permissible for a set of control-and-observe boundary-scan register cells to be distributed in a fanout network from, for instance, one system input buffer to the set of inputs to the on-chip system logic normally driven by that buffer (see Figure 11-11). In such cases, Rule 11.3.1 e) applies.

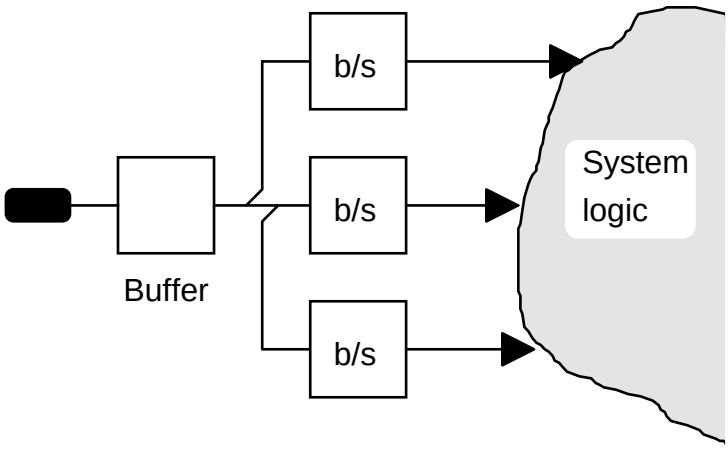


Figure 11-11—Provision of multiple boundary-scan register cells at one input

d) *For clock inputs only:* If the *INTEST* instruction is supported and one or more control-and-observe cells are provided that observe the system clock input signals of a component, each system clock input *I* to the on-chip system logic shall be driven from precisely one control-and-observe cell, and this cell shall be one of those that observe the system clock input that drives *I* during normal (non-test) operation.

NOTES

1—This rule is activated when the option set out in Permission 11.5.1 l) is exercised.

2—It is permissible for a set of control-and-observe boundary-scan register cells to be distributed in a fanout network from, for instance, one system input buffer to the set of inputs to the on-chip system logic normally driven by that buffer (see Figure 11-11). In such cases, Rule 11.3.1 e) applies.

e) The parallel output of a control-and-observe cell that observes a signal received at a system logic input shall drive only one of the following:
   1) One or more system logic inputs;
   2) An output signal driven to a single system logic output pin or external analog/digital circuit block; or
   3) The signal driven to the output control of one or more output buffers.

NOTE—It is a consequence of this rule that where a signal from, for instance, a system input pin would normally fan out to drive more than one of the above options, more than one control-and-observe cell is required. Refer to 11.6 for rules relating to output data signals and output controls.

f) Each cell that observes a signal received at a system logic input shall be designed to route signals as shown in Table 11-1.

g) *For clock inputs only:* When the *INTEST* or *RUNBIST* instruction is selected, the signal driven to the on-chip system logic shall be one of the following:

Table 11-1—Routing of signals in cells at system logic inputs

Instruction	The signal loaded into the shift-register stage of each cell on the rising edge of TCK in the *Capture-DR* controller state is	The signal driven from the parallel output of control-and-observe cells while the instruction is selected is
CLAMP	Not relevant	Not defined[a]
EXTEST	The signal driven to the system logic input from the external source	Not defined[a]
HIGHZ	Not relevant	Not defined[a]
INTEST	Not defined	*For nonclock inputs only:* The parallel output of the shift-register stage *For clock inputs only:* See Rule 11.5.1 g)
PRELOAD	Not defined	The signal received at the connected system pin
RUNBIST	Not defined (Not relevant unless boundary-scan register is selected as the serial path between TDI and TDO)	*For nonclock input only:* Not defined[b] *For clock inputs only:* Defined by Rule 11.5.1 g)
SAMPLE	The signal driven to the system logic input from the external source	The signal received at the connected system pin
BYPASS, *IDCODE*, *USERCODE*	Not relevant	The signal received at the connected system pin

[a]See Rule 11.5.1 h) and Permissions 11.5.1 n) and 11.5.1 o).
[b]See Rules 11.5.1 h) and 11.5.1 k) and Permissions 11.5.1 p) and 11.5.1 q).

1) The signal received at the connected system pin;
2) The TCK signal, controlled such that the on-chip system logic changes state only in the *Run-Test/Idle* controller state; or
3) For *INTEST* only, the parallel output of the shift-register stage.

NOTES

1—Where option 11.5.1 g) 1) is selected, the component designer should assume that the signal applied to the clock input pin will be free-running and not externally controllable. Therefore, to meet Rules 8.9.1 b) and 8.10.1 b), circuitry shall be built into the component to ensure that only appropriate clock transitions cause operation of the on-chip system logic (for example, in the case of the *INTEST* instruction, a "hold" signal may be pulsed internally to ensure single stepping in the *Run-Test/Idle* controller state).

2—Where a component has more than one clock input pin, the component designer should ensure correct operation of the *INTEST* and *RUNBIST* instructions, if provided, for all frequency and phase relationships between the clock input signals that are permissible for correct nontest operation of the component.

h) The component shall not be damaged as a result of signals fed to the on-chip system logic when the *CLAMP*, *EXTEST*, *HIGHZ*, or *RUNBIST* instruction is selected.

NOTE—This may be achieved by disabling operation of the on-chip system logic or by designing the cell such that a constant logic signal is output when these instructions are selected.

i) The design of a boundary-scan register cell that observes the signal received at a system logic input shall be such that if a logic value *V* is present at the system input pin at the time when data is loaded from the signal into the shift-register stage of the boundary-scan register cell (in the *Capture-DR* controller state), the value shifted out of the cell through TDO during the immediate subsequent shifting of the boundary-scan register shall be *V.*

NOTE—For example, where a logic 0 is applied to a system input pin, a logic 0 has to be observed at TDO after loading of the cell that observes that pin. See Figure 11-12.

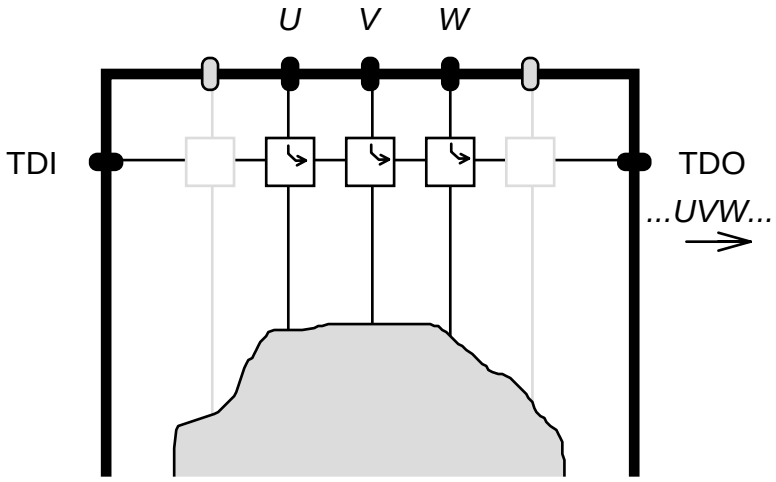


Figure 11-12—Noninversion of data between pin and TDO

j) For each control-and-observe cell that observes a signal received at the system logic from a system input pin *P,* a data value *D* shifted into the cell through TDI and subsequently driven to the on-chip system logic when the *INTEST* instruction is selected shall cause the same result as the application of value *D* at *P* during normal (nontest) operation of the component (Figure 11-13).

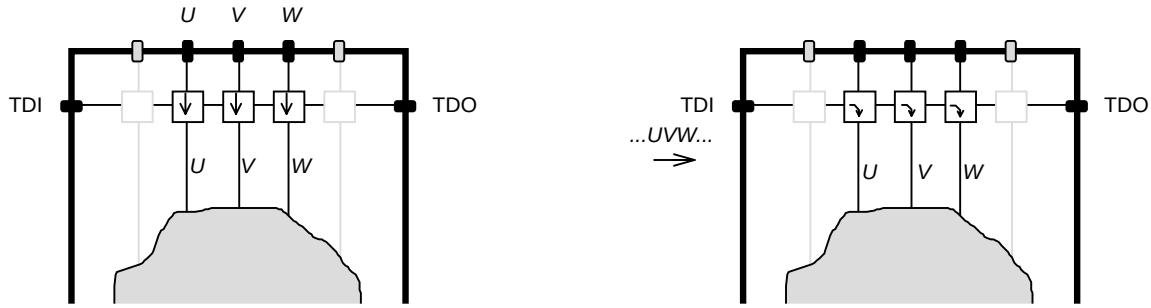


Figure 11-13—Noninversion of data between TDI and the system logic

k) When the *RUNBIST* instruction is provided and selected, the design of boundary-scan register cells that observe signals received at the on-chip system logic shall be such as to prevent interference with self-test execution as a result of data received from external sources (e.g., system input pins).

Permissions

l) *For clock inputs only:* If the *INTEST* instruction is supported, precisely one of the provided boundary-scan register cells that observes the signal received at a clock input to the on-chip system logic also may be able to control that input.

NOTE—See Rule 11.5.1 d).

m) The shift-register stage of a control-and-observe cell that drives an input to the on-chip system logic may be provided with a latched parallel output.

NOTES

1—Where the latched parallel output is omitted, transient data values will be driven to the on-chip system logic during shifting of the boundary-scan register when the *INTEST* instruction is selected. Where this would cause unwanted operation of the on-chip system logic, the parallel output of the shift-register stage can be latched such that the data driven to the system logic changes only on completion of shifting (in the *Update-DR* controller state).

2—Where latched parallel outputs are provided, Rule 11.3.1 c) and Permission 11.3.1 h) apply.

n) Control-and-observe cells that drive inputs to the on-chip system logic may be designed such that when the *CLAMP*, *EXTEST*, or *RUNBIST* instruction is selected, the signal driven to the system logic is the parallel output of the shift-register stage.
o) Control-and-observe cells that drive inputs to the on-chip system logic may be designed such that when the *CLAMP*, *EXTEST*, or *HIGHZ* instruction is selected, a constant signal value is driven to the system logic.
p) Cells may be designed to act as generators of test patterns for the on-chip system logic when the *RUNBIST* instruction (or an alternative self-test instruction) is selected.
q) Cells may be controlled such that during the execution of *RUNBIST* or an alternative self-test instruction, data may flow between the system input pins and the on-chip system logic without modification.

NOTE—However, the results of executing the *RUNBIST* instruction will be independent of data received at nonclock system input pin [see Rule 8.10.1 j)].

r) Inputs to the on-chip system logic may be observed by one or more observe-only boundary-scan register cells in addition to the cell required by Rule 11.5.1 a).

NOTE—Such additional cells are redundant in the sense that they could be omitted from the design without jeopardizing compliance to this standard (see 11.8).

11.5.2 Description

In the example implementations for boundary-scan register cells contained in this clause, the routing of data through each cell is controlled by one or more mode-control signals (labeled *Mode* or *Mode-N*). Different mode-control signals are used for cells at input and output pins of the component, and these signals are derived from the instruction present at the parallel output of the instruction register.

Example designs for boundary-scan register cells located at system input pins are given in Figures 11-14 to 11-18. The value of an appropriate <cell name> from the BSDL Standard VHDL Package that corresponds to each figure is provided for convenience (see B.8.14 and Clause B.9), offset from the main caption within square brackets []. However, it should be noted that the figures represent neither a preferred nor a required implementation of the named BSDL cell type.

Note that Rule 11.4.1 e) permits the input to an observe-only boundary-scan register cell to be taken from any signal that is transparently driven from the system input via a noninverting path, for example, from a point in a signal distribution tree.

Table 11-2 shows the value of the mode signal for the cells illustrated in Figure 11-14 and Figure 11-15 for each of the boundary-scan register instructions defined in Clause 8.

Table 11-2—Mode signal generation for the example cells in Figure 11-14 and Figure 11-15

Instruction	Mode
EXTEST	0
PRELOAD	0
SAMPLE	0
INTEST	1
RUNBIST	X
CLAMP	X

NOTE—When the *EXTEST* instruction is selected, the cell shown in Figure 11-14 feeds data received at the system input pin to the on-chip system logic. In many cases, the on-chip system logic will be tolerant of such signals, which usually will not be representative of those received during normal (nontest) operation. However, in some cases it may be necessary to prevent flow-through of received data (e.g., by adding a logic gate at the output from the cell to the on-chip system logic as shown in Figure 11-16). Alternatively, the cell shown in Figure 11-18 may be used, which allows user control of the data presented to the on-chip system logic while the *EXTEST* instruction is selected.

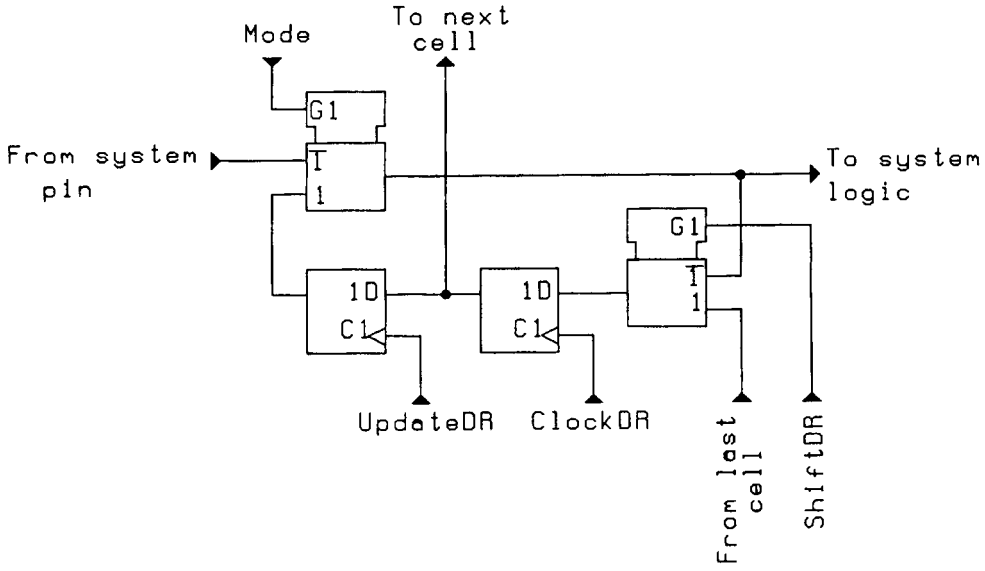


**Figure 11-14—An input cell with parallel output register [BC_2]
(see Table 11-2 for mode signal generation)**

The circuits in Figure 11-14 and Figure 11-15 allow the on-chip system logic to be driven from the boundary-scan register cell when the *INTEST* instruction is selected in accordance with Rule 11.5.1 f) and, for clock inputs, Rule 11.5.1 g) 3). The circuit in Figure 11-17 cannot drive signals into the system logic and may be used at a clock input in accordance with Rule 11.5.1 g) 1). The latter design can be used in circumstances where the delay introduced in the signal path by the multiplexer would cause a design target to be exceeded. (An example would be a high-performance clock pin.)

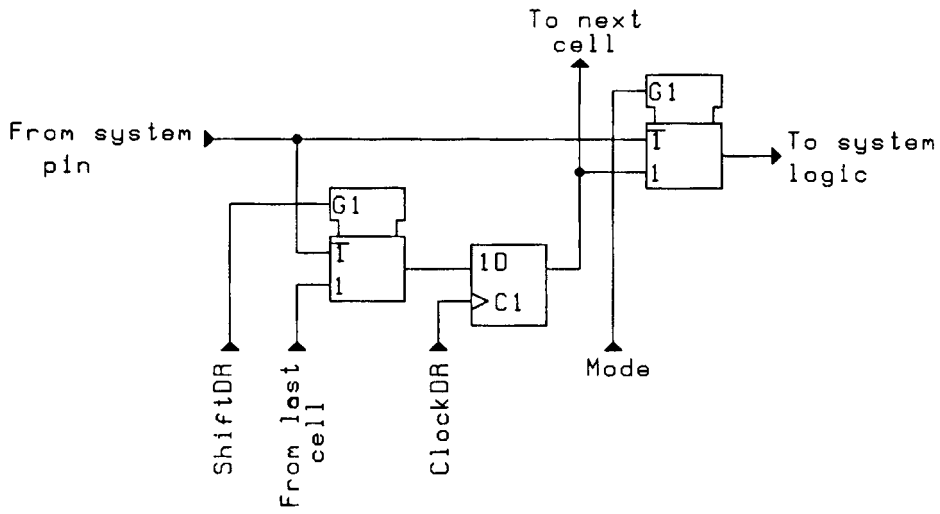


Figure 11-15—An input cell without parallel output register [BC_3]
(see Table 11-2 for mode signal generation)

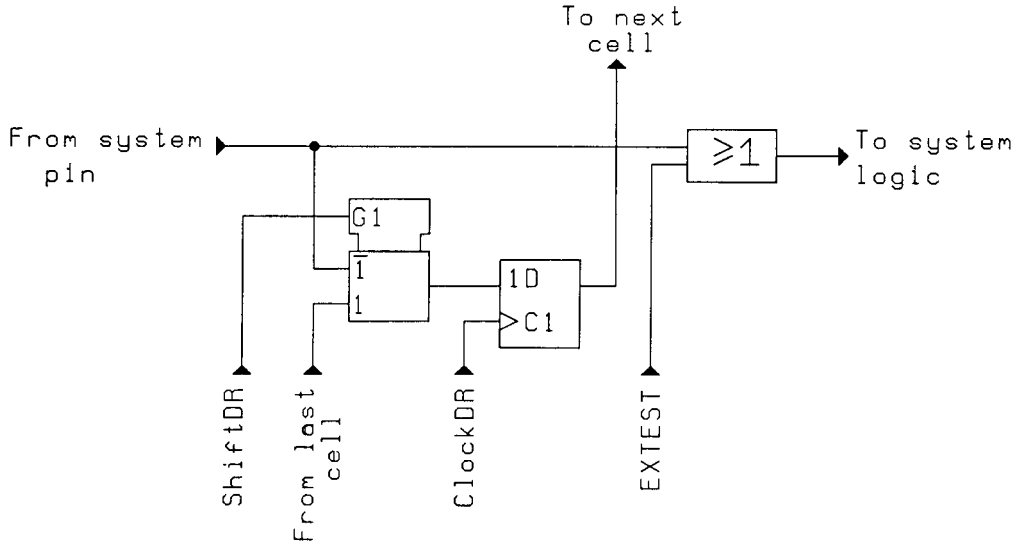


Figure 11-16—A cell that forces the system logic input to 1 during *EXTEST* [BC_4]

The circuit in Figure 11-16 implements Permission 11.5.1 o).

The design in Figure 11-14 includes a parallel output register that is updated from the shift-register stage in the *Update-DR* controller state [Permission 11.5.1 m)]. This register is included to prevent the changes at the output of the shift-register stage during shifting from being applied to the on-chip system logic when the *INTEST* instruction is selected (which could cause unwanted operation). Note that the parallel output could alternatively be held in a level-operated latch, enabled by a logic 1 on the UpdateDR input from the example TAP controller (Figure 6-5 and Figure 6-6).

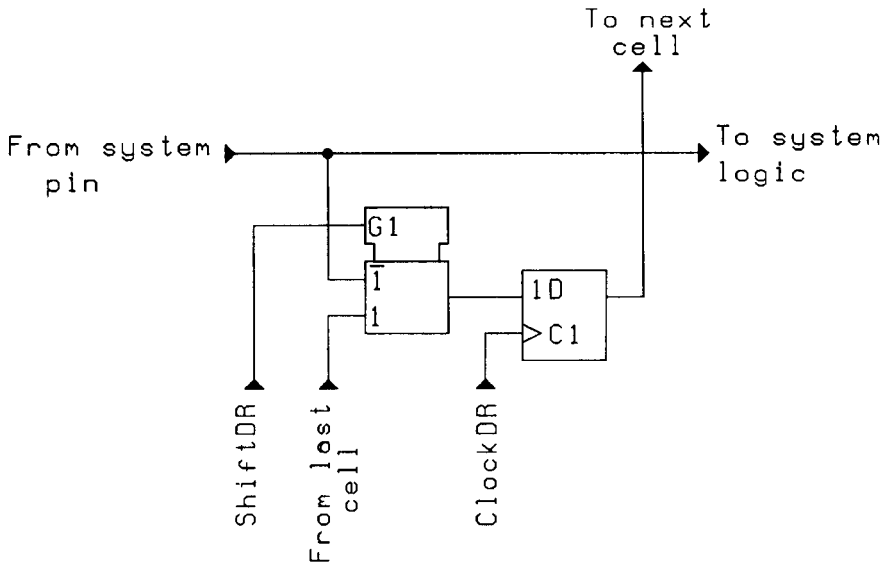


Figure 11-17—An observe-only input cell [BC_4]

The design shown in Figure 11-18 can be used for boundary-scan register cells located at both system input and 2-state system output pins, although the mode signal applied to the cell may need to be different in each case. When the cell is used at a system input pin, the mode signal should be controlled as shown in Table 11-3.

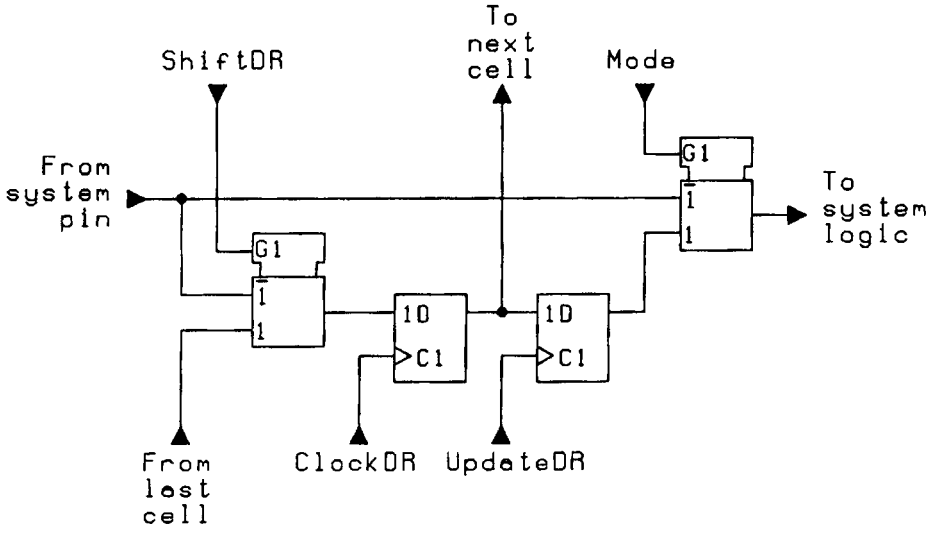


Figure 11-18—An input cell that supports all instructions [BC_1]
(see Table 11-3 for mode signal generation)

Note that the sole difference between the rules that apply to nonclock and clock system inputs is that the provision of control-and-observe cells is not mandatory at clock inputs when the *INTEST* instruction is supported. Thus, there is no requirement for circuitry to be inserted into the signal path between a clock input pin and the on-chip system logic.

Table 11-3—Mode signal generation for the example cell in Figure 11-18

Instruction	Mode
EXTEST	1
PRELOAD	0
SAMPLE	0
INTEST	1
RUNBIST	1
CLAMP	1

11.6 Provision and operation of cells at system logic outputs

The rules in this subclause apply to outputs from the on-chip system logic. These outputs may drive data inputs of buffers at system output pins or at bidirectional system pins when they operate as system outputs. They also may drive output activity controls of buffers located at such pins. Alternatively, on-chip system logic outputs may drive on-chip mixed-signal circuit blocks that are not a part of the on-chip system logic.

Many of the rules presented in 11.6.1 apply to both control and data signals output from the on-chip system logic. However, some rules apply only to on-chip system logic outputs that drive data inputs of buffers at system output pins, while others apply only to outputs that drive control inputs at such buffers. In cases in which a rule is intended to have limited application to one or the other use of a system logic output signal, the rule is prefaced by the phrase "*Control (Data) inputs to buffers only:*."

NOTES

1—Inputs to on-chip analog/digital circuit blocks are considered to be equivalent to data inputs to output buffers.

2—Where the optional *HIGHZ* instruction is provided, selection of this instruction will place every output pin in an inactive drive state, including pins where there is no system requirement for 3-state capability. Where 3-state capability will be provided solely to allow implementation of the *HIGHZ* instruction, the pin should be treated as a 2-state pin for the purposes of this standard.

3—This subclause addresses provision of boundary-scan register cells at outputs from the on-chip system logic in cases where these outputs drive system output pins during normal (nontest) operation. The case in which an output from the on-chip system logic drives a system bidirectional pin during normal (nontest) operation is discussed in 11.7.

11.6.1 Specifications

Rules

a) *Data inputs to buffers only:* For each output of the on-chip system logic that drives the data input of a buffer at a system logic output pin, at least one control-and-observe boundary-scan register cell shall observe only one of the following:
   1) The signal driven from the system logic output (Figure 11-19a); or
   2) The signal at the corresponding output pin (Figure 11-19b).

b) *Control inputs to buffers only:* For each output of the on-chip system logic that drives the control input of a buffer at a system logic output pin, at least one control-and-observe boundary-scan register cell shall observe the signal driven from the system logic output (Figure 11-19c).

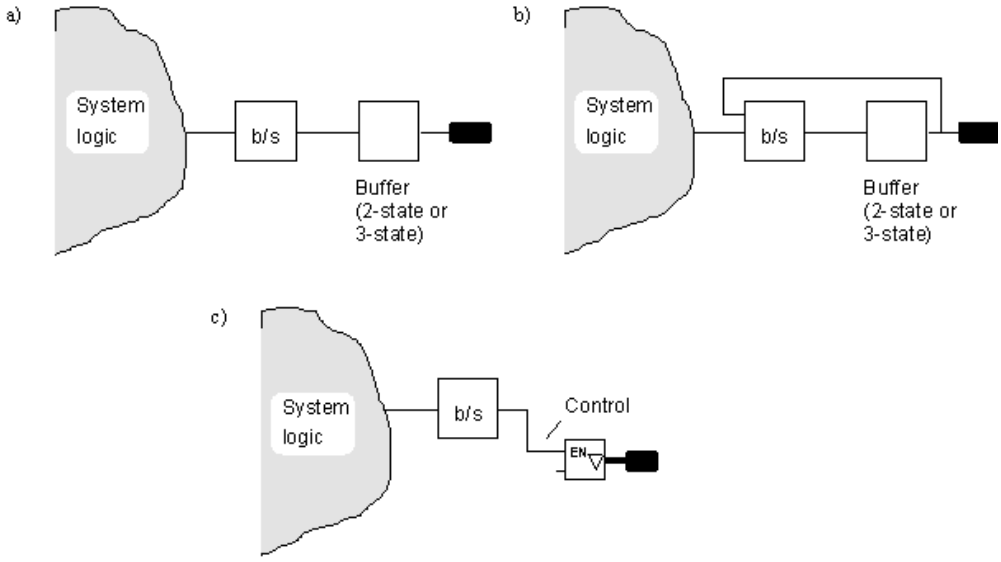


Figure 11-19—Provision of a boundary-scan register cell at a digital system output pin

c) *Data inputs to buffers only:* For a given data input to an output buffer, precisely one of the boundary-scan register cells that satisfy Rule 11.6.1 a) shall be capable of driving that data input (see Figure 11-20).

NOTE—Rule 11.6.1 a) [11.6.1 b)] provides that at least one cell will monitor any given data (control) output from the on-chip system logic. There may be more than one such cell. If, for example, an output from the on-chip system logic fans out to several system output pins, Rules 11.6.1 c) and 11.6.1 e) require precisely one cell capable of driving the connected pin to be placed in each fanout branch. Additional "redundant" observe-only cells may be included in the design [Permission 11.6.1 s)] that can observe the output from the on-chip system logic but that cannot drive any system output pin (see 11.8).

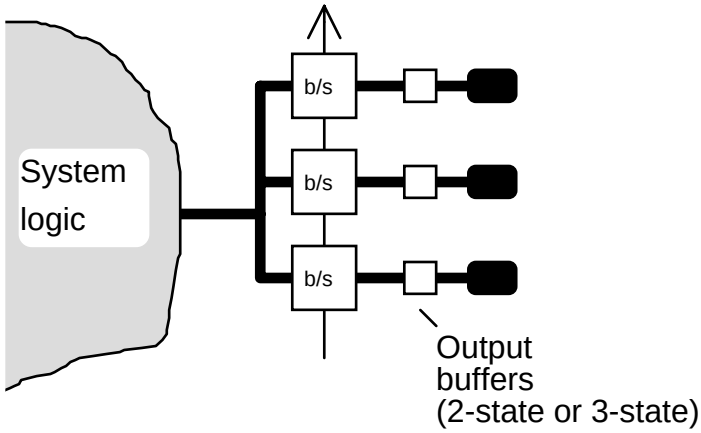


Figure 11-20—Provision of boundary-scan register cells at system logic outputs

d) *Control inputs to buffers only:* For a given control input to an output buffer, precisely one of the boundary-scan register cells that satisfy Rule 11.6.1 b) shall be capable of driving that control input.

NOTE—See also Rules 11.6.1 e) and 11.6.1 f).

e) The parallel output of a control-and-observe boundary-scan register cell that observes a system output from the on-chip system logic shall drive only one of the following:
   1) A single data input to a system output buffer (2-state or 3-state or the output signal for a bidirectional pin); or
   2) The control inputs to a set of output buffers.

NOTE—In the latter case, see Rule 11.6.1 f). Note that where a single output from the on-chip system logic is used both as a control for the output buffers at a group of pins and as the data signal output at one or more other pins, separate boundary-scan register cells are required for the control and noncontrol signal paths, as illustrated in Figure 11-21.

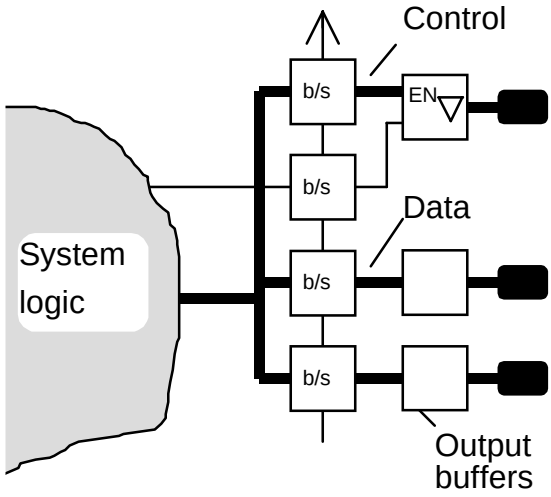


Figure 11-21—Provision of cells when one output is used as both control and data

f) *Control inputs to buffers only:* Where one boundary-scan register cell drives control inputs to output buffers at several system output pins, a given data value (0 or 1) held in the cell shall cause the same operation at all connected system output pins (e.g., a 0 may be defined to place all connected pins in an inactive drive state).

NOTE—See also Recommendation 11.6.1 q).

g) *Control inputs to buffers only:* In cases where a single control signal is driven to a set of system output pins that includes both 3-state and bidirectional pins, in interpreting Rule 11.6.1 f), enabling the 3-state system output(s) shall be considered to be the same as setting bidirectional system pin(s) to output mode; disabling 3-state system output(s) shall be considered to be the same as setting bidirectional pin(s) to input mode.
h) Each cell that observes a system logic output shall be designed to route signals, as shown in Table 11-4.
i) *Control inputs to buffers only:* For each relevant instruction, the option in the right-hand column of Table 11-4 to drive a value that disables connected output buffers shall be selected either for all cells that drive control inputs of output buffers or for none of these cells.

NOTE—For example, if on selection of the *INTEST* instruction one 3-state output pin of the component was forced to the high-impedance state, all other 3-state output pins also would be forced to the high-impedance state when the *INTEST* instruction was selected.

j) Every control-and-observe boundary-scan register cell that observes an output from the on-chip system logic (data or control) shall be provided with a latched parallel output.
k) When the *EXTEST*, *PRELOAD*, or *INTEST* instruction is selected, the latched parallel outputs of control-and-observe boundary-scan register cells that observe outputs from the on-chip system logic shall latch the data held in the shift-register stage on the falling edge of TCK in the *Update-DR* controller state.

Table 11-4—Routing of signals in cells at system logic outputs

Instruction	The signal loaded into the shift-register stage of each cell on the rising edge of TCK in the *Capture-DR* controller state is	The signal driven from the parallel output of each control-and-observe cell while the instruction is selected is
CLAMP	Not relevant	The latched parallel output of shift-register stage
EXTEST	Not defined	The latched parallel output of shift-register stage
HIGHZ	Not relevant	The value that disables connected output buffers
INTEST	The signal output from the on-chip system logic	The latched parallel output of shift-register stage or The value that disables connected output buffers[a]
PRELOAD	Not defined	The signal output from the on-chip system logic
RUNBIST	Not defined (Not relevant unless the boundary-scan register is selected as the serial path between TDI and TDO)	The latched parallel output of shift-register stage; or The value that disables connected output buffers[a]
SAMPLE	The signal output from the on-chip system logic or from the connected output pin	The signal output from the on-chip system logic
BYPASS, IDCODE, USERCODE	Not relevant	The signal output from the on-chip system logic

[a]This option is available only to cells that drive control inputs of output buffers. Where this option is selected for the control input to an output buffer, the data input to that buffer when the instruction is selected may be regarded as "Not defined."

l) When the *CLAMP*, *HIGHZ*, or *RUNBIST* instruction is selected, the latched parallel outputs of control-and-observe boundary-scan register cells that observe outputs from the on-chip system logic shall retain their state unchanged in all controller states.

m) *Data inputs to buffers only*:
   1) If *C* is the control-and-observe boundary-scan register cell that drives data to an output buffer for system output pin *P*; and
   2) If *O* is the output of the on-chip system logic that is observed by *C* and that drives data to *P* during normal (nontest) operation; and
   3) If *C* is the *n*th cell of the boundary-scan register; and
   4) If *V* is the logic signal driven from *P* in normal (nontest) operation (i.e., when the signal driven from *P* is determined by the output from *O*),

   when an instruction is selected that requires the output of *O* to be captured into *C* (e.g., the *SAMPLE* instruction), the logic value observed as the *n*th bit output from the boundary-scan register at TDO in the scan operation immediately after capture shall be *V* (Figure 11-22).

NOTES

1—For example, where the system logic would drive a logic 0 through the system output pin, a logic 0 has to be observed at TDO after loading of the cell that observes that pin. See Figure 11-22.

2—For outputs where only one logic value is actively driven (e.g., open-collector outputs), receipt of one data value (0 or 1) from the on-chip system logic will cause the output to be inactive. In these cases, the data value observed at TDO will be the value that, when fed to the output buffer from the on-chip system logic, will cause the output to be inactive.

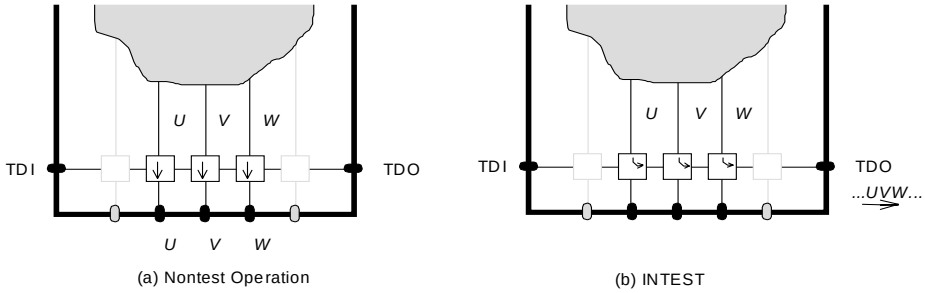


Figure 11-22—Noninversion of data between the system logic and TDO

n) *Data inputs to buffers only*:
   1) If *C* is the control-and-observe boundary-scan register cell that drives data to an output buffer for system output pin *P*; and
   2) If *C* is the *n*th cell of the boundary-scan register; and
   3) If *V* is the value of the *n*th bit of a serial data stream *S* input to the boundary-scan register via TDI; and
   4) If the length of *S* is equal to the number of cells in the boundary-scan register,

   when an instruction is selected that causes the latched parallel output of *C* to determine the value of the data signal driven from *P* (e.g., *EXTEST*) and when the data stream *S* is shifted into the boundary-scan register and, immediately subsequent to the shifting operation, updated to the latched parallel outputs of the boundary-scan register, the value of the data signal output from *P* shall be *V* (Figure 11-23).

NOTE—For outputs where only one logic value is actively driven (e.g., open-collector outputs), receipt of one data value (0 or 1) from the on-chip system logic will cause the output to be inactive. In these cases, the data value input at TDI shall be the value that, when fed to the output buffer from the on-chip system logic, will cause the output to be inactive.

o) *Control inputs to buffers only*:
   1) If *C* is the control-and-observe boundary-scan register cell that drives the control input of the output buffer for a system output pin *P*; and
   2) If *O* is the output of the on-chip system logic that is observed by *C* and that controls the activity of *P* during normal (nontest) operation; and
   3) If *C* is the *n*th cell in the boundary-scan register; and
   4) If *V* (not *V*) is the value of the *n*th bit of *S* output from the boundary-scan register through TDO immediately after capture of *O* into *C* when *P* would be active (inactive) in normal (nontest) operation; and
   5) If the length of *S* is equal to the number of cells in the boundary-scan register,

   when an instruction is selected that causes the latched parallel output of *C* to determine the activity of *P* (e.g., *EXTEST*) and when a data pattern *S* containing *V* (not *V*) as the *n*th bit is shifted onto the latched parallel outputs of the boundary-scan register, *P* shall be active (inactive).

NOTE—For example, where a logic 0 output from the system logic normally would disable a driven output buffer, a logic 0 should be shifted out through TDO whenever the output would have been disabled. Further, a logic 0 shifted into

the cell through TDI should, if driven to the output buffer, cause the output buffer to be disabled. The effect of this rule is similar to that of Rules 11.6.1 m) and 11.6.1 n) (Figure 11-24).

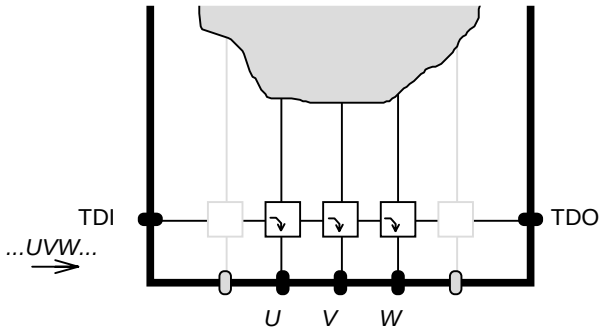


Figure 11-23—Noninversion of data between TDI and a system output pin

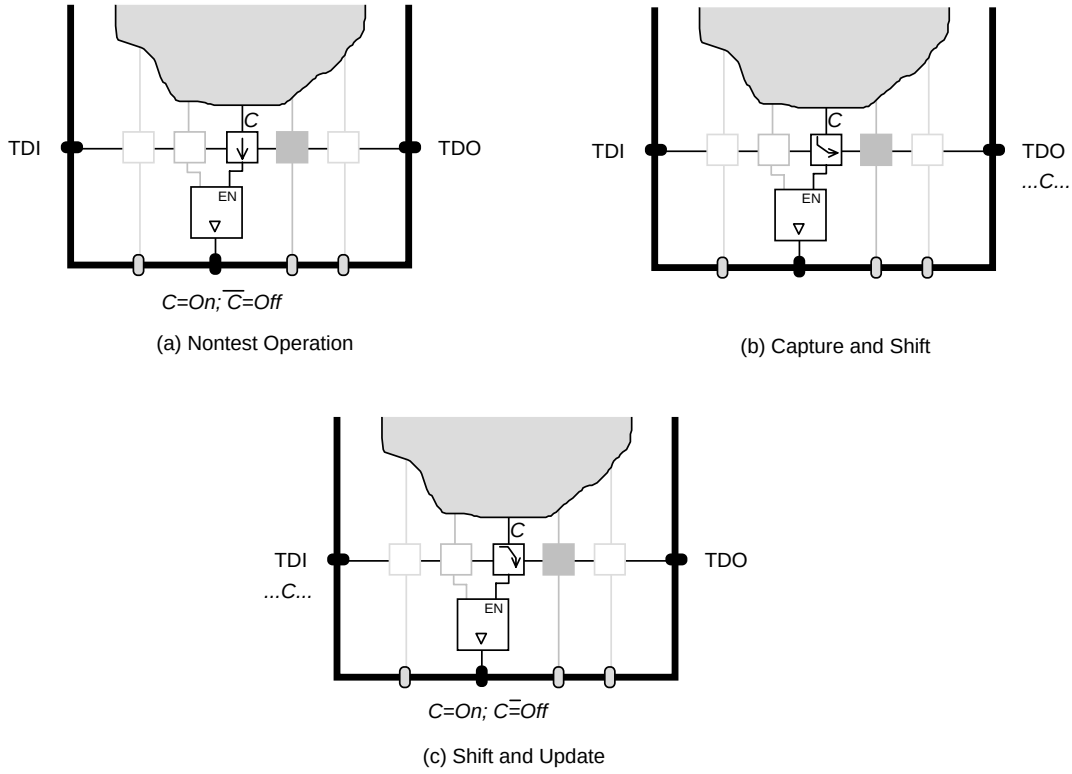


Figure 11-24—Noninversion of control signal values between the system logic and TDO

p) *Control inputs to buffers only*: If the latched parallel output of a boundary-scan register cell that drives a control input to an output buffer is reset in the *Test-Logic-Reset* controller state, it shall be reset to the state that will cause the connected output buffer(s) to be disabled.

NOTE—The timing of the reset is specified in Rule 11.3.1 c) and Permission 11.3.1 h).

Recommendations

q) *Control inputs to buffers only*: The control signal for each functionally distinct group of system output pins (e.g., an address bus or a data bus) should be driven from a distinct boundary-scan register cell dedicated to that purpose, even where the output from the on-chip system logic observed by that cell normally would drive a common control signal to more than one such group of pins.

NOTE—This reduces the complexity of the test generation task because during the test, buffers can be enabled independently as required.

Permissions

r) Boundary-scan register cells that observe outputs from the on-chip system logic may be designed to act as a part of a signature computing register for test results when the *RUNBIST* instruction (or an alternative self-test instruction) is selected.

s) Outputs from the on-chip system logic may be observed by one or more observe-only boundary-scan register cells in addition to the control-and-observe cells required by the preceding rules.

NOTE—Such additional cells are redundant in the sense that they could be omitted from the design without jeopardizing compliance to this standard (see 11.8).

11.6.2 Description

For 2-state output pins, where signals only can be at the high or low logic level at any given instant, one boundary-scan register cell is sufficient to allow the state of the pin to be controlled or observed. However, for 3-state pins the capability exists for data to be driven actively or inactively, such that four states are possible. Data from a minimum of two boundary-scan register cells therefore is required to allow the state (signal value plus active/inactive) of a 3-state pin to be controlled or observed.

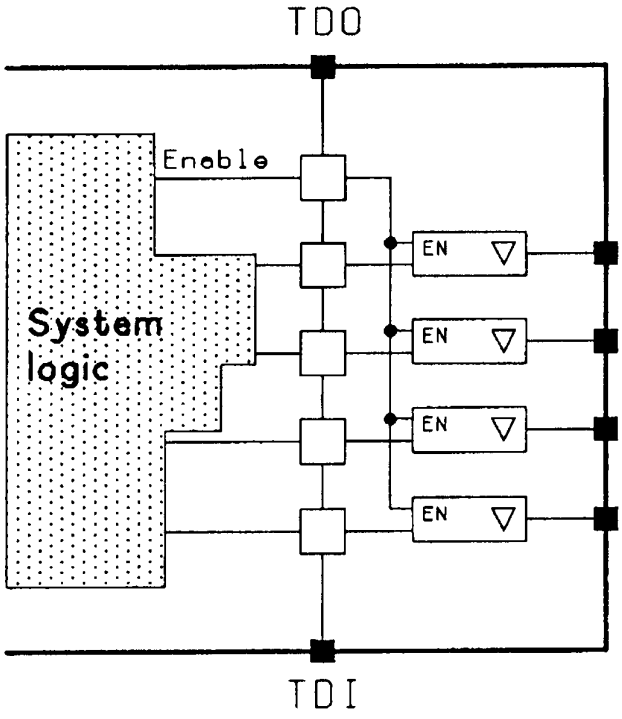


Figure 11-25—Control of multiple 3-state outputs from one signal

Although it would appear that the additional cells might significantly increase the overhead needed to implement boundary scan, it is necessary to provide only one additional cell for each activity control signal generated in the circuit, although a judicious use of a few additional cells is recommended in Recommendation 11.6.1 q) (see, for example, Figure 11-25). Thus, where the activity of many 3-state output pins is controlled from a single source, for example, in a microprocessor address bus, only one additional cell is required to give the necessary control. Since, given the basic design of the circuit, it would be a design error if such 3-state pins were wired together, there is no need for the design of the boundary-scan register to account for this possibility.

The need for the additional cells at output control signals is illustrated in Figure 11-26. This shows a wired junction between 3-state outputs from a number of components. To test this junction, a series of tests shall be performed, each of which checks that one of the outputs can drive either a 0 or a 1 to the receiving devices. During each test, the other outputs have to be set to the opposite data value (1 or 0, respectively) with a high-impedance drive. Table 11-5 shows the pair of tests needed to check the operation of one of the outputs connected to the junction.

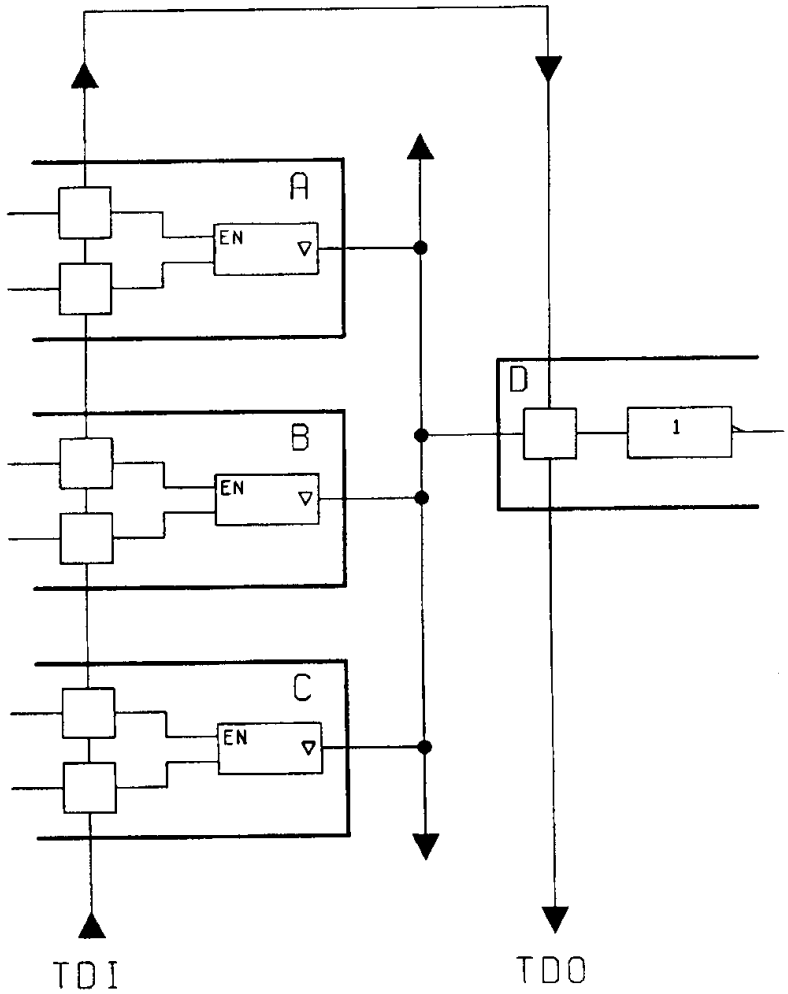


Figure 11-26—Testing board-level bus lines

To apply the test, it is necessary to be able to control both the data value at each output and whether the output is enabled. This can be done via the boundary-scan register independently of the on-chip system logic.

Table 11-5—Test for driver B

Stimulus applied to the bus from			Result seen at
Component A	**Component B**	**Component C**	**Component D**
1/off	0/on	1/off	0
0/off	1/on	0/off	1

For similar reasons, there shall be additional boundary-scan register cells associated with each 3-state bidirectional system pin that control the activity of the associated output buffer. As in the case of 3-state system pins, these cells may be shared across a bus or between any group of 3-state bidirectional system pins that obtain their output control signal from a single source.

Figure 11-27 highlights the following problems that might be encountered when applying tests to logic blocks external to a component by using the boundary-scan register of the component but that are avoided by implementing boundary-scan register cells as defined in this clause.

— The logic block being tested may contain asynchronous sequential logic that will be set into undesirable states if shifting patterns appear at its inputs.

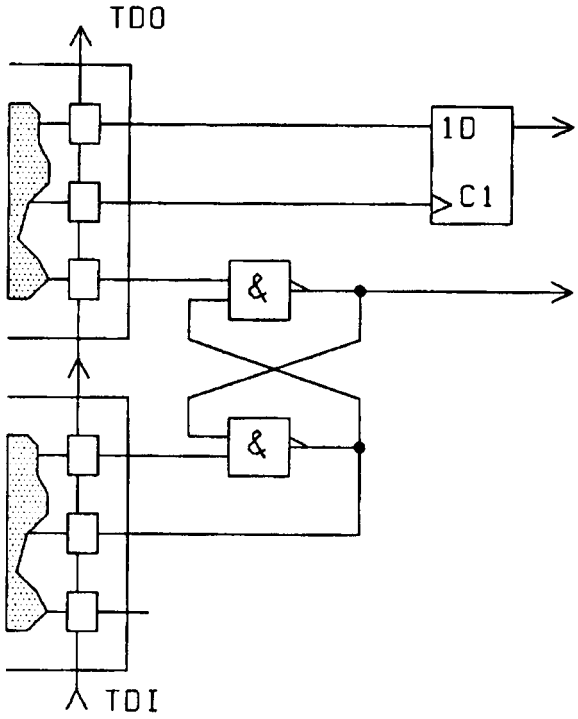


Figure 11-27—Testing external logic via the boundary-scan path

— The signals applied from the boundary-scan register may feed into clock inputs on the logic block being tested, which again will produce undesirable effects if the logic is not shielded from shifting patterns.

Since in a generally applicable architecture it cannot be guaranteed that such features do not exist in the circuitry under test, the boundary-scan design shall be such that these problems are guaranteed to be avoided. A design compatible with this standard ensures this by requiring a parallel output register or latch in each boundary-scan register cell that can affect the state of an output driver at a system pin. The inclusion of this register or latch ensures that while the *EXTEST* instruction is selected, the data driven from a component to neighboring circuitry changes only on completion of the shifting process.

A further potential problem is highlighted by the primitive (noncompliant) boundary-scan register cell design shown in Figure 11-28. During testing of the on-chip system logic (for example, through the *INTEST* or *RUNBIST* instruction), the example cell would allow responses from the system logic to pass through the data-path multiplexer to the shift-register input of the cell. This allows the output response from the on-chip system logic to be loaded into the output boundary-scan register cells and shifted out for inspection. However, a problem arises from the fact that the cell also allows the test response from the on-chip system logic to be output from the host component and hence to be applied to neighboring components on a board assembly.

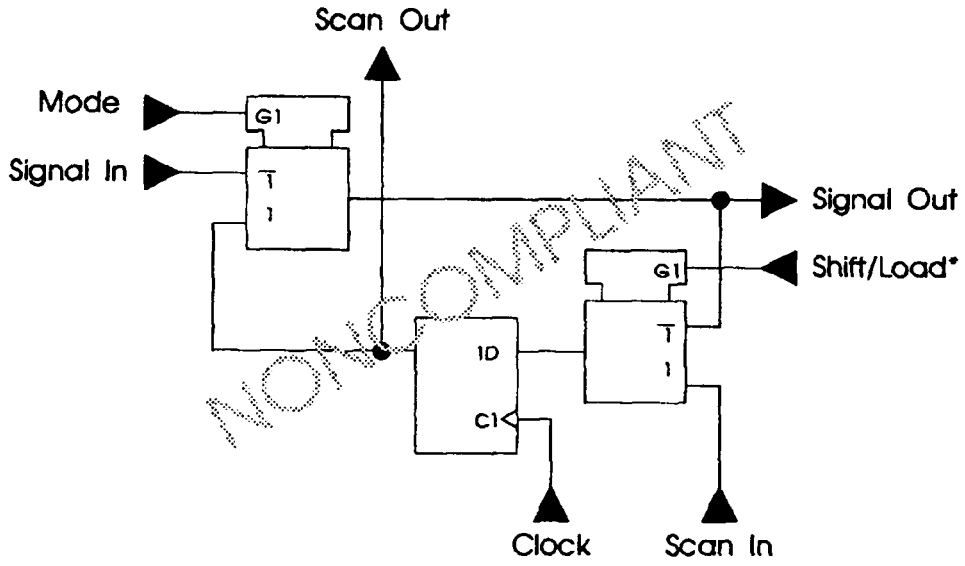


Figure 11-28—A primitive noncompliant output cell design with potential problems

The application of raw test-response data from one component can have a damaging effect on other components in the circuit if it is received at clock or asynchronous data or control inputs. For example, if built-in self-testing is being performed on the memory controller of Figure 11-29, there is a distinct possibility that one or more test-response patterns from the core logic of the memory controller will cause simultaneous activation of the outputs feeding the chip select (CS) inputs of the memory devices. This situation would not occur during normal operation of the complete design, either due to constraints between the logic values applied to the inputs of the memory controller or due to the design of the on-chip system logic. The design would in some way ensure that only one output from the controller was active at any time.

The duration of an on-chip test is dependent on the type of system logic test performed. For static tests applied using the *INTEST* instruction, these potentially damaging output patterns can remain in effect over the interval between successive occurrences of the *Update-DR* controller state. For instance, in a circuit having a scan path length of 500 bits and a TCK rate of 5 MHz, the approximate interval between closest consecutive *Update-DR* controller states is 100 ms. For large board designs, the period could be sufficiently long to cause damage to drivers in contention on a bus.

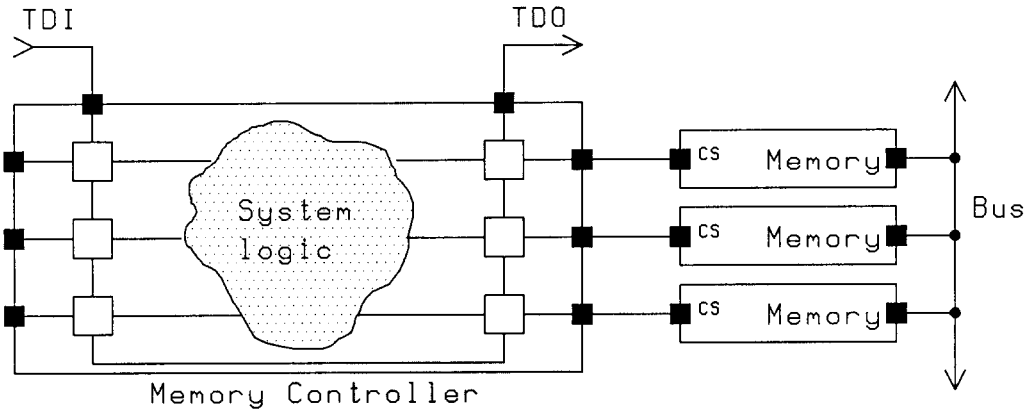


Figure 11-29—A circuit illustrating potential boundary-scan test problem

One solution is to cause the output buffers of the memory controller that feed the memory CS inputs to be placed in a high-impedance state during internal testing of the controller. However, floating inputs can fluctuate between high and low logic levels and are susceptible to induced voltages from adjacent board wiring interconnects. Applying a pull-up resistor on the 3-state buffers will solve the bus contention problem in external components with active-low 3-state enables, but those with active-high 3-state enables are still at risk.

The solution adopted in this standard is to ensure that boundary-scan register cells placed at 2-state output pins are designed such that user-defined logic values can be placed at the associated pins while system logic within the component is tested. Figure 11-30 shows a design that provides this facility and meets the rules defined in this clause. Table 11-6 shows how the mode signal for Figure 11-30 is derived for each of the boundary-scan register instructions defined in Clause 8.

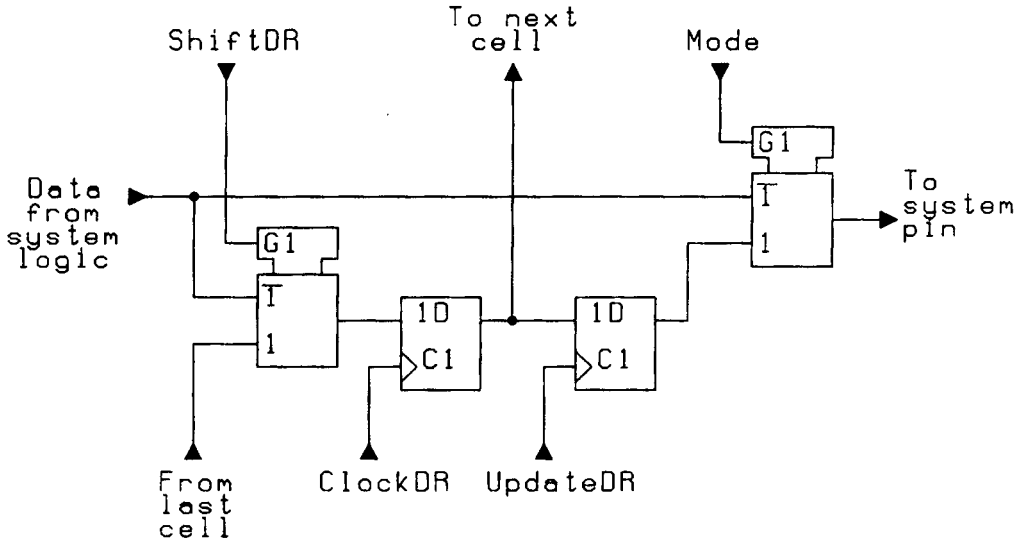


Figure 11-30—An output cell that supports all instructions [BC_1]
(see Table 11-6 for mode signal generation)

Table 11-6—Mode signal generation for the example cells in Figure 11-30, Figure 11-34, Figure 11-36, and Figure 11-46

Instruction	Mode
EXTEST	1
PRELOAD	0
SAMPLE	0
INTEST	1
RUNBIST	1
CLAMP	1

Note that the path in Figure 11-30 between the data input from the system logic and the multiplexer that feeds data to the system pin will not be used during execution of either the *EXTEST* or the *INTEST* instruction. In some cases, it may therefore be necessary to use additional test operations at the board level to test the circuitry within a component fully.

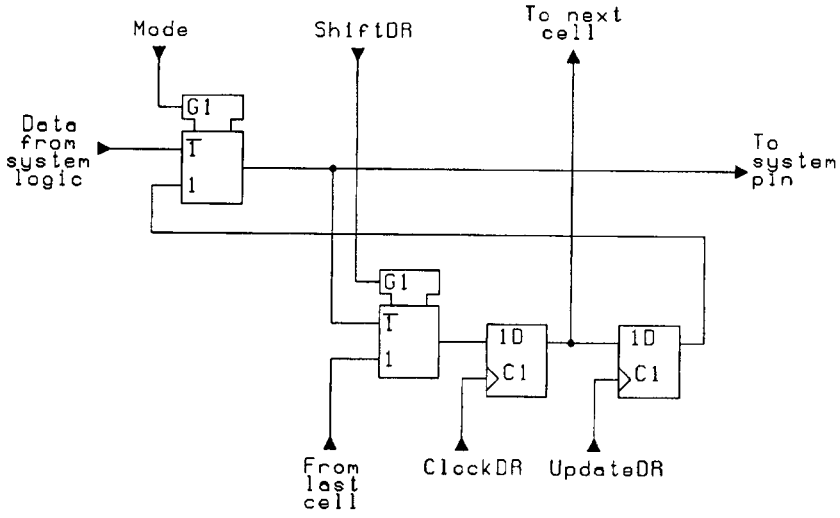


Figure 11-31—An output cell that supports *SAMPLE*, *PRELOAD*, *EXTEST*, and *RUNBIST* [BC_2] (see Table 11-7 for mode signal generation)

The example cell design in Figure 11-31 could be used where the *INTEST* instruction is not supported by a component, since this design does not permit Rule 11.6.1 h) to be met with respect to the *INTEST* instruction. Note that while the cell meets Rule 11.6.1 h) with respect to the *SAMPLE* instruction without additional provision, if the cell drives off-chip via an output buffer, it can be ensured that the signal value captured using the *SAMPLE* instruction is the one intended to be driven off-chip, not the one actually on the off-chip connection at the time. The latter may be affected by faults on the off-chip connection or, for bus connections, by the combination of drivers active at the time. By ensuring that the signal that should have been driven from the chip is sampled at the driving end, while the signal actually driven is sampled at the receiving end, additional diagnostic information is obtained.

Table 11-7 shows how the mode signal for Figure 11-31 is derived for each of the boundary-scan register instructions supported by the cell design.

Table 11-7—Mode signal generation for the example cells in Figure 11-31, Figure 11-33, and Figure 11-39

Instruction	Mode
EXTEST	1
PRELOAD	0
SAMPLE	0
RUNBIST	1
CLAMP	1

On the other hand, it is highly useful that the signal value captured using the *EXTEST* instruction is the one at the corresponding system output pin, according to Rule 11.6.1 a) 2). Doing so allows a connected system network both to be driven and to be captured at the same pin, thus allowing such networks to be tested for shorts to others even where there are no other connected boundary-scan device pins. Output cells of this type are termed self-monitoring. A cell design that implements this option while still capturing the system logic output during *SAMPLE* and *INTEST* is shown in Figure 11-32. An alternative, simpler cell design that also implements this option but cannot capture the system logic output, and thus does not support *INTEST,* is shown in Figure 11-33.

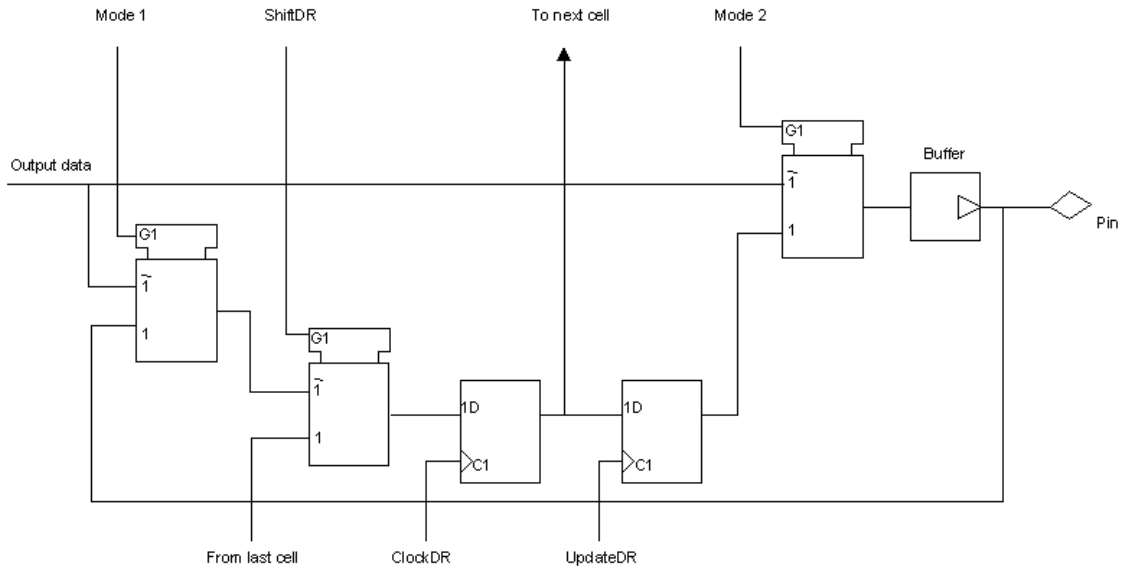


Figure 11-32—A self-monitoring output cell that supports *INTEST* [BC_9] (see Table 11-8 for mode signal generation)

Where a component has 3-state system output pins, these pins may feed onto a wired junction at the board level. In order to test the interconnections forming the wired junction using the *EXTEST* instruction, it shall be possible to drive independently onto the junction from each of the possible driving pins. As was dis-

Table 11-8—Mode signal generation for the example cell in Figure 11-32

Instruction	Mode 1	Mode 2
EXTEST	1	1
PRELOAD	X	0
SAMPLE	0	0
INTEST	0	1
RUNBIST	X	1
CLAMP	X	1

cussed earlier in this clause, to achieve this it is necessary to be able to control the output control signals fed to the output drivers at 3-state or bidirectional system pins.

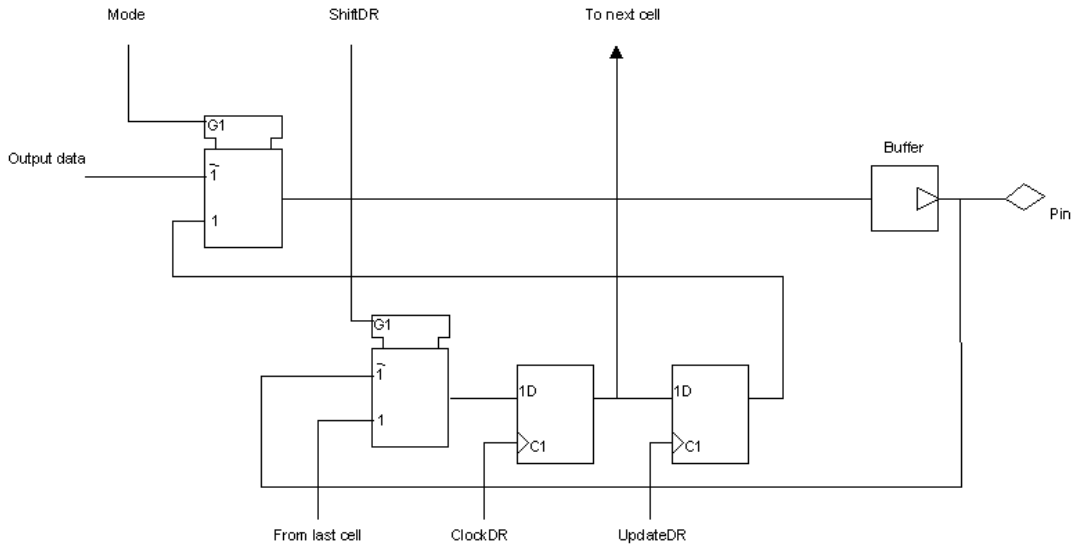


Figure 11-33—A self-monitoring output cell that does not support *INTEST* [BC_10] (see Table 11-7 for mode signal generation)

In addition, it is necessary to ensure that contention does not occur on board-level interconnections when the on-chip system logic is tested using the *INTEST* or *RUNBIST* instruction. This requirement can be met in either of two ways:

a) The state of a system pin can be fully defined by the user by shifting data into the boundary-scan register.
b) A system pin can be forced into the inactive drive state. This additional option is possible since the board-level circuit design shall necessarily be designed such that components driven from the 3-state bus do not erroneously respond to high-impedance conditions during normal system operation. Therefore, the inactive drive state can be safely driven during testing of the system logic within a component.

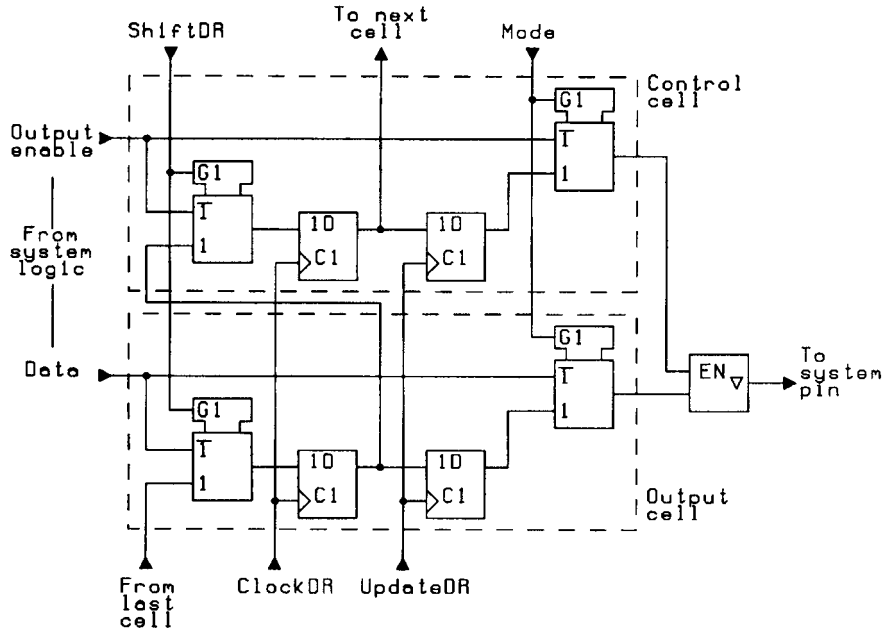


Figure 11-34—Boundary-scan register cells at a 3-state output—Example 1 [BC_1, control and data] (see Table 11-6 for mode signal generation)

The options listed for the *INTEST* and *RUNBIST* instructions in Rule 11.6.1 h) cover these two possibilities. Figure 11-34 and Figure 11-35 give example designs for a boundary-scan register cell that could be used at a 3-state system output pin. Figure 11-34 implements option a), while Figure 11-35 implements option b). In Figure 11-34, the mode signal should be controlled as shown in Table 11-6.

In Figure 11-35, the design of the circuitry around the shift-register stages is such that all paths can be tested if both the *EXTEST* and *INTEST* instructions are executed with appropriate data. The Mode_1 and Mode_2 signals should be controlled as shown in Table 11-9.

Table 11-9—Mode signal generation for the example cell in Figure 11-35

Instruction	Mode_1	Mode_2
EXTEST	1	1
PRELOAD	0	1
SAMPLE	0	1
INTEST	0	0
RUNBIST	0	0
CLAMP	1	1
HIGHZ	0	0

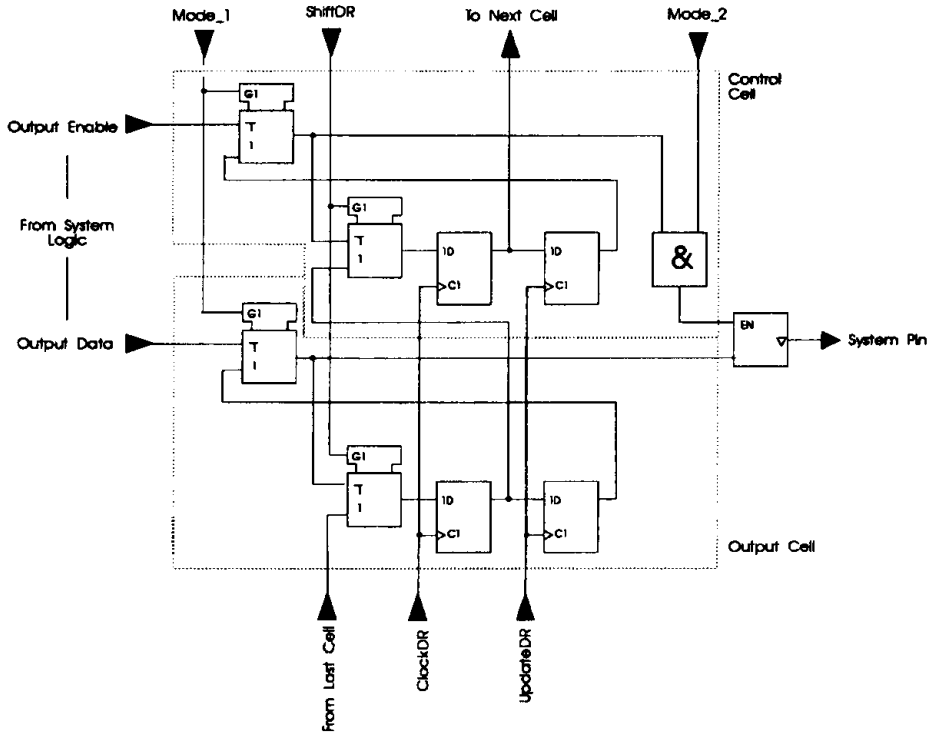


Figure 11-35—Boundary-scan register cells at a 3-state output—Example 2 [BC_2, control and data] (see Table 11-9 for mode signal generation)

11.7 Provision and operation of cells at bidirectional system logic pins

11.7.1 Specifications

Rules

a) Boundary-scan register cells shall be provided at bidirectional system pins such that
   1) Whenever the pin is an input pin, all rules are met for cells provided at system input pins and inputs to the on-chip system logic (see 11.5);
   2) Whenever the pin is an output pin, all rules are met for cells provided at outputs of the on-chip system logic that drive data inputs of system output buffers (see 11.6); and
   3) All rules are met for cells provided at outputs of the on-chip system logic that drive control inputs of buffers at system output pins (see 11.6).

NOTE—In cases where the direction of signal flow is determined by an output *O* of the on-chip system logic, a boundary-scan register cell will exist in the signal path between *O* and the system pin. When the *EXTEST*, *CLAMP*, *INTEST*, or *RUNBIST* instruction is selected, the direction of signal flow will be determined by the data held in the latched parallel output of the shift-register stage of the boundary-scan register cell.

b) Whenever two separate boundary-scan register cells are provided at a bidirectional system pin to meet the requirements of Rules 11.7.1 a) 1) and 11.7.1 a) 2),
   1) The cell that meets the requirements of Rule 11.7.1 a) 1) shall at all times meet all rules for cells provided at system input pins and inputs to the on-chip system logic (see 11.5); and

2) The cell that meets the requirements of Rule 11.7.1 a) 2) shall at all times meet all rules for cells provided at outputs of the on-chip system logic that drive data inputs of system output buffers (see 11.6).

NOTE—A structure of two boundary-scan register cells that would meet Rule 11.7.1 a) while failing to meet Rule 11.7.1 b) not only would require more logic than a structure which meets both Rules 11.7.1 a) and 11.7.1 b) but also would have less test usefulness.

11.7.2 Description

These requirements represent a merging of those for 2-state or 3-state output pins with those for system input pins.

Figure 11-36 and Figure 11-37 give examples of the provision of boundary-scan register cells at 3-state bidirectional pins.

a) Figure 11-36 allows the state of the pin to be fully controlled while the *INTEST* or *RUNBIST* instruction is selected. The mode signal shown in Figure 11-36 should be controlled as indicated in Table 11-6. The *Reset** signal is fed from the example TAP controller in Figure 6-5 to the parallel output register of the control cell in accordance with Permission 11.3.1 h).

Table 11-10—Mode signal generation for the example cells in Figure 11-37

Instruction	Mode_1	Mode_2	Mode_3
EXTEST	1	0	1
PRELOAD	0	0	1
SAMPLE	0	0	1
INTEST	0	1	0
RUNBIST	X	X	0
CLAMP	1	X	1
HIGHZ	X	X	0

b) In Figure 11-37, a single boundary-scan register cell is used to control and observe both output and input data. This cell meets the requirements of 11.6.1 when the pin is defined to be an output and the requirements of 11.5.1 when it is defined to be an input. The various control signals used by the cell should be controlled as shown in Table 11-10.

As discussed in connection with Figure 11-35, the design of the circuitry around the shift-register stages in Figure 11-37 permits all circuitry in the cell to be tested if the *EXTEST* and *INTEST* instructions are executed with appropriate data.

c) Figure 11-38 is similar to Figure 11-37. Note that while this design conforms fully to the rules set out in this standard, it is not recommended for use in new component designs. This is because the combined input and output cell does not capture as much data as possible about the external interconnection when the *EXTEST* instruction is selected. The example design in Figure 11-37 is superior in this respect.

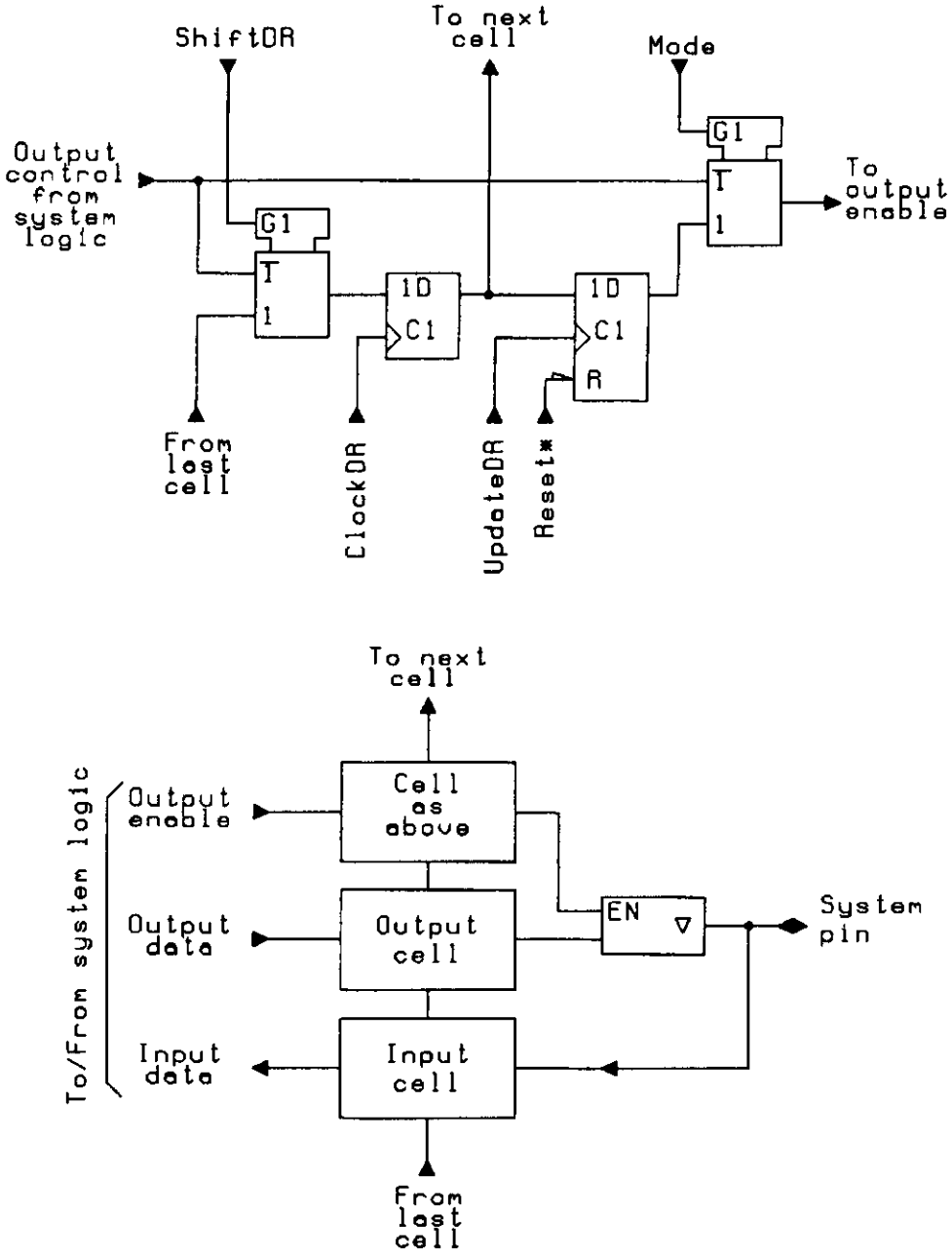


Figure 11-36—Boundary-scan register cells at a bidirectional pin—Example 1 [BC_1, control] (see Table 11-6 for mode signal generation)

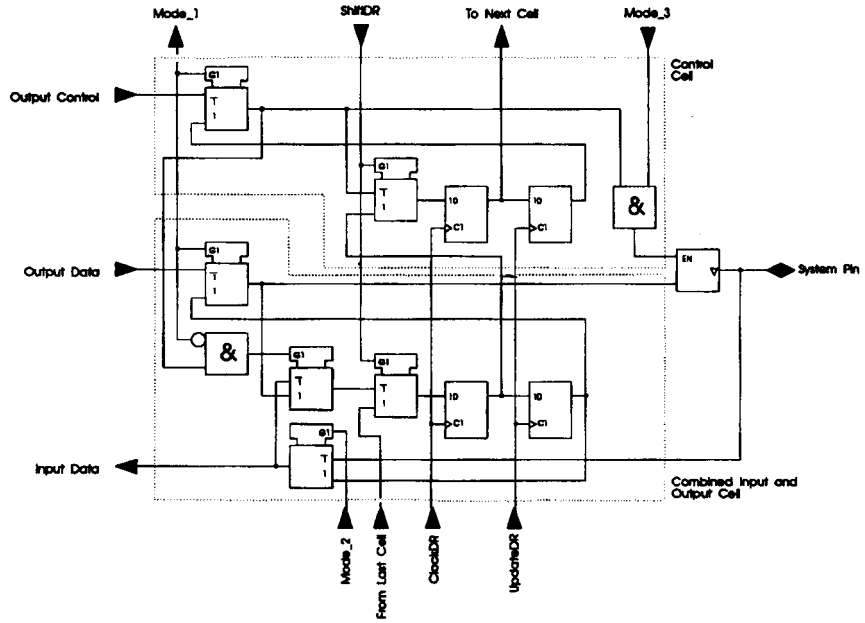


Figure 11-37—Boundary-scan register cells at a bidirectional pin—Example 2 [BC_2, control; BC_7, data] (see Table 11-10 for mode signal generation)

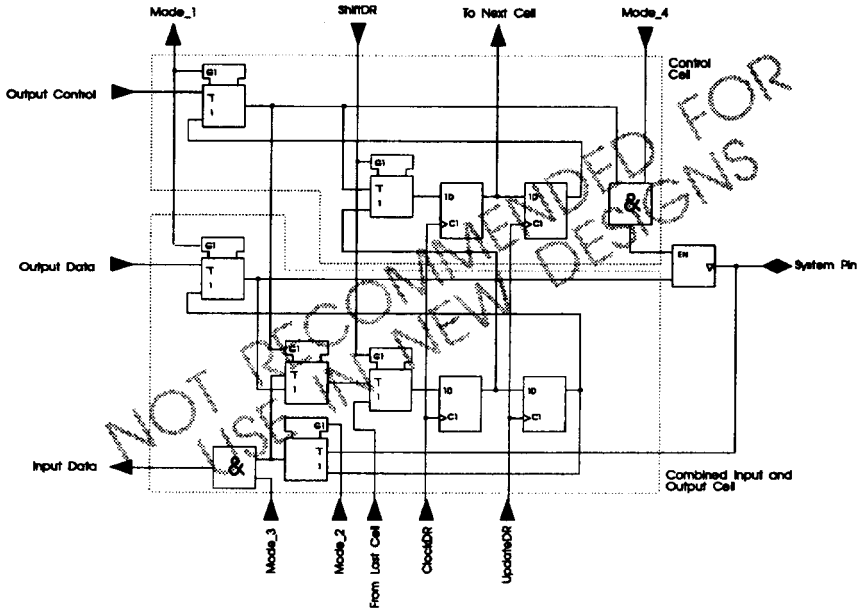


Figure 11-38—Boundary-scan register cells at a bidirectional pin—not recommended for new designs [BC_2, control; BC_6, data] (see Table 11-11 for mode signal generation)

Table 11-11—Mode signal generation for the example cells in Figure 11-38

Instruction	Mode_1	Mode_2	Mode_3	Mode_4
EXTEST	1	0	0	1
PRELOAD	0	0	1	1
SAMPLE	0	0	1	1
INTEST	0	1	1	0
RUNBIST	X	X	X	0
CLAMP	1	X	0	1
HIGHZ	X	X	0	0

Figure 11-39 shows how boundary-scan register cells may be provided at an open-collector bidirectional pin. Note that, like the cell design in Figure 11-31, this cell design does not support the *INTEST* instruction.

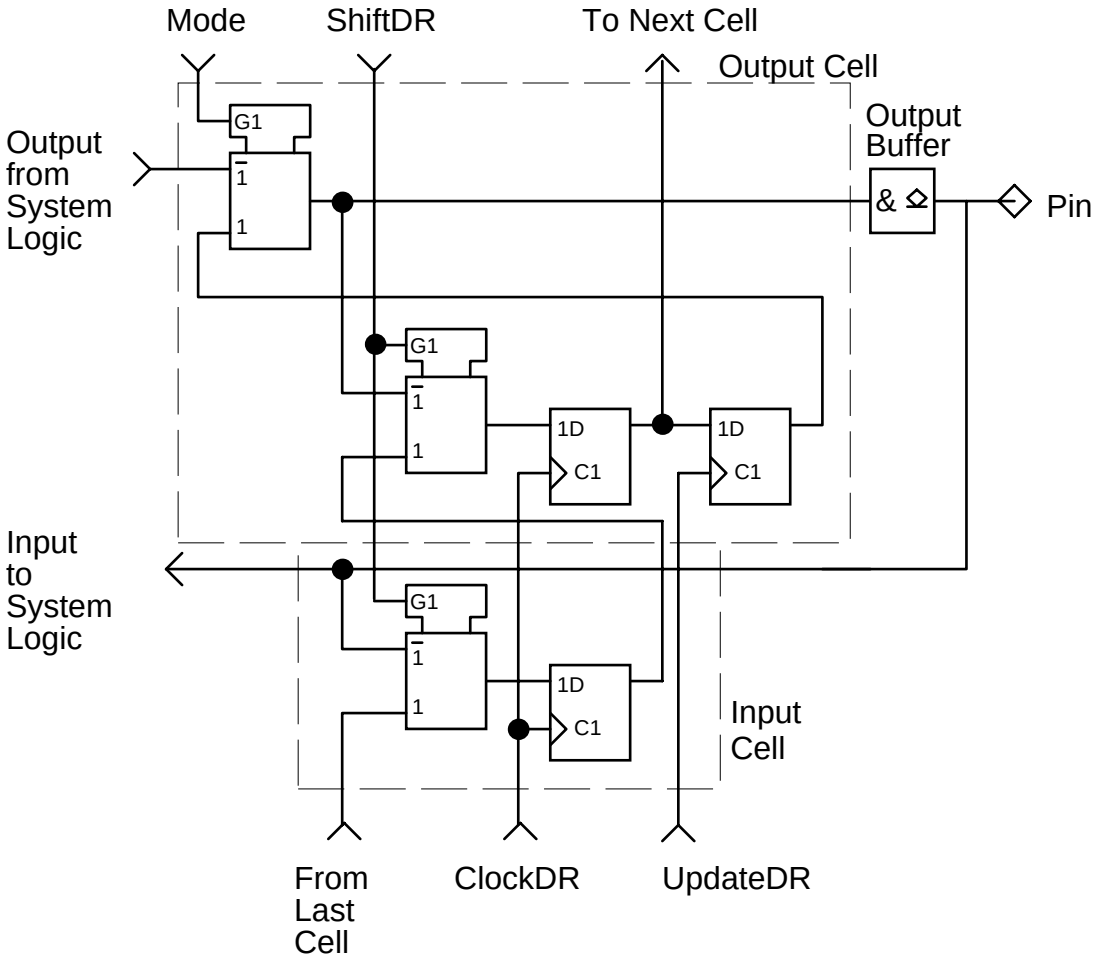


Figure 11-39—Boundary-scan register cells at an open-collector bidirectional pin [BC_4, input; BC_2, output] (see Table 11-7 for mode signal generation)

Note that in cases where a bidirectional pin is provided using an open-drain or open-collector output driver, the pin direction is defined as output for the one state that is actively driven and as input otherwise. Thus, a separate control cell is not required (i.e., a single bidirectional cell can provide for both data and control), as shown in Figure 11-40.

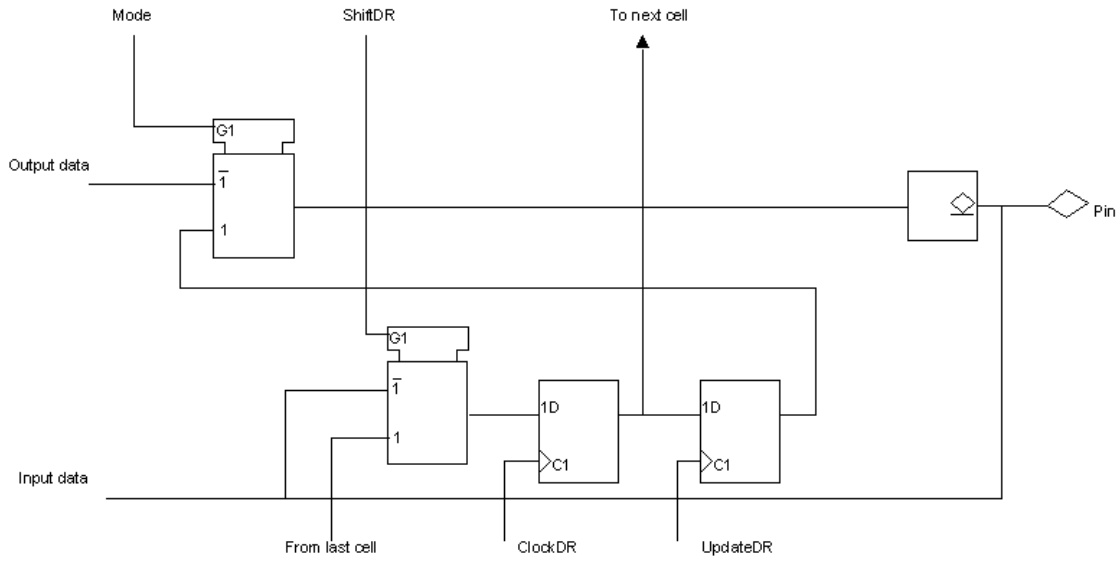


Figure 11-40—A boundary-scan register cell at an open-collector bidirectional pin where no control cell is required [BC_8] (see Table 11-12 for mode signal generation)

Table 11-12—Mode signal generation for the example cells in Figure 11-40 and Figure 11-41

Instruction	Mode
EXTEST	1
PRELOAD	0
SAMPLE	0
RUNBIST	X
CLAMP	1
HIGHZ	X

Figure 11-41 illustrates an alternative, reduced-complexity cell for use at a 3-state bidirectional pin. This cell is designed such that the signal present at the system pin is always captured. Because of this feature, this cell cannot be used where the *INTEST* instruction is to be provided. Where *INTEST* is to be supported, a cell of design similar to that shown in Figure 11-37 is required.

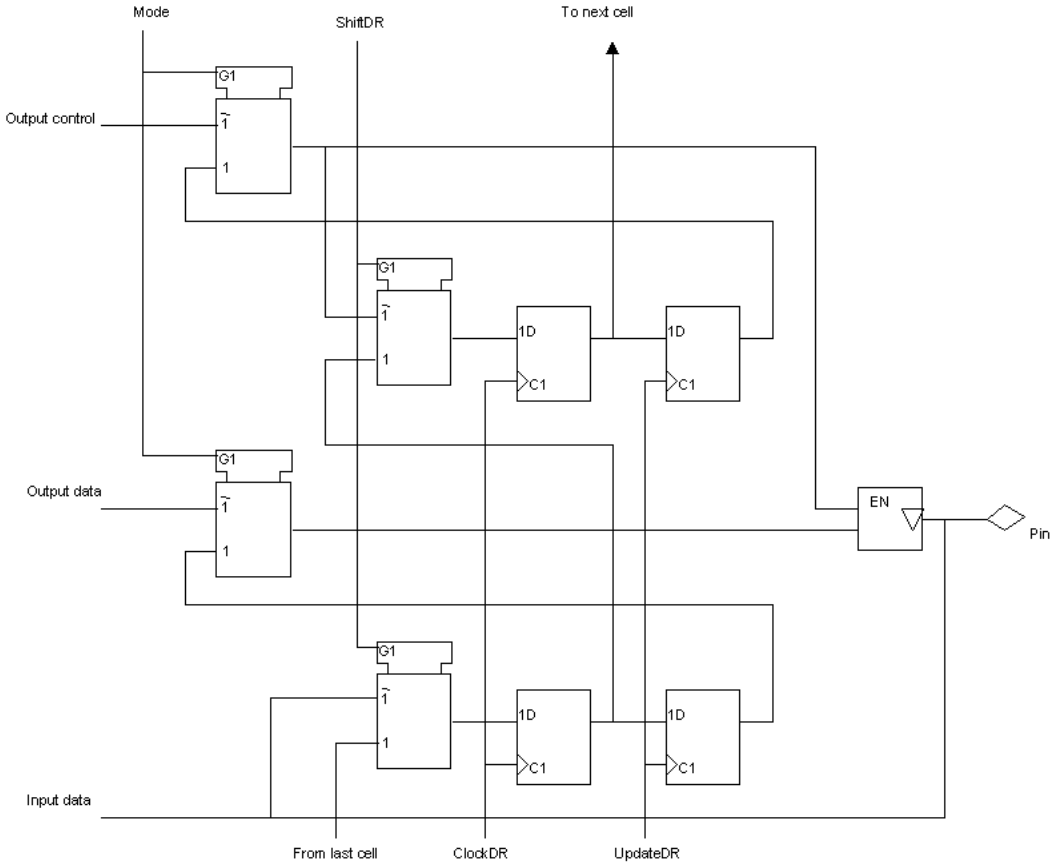


Figure 11-41—Boundary-scan register cells for use at a bidirectional pin where *INTEST* is not provided [BC_2, control; BC_8, data] (see Table 11-12 for mode signal generation)

11.8 Redundant cells

Redundant cells may exist in a component design for a number of reasons. For example:

a) They may be observe-only cells that observe a signal (input or output) that is observed by another boundary-scan register cell.
b) They may be parts of boundary-scan register cells designed for bidirectional system pins in cases where the pin has been programmed or otherwise customized to be permanently an input pin or an output pin. For example, a programmable component may be provided with three boundary-scan register cells at each system pin, sufficient to permit each system pin to be programmed as an input pin, 2-state or 3-state output pin, or bidirectional pin. After programming, certain of these cells may not be logically connected either to a given system pin or to a system logic input or output or both. Alternatively, a vendor of application-specific components may build a boundary-scan register into the basic component design (i.e., the design before the component is “committed”) that provides for a fully bidirectional signal at each possible system pin. When the basic component is “committed,” these cells will be constrained such that only the required functionality is connected.

11.8.1 Specifications

Rules

a) The contents of a redundant boundary-scan register cell shall not affect the behavior of any other part of the component.

Recommendations

b) The number of redundant boundary-scan register cells included in a component should be minimized.
c) Redundant cells should be designed such that the data shifted out through TDO after loading of the shift-register stage in the *Capture-DR* controller state is either a constant or the data just previously shifted into the cell.

11.8.2 Description

Some programmable components (e.g., programmable gate arrays or application-specific ICs) offer input/output circuits that can be programmed as input, output, 3-state, or bidirectional pins. To permit programming as a 3-state or bidirectional pin, two or more boundary-scan register cells would have to be included in each configurable cell to allow access to the data and control signals. However, when the cell is programmed as an input or 2-state output pin, only one cell will be required. In some implementations, the cells not associated with the programmed system function of a pin may be logically disconnected from the pin and from the system logic. Under such circumstances, the disconnected cells could no longer be used during testing and would become redundant. Rule 11.2.1 h) requires that the unused cells remain in the boundary-scan register so that the register has a fixed length regardless of how the component is programmed.

NOTE—In many programmable devices, programmed lack of logical connection(s) may occur only with regard to a boundary-scan register cell and the on-chip system logic. The cells provided for a particular programmable pin may remain logically connectable to that pin during testing, and the bidirectional control cell would then remain functional. The rules of this subclause do not prohibit this “excess” functionality at a pin. Indeed, interconnect test generation may actually be easier when all pins on a board-level net *appear* from *outside* the components to be provided with full bidirectional boundary-scan capability.

To minimize the number of redundant cells contained in the boundary-scan register of a component, the register should contain only cells that, in some programmed configuration of the component, can provide access to signals at the boundary of the on-chip system logic. For example, a cell that receives its parallel data input from the on-chip system logic and sends its parallel data output into the on-chip system logic should not be included in the boundary-scan register (e.g., as shown in Figure 11-42).

11.9 Special cases

11.9.1 Specifications

Permissions

a) In a case in which a system logic input pin is used *solely* as a source of control or *solely* as a source of data for a system output pin, a single cell may be provided that meets the rules of 11.5.1 (for the input pin) and 11.6.1 (for the output pin).

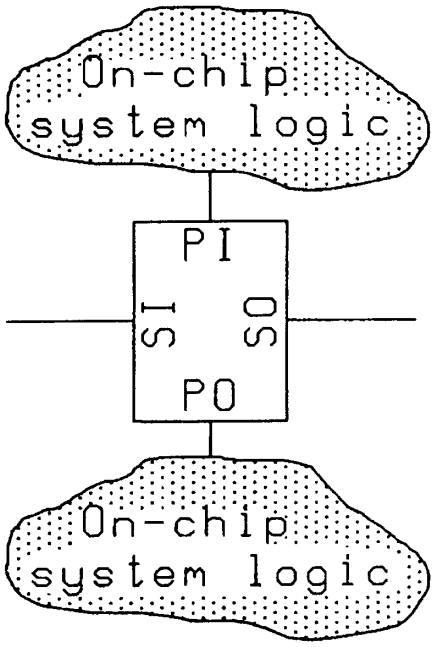


Figure 11-42—A cell that should not be included in the boundary-scan register

11.9.2 Description

Where the signal received at a system logic input pin is used solely to provide data or control for a system output pin, it is possible to use a single boundary-scan register cell to meet both sets of requirements. A common example of a situation where this might arise is one in which a system input pin is used solely to provide an output control signal for 3-state or bidirectional system pins. In such a case, either

a) Two separate boundary-scan register cells may be included, as shown in Figure 11-43; or
b) The functions of both cells may be combined into a single cell, as shown in Figure 11-44.

In the latter case, care should be taken in the design of the cell to ensure that it conforms to all the rules for the set of boundary-scan test instructions supported by the component.

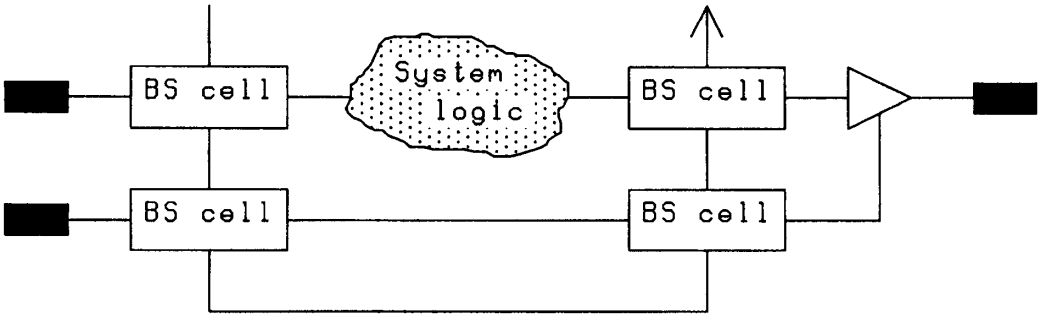


Figure 11-43—Input pins used only to control output pins—Case A

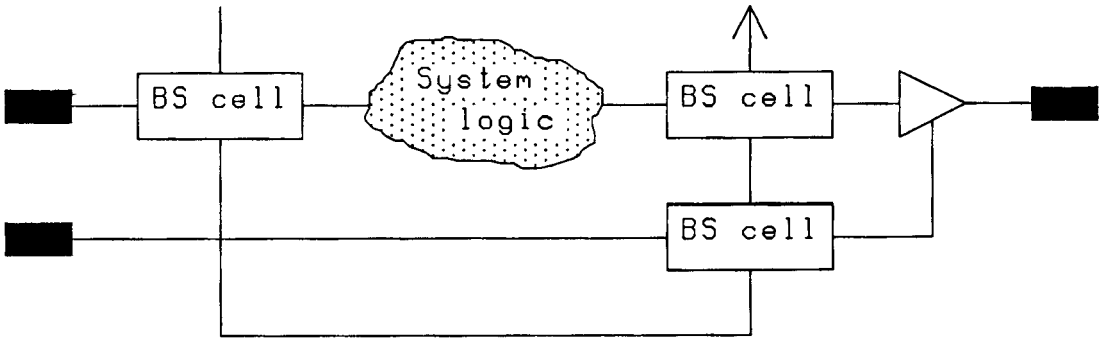


Figure 11-44—Input pins used only to control output pins—Case B

Note that the situation illustrated in Figure 11-45 violates the rules of this standard. In this case, the signal received from the system input pin is used both as an output control and as an input to the on-chip system logic.

In a case in which the signal from a system input pin is used only as a control for the 3-state output buffer and the option has been taken to provide a single boundary-scan register cell as shown in Figure 11-18, the top cell in Figure 11-34 has to be modified if Recommendation 8.9.1 f) is to be met. Specifically, the cell has to reload its own state in the *Capture-DR* controller state when the *INTEST* instruction is selected to avoid taking on a value dependent on off-chip circuitry. Figure 11-46 shows how this could be achieved. For this design, the mode signal should be controlled as shown in Table 11-6.

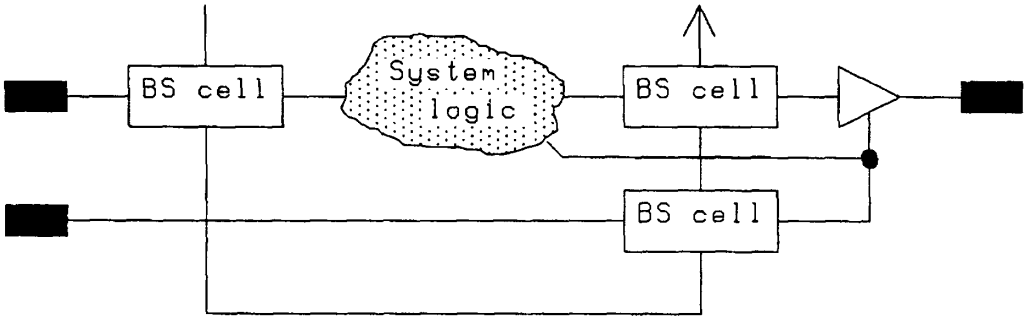


Figure 11-45—Illegal use of a single cell for output control and data

12. The device identification register

This clause defines the design and operation of the optional device identification register. If provided, this register allows the manufacturer, part number, and version of a component to be determined through the TAP. One application of the device identification register is to distinguish the manufacturer(s) of components on a board when multiple sourcing is used. As more components emerge that conform to this standard, it may become desirable to allow for a system diagnostic controller unit to blindly interrogate a board design in order to determine the type of each component in each location. The need to do this becomes more apparent if one considers systems that are configurable by the addition of option boards or by programming certain components, etc. This information is also available for factory process monitoring and failure mode analysis of assembled boards.

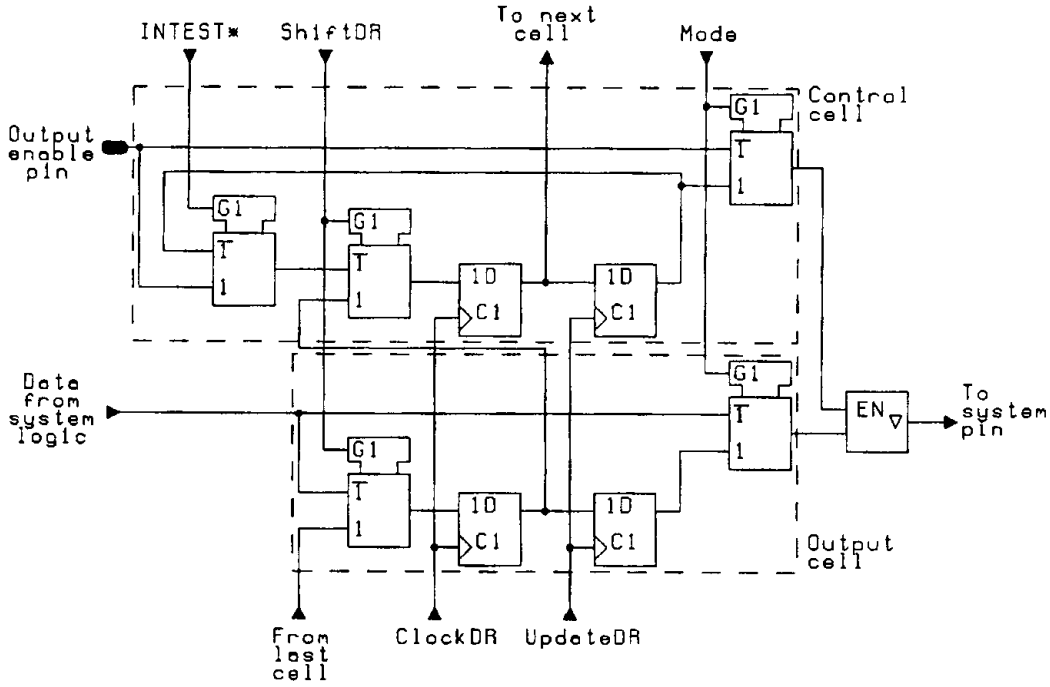


Figure 11-46—Boundary-scan register cells at a 3-state pin where output control is from a system pin [BC_5, control; BC_1, data] (see Table 11-6 for mode signal generation)

NOTE—The design requirements contained in this clause apply only when the optional device identification register is included in a component.

12.1 Design and operation of the device identification register

12.1.1 Specifications

Rules

a) The device identification register shall be a shift-register-based path that has a parallel input but no parallel output.
b) The circuitry used to implement shift-register stages in the device identification register shall not be used to perform any system function (i.e., it shall be a dedicated part of the test logic).
c) On the rising edge of TCK in the *Capture-DR* controller state, the device identification register shall be set such that subsequent shifting causes an identification code to be presented in serial form at TDO.
d) The component shall contain a vendor-defined identification code, containing four fields (see Figure 12-1), which is accessed when the *IDCODE* instruction is entered.
e) For user-programmable components where programming cannot be completely determined by use of public instructions (see 8.2), the ability shall be provided to permit the user to program a supplementary 32-bit identification code that will be loaded into the device identification register in response to the *USERCODE* instruction.
f) The operation of the device identification register shall have no effect on the operation of the on-chip system logic.

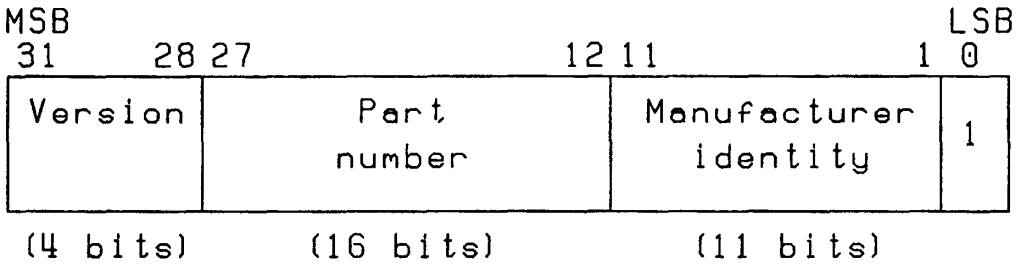


Figure 12-1—Structure of the device identification register

12.1.2 Description

Figure 12-2 shows a design for a device identification register cell that satisfies these requirements.

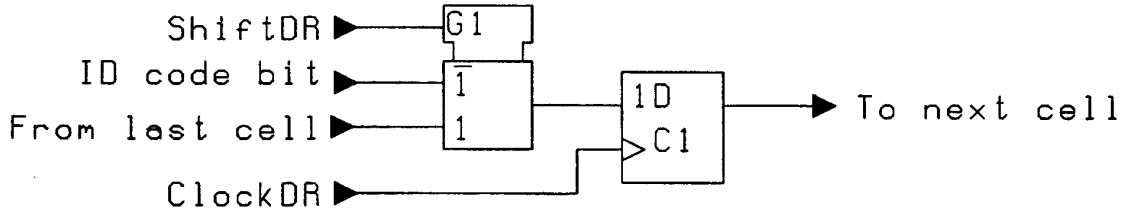


Figure 12-2—Device identification register cell design

The identification code loaded into the device identification register in response to the *IDCODE* instruction allows the manufacturer, part number, and variant for the component to be read in a serial binary form. In situations where blind interrogation of a product is necessary, this information allows the structure of the board to be determined along with, by reference to stored data, the instruction set and other details for each component. (It is assumed that the components in a product will be selected from a limited set.)

Examination of the identification code also allows the structure of the boundary-scan register to be deduced, including the positioning of cells at input and output pins and the location of cells that control 3-state or bidirectional pins. This information is valuable in ensuring that contention between drivers at the board level is avoided (for example, as discussed in 11.6).

For programmable components, however, the configuration of pins as inputs, outputs, etc., may be determined by programming rather than by the basic design of the component. In such cases, therefore, a supplementary identification code is required to indicate how the component has been programmed. This supplementary code shall be user-programmable and accessed through the device identification register in response to the *USERCODE* instruction.

NOTE—The supplementary identification code is required only in cases where the component cannot be reprogrammed through the test logic defined by this standard. In cases where such reprogramming is possible, the ATE or master device controlling the operation of the component can ensure that it is programmed to the correct state at the start of the test sequence.

Since the bypass register (which is selected in the absence of a device identification register by the instruction loaded in the *Test-Logic-Reset* controller state) loads a logic 0 at the start of a scan cycle, whereas a device identification register will load a constant logic 1 into its LSB, examination of the first bit of data

shifted out of a component during a test data scan sequence immediately after exit from the *Test-Logic-Reset* controller state will show whether a device identification register is included in the design.

A requirement of the *IDCODE* and *USERCODE* instructions is that when they are used, the on-chip system logic shall continue its normal operation undisturbed. Rule 12.1.1 b) is included so that this requirement can be met. Note, however, provided that Rule 12.1.1 f) is met, the shift-register stages may be shared resources used by several of the registers defined by this standard and also by any design-specific test data register.

12.2 Manufacturer identity code

12.2.1 Specifications

Rules

a) The 11-bit manufacturer identity code shall be a compressed form of the code specified by EIA/JEP106[3] generated as follows:
   1) Manufacturer identity code bits 7-1. The seven LSBs are derived from the last byte of the EIA/JEP106 code by discarding the parity bit.
   2) Manufacturer identity code bits 11-8. The four MSBs provide a binary count of the number of bytes in the EIA/JEP106 code that contain the continuation character (hex 7F). Where the number of continuation characters exceeds 15, these four bits contain the modulo-16 count of the number of continuation characters.

b) The manufacturer code 00001111111 shall not be used in components that are otherwise compatible with this standard.

Recommendations

c) Where the component is an application-specific integrated circuit (ASIC), the manufacturer identity code should be that of the manufacturer of the component rather than that of the designer.

12.2.2 Description

This scheme utilizes a listing of manufacturer identification codes specified by EIA/JEP106 as administered by Electronic Industries Association/Joint Electron Device Council (EIA/JEDEC).

The EIA/JEP106 code is formed from a variable number of eight-bit bytes. Each byte contains seven data bits and an odd parity bit (the MSB). Bytes other than the last contain continuation characters (hex 7F), while the last contains 127 different codes that, together with a knowledge of the number of preceding continuation code bytes, allow the manufacturer's identity to be determined.

The compressed form of the EIA/JEP106 code used within the device identification register limits the number of bits needed in the device identification register to contain the manufacturer identity code and allows the length of the code to be standardized. The length of the compressed code is fixed at 11 bits (see 12.1), which allows for 2032 different manufacturer codes. (Note that 16 codes are unused since these codes correspond to the hex 7F code in the seven LSBs—the EIA/JEP106 continuation character.)

One of the unused codes (00001111111) should be treated as illegal for components compatible with this standard. By shifting a dummy device identification code containing this manufacturer identity code from the bus master (ATE, board-level controller, etc.) into the board-level serial path set up by moving directly from the *Test-Logic-Reset* controller state into scanning of the test data registers, it is possible to detect the end of the identity code sequence.

[3]Information on references can be found in Clause 2.

When test data register scanning is entered in this way, the serial path at the board level includes

a) The device identification registers of components that provide them; and
b) The bypass registers of components that do not include a device identification register.

As discussed in 12.1, the fact that identification codes begin with a logic 1 whereas the bypass registers load a logic 0 allows the identification codes in the serial stream read out of the board to be detected. By feeding in the dummy identification code at the board's serial input and checking the serial output for the invalid manufacturer identity code 00001111111, it is possible to locate the end of the identification code sequence for a board containing an unknown number of components.

12.3 Part-number code

12.3.1 Specifications

Rules

a) The part-number code shall consist of 16 bits.
b) The manufacturer shall ensure that no two component types that are offered in the same package with the TAP pins in the same location have the same part-number code.

12.3.2 Description

The part-number code may be used to verify the type of the component inserted in a particular location on an assembled product. The use of a 16-bit value for this code gives an acceptably low chance that an incorrect component inserted in the location will return a correct part-number code.

Part-number codes could, for example, be generated from the textual part-number code by using a data compaction scheme.

12.4 Version code

12.4.1 Specifications

Rules

a) The version code shall consist of 4 bits.

Recommendations

b) The value of the version code for a component should be assigned to identify the variant of a component type.

13. Conformance and documentation requirements

13.1 Claiming conformance to this standard

The level of conformance to this standard can vary according to the range of test operations supported.

13.1.1 Specifications

Rules

a) Components that claim conformance to this standard shall comply with all relevant rules in the Specifications subclauses of this standard.

NOTE—Components that were designed before publication of this standard and conform fully with the requirements of this standard except in the control of the TDO output driver with regard to the controller states in which it is active (see the note that follows Table 6-2) also may claim conformance to this standard.

b) When it is claimed that a component conforms to this standard, the claim shall clearly identify the subset of the public instructions defined in this standard that is supported, as listed in Table 13-1 and defined in 8.2.

Table 13-1—Public instructions

Instruction	Status
BYPASS	Mandatory
CLAMP	Optional
EXTEST	Mandatory
HIGHZ	Optional
IDCODE	Optional
INTEST	Optional
PRELOAD	Mandatory
RUNBIST	Optional
SAMPLE	Mandatory
USERCODE	Optional

Recommendations

c) It is recommended that components support either the *INTEST* or the *RUNBIST* instruction or both.

Permissions

d) ASIC vendors may claim conformance to this standard by illustrating an interconnection of cells that, if built, would produce a component that meets the requirements of this standard.

13.1.2 Description

The minimum requirement for conformance to this standard is set to ensure that the user of an integrated circuit can perform two basic functions using the test logic: examine the operation of a prototype system and test assembled products for assembly-induced defects during manufacturing.

To enable efficient and comprehensive verification of internal component operation at the board and system level, it is strongly recommended that either the *INTEST* or *RUNBIST* instruction or both be supported.

13.2 Prime and second source components

13.2.1 Specifications

Rules

a) With the sole exception of the device identification code, the publicly accessible test logic for second source components shall operate in the same manner as that for the prime source component in response to all public instructions.

13.2.2 Description

It is essential that both the system and the test logic of prime and second source components operate in the same manner in the component purchaser's environment. This ensures that test programs created for a printed circuit board containing multiply sourced components produce consistent results regardless of the source of individual components.

The only exceptions to this requirement are the optional device identification register and any test logic that is accessed only in response to private instructions. In the former case, the identification code shall vary to identify the source of the particular component, its part number, and its revision (see Clause 12). In the latter case, test logic that is not publicly accessible is not intended for use other than by the component vendor; therefore, this test logic should not be operated by a board-level test program.

13.3 Documentation requirements

13.3.1 Specifications

Rules

a) For any component that claims conformance to this standard, the operation of all test logic accessed in response to public instructions shall be fully documented.
b) The following information, required by the component purchaser for use in test development and other activities, shall be supplied by the component manufacturer:
    1) *Instruction register.* The following information pertaining to the instruction register is required:
        i) Its length.
        ii) The pattern of fixed values loaded into the register during the *Capture-IR* controller state.
        iii) The significance of each design-specific data bit presented at a parallel input, where provided.
    2) *Instructions.* For each public instruction offered by a component, the following information is required:
        i) The binary code(s) for the instruction.
        ii) A list of test data registers placed in a test mode of operation by the instruction.
        iii) The name of the serial test data register path enabled to shift data by the instruction.
        iv) A definition of any data values that shall be written into test data registers before selection of the instruction and the order in which these values shall be loaded.
        v) The effect of the instruction. Any system pins whose drivers become inactive as a result of loading the instruction should be clearly identified.
        vi) A definition of the test data registers that will hold the result of applying a test and of how they are to be examined.
        vii) A description of the method of performing the test and of how data inputs and their corresponding data outputs are to be computed.

If private instructions are utilized in a component, the vendor shall clearly identify any instruction binary codes that, if selected, would cause hazardous operation of the component.

3) *Self-test operation*. For each instruction that causes operation of a self-test function, the following information is required in addition to that listed under Rule 13.3.1 b) 2):
   i) The minimum duration (e.g., a number of cycles of TCK) required to ensure completion of the test.
   ii) A definition of the test data registers whose states are altered during execution of the test.
   iii) A definition of the results of executing the self-test on a fault-free component.
   iv) An estimate of the percentage (e.g., to the nearest 5%) of the single stuck-at faults in the component's circuitry that will be detected by the self-test function *or* a description of the operation of the self-test function and the circuitry exercised.
4) *Test data registers*. For each test data register available for public use and access in a component, the following information is required:
   i) The name of the register used for reference in other parts of the data sheet.
   ii) The purpose of the register.
   iii) The length.
   iv) A full description of the operating modes of the register.
   v) The result of setting each bit at the parallel output of the register.
   vi) The significance of each bit loaded from the parallel input of the register.
5) *Boundary-scan register*. The following information is required in addition to that listed under Rule 13.3.1 b) 4):
   i) The correspondence between boundary-scan register bits and system pins, system direction controls, or system output enables.
   ii) Whether each pin is an input, a 2-state output, a 3-state output, or a bidirectional pin.
   iii) For each boundary-scan register cell at an input pin, whether the cell can apply tests to the on-chip system logic.
   iv) For each boundary-scan register cell associated with an output or direction control signal, a list of the pins controlled by the cell and the value that shall be loaded into the cell to place the driver at each pin in an inactive state or will be observed using the *SAMPLE*, *PRELOAD*, or *INTEST* instructions when the on-chip system logic causes the driver to be inactive.
   v) The method by which single-step operation is to be achieved while the *INTEST* instruction is selected if this instruction is supported.
   vi) The method of providing clocks to the on-chip system logic while the *RUNBIST* instruction is selected if this instruction is supported.
   vii) For each redundant cell, whether the cell returns either the value shifted in or a constant after loading of the cell in the *Capture-DR* controller state.
6) *Device-identification register*. Where a device identification register is included in a component, the following information is required in addition to that listed under Rule 13.3.1 b) 4):
   i) The value of the manufacturer’s identification code.
   ii) The value of the part-number code.
   iii) The value of the version code.
   iv) The method of programming the value of the supplementary identification code, where required.
7) *Performance*. The performance of the test logic should be fully defined, including the following information:
   i) The maximum acceptable TCK clock frequency.
   ii) A full set of timing parameters for the test logic.
   iii) The logic switching thresholds for TAP input and output pins.
   iv) The load presented by the TCK, TMS, TDI, and TRST* pins.
   v) The drive capability of the TDO output pin.
   vi) The extent to which the TDO driver may be overdriven when active (e.g., using an in-circuit test system).

vii) Whether TCK may be stopped in the logic 1 state.

8) *Compliance enable inputs.* If a component has compliance-enable inputs as defined in 4.8.1, the following documentation shall be provided:
   i) A complete list of these inputs labeled as compliance-enable inputs.
   ii) A complete list of those logic patterns that, when applied at the compliance-enable inputs, will enable compliance to this standard.
   iii) A clear indication of any patterns that, if applied to the compliance-enable inputs, would cause hazardous operation of the component.

c) A BSDL description of the component shall be supplied by the component manufacturer (see Annex B).

13.3.2 Description

Figure 13-1 and Figure 13-2 show how set-up and hold timing parameters and propagation delays should be measured relative to the test clock TCK and a reference voltage V_{ref}. Note that such timing parameters are required for TMS, TDI, and TDO and also for system pins that can be driven from the test logic (e.g., the system data input set-up time for the boundary-scan register before the rising edge of TCK in the *Capture-DR* controller state).

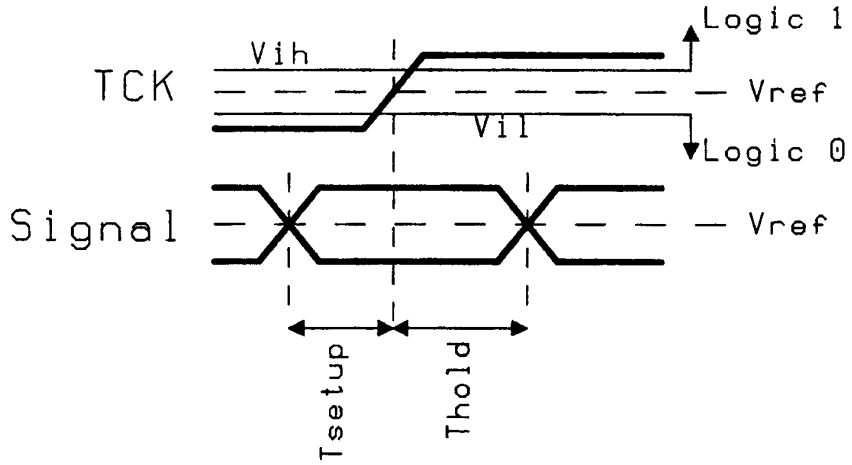


Figure 13-1—Measuring set-up and hold timing

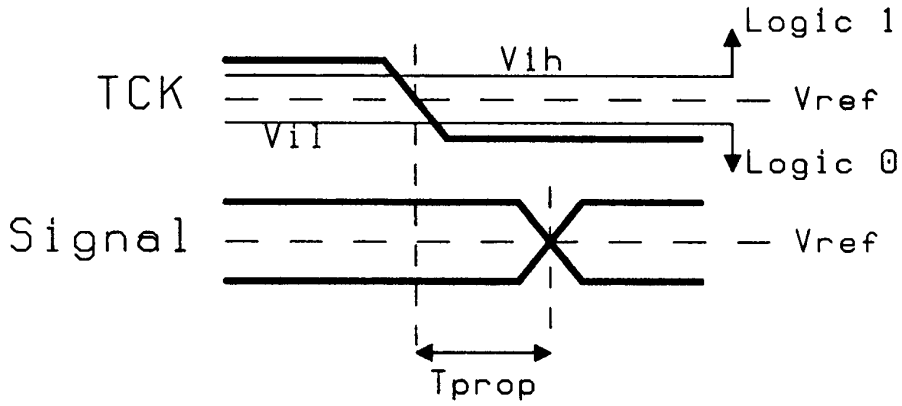


Figure 13-2—Measuring propagation delays

Annex A

(informative)

An example implementation using level-sensitive design techniques

To illustrate how a circuit might be constructed to meet the requirements of this standard, example designs are included in the standard. These examples form a consistent set and could be used as the basis for an implementation. However, it is important to emphasize that *the designs contained in this standard are neither mandatory nor recommended in preference to any other implementation*. Many other implementations are possible. For example, this annex illustrates one of many possible implementations of the test logic that could be based on Level-Sensitive Scan Design (LSSD) techniques. This implementation has two modes of operation:

a) A “chip-on-board” mode where the component responds to the signals received at the TAP inputs in the manner required by this standard; and
b) A “stand-alone” mode that allows the entire component (including both the on-chip system logic and the test logic) to be tested using LSSD techniques, for example, as a part of a postproduction test.

The stand-alone mode extends the functionality of the component beyond that required by this standard while maintaining compatibility with this standard for chip-on-board testing.

A.1 Top-level test logic design

The design for the test logic is shown in Figure A.1. The following design features should be noted:

a) All stored-state devices are constructed from level-sensitive latches. Shift-register stages contained in the test logic require two such latches controlled from a pair of nonoverlapping clocks. The clock generator circuit shown in Figure A.2 generates these clocks, as shown in Figure A.3 and Figure A.4.[4]
b) An internal scan path is provided that visits all shift-register latches in the design, including those in the test logic. The internal scan path is shown as a bold line in Figure A.1.
c) For chip-on-board operation, the LSSD clocks LSSD_A, LSSD_B, and LSSD_P are held at 0, while LSSD_C1 and LSSD_C2 are held at 1. This allows the test logic to operate in response to signals received at the TMS, TDI, and TCK inputs, as defined in this standard.
d) For stand-alone component testing using the internal scan path, TCK and the clocks for the on-chip system logic are operated in concert with the LSSD clocks (LSSD_A, LSSD_B, LSSD_P, LSSD_C1, and LSSD_C2). To ensure that LSSD testing of the test logic is correctly synchronized to that of the on-chip system logic, the signals LSSD_C1 and LSSD_C2 are used. These signals are controlled in concert with the remaining LSSD clocks that enable shifting along the scan path. In this mode, the TCK input to the clock generator is used as a control signal that can enable or disable the signals supplied to LSSD_C1 and LSSD_C2. For example, to permit a positive-going pulse on LSSD_C1 to propagate through to C1, a logic 1 must first be applied at TCK. Similarly, positive-going pulses at LSSD_C2 are allowed through to C2 if TCK is first set to 0. Figure A.4 shows the expected relationships between the various clock signals during LSSD stand-alone testing.

[4]The clocks LSSD_C1 and LSSD_C2 are dedicated test clocks that are used only during stand-alone LSSD testing of the component. They allow logic driven from single-phase clock inputs such as TCK to be controlled completely in accordance with level-sensitive design principles during component testing.

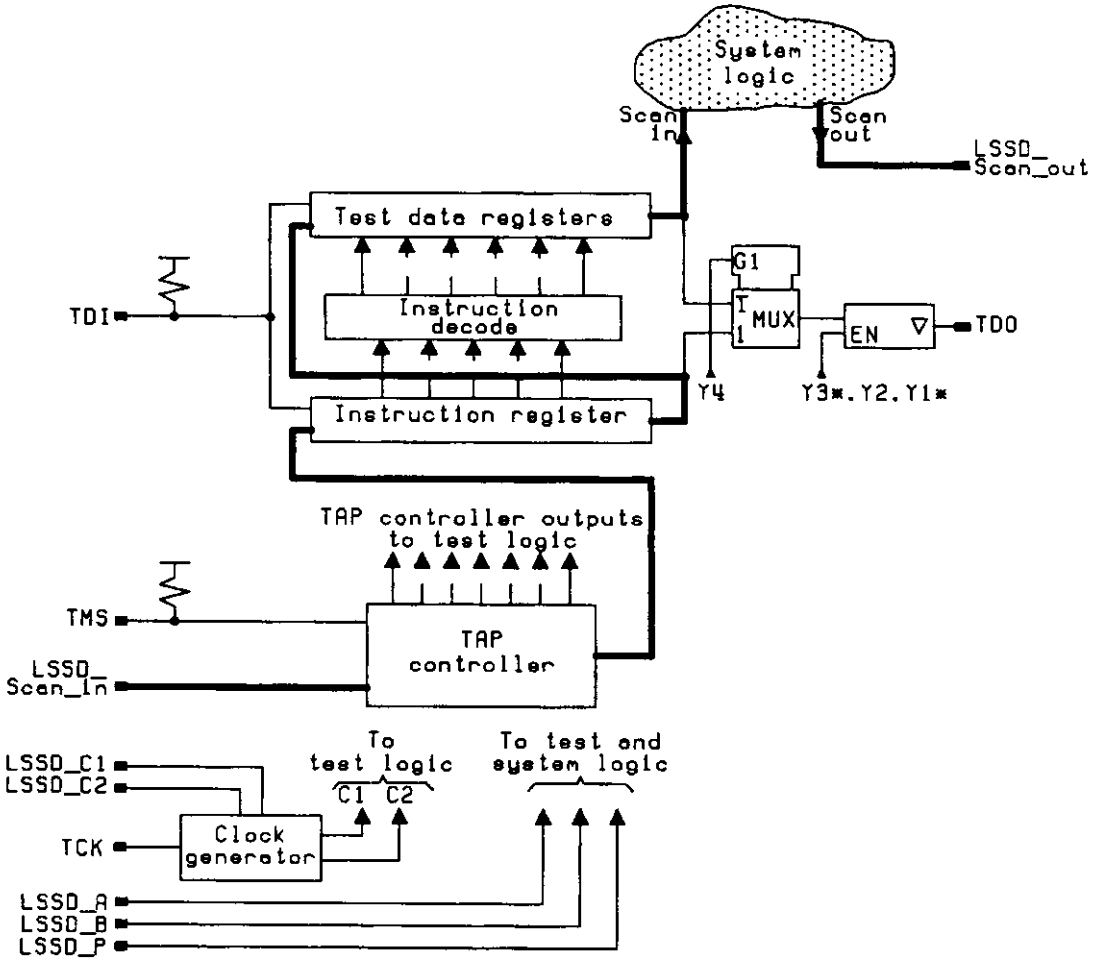


Figure A.1—Test logic schematic

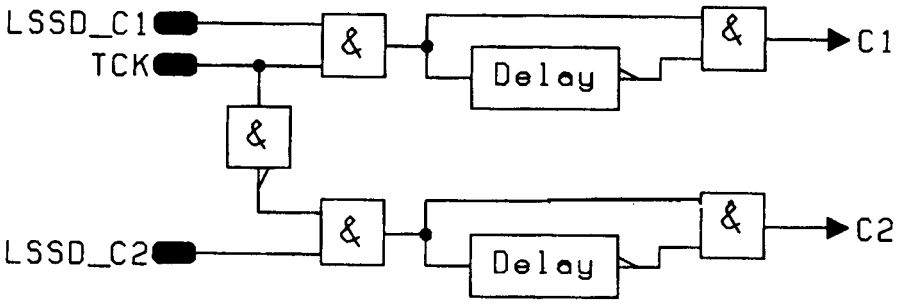


Figure A.2—Generation of nonoverlapping clocks from TCK

e) Since the serial outputs of the instruction and test data registers change state on the falling edge of TCK due to the master-slave operation of the latches that form the shift-register stages, it is not necessary to retime the output fed to TDO.

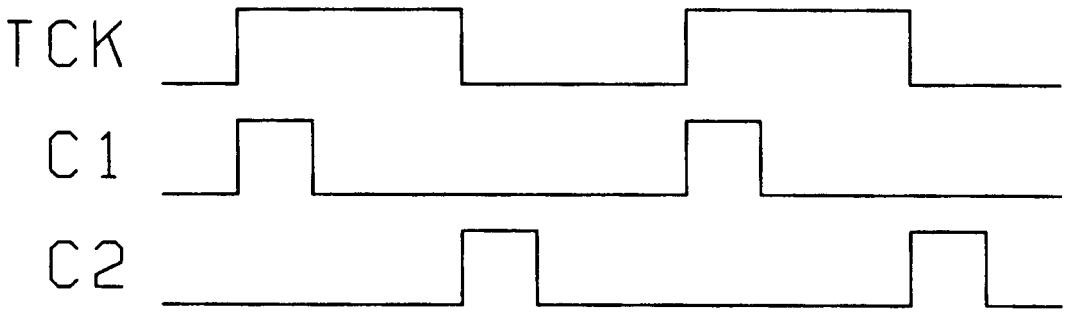


Figure A.3—Operation of the clock generator

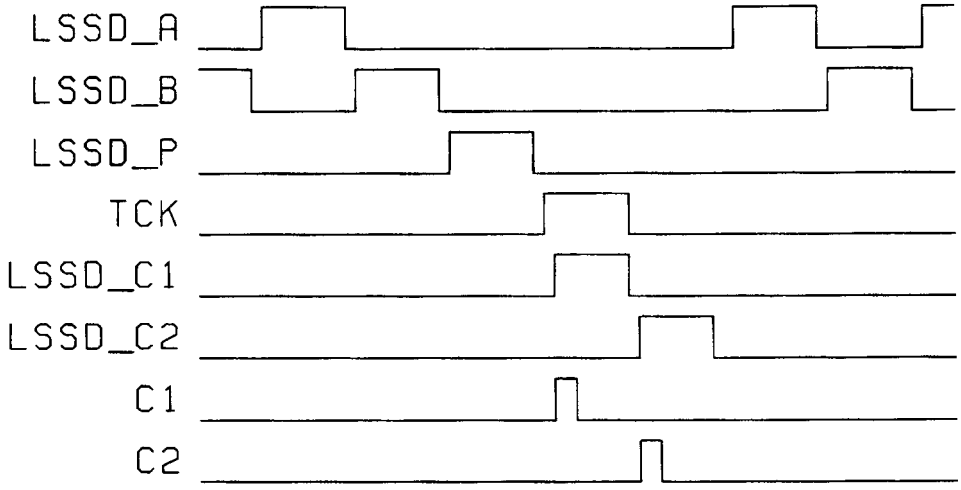


Figure A.4—Control of clocks for "stand-alone" component testing

Note that the capture of input signals to the test logic (e.g., those received at TMS, at TDI, and at system input pins) occurs on the falling edge of C1, which itself occurs at a fixed delay after receipt of a rising edge at TCK. Provided that the width of pulse C1 is independent of the frequency of TCK, the requirements of this standard will be met (see 4.2.2).

A.2 Latch designs

Figure A.5 gives NAND gate equivalent circuits for the latches used in the remainder of the schematics in this annex.

A.3 TAP controller implementation

Figure A.6 and Figure A.7 show the implementation of the TAP controller. Note that for this example, the output decoding logic is defined for each register in the following clauses of this annex. The assignments of logic states to controller states are as for the previous implementation (see Table 6-3).

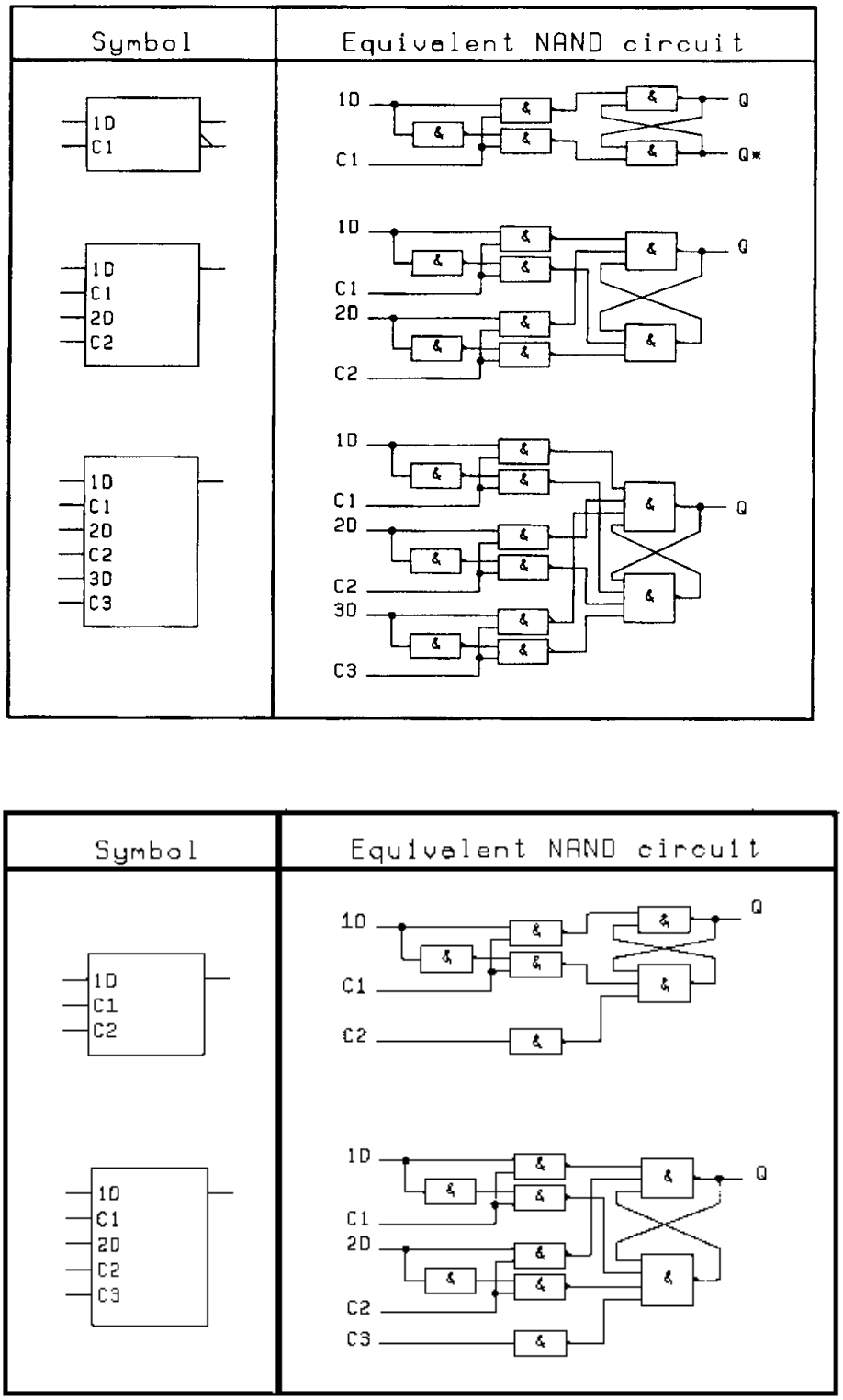


Figure A.5—Schematics for level-sensitive latches

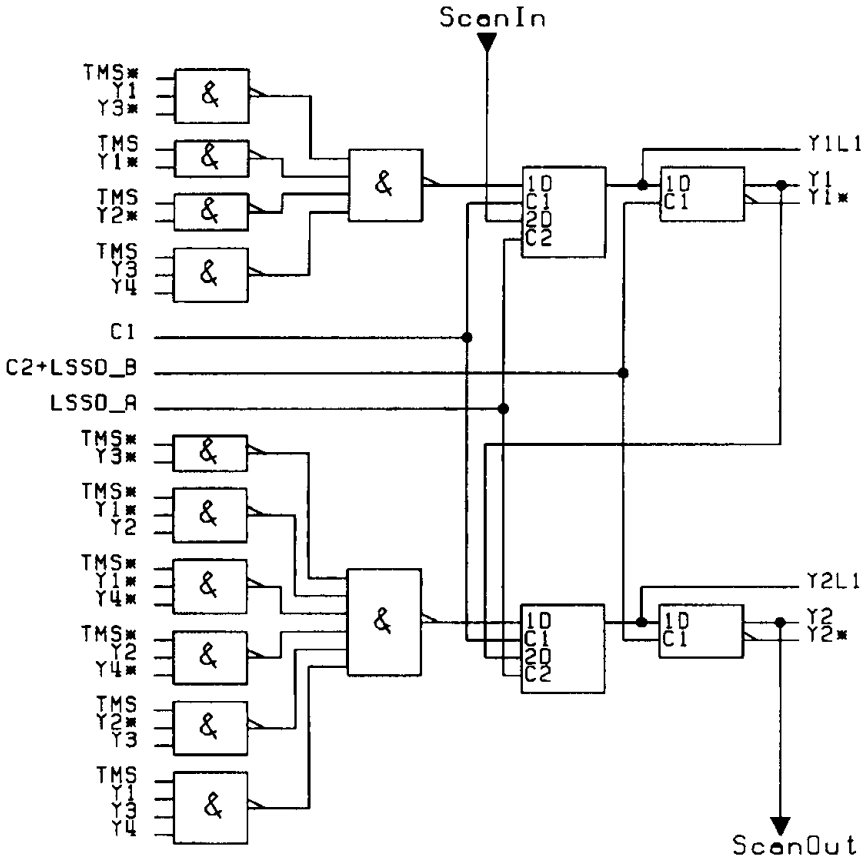


Figure A.6—TAP controller—A

Note the inclusion of the scan test path through the controller latches, which allows the controller to be fully tested as a part of a scan test on the complete integrated circuit.

A.4 Instruction register implementation

To allow for the scan path input to the instruction register, the design of the cell nearest to TDI will differ from that of other cells in the register. Figure A.8 shows a design for the cell nearest to TDI, while Figure A.9 shows a design for other cells.

The control signals for these cell designs are generated from the TAP controller outputs and the various clock signals as follows:

```
CaptureClockIR = C1·Y4·Y3·Y2·Y1*
ShiftClockIR = C1·Y4·Y3*·Y2·Y1*
SetClockIR = C2·Y4L1·Y3L1·Y2L1·Y1L1
L2ClockIR = C2 + LSSD_B
PClockIR = C2·Y4L1·Y3L1·Y2L1*·Y1L1 + LSSD_P
```

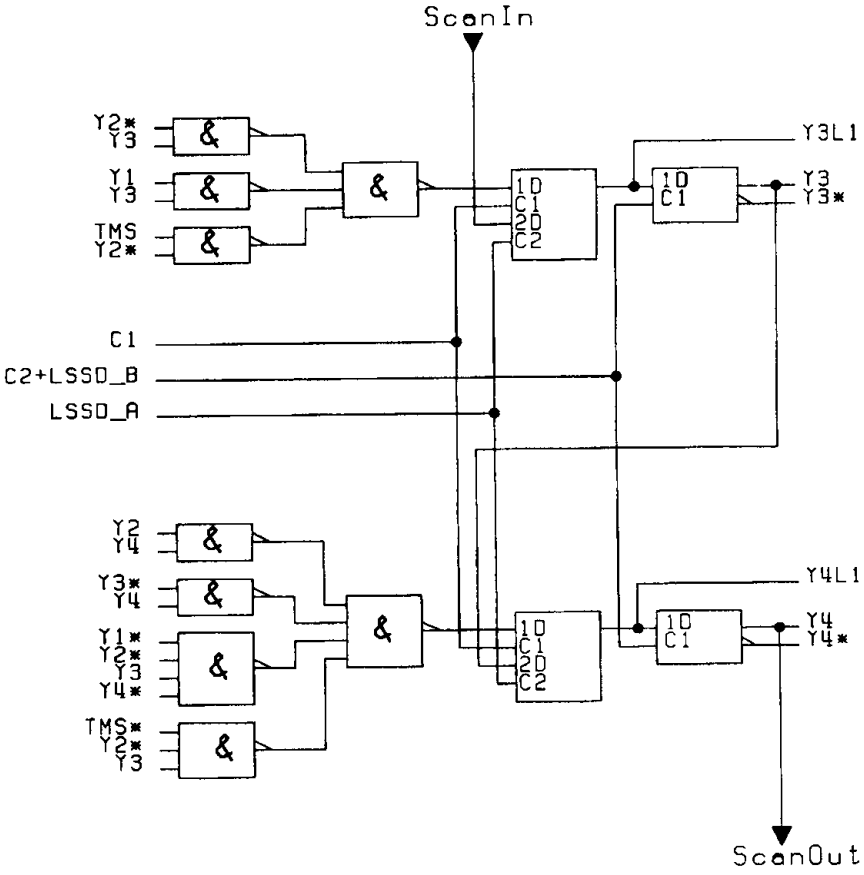



Figure A.7—TAP controller—B

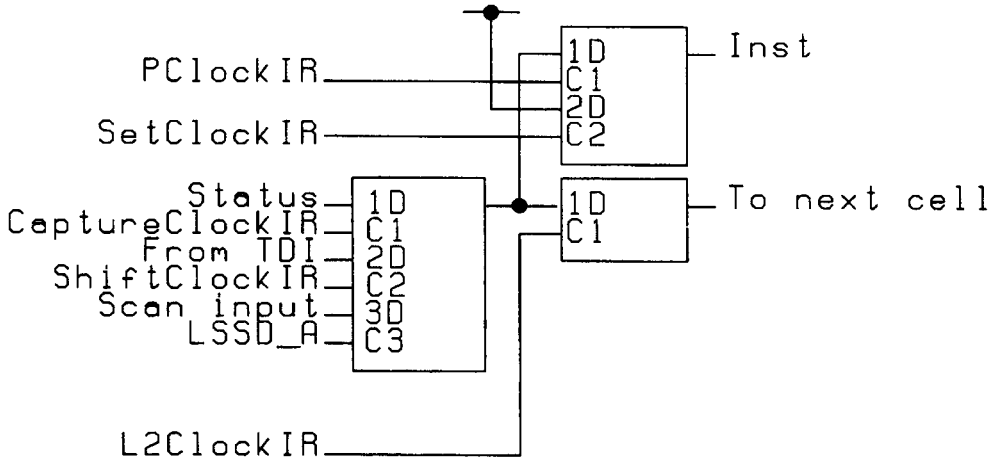


Figure A.8—Instruction register cell nearest to TDI

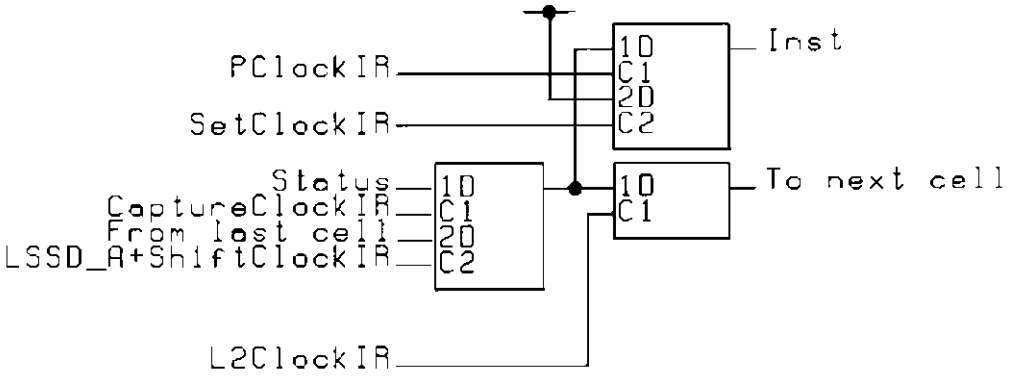


Figure A.9—Other instruction register cells

A.5 Bypass register implementation

The bypass register can be implemented as shown in Figure A.10.

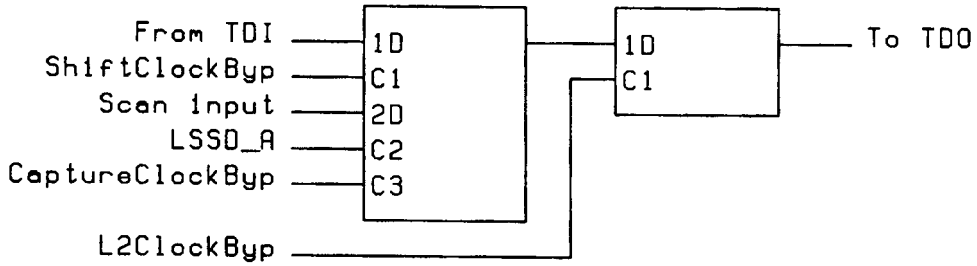


Figure A.10—Bypass register

Control signals for this implementation are generated as follows:

```
CaptureClockByp = C1·Y4*·Y3·Y2·Y1*·BYPASS
ShiftClockByp = C1·Y4*·Y3*·Y2·Y1*·BYPASS
L2ClockByp = C2 + LSSD_B
```

Note that the variable BYPASS is true when the *BYPASS* instruction is present at the instruction register outputs.

A.6 Boundary-scan register implementation

A set of boundary-scan register cell designs is included in the following figures. The control signals required by these cells are as follows:

```
CaptureClockBS = C1·Y4*·Y3·Y2·Y1*·BST
ShiftAClockBS = C1·Y4*·Y3*·Y2·Y1*·BST + LSSD_A
```



```
L2ClockBS = C2 + LSSD_B
PClockBS = C2·Y4L1*·Y3L1·Y2L1*·Y1L1·BST + LSSD_P
ResetClockBS = C2·Y4L1·Y3L1·Y2L1·Y1L1
```

The variable BST is true whenever a boundary-scan test instruction is present at the instruction register's outputs. The signals labeled DriveOut and LogicTest are controlled as follows:

a) DriveOut is true when the *EXTEST*, *INTEST*, or *RUNBIST* instruction is selected.
b) LogicTest is true when the *INTEST* or *RUNBIST* instruction is selected.

Note that the boundary-scan register nearest to TDI should have its first latch modified to accept either the signal received from TDI (when C1·Y4*·Y3*·Y2·Y1*·BST is true) or the signal at the scan input to the register (when LSSD_A is true).

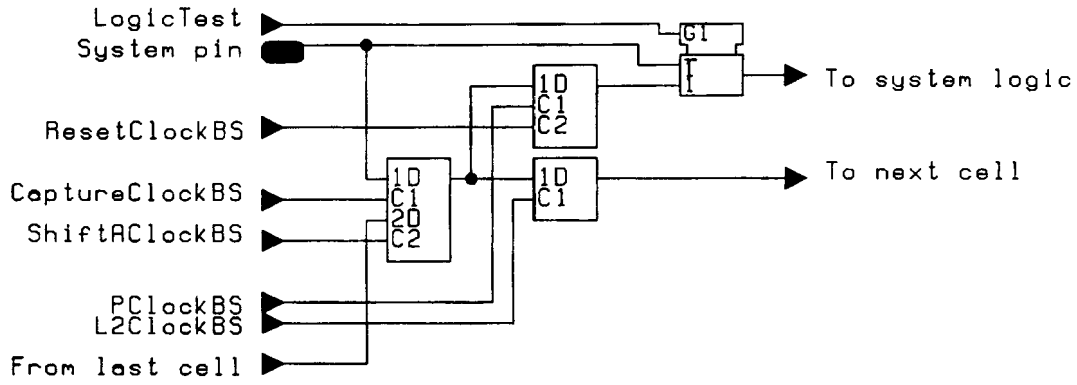


Figure A.11—A level-sensitive input cell design

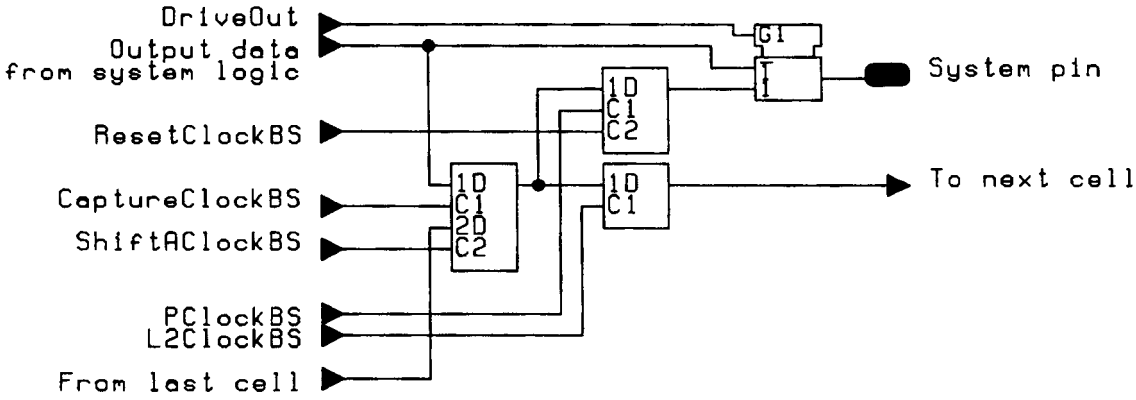


Figure A.12—A level-sensitive output cell design

The parallel output latches in the control cells in Figure A.13 and Figure A.14 are reset in the *Test-Logic-Reset* controller state as allowed by Permission 11.3.1 h) and Rule 11.6.1 p). A similar reset capability is provided in Figure A.11 and Figure A.12.

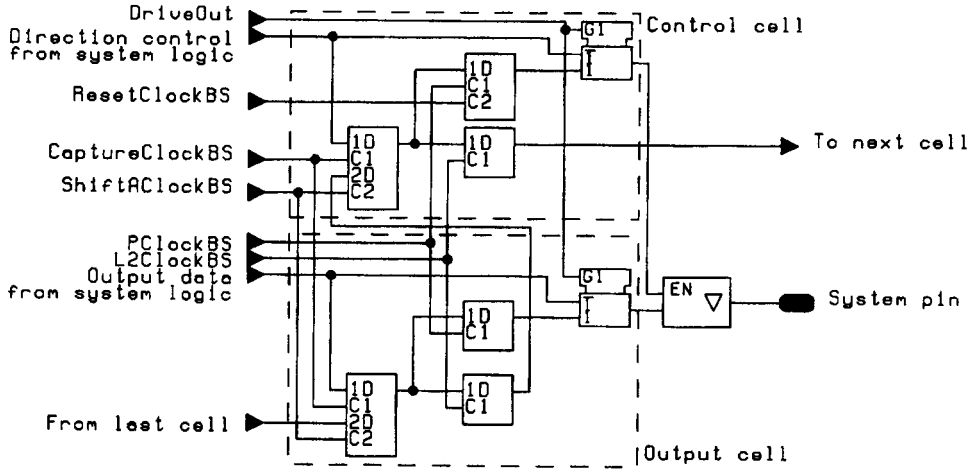


Figure A.13—Level-sensitive cells at a 3-state output

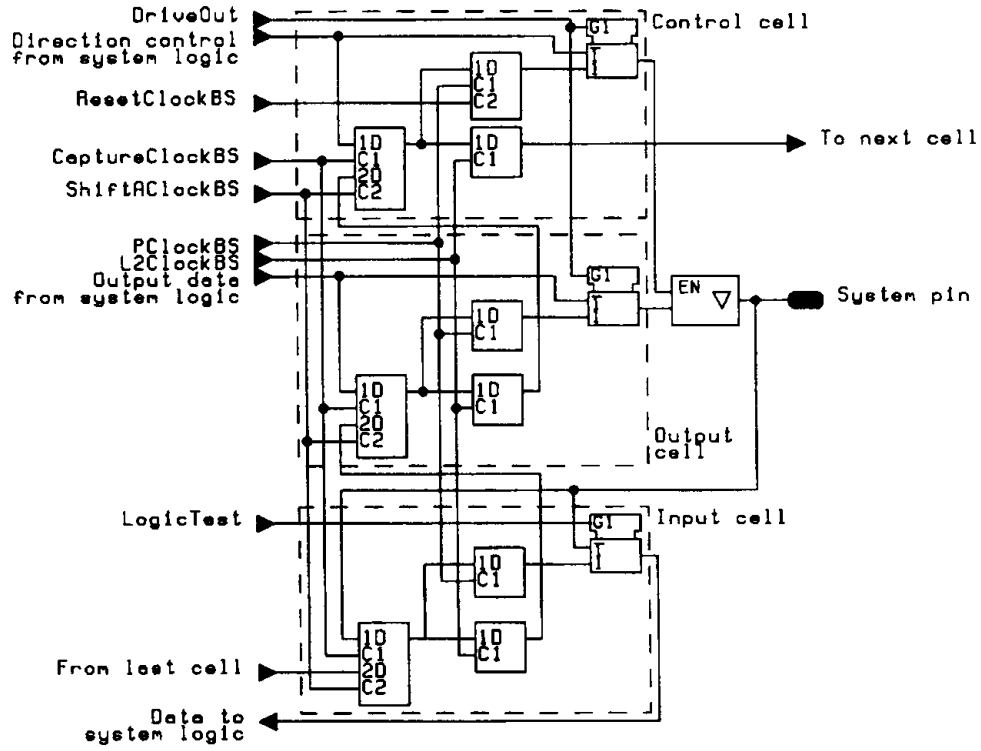


Figure A.14—Level-sensitive cells at a bidirectional pin

Annex B

(normative)

Boundary-scan description language

This annex defines a machine-readable language that allows rigorous description of testability features in components that comply with IEEE Std 1149.1. The language is called the Boundary-Scan Description Language (BSDL). It is a subset and standard practice of the VHSIC Hardware Description Language (VHDL) (IEEE Std 1076-1993).

B.1 General information

B.1.1 Document outline

Clauses B.2 to B.4 define the purpose of BSDL, its scope, and its relationship to VHDL. Clauses B.5 to B.7 describe general characteristics of the language. Clause B.8, the entity description, describes the overall structure of a BSDL description. B.8.2 to B.8.18 are detailed descriptions of each of the mandatory and optional sections of a BSDL description. They are documented in the order in which they should appear in such a description. Clause B.9 describes the Standard VHDL Package, and Clause B.10 describes a user-supplied VHDL package. Clause B.11 includes some special cases for purposes of illustration and indicates how such components can be specified in BSDL. Clause B.12 shows a typical BSDL description. Clause B.13 briefly documents the 1990 version of BSDL (see also B.1.3). Clause B.14 documents the 1994 version of BSDL (the first version approved by the IEEE Standards Board and published by the IEEE).

In B.8.2 to B.8.18, the detailed descriptions of each of the mandatory and optional sections of a BSDL description are organized in the following way:

— Short introduction
— Syntax
— Notes
— Example and additional text
— Semantic checks

The syntax and semantic subclauses are where the language is defined; however, sometimes material in short introductions, notes, and examples is key to a full understanding of the language. Therefore, such text is also part of this definition. The notes often provide the semantic interpretation of various lexical elements.

Commonly used syntactic elements are defined in B.6.3.

B.1.2 Conventions

a) Examples are printed in `Courier` font.
b) See Clause B.6 for conventions relating to syntax.
c) For clarity, all reserved words, predefined words, and punctuation are shown in **bold Helvetica** type within this document. VHDL reserved and predefined words will be shown in **lowercase** letters, and BSDL reserved words will be shown in **UPPERCASE** letters. (BSDL itself is case-insensitive; this convention is adopted for clarity.)

B.1.3 BSDL history

The development of BSDL started soon after the first promulgation of this standard in 1990. When, out of common self-interest, an industrywide group of companies implementing tools to support this standard realized that a single language for describing the boundary-scan implementations in components would be of benefit, many tools were built using early draft specifications of the BSDL language.

Developments made both to this standard and to the BSDL definition since the language was first proposed in 1990 have resulted in the obsolescence of some of the constructs from the first draft versions of the language. Some constructs were rendered unnecessary as a result of the standardization of the *CLAMP* and *HIGHZ* instructions in 1993, while others were found to duplicate information provided elsewhere in a BSDL description and thus were removed as redundant.

Because a significant number of BSDL descriptions have been written based on the 1990 draft version of the language and because these descriptions are likely to remain in circulation for some time, implementors of tools based on BSDL may wish to design them to read both the BSDL language defined in this annex and the earlier 1990 version. Information on how to do this is contained in Clause B.13.

The first IEEE promulgation of a definition for BSDL was the 1994 version. As this standard has been revised for 2001, the definition of BSDL has been modified to correspond. Implementors of tools based on BSDL shall design them to accommodate both the 2001 version of the BSDL language defined in this annex and the earlier 1994 version. Information on how to do this is contained in Clause B.14.

B.2 Purpose of BSDL

BSDL provides a machine-readable means of representing some parts of the documentary information specified in 13.3. The scope of the language is defined in Clause B.3.

The goal of the language is to facilitate communication between companies, individuals, and tools that need to exchange information on the design of test logic that complies with this standard. For example,

— A vendor of a component that supports this standard shall supply a BSDL description to purchasers.
— Automated test-generation tools may use a library of BSDL descriptions to allow the generation of a test for a particular loaded board.
— The test logic defined by this standard could be synthesized with the support of a BSDL description.

BSDL describes “finished” silicon, not “work-in-progress.” For example, when a bare die conforming to this standard is produced, a BSDL description can be provided for it. A die may be inserted into one or more types of component packages as well, with each variation described in BSDL (see B.8.7). BSDL for partially synthesized test logic is considered “work-in-progress,” is not necessarily compliant with this annex, and in most cases should not be transmitted beyond the synthesis environment.

B.3 Scope of BSDL

BSDL is not a general-purpose hardware description language—it is intended solely as a means of describing key aspects of the implementation of this standard within a particular component. A BSDL description is not itself a simulation model. Examples of features that are and are not described using BSDL are listed in Table B.1.

Table B.1—Scope of BSDL

Features described by BSDL	Features that cannot or need not be described
Length and structure of the boundary-scan register	TAP-controller state diagram
Availability of the optional TRST* pin	Bypass register
Physical locations of the TAP pins	Length of the device-identification register
Instruction binary codes	Provision of *BYPASS*, *SAMPLE*, *PRELOAD*, and *EXTEST* instructions
Device-identification code	Operation of user-defined instructions

Note that the language describes only features of the test logic that can vary from component to component, depending on the choice of the component designer. Features that are completely specified by this standard (without option) are not required to be described in BSDL but may be described if users so wish.

Further, BSDL does not have a general means for providing for the specification of logic levels, timing parameters, power requirements, and similar factors. These data do not affect the logical behavior of an implementation and most likely are already described in other parts of the specification of any given component.

B.4 Relationship of BSDL to VHDL

BSDL is a subset of VHDL (IEEE Std 1076-1993) and shall be fully conformant to the requirements of the VHDL standard.

However, the user must be prepared to modify a BSDL description in the face of implementation dependencies in VHDL-based tools. No way has been found to avoid this small amount of effort without introducing further undesirable complications. Specifically, the <standard use statement> (see B.8.4) and the <use statement> (see B.8.5) may require editing because of tool and file system dependencies. The syntax of the statements as defined is legal VHDL; however, an additional prefix (identifying a library in which the Standard VHDL Package will be found) will need to be added for some applications. A syntax lacking such a prefix has been chosen to force an error in such an application rather than risk unpredictable and confusing errors due to the inclusion of an inappropriate prefix.

NOTE—In the event of an error or omission in this annex regarding VHDL syntax, IEEE Std 1076-1993 shall take precedence.

Let it be noted that

a) BSDL does not employ all the syntactic elements of VHDL. Only those elements required to meet the scope of BSDL are used. The following VHDL statements shall be employed in BSDL:

attribute	**generic**	**subtype**
constant	**package**	**type**
end	**package body**	**use**
entity	**port**	

In some cases, only a subset of a particular VHDL language element syntax is used in BSDL. Descriptions of the lexical elements and statement syntax for each syntax element are contained in Clauses B.5 through B.8 and B.10. Historical information on such elements is found in Clause B.13.

b) For cases in which a feature could be described in several ways within VHDL, a restricted set of ways has been selected and defined explicitly as the standard practice for BSDL. This restriction simplifies the application of the VHDL subset for BSDL, particularly for tools that are required only to read or generate BSDL (i.e., tools that have no requirement to read or write the full VHDL language).
c) BSDL imposes additional requirements on the syntax and content of certain character strings, that is, sequences of characters between quotation marks (e.g., "**EXTEST**"). A VHDL parser will not check the information in these strings. In contrast, a BSDL parser shall check that the information in the strings is appropriate for the relevant parameters or attributes for which such strings might be values. There are several string syntaxes defined within BSDL.

B.5 Lexical elements of BSDL

The lexical elements of BSDL shall be a subset and standard practice of those of VHDL as defined in IEEE Std 1076-1993. The following subclauses enumerate the lexical elements needed to understand the BSDL language definition.

B.5.1 Character set

The language shall not be case-sensitive; that is, for example, the character **a** shall be considered identical to the character **A**. Therefore, the following names are identical:

```
FRED        Fred        fred
```

The following characters shall be permitted within the language:

a) Upper- and lowercase letters: **A** to **Z** and **a** to **z** (the language is not case-sensitive)
b) Digits: **0** to **9**
c) Special characters: **" & ' ( ) * , - . : ; < = >** _ (Note that this list is smaller than that of VHDL.) Other special characters shall be allowed as part of a comment (see B.5.3) or as part of the string element of a design warning (see B.8.18).
d) Logical separators: The space character and VHDL format effectors shall be used as logical separators. VHDL format effectors are the characters called horizontal tabulation, vertical tabulation, carriage return, line feed, and form feed.

Logical separators shall have two purposes. They shall be used to eliminate lexical ambiguity by separating lexical tokens such as reserved words and/or identifiers. For example, the reserved word **entity** must be separated from the component name identifier that immediately follows it rather than being run together with it. Logical separators also may be used in combinations to create visually appealing layouts.

B.5.1.1 Identifiers

Identifiers shall be user-supplied names and reserved words functioning as names. Identifiers shall start with a letter and may contain letters, digits, or the underscore character. For example, the following are valid identifiers:

```
BSDL
IEEE_Std_1149_1
```


There shall be no upper limit to the number of characters in an identifier.

The underscore character (_) shall not be allowed to be the last character in an identifier (by VHDL).

```
IEEE_STD_1149_  -- This is not a legal identifier.
```

Adjacent underscore characters (_ _) shall not be allowed (by VHDL).

B.5.1.2 BSDL reserved words

The identifiers listed in this subclause shall be BSDL reserved words and shall have a fixed significance in the language. These identifiers shall not be used for any purpose in a BSDL description other than as part of a comment. For example, a reserved word shall not be used as an explicitly declared identifier. **BC_0** to **BC_99** shall be variable names used in the Standard VHDL Package. Names **BC_0** through **BC_10** are used today, while **BC_11** through **BC_99** shall be reserved for future use. Similarly, all names that start with **STD_1149_** shall be reserved.

AT_PINS
BC_0 to BC_99
BIDIR
BIDIR_IN
BIDIR_OUT
BOTH
BOUNDARY
BOUNDARY_LENGTH
BOUNDARY_REGISTER
BSCAN_INST
BSDL_EXTENSION
BYPASS
CAP
CAP_DATA
CAPTURES
CELL_DATA
CELL_INFO
CELL_TYPE
CLAMP
CLOCK
CLOCK_INFO
CLOCK_LEVEL
COMPLIANCE_PATTERNS
COMPONENT_CONFORMANCE
CONTROL
CONTROLR
INSTRUCTION_OPCODE
INSTRUCTION_PRIVATE
INTERNAL
INTEST
INTEST_EXECUTION
KEEPER
LOW
OBSERVE_ONLY
OBSERVING
ONE
OUTPUT2
OUTPUT3
PHYSICAL_PIN_MAP
PI
PIN_MAP
PIN_MAP_STRING
PO
PORT_GROUPING
PRELOAD
PULL0
PULL1
REGISTER_ACCESS
RUNBIST
RUNBIST_EXECUTION
SAMPLE
STD_1149_*

DESIGN_WARNING	**TAP_SCAN_CLOCK**
DEVICE_ID	**TAP_SCAN_IN**
DIFFERENTIAL_CURRENT	**TAP_SCAN_MODE**
DIFFERENTIAL_VOLTAGE	**TAP_SCAN_OUT**
EXPECT_DATA	**TAP_SCAN_RESET**
EXTEST	**UPD**
HIGHZ	**USERCODE**
ID_BITS	**USERCODE_REGISTER**
ID_STRING	**WAIT_DURATION**
IDCODE	**WEAK0**
IDCODE_REGISTER	**WEAK1**
INPUT	**X**
INSTRUCTION_CAPTURE	**Z**
INSTRUCTION_LENGTH	**ZERO**

NOTE—In the list of reserved words above, the entry **STD_1149_*** is to be interpreted to mean all names that start with **STD_1149_**, for example, **STD_1149_1_2001**.

B.5.1.3 VHDL reserved and predefined words

The identifiers listed below shall be called VHDL (IEEE Std 1076-1993) reserved and predefined words and shall have a fixed significance in the language. These identifiers shall not be used for any purpose in a BSDL description other than as part of a comment. For example, a reserved word shall not be used as an explicitly declared identifier. Reserved words shown in the list below in lowercase letters are part of the BSDL subset of VHDL. Those in uppercase letters are not part of BSDL but shall not be used as identifiers. The VHDL reserved words are shown in this subclause for convenience, *but the latest edition of the VHDL standard shall be consulted as the final authority.*

ABS	**GUARDED**	**REGISTER**
ACCESS	**IF**	**REJECT**
AFTER	**IMPURE**	**REM**
ALIAS	**in**	**REPORT**
all	**INERTIAL**	**RETURN**
AND	**inout**	**ROL**
ARCHITECTURE	**is**	**ROR**
array	**LABEL**	**SELECT**
ASSERT	**LIBRARY**	**SEVERITY**
attribute	**linkage**	**SHARED**
BEGIN	**LITERAL**	**signal**
bit	**LOOP**	**SLA**
bit_vector	**MAP**	**SLL**
BLOCK	**MOD**	**SRA**

body	**NAND**	**SRL**
buffer	**NEW**	**string**
BUS	**NEXT**	**subtype**
CASE	**NOR**	**THEN**
COMPONENT	**NOT**	**to**
CONFIGURATION	**NULL**	**TRANSPORT**
constant	**of**	**true**
DISCONNECT	**ON**	**type**
downto	**OPEN**	**UNAFFECTED**
ELSE	**OR**	**UNITS**
ELSIF	**OTHERS**	**UNTIL**
end	**out**	**use**
entity	**package**	**VARIABLE**
EXIT	**port**	**WAIT**
FALSE	**positive**	**WHEN**
FILE	**POSTPONED**	**WHILE**
FOR	**PROCEDURE**	**WITH**
FUNCTION	**PROCESS**	**XNOR**
GENERATE	**PURE**	**XOR**
generic	**range**	
GROUP	**record**	

B.5.2 Strings

A string shall be defined as a sequence of zero or more characters enclosed between quotation marks. A quotation mark character shall not be allowed within a string in BSDL. For example,

```
"Mary had a little lamb"-- Allowed
"Fred said ""HELP"""-- Not allowed
```

Strings are used extensively in BSDL. Since many of the BSDL strings are potentially much longer than a single line, the concatenation operator **&** shall be used to break them into manageable pieces. For example,

```
"Mary had a little lamb. " &
"Its fleece was white as snow."
```

is a single string identical to

```
"Mary had a little lamb. Its fleece was white as snow."
```

BSDL shall not permit replacement of the quotation mark with any other character.

A string literal shall fit on one line since it is a lexical element. Therefore, the only legal VHDL format effector in a string literal shall be horizontal tabulation.

B.5.3 Comments

Text between a double dash (**--**) symbol and the end of a line shall be treated as a comment. The text shall be allowed to contain any special character allowed by VHDL, not just those given in item c) of B.5.1. Comments syntactically terminate a line of a description. Comments may be interspersed with lexical elements. For example, the following represents a single VHDL string:

```
"This is all" &    -- An example of a string split by a comment
" a single string"
```

B.6 Notes on syntax definition

B.6.1 BNF conventions

The syntax of BSDL shall be presented in Backus-Naur Form (BNF) as follows:

a) Any item enclosed in chevrons (i.e., between the character "<" and the character ">") shall be the name of a syntax item that will be defined in this annex.
b) Items enclosed between braces (i.e., between the character "{" and the character "}") shall be either omitted or included one or more times.
c) Items enclosed between square brackets (i.e., between the character "[" and the character "]") shall be either omitted or included only one time.
d) Items enclosed between the characters "⎧" and "⎫" may appear in any order.
e) Except with regard to case, text shown in **bold Helvetica** type shall be included exactly as it is presented in this annex.
f) Alternative syntaxes shall be separated by a vertical bar ("|").
g) The symbol "::=" shall be read as "shall be defined as." Note that the nonboldface "::=" is only part of a BNF description; in the BSDL file, the boldface characters "**:=**" are used to indicate assignment.
h) White space (spaces, tabulation, carriage returns, etc.) is used in these BNF descriptions to provide enhanced readability and is not part of the syntax. However, white space needed for resolving lexical ambiguity (logical separation) is required as described in B.5.1.

B.6.2 Lexical atoms

a) A <VHDL identifier> shall be any valid identifier chosen as a name for an item (see B.5.1.1).
b) An <integer> shall be any valid unsigned VHDL integer made up of an unsigned numeric character sequence not containing an underscore (_) character and not using an exponent field.
c) A <real number> shall be any valid VHDL real number of the form <integer>**.**<integer> or <integer>**.**<integer>**E**<integer> all written contiguously without spaces or format effectors. Note that 1E3 is not real because it does not contain a decimal point. The number **20.0E6** is real, as is **20000000.0**.
d) A <pattern> shall be a contiguous sequence of one or more **0**, **1**, and **X** characters containing no spaces or format effectors. For example, **001X00** and **XX010X** are legal. However, **100 X00** is not legal because of the embedded space. A low state (see 3.1.15) shall be denoted by **0**, a high state (see 3.1.9) shall be denoted by **1**, and a don't-care value shall be denoted by **X**.
e) A <32-bit pattern> shall be a <pattern> with exactly 32 characters in its character sequence.
f) A <left bracket> shall be the left bracket character (**[**).
g) A <right bracket> shall be the right bracket character (**]**).
h) A <string> shall be any valid string or concatenated string structure meeting the definition in B.5.2.

Lexical ambiguity exists in certain situations and shall be resolved by context. For example, a <pattern> that starts with an **X** shall be differentiated from a <VHDL identifier> by context derived from the syntax. Similarly, a <pattern> that does not contain an **X** shall be differentiated from an integer such as **100** (one hundred).

B.6.3 Commonly used syntactic elements

a) A <port ID> shall identify a component signal that may be used to interface to external signals. A port shall be dimensioned as either a **bit** or a **bit_vector** (see B.8.3). Subscripted names shall be allowed only when **bit_vector**-dimensioned port signals are used.

<port ID>::= <port name> | <subscripted port name>
<port name>::= <VHDL identifier>
<subscripted port name>::= <VHDL identifier> (<subscript>)
<subscript>::= <integer>

b) An <instruction name> shall be an instruction name defined in this standard or an instruction named by the manufacturer of a component.

<instruction name>::= **BYPASS** | **CLAMP** | **EXTEST** | **HIGHZ** |
IDCODE | **INTEST** | **PRELOAD** |**RUNBIST** |
SAMPLE | **USERCODE** | <VHDL identifier>

NOTE—In earlier editions of this standard, the BSDL reserved word **PRELOAD** did not exist and **SAMPLE** was used as an abbreviated name when the *SAMPLE* and *PRELOAD* instructions were merged [see Permission 8.1.1 g) and Rule B.8.11.3 f)].

B.7 Components of a BSDL description

A BSDL description shall be composed of the following parts, as shown in part in Figure B.1. These parts are

a) *The entity description*. An entity description shall be written for each component. It shall specify component-specific parameters of the test logic and might, for example, be written by the designer of a component.
b) *The Standard VHDL Package and Standard VHDL Package Body.* These parts shall contain two types of information:
   1) The Standard VHDL Package shall give a definition of the BSDL subset of VHDL, which is required to allow BSDL descriptions to be read by a VHDL-conformant parser.
   2) The Standard VHDL Package Body shall give definitions of commonly used types of boundary-scan register cells.

Typically, the Standard VHDL Package and Standard VHDL Package Body would reside with system-accompanying software that utilizes BSDL descriptions. Individual BSDL descriptions are not burdened with having to include all the elements contained in the Standard VHDL Package and Standard VHDL Package Body.

NOTE—The Standard VHDL Package and Standard VHDL Package Body shall be exactly as listed in Clause B.9. They shall not be modified and would not normally be shipped along with BSDL files that describe particular components. Normally, the Standard VHDL Package and Standard VHDL Package Body will be supplied as a part of a BSDL-compliant tool and will be maintained by the tool supplier.

The cell definitions provided in the Standard VHDL Package Body could have been given within the Standard VHDL Package in a full VHDL implementation. The advantage of a package body is that the

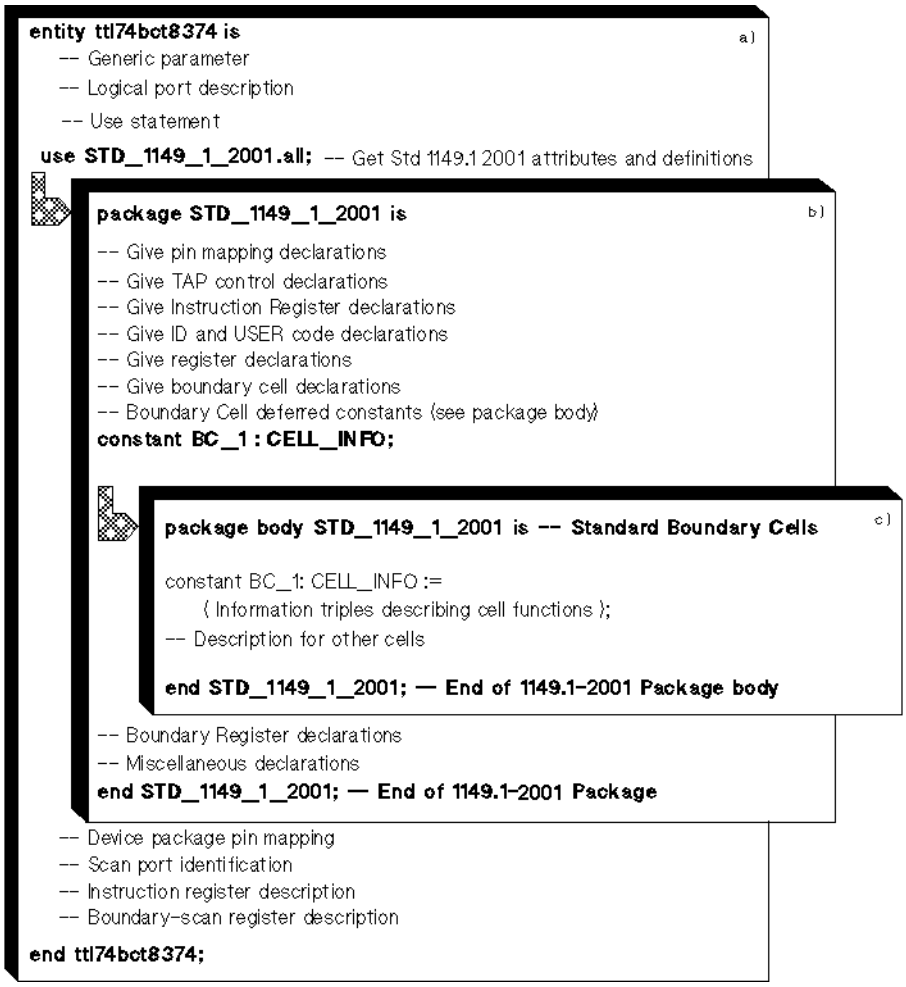


Figure B.1—Components of a BSDL description

information it contains can be updated without causing the need for recompilation of all entities that reference the package. If a package is modified, recompilation is necessary. The package with package body structure is a standard practice of BSDL.

c) *User-specified VHDL packages and VHDL package bodies*. Users may provide VHDL packages and package bodies that define boundary-scan register cell designs specific to any group of components. A vendor of application-specific ICs (ASICs), for example, could provide a user-specified VHDL package and package body to describe the particular boundary-scan register cell designs offered. Global BSDL extensions also could be provided in user-specified VHDL packages (see Clause B.10 and B.8.17).

B.8 The entity description

The entity description and supporting VHDL packages shall make up a BSDL model of the component and are, in effect, the electronic data sheet for its test logic. It shall contain statements through which parameters that may vary from one chip to another are defined, as discussed in Clause B.3.

B.8.1 Overall structure of the entity description

B.8.1.1 Syntax

The BSDL entity description shall have the following structure:

<BSDL description>::=

entity <component name> **is**

<generic parameter>	(see B.8.2)
<logical port description>	(see B.8.3)
<standard use statement>	(see B.8.4)
{<use statement>}	(see B.8.5)
<component conformance statement>	(see B.8.6)
<device package pin mappings>	(see B.8.7)
[<grouped port identification>]	(see B.8.8)
<scan port identification>	(see B.8.9)
[<compliance enable description>]	(see B.8.10)
<instruction register description>	(see B.8.11)
[<optional register description>]	(see B.8.12)
[<register access description>]	(see B.8.13)
<boundary-scan register description>	(see B.8.14)
[<runbist description>]	(see B.8.15)
[<intest description>]	(see B.8.16)
{<BSDL extensions>}	(see B.8.17)
[<design warning>]	(see B.8.18)

end <component name>**;**

<component name>::= <VHDL identifier>

NOTE—While VHDL permits some elements within an entity description to be in an arbitrary order, the fixed ordering above shall be required for BSDL. This ordering is defined to ease the development of tools that are not themselves required to be fully VHDL-compliant.

The <component name> shall identify a particular integrated circuit and should be distinct from the names of all other components that may be used together (i.e., that may be assembled onto a single loaded board).

B.8.1.2 Semantic checks

a) Any <component name> referenced in any attribute statement shall be the same as the component name declared in the entity description.

B.8.2 Generic parameter statement

The <generic parameter> statement shall facilitate selection between multiple component packaging options that are described within the BSDL description (see B.8.7). Each such option shall define a mapping between the physical package pins of the component and the <port ID> elements of the component (see B.6.3).

The component package option relevant to a particular instance of a component shall be identified each time a given BSDL description is referenced.

B.8.2.1 Syntax

<generic parameter>::=
generic (PHYSICAL_PIN_MAP: string); |
generic (PHYSICAL_PIN_MAP: string := <default device package type>**);**

<default device package type>::= **"** <VHDL identifier> **"**

In the first alternative <generic parameter> statement syntax, a <pin mapping name> (see B.8.7) that identifies a component package option shall be supplied with the BSDL description when it is referenced. In the second alternative syntax, a value is given for the <default device package type> that shall be used as the default in the case in which no <pin mapping name> is supplied with the BSDL description when it is referenced. The <default device package type> shall be the quoted name of a <pin mapping name> used to specify the pin mapping for the component (see B.8.7).

B.8.2.2 Examples

```
generic (PHYSICAL_PIN_MAP: string);
```

or

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

NOTE—It is recommended that the component package name from the data sheet of a given component be used as the relevant <pin mapping name>, for example, SSOP_56, PQFP_84, or PGA_18x18.

B.8.2.3 Semantic checks

a) It shall be an error if
   1) No <default device package type> is specified in the <generic parameter> statement of a BSDL description and no package <pin mapping name> (see B.8.7) is specified when a BSDL description is processed
   2) The <default device package type> specified in the <generic parameter> statement is not identical to any <pin mapping name> appearing in any <pin mapping> (see B.8.7) and no <pin mapping name> is specified when the BSDL description is processed
   3) A <pin mapping name> is specified when a given BSDL description is processed and that <pin mapping name> is not identical to any <pin mapping name> appearing in any <pin mapping> (see B.8.7) of that description

B.8.3 Logical port description statement

The BSDL port description shall be a specialization of the VHDL port list. It shall be used to assign meaningful symbolic names to the pins of a component. These symbolic names, which shall be used in subsequent statements in the description, allow the majority of such statements to be independent of a renumbering or other reorganization of the pins of the component.

It is strongly recommended that the existence of nondigital pins (such as power, ground, no-connects, or analog signals) should be recorded in the BSDL description of the component through use of the **linkage** <pin type> defined in the following syntax. BSDL applications may then be able to support substantially improved error detection during board testing; however, only a warning shall be issued in the complete absence of such pins within a description since it is technologically possible for a component not to have any such pins.

B.8.3.1 Syntax

<logical port description>::= **port** (<pin spec> {**;** <pin spec>}**);**

<pin spec>::= <identifier list>: <pin type> <port dimension>
<identifier list>::= <port name> {,<port name>} (see B.6.3)
<pin type>::= **in** | **out** | **buffer** | **inout** | **linkage**
<port dimension>::= **bit** | **bit_vector** (<range>)
<range>::= <integer_1> **to** <integer_2> | <integer_2> **downto** <integer_1>
<integer_1>::= <integer>
<integer_2>::= <integer>

The definitions of the possible values of <pin type> are given in Table B.2, and <port name> is defined in B.6.3.

Table B.2—Pin types

Value	Meaning
in	An input-only pin
out	An output-only pin that may be connected to a bus wire driven by multiple drivers (e.g., a 3-state or open-collector output)
buffer	A 2-state output-only pin where either state is always actively driven (e.g., cannot be connected to a bus wire)
inout	A bidirectional pin
linkage	Other pin types, such as power, ground, analog, and no-connect

NOTE—Where a 2-state output pin has a 3-state mode only to meet the requirements of the *HIGHZ* instruction, it shall be described as a buffer.

The <port dimension> shall define the number of signals that constitute a port. If the <port dimension> is **bit**, one <port name> shall correspond to that one signal. If the <port dimension> is **bit_vector**, one <port name> shall correspond to a collection of *n* signals that shall be individually referenced by subscripting the port name, i.e., by a <subscripted port name> (see B.6.3).

NOTE—The signal types **bit** and **bit_vector** are signal types known to VHDL and shall be the only signal types permitted in BSDL. Also, while VHDL allows multiple, possibly broken ranges to be specified for a **bit_vector**, BSDL syntax shall allow only a single unbroken range.

B.8.3.2 Examples

```
port( TDI, TMS, TCK: in bit;
      TDO: out bit; IN1, IN2: in bit;
      OUT1: out bit; OUT2: buffer bit;
      OUT3: out bit_vector (1 to 8);
      OUT4: out bit_vector (4 downto 1);
      BIDIR1, BIDIR2, BIDIR3: inout bit;
      GND, VCC: linkage bit);
```

B.8.3.3 Semantic checks

a) A **bit_vector** declared in a <logical port description> statement shall have the value of <integer_1> less than or equal to the value of <integer_2>.
b) Each <port name> appearing in an <identifier list> of a <logical port description> shall occur only once within the <logical port description> statement.

B.8.4 Standard use statement

B.8.4.1 Syntax and semantics

The <standard use statement> shall identify one Standard VHDL Package in which attributes, types, constants, and other elements are defined that shall be referenced elsewhere in the BSDL description.

B.8.4.1.1 Syntax

<standard use statement>::= **use** <standard VHDL package identifier>**.all;**

<standard VHDL package identifier>::=
STD_1149_1_1994 | **STD_1149_1_2001** | <other package identifier>

<other package identifier>::= <VHDL identifier>

The <standard VHDL package identifier> shall be the name of the Standard VHDL Package that contains the information to be included. The suffix **.all** shall indicate that all declarations within the VHDL package are to be used.

Values for <other package identifier> may be assigned with future revisions of this annex as the language evolves. For toolmakers desiring to support the earlier draft version of BSDL as well as the version specified in this annex, the value of <other package identifier> may be **STD_1149_1_1990** (see B.13.1).

NOTE—The construct of the <standard use statement> may not be syntactically complete for use by a given VHDL analyzer. This is because a library or default working area has not been specified. In such a case, a complete statement is “use work.STD_1149_1_2001.all;” in which the preface “work.” tells a VHDL analyzer to find the Standard VHDL Package in the current work area. The field “work.” could be replaced by an arbitrary library name such as “BSCAN.” telling a VHDL analyzer where in its system of libraries to find the Standard VHDL Package. Since there is no standardization of library structures from one VHDL environment to another, some editing of BSDL files to specify the location of Standard VHDL Packages is generally unavoidable. The specification used in this subclause may cause an error in a VHDL analyzer, forcing the user to edit the BSDL file for the correct location of the Standard VHDL Package information.

The <standard VHDL package identifier> shall indicate

— An instruction to tools to read a standard package that contains BSDL syntax definitions
— The version of the BSDL language (shown by the year number embedded in the identifier) that was used when creating the BSDL file

As a general rule, future enhancements to the BSDL language will be designed as extensions to previous versions of the language. Therefore, the version information contained in the <standard VHDL package identifier> may be used by BSDL-specific tools to indicate which tool versions will be able to process the information in the input file. For example, if the input file records the version as “**_1994**”, tools that can process any language variant “**_1994**” or later will be able to process the input file correctly.

NOTE—Versions of the BSDL language should not be confused with the version of the standard itself to which a given IC may conform. The <component conformance statement> identifies the version of the standard (see B.8.6).

Within a specific system processing BSDL descriptions, the Standard VHDL Package may be a file somewhere in the file system of the host computer. The **.all** suffix is meaningful to VHDL and is not part of a file name. While VHDL permits the use of a wider range of suffixes, **.all** shall be the only suffix permitted in BSDL.

The content of the Standard VHDL Package shall be the current definition of the BSDL language and shall not be modified by users.

The <standard use statement> shall appear in every BSDL description before any <use statement> (see B.8.5) so that tools that are fully VHDL-compliant can locate information relevant to all components that conform to this standard. (Tools that are limited to the BSDL language shall always use this information.)

B.8.4.1.2 Examples

```
use STD_1149_1_2001.all;  -- Today
```

or

```
use STD_1149_1_2012.all;  -- Sometime in the future
```

The contents of the **STD_1149_1_2001** Standard VHDL Package and its associated Standard VHDL Package Body shall be as listed in Clause B.9. (The contents of the **STD_1149_1_1990** Standard VHDL Package and its associated Standard VHDL Package Body shall be as listed in B.13.1. The contents of the **STD_1149_1_1994** Standard VHDL Package and its associated Standard VHDL Package Body shall be as listed in B.14.1.)

B.8.4.1.3 Semantic checks

a) The Standard VHDL Package content shall be used when processing a BSDL description.

The <standard use statement> also shall be used for version control (see B.8.4.2).

B.8.4.2 Version control

At the time this standard was approved, there were three versions of BSDL descriptions: the 1990 version (see Clause B.13), the 1994 version (approved by the IEEE Standards Board and published by the IEEE in 1994, see Clause B.14), and the 2001 version described here. The <standard use statement> shall indicate whether the BSDL description has been written using the provisional syntax published in the *Proceedings of the International Test Conference* in 1990 or according to this annex. This additional application of the <standard use statement> is intended to provide backwards compatibility to all BSDL descriptions already written and can be used in a similar manner for BSDL descriptions based on future revisions of this annex. It is intended that a parser handling a form of BSDL described in a version of this annex *approved by the IEEE Standards Board and published by the IEEE* shall handle all forms of BSDL described in previous versions of this annex *approved by the IEEE Standards Board and published by the IEEE*.

B.8.4.2.1 Examples

```
use STD_1149_1_2001.all;
```

In the 1990 version of BSDL, the following syntactic elements are not supported:

<component conformance statement>
<grouped port identification>
<compliance enable description>

<runbist description>
<intest description>
<BSDL extensions>

Also, in the 1990 version, cell types **BC_0** and **BC_7** that are defined in this 2001 version (as originally identified and defined by the 1994 version) need to be specified in a user-supplied VHDL package if they are to be referenced. Further, in the 1990 and 1994 versions, cell types **BC_8**, **BC_9**, and **BC_10,** which are identified and defined in this 2001 version, need to be specified in a user-supplied VHDL package if they are to be referenced.

In the 1990 and 1994 versions, the identifiers **KEEPER** and **PRELOAD** were not BSDL reserved words. **KEEPER** and **PRELOAD** become BSDL reserved words in this 2001 version.

Software that processes BSDL descriptions needs to have access to all Standard VHDL Packages related to BSDL. See Clauses B.9, B.13, and B.14 for the three that are currently defined.

B.8.4.2.2 Semantic checks

a) A BSDL parser shall be expected to reject and/or issue warning/error messages when it encounters unrecognized text.

NOTE—Specifically constructed foreign attributes may be added via BSDL extensions (see B.8.17).

B.8.5 Use statement

The optional <use statement> shall identify a VHDL package in which specific attributes or constants are defined so that they can be referenced elsewhere in a BSDL description.

B.8.5.1 Syntax

<use statement>::= **use** <user VHDL package identifier>**.all;**

<user VHDL package identifier>::= <VHDL identifier>

The <user VHDL package identifier> in the <use statement> shall be the name of the VHDL package that contains the information to be included. The suffix **.all** shall indicate that all declarations within the VHDL package are to be used. The **.all** suffix is meaningful to VHDL and is not part of a file name. While VHDL permits the use of a wider range of suffixes, **.all** shall be the only suffix permitted in BSDL. See B.10.1 for the content of a user-supplied package.

B.8.5.2 Examples

```
use Private_Package.all;      -- Identifies the proprietary VHDL
                              -- package of a user
```

NOTE—The construct of the <use statement> may not be syntactically complete for use by a given VHDL analyzer. This is because a library or default working area has not been specified. In such a case, a complete statement is “use work.Private.all;” in which the preface “work.” tells a VHDL analyzer to find the user-supplied VHDL package in the current work area. The field “work.” could be replaced by an arbitrary library name such as “BSCAN.” telling a VHDL analyzer where in its system of libraries to find the user-supplied VHDL package. Since there is no standardization of library structures from one VHDL environment to another, some editing of BSDL files to specify the location of a user-supplied VHDL package is generally unavoidable. This specification may cause an error in a VHDL analyzer, forcing the user to edit the BSDL file for the correct location of the user-supplied VHDL package information.

In full VHDL, two or more use statements may be given which reference VHDL packages that contain different definitions of the same item, such as a Boundary-Scan Register Cell. Later, the ambiguity of which

package is being referenced can be removed by qualifying each reference in a specified manner. This facility shall not be supported in BSDL.

B.8.5.3 Semantic checks

a) Information in a VHDL package named in a <use statement> always shall be used when processing a BSDL description (see Clause B.10).
b) Identifiers declared in VHDL packages shall not be defined
   1) More than once in a single VHDL package
   2) In more than one VHDL package referenced by a <use statement> or <standard use statement> in a single BSDL description

B.8.6 Component conformance statement

The <component conformance statement> shall identify the edition of this standard to which the testability circuitry of a physical component conforms.

It is possible for a component designed in 1990 to be described by the version of BSDL defined by this annex, but this cannot imply that the component conforms to the rules of IEEE Std 1149.1-2001. For example, IEEE Std 1149.1-1990 allowed (by omission of rules) two drivers controlled by a single control cell to be disabled by opposing values loaded into that cell. This is explicitly forbidden in IEEE Std 1149.1-2001 (as it was originally in IEEE Std 1149.1a-1993). The <component conformance statement> allows tools to account for changes in the rules that may occur in past and future editions of this standard.

B.8.6.1 Syntax

<component conformance statement>::= **attribute COMPONENT_CONFORMANCE of**
<component name>**: entity is** <conformance string>**;**

<conformance string>::= **"** <conformance identification> **"**
<conformance identification>::=
STD_1149_1_1990 | **STD_1149_1_1993** | **STD_1149_1_2001**

The symbol **STD_1149_1_1990** shall refer to IEEE Std 1149.1-1990. **STD_1149_1_1993** shall refer to to IEEE Std 1149.1a-1993. **STD_1149_1_2001** shall refer to IEEE Std 1149.1-2001. Subsequent editions of this annex may add new values to the <conformance identification> element.

B.8.6.2 Examples

```
attribute COMPONENT_CONFORMANCE of My_Old_IC:entity is
      "STD_1149_1_1990";     -- 1990 component described
                             -- in this annex
```

B.8.6.3 Semantic checks

No semantic check is necessary for this statement. However, some semantic checks described in this annex shall be influenced by the value that appears in the <conformance identification> element [see, for example, semantic check B.8.14.4 p)].

B.8.7 Device package pin mappings

The mapping of logical signals onto the physical pins of a particular component package shall be defined through the use of an attribute statement and an associated BSDL string.

B.8.7.1 Syntax

<device package pin mappings>::= <pin map statement> <pin mappings>

<pin map statement>::= **attribute PIN_MAP of** <component name>**: entity is PHYSICAL_PIN_MAP;**

<pin mappings>::= <pin mapping> {<pin mapping>}
<pin mapping>::= **constant** <pin mapping name>**: PIN_MAP_STRING:=** <map string>**;**
<pin mapping name>::= <VHDL identifier>
<map string>::= **"** <port map> {**,** <port map>} **"**
<port map>::= <port name>**:** <pin list> (see B.6.3)
<pin list>::= <pin ID> | **(**<pin ID> {, <pin ID>}**)**
<pin ID>::= <VHDL identifier> | <integer>

NOTE—No subscripting of <port name> shall be allowed. Only the base name of a port shall appear if the port was described to be a **bit_vector**.

B.8.7.2 Examples

```
attribute PIN_MAP of ttl74bct8374: entity is PHYSICAL_PIN_MAP;
constant DW:PIN_MAP_STRING:=
      "CLK:1, Q:(2,3,4,5,7,8,9,10), " &
      "D:(23,22,21,20,19,17,16,15), " &
      "GND:6, VCC:18, OC_NEG:24, " &
      "TDO:11, TMS:12, TCK:13, TDI:14";
constant FK:PIN_MAP_STRING:=
      "CLK:9, Q:(10,11,12,13,16,17,18,19), " &
      "D:(6,5,4,3,2,1,26,25), " &
      "GND1:15, VCC1:8, GND2:14, VCC2:22, OC_NEG:7, " &
      "TDO:20, TMS:21, TCK:23, TDI:24";
constant DIE_BOND:PIN_MAP_STRING:=
      "CLK:Pad01, " &
      "Q:(Pad02,Pad03,Pad04,Pad05,Pad06,Pad07,Pad08,Pad09), " &
      "D:(Pad10,Pad11,Pad12,Pad13,Pad14,Pad15,Pad16,Pad17), " &
      "GND1:Pad18, VCC1:Pad19, GND2:Pad20, VCC2:Pad21, " &
      "OC_NEG:Pad22, " &
      "TDO:Pad23, TMS:Pad24, TCK:Pad25, TDI:Pad26";
```

Attribute **PIN_MAP** shall be a string that is set to the value of the parameter **PHYSICAL_PIN_MAP**, which is defined by the generic statement. VHDL constants shall then be declared, one for each packaging variation. In the example description, three packaging variations exist: `DW`, `FK`, and `DIE_BOND`.

The constants shall identify component packages that are typified for BSDL purposes by the mapping between logical port names and the physical pins of a component. A BSDL parser shall look for the constant with a name matching the value of **PIN_MAP**. The standard practice for BSDL mandates that the type of the constant shall be **PIN_MAP_STRING**.

The following is an example of a <map string>:

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


Notice that this is the concatenation of two smaller strings. A BSDL parser reads the contents of the string. It matches signal names, such as `CLK`, with the names in the port definition.

For a given <port map>, the <pin list> shall identify the physical pin (or set of physical pins) associated with the port called <port name>. A <pin ID> may be a number or an alphanumeric identifier because some component packages, such as Pin-Grid Arrays (PGAs), use coordinate identifiers, such as A07 or H13. Note, however, that names such as 7A and 13H are illegal since they are not valid VHDL identifiers.

If signals such as that having <port name> `Q` in the example are identified as **bit_vector** in the <logical port description>, there must be a one-to-one mapping between the members of that **bit_vector** and the members of the <pin list> associated with `Q`.

The ordering of items in the <pin list> shall be significant, and the ordering shall provide correlation between a given <subscripted port name> and its associated <pin ID>.

For example, if `Q` were defined to have members `"1 to 8"`, the physical pin mapped onto port `Q(1)` in the example (for the `DW` component package) would be pin `2`, and `Q(2)` would be pin `3`, etc. If `Q` were defined to have members `"8 downto 1"`, `Q(1)` would be pin `10` and `Q(2)` would be pin `9`, etc.

Of the three mappings shown in the example, the first two are for packaging variations of a finished IC and the third shows a die-bond mapping for a finished bare die that might be used in a multichip module.

B.8.7.3 Semantic checks

a) Within a given <pin mapping>, each <pin ID> shall appear only once.
b) All nonlinkage ports in the <logical port description> of a given BSDL description shall be referenced in each <pin mapping> of that description, and vice versa.
c) For a given <port map>, the number of members in the <pin list> shall be the same as the number of bits of the **bit_vector** of the port or shall be 1 if the relative <port dimension> is **bit**.
d) The <port map> for a <port name> defined as **bit** in the <logical port description> shall be defined using a single <pin ID> without a subscript. The <port map> for a <port name> defined as a **bit_vector** in the <logical port description> shall be defined using one or more <pin ID> elements within parentheses.
e) Each <pin mapping name> shall be unique within a <pin mappings>.
f) Any <port name> element in a <port mapping> element of a given BSDL description shall appear in an <identifier list> element of the <logical port description> statement of the description.

Semantic check B.8.7.3 b) requires separate BSDL descriptions for components that do not have pin connections for some nonlinkage signals. [Concomitant differences in the boundary-scan register description (see B.8.14) will also result from these differences.]

There may be different numbers of linkage ports among packaging variants for a component. In the given example, the `FK` package had two more power/ground pins than the `DW` package. All ports `GND`, `GND2`, `VCC`, and `VCC2` must have appeared in the port definition. It is strongly recommended that all physical linkage pins should be included in a <pin mapping>.

B.8.8 Grouped port identification

The optional <grouped port identification> shall be used to identify system I/O signals that have the special characteristic of using more than one pin to carry a bit of data and for which the allowed states of the pins are restricted. Typically, these are differential pairs of signals operating in either the voltage domain or the current domain. The syntax shown below allows for future expansion if it is later determined that the description of signal groups containing more than two signals is important.

In the case of differential pairs where one signal is always the complement of the other (a state restriction), there is a “Plus” pin and a “Minus” pin. This standard states that a differential pair may need to be treated as an analog circuit with a single boundary-scan register cell providing or receiving data (see Figure 11-7 and Figure 11-9). However, due to the prevalence of differential signaling and the fact that digital data is indeed being transmitted, it is desirable to accomplish boundary-scan interconnect testing on each pin of a differential signal pair. The grouped port identification shall be used to describe such a situation to test generation software.

Rules 11.5.1 i) and 11.6.1 n) mandate that the data provided by or captured by a boundary-scan register cell shall have the same polarity as the data bit transmitted by the associated I/O pin. Clearly, the “Minus” signal cannot do this since it always transmits the complement of the “Plus” signal. The “Plus” signal can be identified to software so that it can be directly associated with the boundary-scan register. Then the “Minus” signal is linked to the “Plus” signal to indicate that a pairing exists and that the “Minus” signal is *not* associated with a cell in the boundary-scan register.

Ports shall have a <grouped port identification> when the state restrictions apply during boundary-scan operation, e.g., during *EXTEST*. If the ports are restricted during normal system operation but *not* during boundary-scan operation, the <grouped port identification> will lead software to produce incomplete results, e.g., an inadequate board interconnect test.

B.8.8.1 Syntax

<grouped port identification>::=
attribute PORT_GROUPING of <component name>**: entity is** <group table string>**;**

<group table string>::= **"** <group table> **"**
<group table>::= <twin group entry> {**,** <twin group entry>}
<twin group entry>::= <twin group type> **(**<twin group list>**)**
<twin group type>::= **DIFFERENTIAL_VOLTAGE** | **DIFFERENTIAL_CURRENT**
<twin group list>::= <twin group> {**,** <twin group>}
<twin group>::= **(**<representative port>, <associated port>**)**
<representative port>::= <port ID> (see B.6.3)
<associated port>::= <port ID>

The <representative port> pin shall be treated as the positive (“Plus”) pin of a differential pair. The <associated port> shall be the negative (“Minus”) pin of a differential pair.

Software must determine how to handle **DIFFERENTIAL_CURRENT** signals that are directly accessible to tester resources. The **DIFFERENTIAL_VOLTAGE** signals typically would be physically similar to other logic signals being tested. These two keywords help to inform software as to the physical nature of the way data is transmitted.

B.8.8.2 Examples

This example includes a <boundary-scan register description> (see B.8.14) for purposes of illustrating certain semantic relationships.

```
entity diff is
generic(Physical_Pin_Map:string:= "Pack");

port (CLK:in bit;
      D_Pos:in bit_vector(1 to 4);
      D_Neg:in bit_vector(1 to 4);
      Q_Pos:out bit_vector(1 to 4);
```



```
      Q_Neg:out bit_vector(1 to 4);
      GND, VCC:linkage bit; OC_NEG:in bit;
      TDO:out bit; TMS, TDI, TCK:in bit);

-- Get IEEE Std 1149.1-2001 attributes/definitions

use STD_1149_1_2001.all;

attribute COMPONENT_CONFORMANCE of diff : entity is
           "STD_1149_1_2001";

attribute PIN_MAP of diff : entity is PHYSICAL_PIN_MAP;

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

attribute PORT_GROUPING of diff : entity is
"Differential_Voltage ((Q_Pos(1),Q_Neg(1)),"& -- Voltage signals
                      "(Q_Pos(2), Q_Neg(2)),"&
                      "(Q_Pos(3), Q_Neg(3)),"&
                      "(Q_Pos(4), Q_Neg(4))),"&
"Differential_Current ((D_Pos(1),D_Neg(1)),"& -- Current signals
                      "(D_Pos(2), D_Neg(2)),"&
                      "(D_Pos(3), D_Neg(3)),"&
                      "(D_Pos(4), D_Neg(4)))";
```

(... some BSDL deleted for brevity ...)

```
attribute BOUNDARY_REGISTER of diff : entity is
-- num cell  port     function safe [ccell disval rslt]
   "9 (BC_1, CLK,      input,   X)," &
   "8 (BC_1, OC_NEG,   input,   X)," & -- Merged input/control
   "8 (BC_1, *,        control, 1)," & -- Merged input/control
   "7 (BC_1, D_Pos(1), input,   X)," &
   "6 (BC_1, D_Pos(2), input,   X)," &
   "5 (BC_1, D_Pos(3), input,   X)," &
   "4 (BC_1, D_Pos(4), input,   X)," &
   "3 (BC_1, Q_Pos(1), output3, X,   8, 1, Z),"&-- Also bussable
   "2 (BC_1, Q_Pos(2), output3, X,   8, 1, Z),"&
   "1 (BC_1, Q_Pos(3), output3, X,   8, 1, Z),"&
   "0 (BC_1, Q_Pos(4), output3, X,   8, 1, Z)";

end diff;
```

Software processing this BSDL description example is able to identify four pairs of voltage-differential pins and four pairs of current-differential pins. Each differential signal shall be separated into positive and negative pins, with the positive pins associated with cells in the boundary-scan register.

B.8.8.3 Semantic checks

a) Any <port ID> used in a <grouped port identification> shall have been declared in the <logical port description> statement and name a signal of a nonlinkage type.
b) If a <port ID> is a <subscripted port name>, the <subscript> (see B.6.3) shall lie within the range specified for the **bit_vector** of the relevant port (see B.8.3).
c) Any <port ID> appearing as a <representative port> shall also appear as a <port ID> in a <cell spec> in the subsequent <boundary-scan register description> (see B.8.14).
d) Any <port ID> appearing as an <associated port> *shall not* appear as a <port ID> in a <cell spec> in the subsequent <boundary-scan register description> [this is an exception to semantic check B.8.14.4 s)].
e) The two ports listed in a <twin group> shall have the same pin type (see Table B.2).
f) Any <port ID> appearing in a <twin group> shall be a <subscripted port name> or a <port name> defined in a <logical port description> statement with a corresponding <port dimension> equal to **bit**.

B.8.9 Scan port identification

The scan port identification statements shall define the TAP of the component.

B.8.9.1 Syntax

<scan port identification>::=
⦃<TCK stmt> <TDI stmt> <TMS stmt> <TDO stmt> [<TRST stmt>]⦄

<TCK stmt>::= **attribute TAP_SCAN_CLOCK of** <port ID> **: signal is (**<clock record>**);**
<TDI stmt>::= **attribute TAP_SCAN_IN of** <port ID> **: signal is true;**
<TMS stmt>::= **attribute TAP_SCAN_MODE of** <port ID>**: signal is true;**
<TDO stmt>::= **attribute TAP_SCAN_OUT of** <port ID>**: signal is true;**
<TRST stmt>::= **attribute TAP_SCAN_RESET of** <port ID> **: signal is true;**

<clock record>::= <real number> **,** <halt state value>
<halt state value>::= **LOW** | **BOTH**

The statements shall identify specific logical signals of the component as being signals of the TAP.

A <clock record> shall be a pair consisting of

— A real number that gives the maximum operating frequency for TCK in hertz. In the example of B.8.9.2, 20.0 MHz is specified as the maximum operating frequency.
— A VHDL type that shall have one of two values—**LOW** and **BOTH**. This shall specify the state(s) in which the TCK signal may be stopped without causing loss of data held in the test logic. **BOTH** shall indicate that the clock can be stopped in either state. Components that allow TCK to be stopped only in the high state do not conform to this standard.

B.8.9.2 Examples

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


Signal names TDI, TDO, TMS, TRST, and TCK are those that were used in the <logical port description> in the above example (see B.8.8.2). The names used in the above examples are those defined in this standard; however, arbitrary names could have been used.

NOTE—The meaning of each signal in relation to this standard shall be deduced from the attribute name, not from the value of a <port ID> element in the <scan port identification>.

B.8.9.3 Semantic checks

a) With respect to the <port ID> elements in the <scan port identification>,
   1) The unique <port name> element in a <port ID> occurring in a <TDO stmt> shall appear as a value in an <identifier list> of a <pin spec> (see B.8.3) in the <logical port description>. Furthermore, the value of <pin type> on that <pin spec> shall be **out**.
   2) The unique <port name> element in a <port ID> occurring in a <TCK stmt>, a <TDI stmt>, a <TMS stmt>, or a <TRST stmt> shall appear as a value in an <identifier list> of a <pin spec> (see B.8.3) in the <logical port description>. Furthermore, the value of <pin type> on that <pin spec> shall be **in**.
   3) Any <port ID> appearing in the <scan port identification> shall be a <subscripted port name> or a <port name> defined in a <logical port description> statement with a corresponding <port dimension> equal to **bit**.

b) No <port ID> in the <scan port identification> shall later appear in the <boundary-scan register description> [this is an exception to semantic check B.8.14.4 s)].
c) A given <port ID> shall occur at most once in the <scan port identification>.
d) If a <port ID> is a <subscripted port name>, the <subscript> (see B.6.3) shall lie within the range specified for the **bit_vector** of the relevant port.
e) No value of a <port ID> element in the <scan port identification> shall have appeared as a <representative port> or an <associated port> in a <twin group> (see B.8.8).

B.8.10 Compliance-enable description

This portion of a BSDL description shall appear in the description of a component or bare chip if the optional compliance-enable feature described by this standard (see 4.8) has been implemented in that component or bare chip. Otherwise, improper operation of the part may occur during an automatically generated test.

B.8.10.1 Syntax

<compliance enable description>::= **attribute COMPLIANCE_PATTERNS of**
<component name> **: entity is** <compliance pattern string> **;**

<compliance pattern string>::= **" (** <compliance port list> **) (** <pattern list> **)"**
<compliance port list>::= <port ID> { **,** <port ID>} (see B.6.3)
<pattern list>::= <pattern> { **,** <pattern> }

NOTE—A number of <pattern> elements may be specified, reflecting the fact that there may be many combinations of bits that enable compliance. For convenience, a <pattern> may contain X bits to reduce the size of a <pattern list>. As an example, the <pattern list> 1111X, 0XXX0 specifies 10 unique bit patterns that enable compliance.

A semantic relationship shall positionally associate <port ID> elements in a <compliance port list> with the bits in a <pattern>. For example, the first appearing <port ID> is associated with the first (leftmost) bit in <pattern>.

The following example is given for a component that implements an LSSD test mode as well as standard boundary scan, as described in Annex A [see A.1 c)]. When the LSSD test mode is used, the test logic

defined by this standard becomes configured as a part of a scan path such that LSSD techniques can be used to verify the operation of the complete component. Thus, the LSSD control pins are compliance-enable pins.

B.8.10.2 Examples

```
attribute COMPLIANCE_PATTERNS of Annex_A_IC: entity is
      "(LSSD_A, LSSD_B, LSSD_P, LSSD_C1, LSSD_C2) (00011)";
```

Software that generates tests using test facilities defined by this standard must assure that any compliance-enable conditions are first set up before exercising the TAP of the affected IC. Any compliance-enable pattern (provided as a <pattern> within the <pattern list>) must be held constant for the duration of all boundary-scan testing.

B.8.10.3 Semantic checks

a) The number of <port ID> elements in the <compliance port list> shall be equal to the number of bits in each <pattern> within the <pattern list>.
b) No <port ID> value shall occur more than once in the <compliance port list>.
c) No <port ID> value in the <compliance port list> shall also appear as a <port ID> value in the <scan port identification> (see B.8.9).
d) Each <port ID> value in the <compliance port list> shall appear as a value in an <identifier list> of a <pin spec> (see B.8.3) in the <logical port description>. Furthermore, the value of <pin type> in that <pin spec> shall be **in**.
e) No <port ID> value in the <compliance port list> shall later appear in the <boundary-scan register description> [this is an exception to semantic check B.8.14.4 s)].
f) If a <port ID> value in the <compliance enable description> is a <subscripted port name>, the <subscript> (see B.6.3) shall lie within the range specified for the **bit_vector** of the relevant port.
g) No <port ID> value in the <compliance port list> shall appear as a <port ID> in the <grouped port identification> (see B.8.8).

B.8.11 Instruction register description

The next segment of the BSDL description concerns the component-dependent characteristics of the instruction register. The details that shall be specified to characterize the implementation of the instruction register in a particular component are

— *Length*. The instruction register will be at least two bits long. Its length is not otherwise limited.
— *Instructions*. The register is required to support certain instructions. A designer may add any or all of the optional instructions defined by this standard and/or any number of design-specific instructions. Also, this standard provides for private instructions, which shall be marked as such to warn applications not to use them in order to prevent unsafe or undocumented behavior.
— *Instruction binary codes (opcodes)*. The *BYPASS* instruction is decoded from a bit pattern fixed by this standard [see Rule 8.4.1 b)]. Bit patterns for other instructions shall be specified by the test logic designer. Each instruction may be decoded from several bit patterns.
— *Instruction capture*. On passing through the *Capture-IR* controller state, the instruction register will load data from its parallel input. In some register stages, certain fixed values are required, but in other register stages, design-dependent values may be loaded.

BSDL shall provide a means of describing these characteristics and take advantage of opportunities for semantic checks, thus verifying that the component is in compliance with this standard (that is, it has implemented the required instruction binary codes properly).

The characteristics of the instruction register that shall be specified using BSDL are its length, the opcodes, the pattern captured in the *Capture-IR* controller state, and whether any given instruction is public or private.

B.8.11.1 Syntax

<instruction register description>::= <instruction length stmt>
<instruction opcode stmt>
<instruction capture stmt>
[<instruction private stmt>]

<instruction length stmt>::= **attribute INSTRUCTION_LENGTH of** <component name>
: entity is <integer>**;**
<instruction opcode stmt>::= **attribute INSTRUCTION_OPCODE of** <component name>
: entity is <opcode table string>**;**
<instruction capture stmt>::= **attribute INSTRUCTION_CAPTURE of** <component name>
: entity is <pattern list string>**;**
<instruction private stmt>::= **attribute INSTRUCTION_PRIVATE of** <component name>
: entity is <instruction list string>**;**

<opcode table string>::=**"** <opcode description> {, <opcode description>} **"**
<opcode description>::=<instruction name> (<pattern list>)
<pattern list>::= <pattern> {**,** <pattern>}
<pattern list string>::=**"** <pattern list> **"**
<instruction list string>::= **"** <instruction list> **"**
<instruction list>::= <instruction name> {**,** <instruction name>} (see B.6.3)

B.8.11.2 Examples

```
attribute INSTRUCTION_LENGTH of My_IC:-- Must be first
                                entity is 4;
attribute INSTRUCTION_OPCODE of My_IC:-- Must be second
                                entity is
      "EXTEST (0011), " &
      "EXTEST (1011), " &
      "BYPASS (1111), " &
      "SAMPLE (0001, 1000), " &
      "PRELOAD(1001, 1000)," &
      "HIGHZ  (0101), " &
      "SECRET (1010) ";
attribute INSTRUCTION_CAPTURE of My_IC:-- Must be third
                                entity is "0001";
attribute INSTRUCTION_PRIVATE of My_IC:-- Optional
                                entity is "Secret";
```

NOTES

1—In the above example, *BYPASS* was shown to be decoded from “1111”. Because this is the mandatory pattern specified by this standard, its expression is redundant and not required. In addition to any explicitly assigned patterns, all unassigned patterns will also be decoded as *BYPASS*.

2—For devices designed to conform to editions of this standard before IEEE Std 1149.1-2001, *EXTEST* had a mandatory pattern of all zeros (for example, “0000”), and so, where the value of <conformance identification> in a given BSDL description is **STD_1149_1_1990** or **STD_1149_1_1993**, the expression of this mandatory decode is redundant and not required. For devices designed to conform to this edition of the standard, *EXTEST* has no mandatory bit pattern and

so, where the value of <conformance identification> in a given BSDL description is **STD_1149_1_2001**, **EXTEST** and its bit pattern must occur in the <instruction opcode stmt>.

3—Also, notice that **EXTEST** is given on two lines with two decodes. This shows that multiple lines may be used for each instruction if needed (for instance, when this attribute is written by a computer program).

4—As would be required in the case that the <conformance_identification> were **STD_1149_1_2001**, the above example illustrates the specification of a pattern for **PRELOAD**. It also illustrates options for *SAMPLE* and *PRELOAD* instructions both where the same pattern is used for both and where unique patterns are used for each. It should be further noted that where a pattern for *SAMPLE* is the same as a pattern for PRELOAD, all rules for both instructions must be met. This combination is equivalent to the *SAMPLE/PRELOAD* instruction in previous editions of this standard.

The purpose of each attribute shall be as follows:

— **INSTRUCTION_LENGTH**. The <instruction length stmt> shall define the length of the instruction register and, hence, the number of bits that each opcode pattern shall contain in subsequent statements of the <instruction register description>.
— **INSTRUCTION_OPCODE**. The <instruction opcode stmt> shall be a BSDL string containing instruction identifiers and their associated bit patterns. The rightmost bit in the pattern shall be that closest to TDO (i.e., that shifted in first). Each <opcode description> shall be such a pair. This standard mandates the existence of *BYPASS*, *SAMPLE*, *PRELOAD*, and *EXTEST* instructions, with a mandatory bit pattern for *BYPASS*. Decoding of unused bit patterns shall default to the *BYPASS* instruction.
— **INSTRUCTION_CAPTURE**. The <instruction capture stmt> shall specify the bit pattern that is loaded into the instruction register when the TAP controller passes through the *Capture-IR* state. This bit pattern shall be shifted out whenever a new instruction is shifted in. This standard mandates that the two least significant bits must be “01”. The remainder of this bit pattern is design-specific.
— **INSTRUCTION_PRIVATE**. The optional <instruction private stmt> shall identify instructions that are private and potentially unsafe for use by other than the manufacturer of the component. By definition, the effects of these instructions are undefined to the general public; their use should be avoided. Note that failure to follow warnings about private instructions can result in damage to the component, circuit board, or system.

B.8.11.3 Semantic checks

a) The integer value of the attribute **INSTRUCTION_LENGTH** shall be greater than or equal to 2. This value shall be interpreted as the length of the instruction register.
b) All opcodes defined within a <pattern list> shall have length equal to that of the instruction register.
c) The opcode made up of all 1s shall decode to *BYPASS* or not be defined explicitly for any instruction.
d) Where the value of <conformance identification> is **STD_1149_1_2001**, an opcode for *EXTEST* shall be defined, and the <opcode description> in which it is defined shall have **EXTEST** as the value of its <instruction name> element.

NOTE—Where a device conforms to this 2001 edition of this standard, as indicated by a <conformance identification> value of **STD_1149_1_2001**, the all 0s opcode is not mandated for *EXTEST*. Therefore, an opcode bit pattern for *EXTEST* is required to be explicitly defined. Further, if the all 0s opcode is not otherwise assigned, it decodes to *BYPASS*.

e) Where the value of <conformance identification> is **STD_1149_1_1990** or **STD_1149_1_1993**, the opcode made up of all 0s shall decode to *EXTEST* or shall not be defined explicitly for any instruction.

NOTE—Where a device conforms to earlier editions of this standard, as indicated by a <conformance identification> value of **STD_1149_1_1990** or **STD_1149_1_1993**, the all 0s opcode is mandated for *EXTEST* and so shall be defined as such or be not explicitly defined (in which case it is presumed by implication).

f) Where the value of <conformance identification> is **STD_1149_1_2001**, an opcode for *SAMPLE* shall be defined, and the <opcode description> in which it is defined shall have **SAMPLE** as the value of its <instruction name> element.
g) Where the value of <conformance identification> is **STD_1149_1_2001**, an opcode for *PRELOAD* shall be defined, and the <opcode description> in which it is defined shall have **PRELOAD** as the value of its <instruction name> element.
h) Where the value of <conformance identification> is **STD_1149_1_1990** or **STD_1149_1_1993**, an opcode for *SAMPLE/PRELOAD* shall be defined as follows:
    1) At least one <opcode description> shall have **SAMPLE** as the value of its <instruction name> element.
    2) Any <opcode description> that has **PRELOAD** as the value of its <instruction name> element shall contain only such values for <pattern> as have been defined for any <opcode description> that fulfills the requirement for B.8.11.3 h) 1).

NOTE—Where a device conforms to earlier editions of this standard, as indicated by a <conformance identification> value of **STD_1149_1_1990** or **STD_1149_1_1993**, the functions of *SAMPLE* and *PRELOAD* were required to be implemented in a merged fashion, *SAMPLE/PRELOAD*. Therefore, every opcode which is defined for *SAMPLE* is implicitly defined for *PRELOAD*. Further, thus, where an opcode is defined for *PRELOAD*, it must have a binary value that matches one defined for *SAMPLE*.

i) Where Permission 8.1.1 g) is met, the length of the selected registers shall be identical for each of the instructions that share the same opcode.
j) Where Permission 8.1.1 g) is not met, opcodes containing **X** bits shall not be ambiguous (i.e., decodable as two or more different instructions). That is, if there are two <opcode description> elements and two <pattern> elements such that an **X** appears in each of the two <opcode description> elements, the two <pattern> elements shall differ in some character position in which *neither* pattern contains the character **X**.
k) The <pattern> value in the <instruction capture stmt> shall have a length equal to that of the instruction register. The two least significant bits of this <pattern> shall be **01**.
l) Only user-defined instructions shall be defined as private.
m) Any <instruction name> appearing in an <instruction list> shall appear only once and also shall appear in the <opcode table>.

B.8.12 Optional register description

This section shall identify whether the optional device identification register is provided by specifying the bit patterns returned in response to selection of the device identification register instructions—*IDCODE* and, if implemented, *USERCODE*.

B.8.12.1 Syntax

<optional register description>::= ⎧ <idcode statement> [<usercode statement>] ⎫

<idcode statement>::= **attribute IDCODE_REGISTER of** <component name>
: entity is <32-bit pattern list string>**;**

<usercode statement>::= attribute USERCODE_REGISTER of <component name>
: entity is <32-bit pattern list string>**;**

<32-bit pattern list string>::= **"** <32-bit pattern list> **"**
<32-bit pattern list>::= <32-bit pattern> {, <32-bit pattern>} (see B.6.2)

B.8.12.2 Examples

```
attribute IDCODE_REGISTER of My_IC: entity is
      "0011" &                -- Version
      "1111000011110000" &    -- Part number
      "00001010100" &         -- Identity of the manufacturer
      "1";                    -- Required by IEEE Std 1149.1-1990

attribute USERCODE_REGISTER of My_IC: entity is
  "10XX" & "0011" & "1100" & "1111" & -- Start 1st 32-bit pattern
  "0000" & "0000" & "0000" & "1111,"& -- End 1st 32-bit pattern
  "111X" & "0011" & "1001" & "1000" & -- Start 2nd 32-bit pattern
  "0000" & "0100" & "1001" & "1000";  -- End 2nd 32-bit pattern
```

In these examples, concatenation is used to delimit fields within the codes.

An "**X**" shall be used to mask subfields within a code that are not important for testing purposes; "**X**" specifies a "don't care." It is also possible that a single component type may be sourced from different manufacturers with different version fields. The <32-bit pattern list> shall allow all desired sourcing to be fully specified without using "**X**" bits.

B.8.12.3 Semantic checks

a) If a device identification register is specified by the inclusion of an <idcode statement>, the least significant bit in any <32-bit pattern> element within the <idcode statement> shall be **1**.
b) An <idcode statement> shall appear in a BSDL description if and only if **IDCODE** appears as the value of an <instruction name> element in an <opcode description> of the <instruction opcode stmt>.
c) A <usercode statement> shall appear in a BSDL description if and only if both **IDCODE** and **USERCODE** appear as the value of <instruction name> elements in an <opcode description> of the <instruction opcode stmt>.
d) An illegal bit pattern in the manufacturer code shall cause an error [see Rule 12.2.1 b)].

B.8.13 Register access description

All instructions shall place a test data register between TDI and TDO. User-defined instructions may access test data registers mandated by this standard or design-specific registers. This standard allows a designer to place additional test data registers, referenced by user-defined instructions, in the component.

It is important for test development software to know of the existence and length of all test data registers and the names of their associated instructions.

B.8.13.1 Syntax

<register access description>::=
attribute REGISTER_ACCESS of <component name>**: entity is**
<register string>**;**

<register string>::= **"** <register association> {**,** <register association>} **"**
<register association>::= <register> **(**<instruction capture list>**)**
<instruction capture list>::= <instruction capture> {, <instruction capture>}
<instruction capture>::= <instruction name> [**CAPTURES** <pattern>](see B.6.3)
<register>::= **BOUNDARY** | **BYPASS** | **DEVICE_ID** |

<VHDL identifier> <left bracket> <register length> <right bracket>
<register length>::= <integer>

B.8.13.2 Examples

```
attribute REGISTER_ACCESS of ttl74bct8374: entity is
      "BOUNDARY (READBN, READBT, CELLTST), " &
      "BYPASS (TOPHIP, SETBYP, RUNT, TRIBYP), " &
      "BCR[2] (SCANCN, SCANCT CAPTURES 0X)";
```

In this example, `READBN`, `READBT`, `CELLTST`, `TOPHIP`, `SETBYP`, `RUNT`, `TRIBYP`, `SCANCN`, and `SCANCT` must have been defined in the <instruction opcode stmt> (see B.8.11). The first three instructions select the boundary-scan register, while the next four instructions select the bypass register. Note that the last two instructions (`SCANCN` and `SCANCT`) select a two-bit register called `BCR[2]`. This is a design-specific test data register. The `SCANCT` instruction also shows a capture value `0X` that will be loaded into `BCR[2]` when passing through the *Capture-DR* controller state. No capture value is specified for the `SCANCN` instruction.

By identifying the association between instructions and test data registers, software that can read BSDL descriptions can determine the length of the test data scan sequence for a given instruction. The lengths of the mandatory boundary-scan register, bypass register, and instruction register, as well as the optional identification register, are known from other statements as well as from their relationship to the instructions for the component. A semantic check shall be made to ensure that each instruction has an associated test data register as required by this standard. Exceptions to this are private instructions (see B.8.11). They are not to be accessed, nor do their target registers need to be identified. Because these instructions should not be used by component users, their associated test data registers need not be defined in the BSDL description. (However, they may be defined if the user wishes.)

Note that this standard allows user instructions to reference several registers at once in a concatenated mode [see Permission 9.2.1 e)]. In BSDL, the logical register resulting from such a concatenation shall be treated as if it were a separate register with a distinct name and length.

B.8.13.3 Semantic checks

a) The association of registers with the *BYPASS*, *CLAMP*, *EXTEST*, *HIGHZ*, *IDCODE*, *INTEST*, *PRELOAD*, *SAMPLE*, and *USERCODE* instructions is mandated by this standard. Descriptions of these assignments are redundant and not needed in BSDL, but if they are given,
   1) They shall be checked against the mandatory assignment specified in this standard and an error issued if they are not correct; and
   2) They shall not have a "**CAPTURES** <pattern>" element in their description. (This capture data is specified either in this standard or elsewhere in BSDL.)

b) Any public instruction (i.e., an instruction whose name appears in the <instruction opcode stmt> but does not appear in the <instruction private stmt>, see B.8.11) not listed in semantic check B.8.13.3 a) shall have an associated test data register defined.

c) The length of each publicly accessible design-specific test data register shall be specified and shall be greater than 0; if the <register> appears in more than one <register association>, the first appearance shall define the length, and subsequent appearances shall either not define it again or define it identically.

d) All instructions, the names of which appear as the value of an <instruction name> element in any <instruction capture> element of the <register string>, shall appear as an <instruction name> element in an <opcode description> element in the <instruction opcode stmt> (see B.8.11).

e) Any <instruction name> element shall appear in only one <instruction capture list> within the <register string>.

f) The <pattern> value in an <instruction capture> element shall contain the same number of bits as the <register> in the same <register association>.

B.8.14 Boundary-scan register description

The <boundary-scan register description> shall contain a list of boundary-scan register cells numbered 0 to LENGTH-1 (where the total number of cells in the register is LENGTH). The cells may be listed in any order, but all shall be defined. Cell 0 shall be that closest to TDO. The boundary-scan register cells can vary in design and purpose. Clause 11 shows many example cell designs, but many others are possible under the rules of this standard.

The characteristics of each cell design used in a component shall be specified before the cells can be referenced in the <boundary-scan register description>. For the example cell designs included in this standard, cell descriptions shall be contained in the Standard VHDL Package **STD_1149_1_2001** (see Clause B.9). Cells defined in this VHDL package shall be referenced through the simple set of names listed in Table B.3.

Table B.3—List of cells defined in the Standard VHDL Package and relevant figure numbers

Name	Figures[a]	Comments
BC_0	Special cell	Degenerate form[b]
BC_1	11-18, 11-30, 11-34c, 11-34d, 11-36c, 11-46d	Design usable for many functions
BC_2	11-14, 11-31, 11-35c, 11-35d, 11-37c, 11-38c, 11-39(output), 11-41c	*INTEST* unsupported on Output2
BC_3	11-15	Input or Internal only
BC_4	11-16, 11-17, 11-39(input)	Input, Observe-Only, Clock or Internal only
BC_5	11-46c	Combined Input/Control
BC_6	11-38d	Bidirectional, deprecated[c]
BC_7	11-37d	Bidirectional
BC_8	11-40, 11-41d	Simpler bidirectional, lacking *INTEST* support; observes the signal at the corresponding pin even while operating in output mode
BC_9	11-32	Output that observes the signal at the corresponding pin for *EXTEST* and observes the signal driven from the system logic for *INTEST* and *SAMPLE*
BC_10	11-33	Simpler output, lacking *INTEST* support; always monitors the signal at the corresponding pin instead of the signal driven from the system logic

[a]The suffix “c” is used to denote a control cell shown in a cited figure. The suffix “d” denotes a data cell.
[b]**BC_0** is a cell that captures the value specified by the rules of this standard and that captures a don't-care value whenever this standard allows options. It can be used whenever there is uncertainty about the exact behavior of a compliant cell.
[c]**BC_6** meets the requirements of this standard, but it lacks desirable features and is not recommended for use in new designs. The design for **BC_7** (see Figure 11-37d) is preferred.

The method of describing cells other than those depicted in this standard is described in B.10.2. When such cell designs are to be used, their descriptions shall be given in a user-supplied VHDL package and VHDL package body.

Several rules shall be observed when combining cells to create a boundary-scan register conformant to this standard. Adherence to some rules shall be checked during processing of the BSDL description of a component. For example, some cell designs may be used only on a component input. Some will not support the *INTEST* instruction—this is allowable if **INTEST** does not appear in the <instruction opcode stmt> (see B.8.11). Some cells require the aid of another to control 3-state enables or the direction of signal flow.

A very general cell design from this standard (see Figure 11-18) is shown in Figure B.2. Figure B.3 shows a symbolic representation of the same cell design.

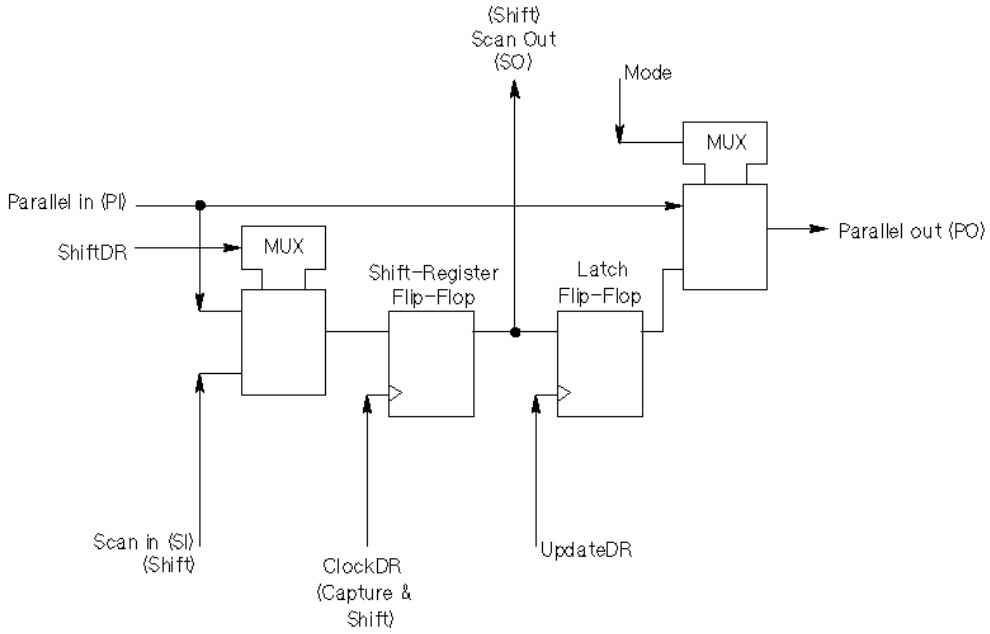


Figure B.2—Cell design corresponding to Figure 11-18

The design in Figure B.2 consists of a parallel input (PI), a parallel output (PO), a multiplexer controlled by a mode signal, and two flip-flops. The mode signal is a function of the current instruction. A serial input (SI) and a serial output (SO) form the shift path through the cell. The mode signal is a logic 0 or 1 that tells a cell what test function to perform (see Table 11-3). Note that the symbolic representation does not include

— The multiplexer controlled by signal ShiftDR;
— The mode signal and multiplexer; and
— The clock signals, ClockDR, and UpdateDR.

These parts of the cell design do not need to be considered in BSDL because the control and operation of boundary-scan register cells are fully defined by this standard. Thus, Figure B.3 is a full representation of the cell design shown in Figure B.2 for the purposes of BSDL. The parallel input and output are shown in the figure, and they are connected to various places, depending on the application. The two flip-flops are labeled CAP and UPD to represent their uses. The CAP flip-flop captures data from the system data input of the cell in the *Capture-DR* controller state and lies on the shift register path. The UPD flip-flop loads data from the CAP flip-flop in the *Update-DR* controller state. The shift path is shown because many such cells will be linked together in a shift chain that makes up the boundary-scan register. The shift path links only the CAP flip-flops.

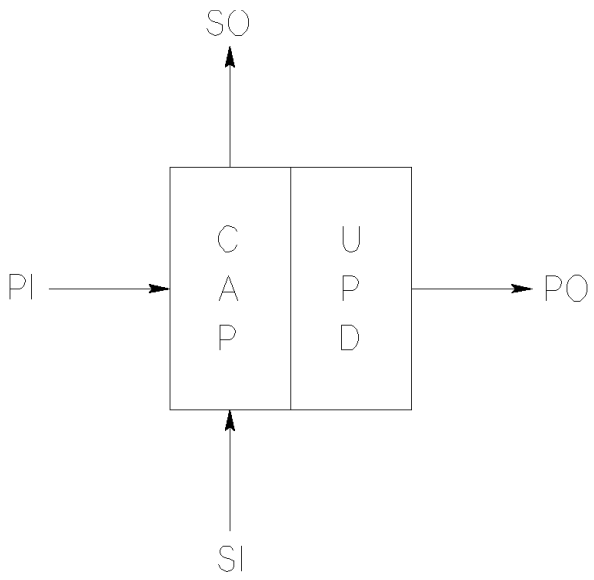


Figure B.3—A symbolic representation of a boundary-scan register cell

One cell design, shown in Figure 11-17 in this standard, is an observe-only cell and has a symbol with no UPD flip-flop (Figure B.4). This cell can be used at a system input pin and has the advantage of lower propagation delay in some implementations. However, it does not support the optional *INTEST* instruction if provided at a nonclock input.

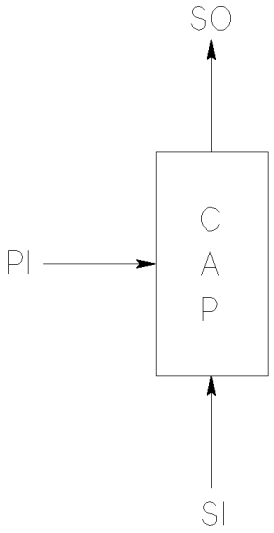


Figure B.4—A symbolic representation of a boundary-scan register cell without an update stage

The symbols in Figure B.5 and Figure B.6 show a bubble on top of the UPD flip-flop (indicating that the flip-flop may be preset to 1) or a bubble on the bottom (indicating that the flip-flop may be cleared to 0) when the *Test-Logic-Reset* controller state is entered. No signal connection is made to these bubbles.

NOTE—The resetting or clearing of such cells shall be consistent with Rule 11.6.1 p), which states that the value forced into the cell shall be the one that would disable the associated drivers.

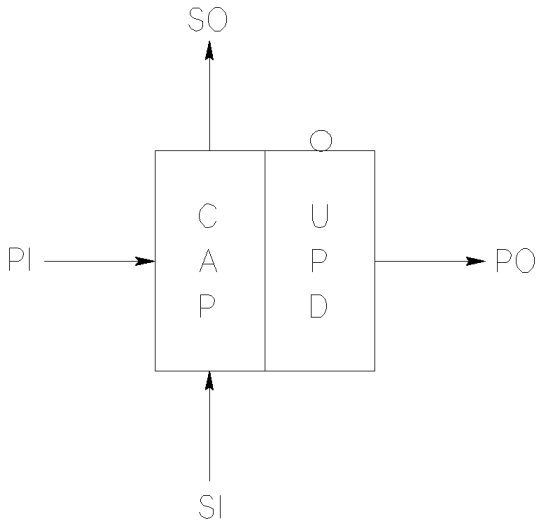


Figure B.5—A symbol for a boundary-scan register cell with preset

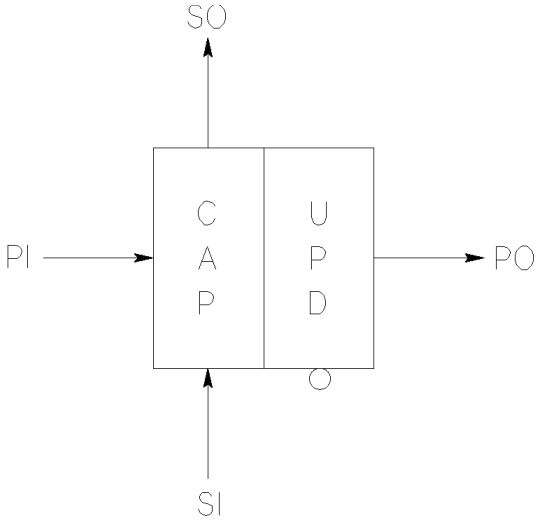


Figure B.6—A symbol for a boundary-scan register cell with clear

B.8.14.1 Syntax

<boundary-scan register description>::= <boundary length stmt> <boundary register stmt>

<boundary length stmt>::=
attribute BOUNDARY_LENGTH of <component name>**: entity is** <integer>**;**

<boundary register stmt>::= **attribute BOUNDARY_REGISTER of** <component name>
: entity is <cell table string>**;**

<cell table string>::= **"** <cell table> **"**
<cell table>::= <cell entry> { **,** <cell entry> }
<cell entry>::= <cell number> **(** <cell info> **)**
<cell number>::= <integer>
<cell info>::= <cell spec> [**,** <disable spec>]

<cell spec>::= <cell name> **,** <port ID or null> **,** <function> **,** <safe bit>
<cell name>::= <VHDL identifier>
<port ID or null>::= <port ID> | * (see B.6.3)
<function>::= **INPUT** | **OUTPUT2** | **OUTPUT3** | **CONTROL** | **CONTROLR** | **INTERNAL** | **CLOCK** | **BIDIR** | **OBSERVE_ONLY**
<safe bit>::= **0** | **1** | **X**
<disable spec>::= <ccell> , <disable value> , <disable result>
<ccell>::= <integer>
<disable value>::= **0** | **1**
<disable result>::= **Z** | **WEAK0** | **WEAK1** | **PULL0** | **PULL1** | **KEEPER**

B.8.14.2 Examples

```
attribute BOUNDARY_LENGTH of ttl74bct8374: entity is 18;
```

The <boundary length stmt> shall define the number (LENGTH) of cells in the boundary-scan register. This number shall match the number of <cell entry> elements in the <boundary register stmt>, which describes the structure of the boundary-scan register. Some cells may require two lines of description (see B.11.1.3).

```
attribute BOUNDARY_REGISTER of ttl74bct8374: entity is
--
--num cell  port/*  function safe [ccell disval rslt]
      --
"17  (BC_1, CLK,    input,   X)," &
"16  (BC_1, OC_NEG, input,   X)," &
"16  (BC_1, *,      control, 1)," &
. . .
. . .
"3   (BC_1, Q(5),   output3, X,   16,    1,    Z)," &
"2   (BC_1, Q(6),   output3, X,   16,    1,    Z)," &
"1   (BC_1, Q(7),   output3, X,   16,    1,    Z)," &
"0   (BC_1, Q(8),   output3, X,   16,    1,    Z)";
```

B.8.14.3 Cell entry elements

The <boundary register stmt> shall contain a string (<cell table string>) within which there is a list of elements (<cell entry>), each with two fields. The <cell entry> elements may be listed in any order, but all shall be listed.

— The <cell number> element shall be in the range from 0 to LENGTH-1.
— The <cell info> shall contain a list of either four or seven elements contained within parentheses. (In the above example, the elements are labeled—`cell`, `port/*`, `function`, `safe`, `ccell`, `disval`, and `rslt`—as indicated by the commented header.)

All <cell entry> elements shall include values for the first four elements of the second field. Only <cell entry> elements for cells that drive system outputs that can be set to an inactive drive state (e.g., open-collector or 3-state outputs) shall have the remaining three elements of the second field, which specify how the output may be disabled. If the <function> element is **OUTPUT3** or **BIDIR**, the last three elements shall be defined. If the <function> element is **BIDIR**, the action of placing the relevant driver in the inactive drive state shall be taken as equivalent to setting the cell to operate as a receiver. If the <function> element is **OUTPUT2**, the last three elements either shall or shall not be defined, depending on whether the described driver is an asymmetrical driver, e.g., open-collector (VHDL <port type> equal to out) or capable of actively driving both states (VHDL <port type> equal to **buffer**), respectively.

The <cell spec> and <disable spec> elements shall be as defined in the following subclauses.

B.8.14.3.1 The <cell name> element

This shall identify the cell design used. It shall match a cell described in the Standard VHDL Package or in a user-supplied VHDL package.

B.8.14.3.2 The <port id or null> element

This element shall identify the system input or output connected to a given cell. Any name supplied for this element shall match one specified in the <logical port description>. A cell serving as an output control or internal cell shall have an asterisk (*) supplied for this element. Either a <port name> element with the corresponding <port dimension> previously described as bit or a <subscripted port name> shall be supplied as the value of a <port ID or null> element.

B.8.14.3.3 The <function> element

This element shall define the primary function of the relevant cell. Table B.4 lists the possible values of the <function> element.

Table B.4—Function element values and meanings

Value	Meaning	Example figure in this standard[a]
INPUT	Control-and-observe cell Observe-only cell not supporting *INTEST*	11-18 11-16
CLOCK	Cell at a clock input	11-17
OUTPUT2	A cell that drives a 2-state (either symmetric or asymmetric) output	11-30
OUTPUT3	A cell that drives data to a 3-state output	11-34d
CONTROL	A cell that controls a 3-state enable or direction control	11-34c
CONTROLR	A control cell that is forced to its disable state in the *Test-Logic-Reset* controller state	11-36c
INTERNAL	Cell not associated with a device signal pin that captures constants “1”, “0”, or “X”	–
BIDIR	A reversible cell for a bidirectional pin	11-37d
OBSERVE_ONLY	A solitary observe-only cell on an input Additional cell observing any device signal pin	11-17

[a]The suffix “c” is used to denote a control cell shown in a cited figure. The suffix “d” denotes a data cell.

Notice that many of the cell designs of this standard are somewhat general, meaning that they can be used in more than one context. For example, the <function> element in a description of the cell depicted in Figure 11-30 can have as its value **INPUT**, **OUTPUT2**, **OUTPUT3**, **CONTROL**, or **INTERNAL**. The value of the <function> element has important implications in describing a given cell (see B.10.2).

An **INPUT** function shall indicate that the cell has observe capability (control-and-observe capability if *INTEST* is implemented in the device) and is connected to a system pin. This pin shall have a <pin type> value of **in** or **inout** only.

A **CLOCK** function shall indicate an observe-only cell that is connected to a system input clock pin that allows support of *INTEST* [see Rule 11.5.1 g) 1)].

An **OUTPUT2** function shall indicate that the cell (which must have control-and-observe capability) provides data for a 2-state (symmetric or asymmetric) driver and is connected to a system pin. This pin shall have a <pin type> value of **out**, **buffer**, or **inout**.

An **OUTPUT3** function shall indicate that the cell (which must have control-and-observe capability) provides data for a 3-state driver and is connected to a system pin. This pin shall have a <pin type> value of **out** or **inout**.

A **CONTROL** function shall indicate that the cell (which must have control-and-observe capability) provides output enable control and/or direction control to one or more output drivers or bidirectional pins. A **CONTROL** cell shall not be referenced to a system pin in the <cell spec> element; see B.11.1.3 for detail on system input pins that are used to control system output drivers.

A **CONTROLR** function shall be identical to a **CONTROL** function with the exception that it also has the capability to be reset or cleared when the TAP passes through the *Test-Logic-Reset* state (see Figure 11-36).

A **BIDIR** function shall indicate a control-and-observe cell that is connected to a system pin with <pin type> **inout**.

An **INTERNAL** function shall indicate that the cell is either a placeholder cell that has gone unused because of the programming of user-programmable logic, or a cell that sits at the interface between digital and analog portions of the core circuitry of a component [see Rule 11.4.1 b)]. An **INTERNAL** cell shall not be connected to a system pin.

An **OBSERVE_ONLY** function shall indicate that the cell (which has observe-only capability) is either a cell like **INPUT** without *INTEST* support capability or an additional cell that can monitor any kind of nonlinkage system pin not exempted by semantic checks B.8.8.3 d) (grouped ports), B.8.9.3 b) (scan port identification), or B.8.10.3 e) (compliance-enable description).

Clause 11 effectively classifies system pins as INPUT, CLOCK, TWO-STATE OUTPUT, THREE-STATE OUTPUT, and BIDIRECTIONAL. For a system pin classification of

a) INPUT pin, a cell shall have a <function> of **INPUT** or **OBSERVE_ONLY**. Additional cells with <function> **INPUT** or **OBSERVE_ONLY** may be connected to the INPUT pin. If INTEST is a supported instruction, there shall be at least one cell with <function> of **INPUT** connected to the INPUT pin.
b) CLOCK pin, there are two cases:
    1) At least one cell shall have a <function> of **CLOCK**. Additional cells with <function> **OBSERVE_ONLY** also may be connected to the pin. This case is used where external clocking must be supplied to support *INTEST* or *RUNBIST*.
    2) At least one cell shall have a <function> of **INPUT**. Additional cells with <function> **INPUT** (see Figure 11-11) or **OBSERVE_ONLY** also may be connected to the CLOCK pin. This case is used where clocking must be supplied by shifting to support *INTEST* [see Rule 11.5.1 g) 3)].
c) TWO-STATE OUTPUT pin, one cell shall have a <function> of **OUTPUT2**. Additional cells with <function> **OBSERVE_ONLY** also may be connected to the pin.
d) THREE-STATE OUTPUT pin, one cell shall have a <function> of **OUTPUT3**. Additional cells with <function> **OBSERVE_ONLY** also may be connected to the pin.

e) BIDIRECTIONAL pin, there are two cases:
   1) A single cell with <function> **BIDIR** shall be attached to the pin. Additional cells with <function> **OBSERVE_ONLY** also may be connected to the pin; or
   2) A two-cell structure shall be used to create bidirectionality; one of these cells shall have a <function> of either **OUTPUT2** or **OUTPUT3**, and the other cell shall have a <function> of either **INPUT** or **OBSERVE_ONLY**. Additional cells with <function> **OBSERVE_ONLY** also may be connected to the pin.

Clause 11 classifies system logic (different from system pin) inputs and outputs as INPUT, CLOCK, OUTPUT DATA, and OUTPUT CONTROL. For a system logic classification of

a) INPUT or CLOCK, there are two cases:
   1) Cell provisions shall be the same as noted for system INPUT or CLOCK pins, or
   2) For system logic receiving data from analog circuitry, cells with <function> **INTERNAL** shall be connected, but they shall not be referenced to a system pin in the <cell spec> element.
b) OUTPUT DATA or OUTPUT_CONTROL, there are two cases:
   1) Cell provisions shall be the same as noted for system pins above; additional cells with <function> **INTERNAL** also may be connected, but they shall not be referenced to a system pin in the <cell spec> element, or
   2) For system logic providing data to analog circuitry, cells with <function> **INTERNAL** shall be connected, but they shall not be referenced to a system pin in the <cell spec> element.

Clause 11 also specifies that cells may exist that are connected neither to system pins nor to system logic due to the programming of programmable system logic (see 11.8). Such cells shall have <function> values of **INTERNAL**.

B.8.14.3.4 The <safe bit> element

This element shall supply a value that should be loaded into the CAP flip-flop (and UPD flip-flop if it exists) of a given cell when board-level test generation software might otherwise choose a value randomly.

The <safe bit> value is not intended to force software to use particular values for cells. Rather, it provides values for cells where software would otherwise choose a 0 or 1 at random. An "**X**" shall signify that the value does not matter and that test generation software may assign either a 1 or a 0 in a case where there is no value that the algorithm requires.

For control cells, the <safe bit> value shall be that which turns off the associated drivers. Other examples where the <safe bit> value might be defined as "**0**" or "**1**" (rather than "**X**") are

— The value that an output should have during *INTEST* that minimizes driver current, or
— A preferred value to present to on-chip logic at a component input during *EXTEST.*

B.8.14.3.5 The <ccell> element

This element shall identify for the relevant <port ID> the <cell number> of the control cell that can disable the output.

B.8.14.3.6 The <disable value> element

This element shall give the value that must be scanned into the control cell identified by the previous <ccell> element to disable the port named by the relevant <port ID>.

B.8.14.3.7 The <disable result> element

The value of <port ID> within a given <cell entry> element is the name of a signal. If that signal can be disabled, the value of the <disable result> element within the same <cell entry> shall specify the condition of the driver of that signal when it is disabled. The permissible values shall be

— A high-impedance state (**Z**); or
— A weak "0" external pull down (**WEAK0**); or
— A weak "1" external pull up (**WEAK1**); or
— A weak "0" internal pull down (**PULL0**); or
— A weak "1" internal pull up (**PULL1**); or
— A "kept" state memory of the last strongly driven logic state (**KEEPER**)

The values **WEAK1** and **WEAK0** would be used for asymmetrical drivers, such as TTL open-collector or ECL open-emitter outputs, when a pull-down or a pull-up is external to the component. The values **PULL0** and **PULL1** would be used for asymmetrical drivers, such as TTL open-collector or ECL open-emitter outputs, when a pull-down or a pull-up is internal to the component. The value **KEEPER** would be used for drivers that maintain a weakly driven memory of the state last seen on the board network to which such a driver is connected.

NOTES

1—It must be emphasized that bus keepers generally do not retain a reliable logic state useful as part of a logic implementation. Indeed, any glitch or system noise on a "kept" bus may upset the state of any connected keepers. Drivers with bus keepers can be thought of as types that disable to a high-impedance state that always stays out of the forbidden voltage zone between defined low and high logic values. Just as a high-impedance state conveys no information, neither does a kept state. Implementers of board or system application software probably will choose the same treatment of the <disable result> values **KEEPER** and **Z**.

2—The keeper feature is an important parametric option in the design of a device's drivers. An IC vendor using such drivers would want to verify their action on an IC tester. By giving the ability for BSDL to denote the existence of such drivers, IC test software can automatically set up tests (at the logical level) for these features that are similar to hysteresis measurements. Of course, the analog parameters of a keeper, like other analog information, are not described in BSDL.

By processing the <boundary-scan register description>, it is possible for software to check that every non-linkage, non-TAP controller, non-compliance enable, non-grouped port name in the port statement has been named by a <port ID> of the <boundary-scan register description>. Missing <port ID> values (other than linkage, TAP controller ports, compliance-enable ports, and grouped ports) identify digital system signals lacking corresponding cells in the boundary-scan register, which indicates a noncompliant device or an error in entering the BSDL description.

B.8.14.4 Boundary-scan register semantic checks

a) The value of the <integer> (LENGTH) in the <boundary length stmt> shall be greater than zero.
b) Every <cell entry> element of the <cell table> shall include a <cell number> element with a value in the range from 0 to LENGTH - 1.
c) Every <integer> with a value between 0 and LENGTH - 1 shall appear as a <cell number> in some <cell entry> of the <boundary register stmt>.
d) Only a pair of merged cells (see B.11.1.3) shall correspond to two <cell entry> elements containing identical <cell number> elements in the <cell table>. Moreover,
    1) The only possible mergers shall be of cells with <function> equal to **INPUT** and cells with <function> equal to **OUTPUT2**, **OUTPUT3**, **CONTROL**, or **CONTROLR**.
    2) The value of the <cell name> element in both <cell entry> elements shall be equal.
    3) The <safe bit> values for these two cells shall be identical unless one value is **X**.
    4) The <data source> values of the <capture descriptor> values (see B.10.2) for these two cells shall be identical for all supported instructions.

e) Every <cell name> appearing in the <cell table> shall be the name of a cell described in either the Standard VHDL Package or a user-supplied VHDL package.

f) Any value of a <port ID> element in a <cell spec> that is not a <subscripted port name> shall be a <port name> in an <identifier list> element of a <pin spec> in the <logical port description> such that the value of the <port dimension> element in that <pin spec> is **bit**.

g) Any <port name> of a <port ID> appearing in a <cell spec> shall appear also in the <logical port description>; if a given <port ID> in a <cell spec> is a <subscripted port name>, its <subscript> shall be in the range of the <range> element in the <pin spec> element of the <logical port description> in which the <VHDL identifier> of that <subscripted port name> appears as a value of a <port name> element.

h) A <port ID or null> shall have the value asterisk (*) when, in the same <cell spec> element, the <function> element has the value **CONTROL**, **CONTROLR**, or **INTERNAL**.

i) Any <cell info> element containing a <function> element equal to **INPUT**, **CONTROL**, **CONTROLR**, **INTERNAL**, **OBSERVE_ONLY**, or **CLOCK** shall not also contain a <disable spec> element.

j) Any <cell entry> element containing a <function> element equal to **OUTPUT3** or **BIDIR** also shall contain a <disable spec> element.

k) Any <cell entry> element including a <function> element equal to **OUTPUT2** and also containing a <disable spec> element shall satisfy the following conditions:

   1) The value of <cell number> shall equal the value of <ccell>. (This implies that an **OUTPUT2** cell may control itself.)
   2) The value of the <disable value> shall be equivalent to the (weak) logical value of the <disable result>. That is, when the <disable value> is **0**, the <disable result> be **WEAK0** or **PULL0**, and when the <disable value> is **1**, the <disable result> shall be **WEAK1** or **PULL1**.

l) Any <cell entry> element containing a <function> element equal to **BIDIR** and a <ccell> element value equal to the value of the <cell number> element (implying a bidirectional cell that controls itself) shall have the value of the <disable value> equivalent to the (weak) logical value of the <disable result> element. That is, when the <disable value> is **0**, the <disable result> shall be **WEAK0** or **PULL0**, and when the <disable value> is **1**, the <disable result> shall be **WEAK1** or **PULL1**.

m) For any <cell entry> including a <function> element equal to **CONTROL** or **CONTROLR**, the value of the <cell number> element of that <cell entry> shall appear as the value of the <ccell> element of the <disable spec> element of other <cell entry> elements.

n) For any <cell entry> including a <function> element equal to **CONTROL** or **CONTROLR**, the value of the <safe bit> element of that <cell entry> shall be equal to the value of the <disable value> element of the <disable spec> element of the other <cell entry> elements that satisfy semantic check B.8.14.4 m) (i.e., the controlled cells).

o) The <ccell> element of a <disable spec> element shall have only the values permitted under the conditions of semantic checks B.8.14.4 k), B.8.14.4 l), and B.8.14.4 m).

p) When the value of <conformance identification> (see B.8.6) is **STD_1149_1_1993** or **STD_1149_1_2001**, if two distinct <disable spec> elements in the <boundary register stmt> have <ccell> elements with a common value, the values of the <disable value> elements of these two <disable spec> elements also shall be equal.

q) For any <port ID> element appearing in a <cell entry> element of the <boundary register stmt>, a <VHDL identifier> with the same value as the given <port ID> element shall appear in an <identifier list> of a <pin spec> in the <logical port description>.

r) If a <VHDL identifier> appears in an <identifier list> of a <pin spec> in the <logical port description>, and if the <pin type> appearing in that <pin spec> has the value **linkage**, the given <VHDL identifier> shall never appear as

   1) The <port ID> in any <cell entry> of the <boundary register stmt>, or
   2) The initial <VHDL identifier> of a <subscripted port name> serving as a <port ID> in any <cell entry> of the <boundary register stmt>

s) Excepting those elements explicitly mentioned in the following list, all <port ID> elements with a <VHDL identifier> appearing in an <identifier list> of a <pin spec> also including a <pin type> not

equal to **linkage** [in the <logical port description> (see B.8.3) of the BSDL description] shall appear as <port ID> elements in the <boundary register stmt>. Specifically exempted from this check are any <port ID> elements satisfying any of the following conditions:

1) Semantic check B.8.8.3 d) (grouped ports)
2) Semantic check B.8.9.3 b) (scan port identification)
3) Semantic check B.8.10.3 e) (compliance-enable description)

NOTE—This semantic check means that all nonexempted system pins shall be associated with cell(s) in the boundary-scan register. Further, this semantic check means that all exempted system pins shall not be associated with cell(s) in the boundary-scan register.

Semantic checks B.8.14.4 q), B.8.14.4 r), and B.8.14.4 s) state which <port ID> elements in the <logical port description> shall appear in the <boundary register stmt>, and vice versa. The next semantic checks state the properties that shall exist for <function> elements within <cell entry> elements.

t) It is allowed that additional <cell entry> elements with the <function> **OBSERVE_ONLY** may have <port ID> elements in common with any other <cell entry> obeying semantic check B.8.14.4 u).

NOTE—Additional **OBSERVE_ONLY** cells shall always monitor the state of a system I/O pin.

u) For any <port ID> element appearing in a <cell entry> element of the <boundary register stmt>, when the <pin type> in the <pin spec> of that <port ID> is
   1) **in**, the <function> of the <cell entry> shall be **INPUT**, **CLOCK**, or **OBSERVE_ONLY** only.
   2) **out**, the <function> of the <cell entry> shall be **OUTPUT2** or **OUTPUT3** only; furthermore, no other <cell entry> containing the same <port ID> shall have <function> **OUTPUT2** or **OUTPUT3**; also, when the <function> is **OUTPUT2**, the <cell entry> shall have a <disable spec> according to semantic check B.8.14.4 k).
   3) **buffer**, the <function> of the <cell entry> shall be **OUTPUT2** only, and the <cell entry> shall not contain a <disable spec>; furthermore, no other <cell entry> containing the same <port ID> shall have <function> **OUTPUT2**.
   4) **inout**, the <function> of the <cell entry> shall be **BIDIR**, **OUTPUT2**, **OUTPUT3**, **INPUT**, or **OBSERVE_ONLY** only; if the <function> value is **BIDIR**, no other <cell entry> containing the same <port ID> shall have the <function> **BIDIR**, **OUTPUT2**, or **OUTPUT3**; if the <function> of the <cell entry> is **OUTPUT2** or **OUTPUT3**, no other <cell entry> containing the same <port ID> shall exist with the <function> value of **OUTPUT2** or **OUTPUT3**, and at least one other <cell entry> containing the same <port ID> but a different <cell number> shall exist with the <function> value of **INPUT** or **OBSERVE_ONLY**, and vice versa.

v) The <function> in a <cell entry> shall be an existing <cell context> (see B.10.2) within the <capture descriptor> of the cell named by <cell name>.

w) If **INTEST** occurs as the value of an <instruction name> element in an <opcode description> element of the <instruction opcode stmt>, for each <port ID> element satisfying semantic check B.8.14.4 s), a <cell entry> shall exist that references that <port ID> *and* that possesses *INTEST support capability.*

NOTES

1—For this semantic check, a given <cell entry> does not possess *INTEST support capability* unless the <capture descriptor list> (see B.10.2) of the cell design named by the <cell name> element meets one of the following conditions:

—for a <function> element value of **INPUT**, **CLOCK**, **OUTPUT2**, **OUTPUT3**, **CONTROL**, or **CONTROLR**, a <capture descriptor> element contains a <cell context> element value that matches the <function> element value *and* has a <capture instruction> element value of **INTEST**; or,

—for a <function> element value of **BIDIR**, one <capture descriptor> element contains a <cell context> element value of **BIDIR_IN** *and* another <capture descriptor> element contains a <cell context> element value of **BIDIR_OUT**, *and* both such <capture descriptor> elements have a <capture instruction> element value of **INTEST**.

2—For this semantic check, a given <cell entry> does not possess *INTEST support capability* if its <function> element value is **OBSERVE_ONLY**. An **OBSERVE_ONLY** cell cannot by itself provide *INTEST support capability.*

See Clause B.11 for more information given by example for describing the boundary-scan register.

B.8.15 RUNBIST description

The goal of this portion of a BSDL description shall be to provide support for the *RUNBIST* instruction as specified within this standard. The intent is to describe only those aspects of the *RUNBIST* instruction that this standard specifies. In some cases, this may not completely define the Built-In Self-Test (BIST) operational environment. In such cases, additional information must be supplied externally.

Note that the following features of a BIST implementation are not supported explicitly by BSDL:

— Timing-related information (beyond active clock and number of clock cycles)
— Frequency and phase relationship(s) of clock(s)

B.8.15.1 Syntax

<runbist description>::=
attribute RUNBIST_EXECUTION of <component name>
: entity is " <runbist spec> **";**

<runbist spec>::= <wait spec> **,** <pin spec> **,** <signature spec>
<wait spec>::= **WAIT_DURATION (**<duration spec>**)**
<duration spec>::= <clock cycles list> | <time> [**,** <clock cycles list>]
<clock cycles list>::= <clock cycles> { **,** <clock cycles> }
<time>::= <real number>
<clock cycles>::= <port ID> <integer> (see B.6.3)
<pin spec>::= **OBSERVING** <condition> **AT_PINS**
<condition>::= **HIGHZ** | **BOUNDARY**
<signature spec>::= **EXPECT_DATA** <det pattern>
<det pattern>::= <bit> { <bit> }
<bit>::= **0** | **1**

A <det pattern> shall be a contiguous sequence of one or more **0** and **1** characters containing no spaces or format effectors. For example, **001100** and **110101** are legal. However **100 X00** is not legal because of the embedded space and the **X** character.

Time shall be specified in seconds (via the value of the <time> element). Where both time and clock cycles are specified, they shall be interpreted as the maximum of the time specified or the time required to apply the required number of clock cycles. Where more than one clock is specified, the duration shall be the time required for all of the clock inputs to receive the specified number of clock cycles.

B.8.15.2 Examples

Example 1

```
attribute RUNBIST_EXECUTION of BIST_IC1: entity is
      "Wait_Duration (1.0e-3)," &
      "Observing HIGHZ At_Pins," &
      "Expect_Data 010101";
```

In this example, the value of <time> in the <wait spec> is specified at 1 ms, which is the minimum duration the device needs to stay in the *Run-Test/Idle* controller state. Also, note that the output pins are forced to

high impedance, which implies that there is no need to initialize the update latches of the boundary-scan register.

Example 2

```
attribute RUNBIST_EXECUTION of BIST_IC2: entity is
      "Wait_Duration (TCK 23000)," &
      "Observing HIGHZ At_Pins," &
      "Expect_Data 010101";
```

In this example, the device needs to wait in the *Run-Test/Idle* controller state for the duration sufficient for the application of 23 000 clock cycles at TCK.

Example 3

```
attribute RUNBIST_EXECUTION of BIST_IC3: entity is
      "Wait_Duration (1.0e-3, TCK 23000)," &
      "Observing HIGHZ At_Pins," &
      "Expect_Data 010101";
```

In this example, <wait spec> is to be interpreted as 1 ms or the time required for TCK to receive 23 000 cycles, whichever is greater.

Example 4

```
attribute RUNBIST_EXECUTION of BIST_IC4: entity is
      "Wait_Duration (CLK 100000, SYSCK 24000)," &
      "Observing BOUNDARY At_Pins," &
      "Expect_Data 010101";
```

In this example, <wait spec> is to be interpreted as the time required for CLK and SYSCK to receive 100 000 and 24 000 clock cycles, respectively. Also note that the boundary-scan register is visible at the pins, indicating that it needs to be initialized before the execution of *RUNBIST*.

B.8.15.3 Semantic checks

a) The number of bits in the value of the <det pattern> element of the <signature spec> element shall be equal to the length of the register whose name appears in the <register> element of that <register association> element of the <register access description> in which **RUNBIST** appears as the value of an <instruction name> element.
   1) If the value of the associated <register> element is **BOUNDARY**, the register length shall be specified by the value of the <integer> element of the <boundary length stmt>.
   2) If the value of the associated <register> element is *not* **BOUNDARY**, the register length shall be specified by the explicitly defined value of the <integer> element in that same <register> element.

b) Any value of <port ID> in the <wait spec> statement shall
   1) Appear as the value of <port ID> in the <TCK stmt> of the BSDL description (see B.8.9); or
   2) Appear as the value of <port ID> in a <cell spec> of the <boundary register stmt> in which the <function> element has the value **CLOCK** (see B.8.14).

c) If the <runbist description> statement occurs in a BSDL description, **RUNBIST** shall be the value of some <instruction name> element in the <opcode table> of the <instruction opcode stmt>.

d) Values of <time> and <clock cycles> shall be greater than 0.

e) A given <port ID> shall not appear more than once in the <runbist description> element.

NOTE—The existence of **RUNBIST** in the **INSTRUCTION_OPCODE** table shall not require <runbist description> to be specified in a BSDL description.

B.8.16 INTEST description

The goal of this portion of a BSDL description shall be to describe

— How test patterns are to be applied to the component when the *INTEST* instruction is selected (i.e., the source of clock pulses for the component and the time for which the test logic must remain in the *Run-Test/Idle* controller state to permit execution of each applied test); and
— The external behavior of the component while the *INTEST* instruction is selected.

Note that the test patterns themselves are not specified and are assumed to be provided by an alternative method not specified in this annex. For *INTEST*, the duration shall be not the duration for the entire test (as is the case of *RUNBIST*) but the time required for the application of a single vector. With the application of each vector via the boundary-scan register, this standard permits the device to execute a single step of the operation that may require several clock cycles to complete. Otherwise, the interpretation of <pin spec> shall be identical to that in **RUNBIST_EXECUTION**. The syntax of the <wait spec> and <pin spec> elements shall be as given in B.8.15.

B.8.16.1 Syntax

<intest description>::=
attribute INTEST_EXECUTION of <component name>
: entity is " <intest spec> **";**

<intest spec>::= <wait spec> **,** <pin spec> (see B.8.15)

B.8.16.2 Examples

Example 1

```
attribute INTEST_EXECUTION of IC1: entity is
      "Wait_Duration (1.0e-3)," &
      "Observing HIGHZ At_Pins";
```

In this example, the value of <time> in the <wait spec> is specified at 1 ms, which is the minimum duration the device needs to stay in the *Run-Test/Idle* controller state. Also, note that the output pins are forced to high impedance.

Example 2

```
attribute INTEST_EXECUTION of IC2: entity is
      "Wait_Duration (TCK 250)," &
      "Observing HIGHZ At_Pins";
```

In this example, the device needs to wait in the *Run-Test/Idle* controller state for a duration sufficient for the application of 250 clock cycles of TCK to permit the device to complete one single step of operation.

Example 3

```
attribute INTEST_EXECUTION of IC3: entity is
      "Wait_Duration (CLK 100, SYSCK 200)," &
      "Observing BOUNDARY At_Pins";
```

In this example, <wait spec> is to be interpreted as the time required for CLK and SYSCK to receive 100 and 200 clock cycles, respectively. Also note that the state of the pins is controlled by the data held in the boundary-scan register.

B.8.16.3 Semantic checks

a) Any value of <port ID> in the <wait spec> statement shall
   1) Appear as the value of <port ID> in the <TCK stmt> of the BSDL description (see B.8.9); or,
   2) Appear as the value of <port ID> in a <cell spec> of the <boundary register stmt> in which the <function> element has the value **CLOCK** (see B.8.14).
b) If the <intest description> statement occurs in a BSDL description, **INTEST** shall be the value of some <instruction name> element in the <opcode table> of the <instruction opcode stmt>.
c) Values of <time> and <clock cycles> shall be greater than 0.
d) A given value of <port ID> shall not appear more than once in the <intest description> element.

NOTE—The existence of **INTEST** in the **INSTRUCTION_OPCODE** table shall not require <intest description> to be specified in a BSDL description.

B.8.17 User extensions to BSDL

Optional BSDL extensions shall provide a way to expand BSDL for proprietary needs without losing compatibility with the general definition of BSDL. The Standard VHDL Package **STD_1149_1_2001** defines a VHDL subtype **BSDL_EXTENSION** (as originally defined in Standard VHDL Package **STD_1149_1_1994**). It allows the user to define foreign attributes as being “BSDL extensions.” These generally may be ignored by a BSDL parser. BSDL extensions shall appear in an entity description as the last portion before the (optional) **DESIGN_WARNING** (see B.8.18). In this manner, they may reference any data items defined previously.

B.8.17.1 Syntax

<BSDL extensions>::= <BSDL extension> { <BSDL extension> }

<BSDL extension>::= <extension declaration> | <extension definition>

<extension declaration>::= **attribute** <extension name> **: BSDL_EXTENSION;**

<extension definition>::= **attribute** <extension name> **of** <component name>
: entity is <extension parameter string>**;**

<extension name>::= <entity defined name> | <VHDL package defined name>
<entity defined name>::= <VHDL identifier>
<VHDL package defined name>::= <VHDL identifier>
<extension parameter string>::= <string>

An <extension definition> shall appear after its corresponding <extension declaration>. The <extension declaration> may appear in the description itself or in a user-supplied VHDL package. If several BSDL extensions exist in the description, they may be intermixed in any manner as long as the declaration of an attribute precedes the definition of that attribute. The ability to define BSDL extensions in user-supplied VHDL packages allows global definition of extensions.

B.8.17.2 Examples

Example 1

```
      Package Global_extension is -- An example BSDL extension package
                                  -- Does not define boundary cells,
                                  --   just extensions

      use STD_1149_1_2001.all;

   -- Deferred constant declarations go here, if any (see Clause B.10)

      attribute First_extension  : BSDL_EXTENSION; -- Declare BSDL
      attribute Second_extension : BSDL_EXTENSION; --  extensions here
      attribute Third_extension  : BSDL_EXTENSION;

      end Global_extension;

      package body Global_extension is

   -- Deferred constant definitions go here, if any (see Clause B.10)

      end Global_extension;
```

In this example, a user-supplied VHDL package containing a BSDL extension is given; this package will be referenced by the entity of the next example.

Example 2

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

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

      use STD_1149_1_2001.all;
      use Global_extension.all; -- Get declarations of
                                --   global extensions
```

. . . Many deleted lines . . .

```
      -- Local declarations

      attribute Local_extension1: BSDL_extension; --Declare local BSDL
      attribute Local_extension2: BSDL_extension; --  extensions here

      -- Now, define some proprietary extensions that were declared
      --   in package - Global_Extension

      attribute First_extension of example:entity is   -- Define attr.
            " String of data " &                 -- (global extension)
            " in proprietary form. ";
      attribute Second_extension of example:entity is
            " More data, etc. ";
```

```
      -- Local definition

      attribute Local_extension1 of example:entity is  -- Define attr.
            " Finally defined ";               -- (local extension)

      -- Optional design warning still located here --

end example;
```

In this example, an entity is shown that uses global extensions as well as local extensions defined in the entity. Note that not all declared extensions are defined (e.g., `Third_extension`).

B.8.17.3 Semantic checks

a) Any <VHDL identifier> appearing as a value of the <extension name> element in an <extension definition> shall appear also as the value of the <extension name> element of an <extension declaration> that occurs earlier in the BSDL description or in a given VHDL package. In the case in which the <extension declaration> occurs in a VHDL package, appearance of the corresponding <extension definition> in the BSDL description shall be considered as an “appearance after” the given <extension declaration>.
b) Each <extension name> value in an <extension declaration> shall be unique even if the <extension name> values are defined in separate places, i.e., in separate user-supplied VHDL packages or one in the BSDL entity and one in a user-supplied VHDL package.

If a value of an <extension name> element appearing in an <extension declaration> never appears as the value of an <extension name> element in any <extension definition> within a given BSDL description, no error shall be generated by an application parsing that description.

NOTE—When inventing names for <extension name> elements, take care to assure uniqueness of the names with respect to names created in other organizations that are also inventing extensions by avoiding common names that might be thought of by others. A company name could be appended to the name to maximize uniqueness.

B.8.18 Design warning

A component designer may know of situations in which the system usage of a component can be subverted via the boundary-scan feature and cause circuit problems. As a simple example, a component may have dynamic system logic that requires clocking to maintain its state. Thus, clocking must be maintained when bringing the component out of system mode and into test mode for *INTEST*. The DESIGN_WARNING attribute shall be assigned a string message to alert future consumers of the potential for problems.

B.8.18.1 Syntax

<design warning>::=
attribute DESIGN_WARNING of <component name> **: entity is** <string> **;**

B.8.18.2 Examples

```
      attribute DESIGN_WARNING of My_IC:entity is
            "Dynamic device, " &
            "maintain clocking for INTEST.";
```

This warning is for application-specific display purposes only. It shall be a textual message of arbitrary length with no specified syntax and is not intended for software analysis.

B.8.18.3 Semantic checks

No semantic checks are necessary.

B.9 The Standard VHDL Package STD_1149_1_2001

The following shall be the complete content of Standard VHDL Package STD_1149_1_2001. Note that both the VHDL package and the VHDL package body are shown. This information shall define the basis of BSDL and typically would be write-protected by a system administrator. An explanation of the cell definitions (e.g., **BC_1**, **BC_2**, etc.) in the package body is given in B.10.2. BSDL descriptions that use <standard VHDL package identifier> **STD_1149_1_2001** shall be processed using this Standard VHDL Package.

NOTE—Where figures are cited, the suffix “c” is used to denote a control cell. The suffix “d” denotes a data cell.

```
package STD_1149_1_2001 is

-- Give component conformance declaration

attribute COMPONENT_CONFORMANCE : string;
-- Give pin mapping declarations

attribute PIN_MAP : string;
subtype PIN_MAP_STRING is string;

-- Give TAP control declarations

type CLOCK_LEVEL is (LOW, BOTH);
type CLOCK_INFO  is record
  FREQ : real;
  LEVEL: CLOCK_LEVEL;
end record;

attribute  TAP_SCAN_IN   : boolean;
attribute  TAP_SCAN_OUT  : boolean;
attribute  TAP_SCAN_CLOCK: CLOCK_INFO;
attribute  TAP_SCAN_MODE : boolean;
attribute  TAP_SCAN_RESET: boolean;

-- Give instruction register declarations

attribute  INSTRUCTION_LENGTH : integer;
attribute  INSTRUCTION_OPCODE : string;
attribute  INSTRUCTION_CAPTURE : string;
attribute  INSTRUCTION_PRIVATE : string;

-- Give ID and USER code declarations

type ID_BITS is ('0', '1', 'x', 'X');
type ID_STRING is array (31 downto 0) of ID_BITS;
attribute IDCODE_REGISTER  :  ID_STRING;
attribute USERCODE_REGISTER:  ID_STRING;

-- Give register declarations
```

```
attribute REGISTER_ACCESS : string;

-- Give boundary cell declarations

type BSCAN_INST is (EXTEST, SAMPLE, INTEST);
type CELL_TYPE is (INPUT, INTERNAL, CLOCK, OBSERVE_ONLY,
                   CONTROL, CONTROLR, OUTPUT2,
                   OUTPUT3, BIDIR_IN, BIDIR_OUT);
type CAP_DATA is (PI, PO, UPD, CAP, X, ZERO, ONE);
type CELL_DATA is record
  CT : CELL_TYPE;
  I  : BSCAN_INST;
  CD : CAP_DATA;
end record;
type CELL_INFO is array (positive range <>) of CELL_DATA;

-- Boundary cell deferred constants (see package body)

constant BC_0  : CELL_INFO;
constant BC_1  : CELL_INFO;
constant BC_2  : CELL_INFO;
constant BC_3  : CELL_INFO;
constant BC_4  : CELL_INFO;
constant BC_5  : CELL_INFO;
constant BC_6  : CELL_INFO;
constant BC_7  : CELL_INFO;
constant BC_8  : CELL_INFO;
constant BC_9  : CELL_INFO;
constant BC_10  : CELL_INFO;

-- Boundary register declarations

attribute BOUNDARY_LENGTH : integer;
attribute BOUNDARY_REGISTER : string;

-- Miscellaneous

attribute PORT_GROUPING : string;
attribute RUNBIST_EXECUTION : string;
attribute INTEST_EXECUTION : string;
subtype BSDL_EXTENSION is string;
attribute COMPLIANCE_PATTERNS : string;
attribute DESIGN_WARNING : string;

end STD_1149_1_2001;  -- End of 1149.1-2001 Package


package body STD_1149_1_2001 is   -- Standard boundary cells

-- Generic cell capturing minimum allowed data

constant BC_0 : CELL_INFO :=
 ((INPUT,   EXTEST,  PI),      (OUTPUT2,  EXTEST,  X),
  (INPUT,   SAMPLE,  PI),      (OUTPUT2,  SAMPLE,  PI),
```

```
  (INPUT,   INTEST,  X),       (OUTPUT2,  INTEST,  PI),
  (OUTPUT3, EXTEST,  X),       (INTERNAL, EXTEST,  X),
  (OUTPUT3, SAMPLE,  PI),      (INTERNAL, SAMPLE,  X),
  (OUTPUT3, INTEST,  PI),      (INTERNAL, INTEST,  X),
  (CONTROL, EXTEST,  X),       (CONTROLR, EXTEST,  X),
  (CONTROL, SAMPLE,  PI),      (CONTROLR, SAMPLE,  PI),
  (CONTROL, INTEST,  PI),      (CONTROLR, INTEST,  PI),
  (BIDIR_IN,EXTEST,  PI),      (BIDIR_OUT, EXTEST, X ),
  (BIDIR_IN,SAMPLE,  PI),      (BIDIR_OUT, SAMPLE, PI),
  (BIDIR_IN,INTEST,  X ),      (BIDIR_OUT, INTEST, PI),
  (OBSERVE_ONLY, SAMPLE, PI),  (OBSERVE_ONLY, EXTEST, PI) );

-- Description for f11-18, f11-30, f11-34c, f11-34d, f11-36c, f11-46d

constant BC_1 : CELL_INFO :=
 ((INPUT,   EXTEST,  PI),  (OUTPUT2,  EXTEST,  PI),
  (INPUT,   SAMPLE,  PI),  (OUTPUT2,  SAMPLE,  PI),
  (INPUT,   INTEST,  PI),  (OUTPUT2,  INTEST,  PI),
  (OUTPUT3, EXTEST,  PI),  (INTERNAL, EXTEST,  PI),
  (OUTPUT3, SAMPLE,  PI),  (INTERNAL, SAMPLE,  PI),
  (OUTPUT3, INTEST,  PI),  (INTERNAL, INTEST,  PI),
  (CONTROL, EXTEST,  PI),  (CONTROLR, EXTEST,  PI),
  (CONTROL, SAMPLE,  PI),  (CONTROLR, SAMPLE,  PI),
  (CONTROL, INTEST,  PI),  (CONTROLR, INTEST,  PI) );

-- Description for f11-14, f11-31, f11-35c, f11-35d, f11-37c,
--   f11-38c, f11-39(output) and f11-41c

constant BC_2 : CELL_INFO :=
 ((INPUT,   EXTEST,  PI),  (OUTPUT2, EXTEST,   UPD),
  (INPUT,   SAMPLE,  PI),  (OUTPUT2, SAMPLE,   PI),
  (INPUT,   INTEST,  UPD),  -- Intest on output2 not supported
  (OUTPUT3, EXTEST,  UPD), (INTERNAL, EXTEST,  PI),
  (OUTPUT3, SAMPLE,  PI),  (INTERNAL, SAMPLE,  PI),
  (OUTPUT3, INTEST,  PI),  (INTERNAL, INTEST,  UPD),
  (CONTROL, EXTEST,  UPD), (CONTROLR, EXTEST,  UPD),
  (CONTROL, SAMPLE,  PI),  (CONTROLR, SAMPLE,  PI),
  (CONTROL, INTEST,  PI),  (CONTROLR, INTEST,  PI) );

-- Description for f11-15

constant BC_3 : CELL_INFO :=
 ((INPUT, EXTEST,  PI),    (INTERNAL, EXTEST,  PI),
  (INPUT, SAMPLE,  PI),    (INTERNAL, SAMPLE,  PI),
  (INPUT, INTEST,  PI),    (INTERNAL, INTEST,  PI) );

-- Description for f11-16, f11-17, f11-39(input)

constant BC_4 : CELL_INFO :=
 ((INPUT, EXTEST,  PI),        -- Intest on input not supported
  (INPUT, SAMPLE,  PI),
  (OBSERVE_ONLY, EXTEST, PI),
  (OBSERVE_ONLY, SAMPLE, PI),  -- Intest on observe_only not supported
  (CLOCK, EXTEST,  PI),   (INTERNAL, EXTEST,  PI),
```

```
  (CLOCK, SAMPLE,  PI),   (INTERNAL, SAMPLE,  PI),
  (CLOCK, INTEST,  PI),   (INTERNAL, INTEST,  PI) );

-- Description for f11-46c, a combined input/control

constant BC_5 : CELL_INFO :=
 ((INPUT, EXTEST,  PI),   (CONTROL, EXTEST,  PI),
  (INPUT, SAMPLE,  PI),   (CONTROL, SAMPLE,  PI),
  (INPUT, INTEST,  UPD),  (CONTROL, INTEST,  UPD) );

-- Description for f11-38d, a reversible cell
-- !! Not recommended; replaced by BC_7 below !!

constant BC_6 : CELL_INFO :=
 ((BIDIR_IN, EXTEST,  PI),  (BIDIR_OUT, EXTEST,  UPD),
  (BIDIR_IN, SAMPLE,  PI),  (BIDIR_OUT, SAMPLE,  PI),
  (BIDIR_IN, INTEST,  UPD), (BIDIR_OUT, INTEST,  PI) );

-- Description for f11-37d, self monitor reversible
-- !! Recommended over cell BC_6 !!

constant BC_7 : CELL_INFO :=
 ((BIDIR_IN, EXTEST,  PI),  (BIDIR_OUT, EXTEST,  PO),
  (BIDIR_IN, SAMPLE,  PI),  (BIDIR_OUT, SAMPLE,  PI),
  (BIDIR_IN, INTEST,  UPD), (BIDIR_OUT, INTEST,  PI) );

-- Description for 11-40, f11-41d

constant BC_8 : CELL_INFO :=
 -- Intest on bidir not supported
 ((BIDIR_IN, EXTEST,  PI),  (BIDIR_OUT, EXTEST,  PO),
  (BIDIR_IN, SAMPLE,  PI),  (BIDIR_OUT, SAMPLE,  PO) );

-- Description for f11-32

constant BC_9 : CELL_INFO :=
 -- Self-monitoring output that supports Intest
 ((OUTPUT2, EXTEST,  PO),  (OUTPUT3, EXTEST,  PO),
  (OUTPUT2, SAMPLE,  PI),  (OUTPUT3, SAMPLE,  PI),
  (OUTPUT2, INTEST,  PI),  (OUTPUT3, INTEST,  PI) );

-- Description for f11-33

constant BC_10 : CELL_INFO :=
 -- Self-monitoring output that does not support Intest
 ((OUTPUT2, EXTEST,  PO),  (OUTPUT3, EXTEST,  PO),
  (OUTPUT2, SAMPLE,  PO),  (OUTPUT3, SAMPLE,  PO) );

end STD_1149_1_2001;  -- End of IEEE Std 1149.1-2001 Package Body
```

B.10 User-supplied VHDL packages

B.10.1 User-supplied VHDL package structure

A user-supplied VHDL package shall be used to express the behavior of user-designed boundary-scan register cells. It has a VHDL package section and a VHDL package body section. The VHDL package section is abbreviated compared to the Standard VHDL Package, since the definitions of BSDL are supplied in the Standard VHDL Package specified by the <standard use statement>. The names of the cells that are defined in the VHDL package body shall be given (they are called deferred constants).

NOTE—When writing a user-supplied VHDL package, it is possible to create an error if a later (e.g., 2001) construct such as a **BSDL_EXTENSION** is referenced that is not defined in an earlier-defined (e.g., 1990) Standard VHDL Package specified in the <standard use statement>. To avoid such an error, a 2001 user-supplied package should reference **STD_1149_1_2001**.

B.10.1.1 Syntax

<user package>::= **package** <user package name> **is**
<standard use statement>
{ <extension declaration> } (see B.8.17)
{ <deferred constant> }
end <user package name> **;**

<user package body>::= **package body** <user package name> **is**
<standard use statement>
{ <cell description constant> } (see B.10.2)
end <user package name> **;**

<user package name>::= <VHDL identifier>
<deferred constant>::= **constant** <cell name> **: CELL_INFO;**
<cell name>::= <VHDL identifier>

For <cell description constant>, see B.10.2.

B.10.1.2 Examples

For an example, see B.10.3.

B.10.1.3 Semantic checks

a) The <user package name> value shall be unique.
b) All <cell name> values shall be unique.
c) The <user package name> value in the <user package> shall match the <user package name> value in the <user package body>.

B.10.2 Cell description constants

The syntax of cell description constants shall be as given in B.10.2.1.

B.10.2.1 Syntax

<cell description constant>::=
constant <cell name> **: CELL_INFO := (** <capture descriptor list> **) ;**

<cell name>::= <VHDL identifier>
<capture descriptor list>::= <capture descriptor> { , <capture descriptor> }
<capture descriptor>::= **(** <cell context> **,** <capture instruction> **,** <data source> **)**
<cell context>::= **INPUT** | **OUTPUT2** | **OUTPUT3** | **INTERNAL** | **CONTROL** | **CONTROLR** | **CLOCK** | **BIDIR_IN** | **BIDIR_OUT** | **OBSERVE_ONLY**
<capture instruction>::= **EXTEST** | **SAMPLE** | **INTEST**
<data source>::= **PI** | **PO** | **CAP** | **UPD** | **ZERO** | **ONE** | **X**

NOTE—Although the standard-defined instruction *PRELOAD* does operate the boundary-scan register, it is not listed as a <capture instruction> element since all cells capture unspecified data ("**X**") when *PRELOAD* is in effect (see 8.7).

A cell description constant shall be a specific VHDL constant record made up of a variable number of data triples containing VHDL enumerated types. For example, a capture descriptor looks like the following:

B.10.2.2 Examples

Example 1

```
(INPUT, EXTEST, PI)
```

This can be read as, "for this cell used as an **INPUT** cell, while *EXTEST* is in effect, the capture flip-flop loads the Parallel Input (**PI**) data at the *Capture-DR* controller state."

Example 2

```
(BIDIR_IN, INTEST, UPD)
```

This can be read as, "for this cell used as a bidirectional cell acting as an input (**BIDIR_IN**), while *INTEST* is in effect, the capture flip-flop loads the value of the Update flip-flop (or latch) data (**UPD**) at the *CAPTURE-DR* controller state."

Example 3

```
(OUTPUT2, SAMPLE, PI)
```

This can be read as, "for this cell used as a (2-state) output cell (**OUTPUT2**), while *SAMPLE* is in effect, the capture flip-flop loads the Parallel Input (**PI**) data at the *Capture-DR* controller state."

Example 4

```
(OUTPUT3, EXTEST, ZERO)
```

This can be read as, "for this cell used as a (3-state) output cell (**OUTPUT3**), while *EXTEST* is in effect, the capture flip-flop loads a logic 0 (**ZERO**) at the *Capture-DR* controller state."

B.10.2.3 Cell context values

Table B.5 lists <cell context> values.

With the exception of <cell context> values of **BIDIR_IN** and **BIDIR_OUT**, all the <cell context> values in Table B.5 shall map onto the like-named <function> values in Table B.4 and supporting text (see B.8.14.3.3). The <cell context> values of **BIDIR_IN** and **BIDIR_OUT** both shall map onto the <function> value **BIDIR**. The behavior of a **BIDIR** cell shall be dependent on whether it is currently set to be driving

Table B.5—Cell context element values and meanings

Cell context value	Meaning
INPUT	Control-and-observe input cell
CLOCK	Observe-only cell for clock pins (supports *INTEST* instruction)
OUTPUT2	2-state output cell
OUTPUT3	3-state output cell
CONTROL	Output enable or direction control cell
CONTROLR	**CONTROL** with preset/clear at *Test-Logic-Reset*
BIDIR_IN	Single-cell bidirectional pin acting as input
BIDIR_OUT	Single-cell bidirectional pin acting as output
INTERNAL	Internal cell, not associated with a system pin
OBSERVE_ONLY	Observe-only cell, associated with a system pin

data out (**BIDIR_OUT**) or receiving data in (**BIDIR_IN**), as determined by the data value contained in the controlling cell identified by the <ccell> value.

B.10.2.4 Data source values

Table B.6 gives the values for <data source>. Figure B.7 gives a general model of the data source possibilities.

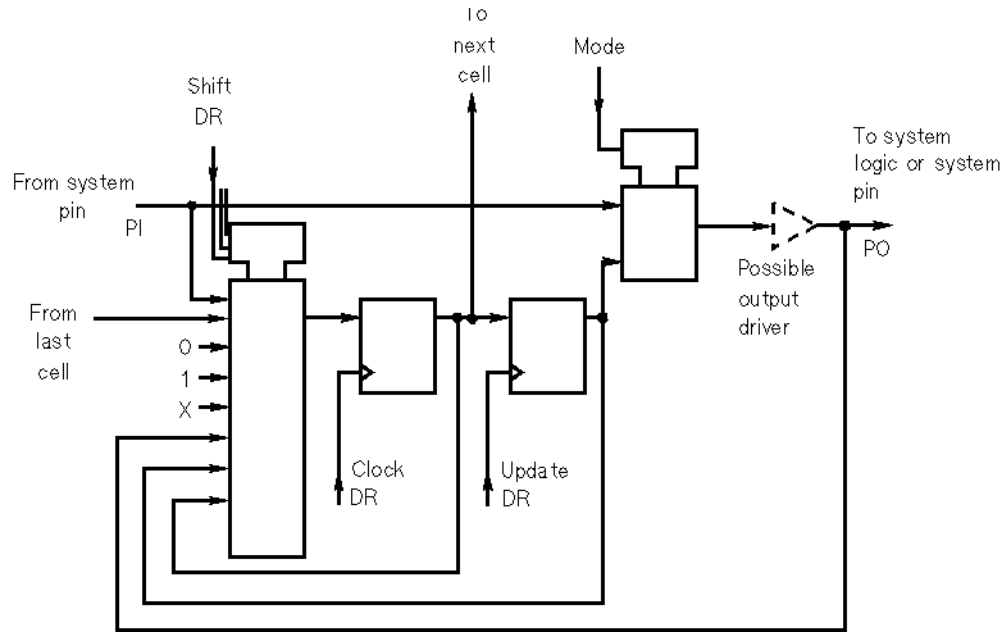


Figure B.7—A general model of a boundary-scan register cell showing CAP inputs

It is important to know the context of a cell to know how to interpret the data source. For example, the cell design in this standard at Figure 11-18 (called **BC_1**) can be used as an input cell (**INPUT**), an output cell for a 2-state pin (**OUTPUT2**), a 3-state pin (**OUTPUT3**), an internal cell (**INTERNAL**), or a control cell

Table B.6—Data source element values and meanings

Data source value	Meaning
PI	Parallel input
PO	Parallel output (the output pad if a driver is present)
CAP	Capture flip-flop data
UPD	Update flip-flop (or latch) data
ZERO	Constant “0”
ONE	Constant “1”
X	Unknown data

(**CONTROL**). This context determines how software shall interpret **PI** and **PO**. If the cell is used as an input (or is a bidirectional cell acting as an input), **PI** shall be interpreted as a system pin whose data is being captured. If the cell is used as an output (or a bidirectional cell acting as an output), **PI** shall be interpreted as the output from the system logic; during *EXTEST*, the cell would capture **X** unless a full simulation of the system logic were used to predict the system logic output. If the cell is used as an output, **PO** shall be the system pin; during *EXTEST*, the cell would capture board levels outside the component. When the cell is used as an input, **PO** will capture **X**.

B.10.2.4.1 Semantic checks

In the following tables, an “L” shall indicate legality. An “M” shall indicate legality in the case of merged cells only. An “A” shall indicate legality when the cell is an additional cell not mandated by this standard. A <capture descriptor> shall be a (<cell context> <capture instruction> <data source>) element. An example of an illegal <capture descriptor> is (**INPUT**, **EXTEST**, **UPD**).

a) For a <capture descriptor> element with a <cell context> element equal to **INPUT**, **CLOCK**, or **BIDIR_IN**, legal <data source> values for given <capture instruction> elements shall be as given in Table B.7.

Table B.7—Legal capture sources for <cell context> of INPUT, CLOCK, and BIDIR_IN

<capture instruction>	PI	PO	UPD	CAP	X	ZERO	ONE
EXTEST	L	–	–	–	–	–	–
SAMPLE	L	–	–	–	–	–	–
INTEST	L	L	L	L	L	L	L

b) For a <capture descriptor> element with a <cell context> element equal to **OUTPUT2**, **OUTPUT3**, or **BIDIR_OUT**, legal <data source> values for given <capture instruction> elements shall be as given in Table B.8.

c) For a <capture descriptor> element with a <cell context> element equal to **CONTROL** or **CONTROLR**, legal <data source> values for given <capture instruction> elements shall be as given in Table B.9.

Table B.8—Legal capture sources for <cell context> of OUTPUT2, OUTPUT3, and BIDIR_OUT

<capture instruction>	PI	PO	UPD	CAP	X	ZERO	ONE
EXTEST	L	L	L	L	L	L	L
SAMPLE	L	L[a]	–	–	–	–	–
INTEST	L	–	–	–	–	–	–

[a]This combination becomes legal with this 2001 version of BSDL. Beginning with edition IEEE Std 1149.1-2001, this standard now allows an output of the system logic to be sampled at the data output (pin) of the associated output buffer as well as at the data input of the associated output buffer [see Rules 11.6.1 a) and 11.6.1 h)].

Table B.9—Legal capture sources for <cell context> of CONTROL and CONTROLR

<capture instruction>	PI	PO	UPD	CAP	X	ZERO	ONE
EXTEST	L	L	L	L	L	L	L
SAMPLE	L	–	–	–	–	–	–
INTEST	L	–	M	–	–	–	–

d) For a <capture descriptor> element with a <cell context> element equal to **INTERNAL**, legal <data source> values for given <capture instruction> elements shall be as given in Table B.10.

Table B.10—Legal capture sources for <cell context> of INTERNAL

<capture instruction>	PI	PO	UPD	CAP	X	ZERO	ONE
EXTEST	L	L	L	L	L	L	L
SAMPLE	L	L	L	L	L	L	L
INTEST	L	L	L	L	L	L	L

NOTE—For a <cell context> of **INTERNAL**, a <capture descriptor> value of **PI** is essentially identical to **X** since internal cells do not capture anything other than constant 0s (**ZERO**), 1s (**ONE**), or the values previously shifted in (**CAP**, **UPD**, or **PO**).

e) For a <capture descriptor> element with a <cell context> element equal to **OBSERVE_ONLY**, legal <data source> values for given <capture instruction> elements shall be as given in Table B.11.

Table B.11—Legal capture sources for <cell context> of OBSERVE_ONLY

<capture instruction>	PI	PO	UPD	CAP	X	ZERO	ONE
EXTEST	L	–	–	–	–	–	–
SAMPLE	L	–	–	–	–	–	–
INTEST	A	–	–	–	–	–	–

f) No combination of a <cell context> value and a <capture instruction> value shall appear more than once in a single <capture descriptor list>.
g) The <cell name> value of a <cell description constant> in a <user package body> shall match the <cell name> value of a <deferred constant> in the <user package>, where the <user package body> and <user package> specify the same <user package name>.

NOTE—In the 1990 version of BSDL, the *RUNBIST* instruction was included as one of the <capture instruction> elements, but it does not appear in this 2001 version (nor in the earlier 1994 version). This reflects the fact that *RUNBIST* may or may not reference the boundary-scan register and that the **RUNBIST_EXECUTION** attribute has been added to describe *RUNBIST* capture behavior.

B.10.3 Example of a user-supplied VHDL package

The following is an example of a user-supplied VHDL package that describes two new cells. These cells are able to capture constants (“0” and “1”) during certain situations. For example, used as outputs during *EXTEST*, they capture constant data rather than the system logic values (usually interpreted as “X”). By using these cells in the output cell positions of a boundary-scan register, it is possible to implement an informal ID code. They will capture a pattern of “1” and “0” bits.

```
-- User-defined package describing two new cells

package USER_PACKAGE is

      use STD_1149_1_2001.all;   -- Get definition of "Cell_Info"

      -- Boundary Cell deferred constants (defined in package body)

      constant USER_0  : CELL_INFO;
      constant USER_1  : CELL_INFO;

      end USER_PACKAGE;          -- End of user package


package body USER_PACKAGE is     -- User boundary cells

      use STD_1149_1_2001.all;

      constant USER_0 : CELL_INFO :=
      ((OUTPUT2, EXTEST,  ZERO),
       (OUTPUT2, SAMPLE,  PI),
       (OUTPUT3, EXTEST,  ZERO), (INTERNAL, EXTEST,  ZERO),
       (OUTPUT3, SAMPLE,  PI),   (INTERNAL, SAMPLE,  PI),
       (OUTPUT3, INTEST,  PI),   (INTERNAL, INTEST,  PI),
       (CONTROL, EXTEST,  ZERO), (CONTROLR, EXTEST,  ZERO),
       (CONTROL, SAMPLE,  PI),   (CONTROLR, SAMPLE,  PI),
       (CONTROL, INTEST,  PI),   (CONTROLR, INTEST,  PI) );

      constant USER_1 : CELL_INFO :=
      ((OUTPUT2, EXTEST,  ONE),
       (OUTPUT2, SAMPLE,  PI),
       (OUTPUT3, EXTEST,  ONE),  (INTERNAL, EXTEST,  ONE),
       (OUTPUT3, SAMPLE,  PI),   (INTERNAL, SAMPLE,  PI),
       (OUTPUT3, INTEST,  PI),   (INTERNAL, INTEST,  PI),
       (CONTROL, EXTEST,  ONE),  (CONTROLR, EXTEST,  ONE),
```



```
       (CONTROL, SAMPLE,  PI),   (CONTROLR, SAMPLE,  PI),
       (CONTROL, INTEST,  PI),   (CONTROLR, INTEST,  PI) );

end USER_PACKAGE;                -- End of user package body
```

B.11 Example applications of BSDL

B.11.1 Boundary-scan register description

The following examples illustrate a number of “special case” boundary-scan register structures.

B.11.1.1 Multiple cells per pin

Component pins can be serviced by more than one cell. Each cell can perform a different function. Note that this function is with respect to the boundary-scan register cell, not the component pin. For example, on a bidirectional pin (see Figure 11-36), one cell can serve as an input receiver while the other serves as an output driver. Additional **OBSERVE_ONLY** cells may be connected to any I/O pin.

The component shown in Figure B.8 will be used to illustrate a boundary-scan register with several **OBSERVE_ONLY** cells.

Cell 2 is an additional **OBSERVE_ONLY** cell associated with bidirectional pin 9 of the component.

Cell 7 is an additional **OBSERVE_ONLY** cell associated with output pin 8 of the component. Cells 7, 8, and 9 may have been a programmable two-cell bidirectional implementation that has been reprogrammed to a 2-state output.

Cell 15 is an additional **OBSERVE_ONLY** cell associated with input pin 6 of the component. Cells 13 and 14 are also associated with pin 6, but they are described as **INPUT** cells and are connected to the system logic.

Cell 17 is an additional **OBSERVE_ONLY** cell associated with input pin 5 of the component. Cell 16 is the normal **INPUT** cell for pin 5.

If the component in Figure B.8 supports *INTEST*, cell 18 shall be of type **CLOCK**. If the component does not support *INTEST*, cell 18 could be either be an **INPUT** cell or an **OBSERVE_ONLY** cell.

B.11.1.2 Internal cells

Internal cells can be used to capture “constants” or system-logic-dependent values (0s and 1s) within a design. One proposed use of this possibility is the capturing of a hard-coded value (perhaps in the first few bits of the boundary-scan register) as an informal identification code. Another application is to monitor power connections to ensure that they are receiving the correct input supply and to capture a data bit based on the measured results. If the power connections are good, the data loaded will be 1, for example, while a faulty power input would cause a 0 to be captured. Internal cells, with either control-and-observe capability or observe-only capability, may sit at the boundary between digital and analog sections of the core circuitry. Finally, there may be “extra,” unused cells in a programmable component (see Clause 11).

Note that this standard does not allow system logic to surround internal cells, as shown in Figure 11-42.

The component shown in Figure B.8 will be used to illustrate a boundary-scan register with several **INTERNAL** cells.

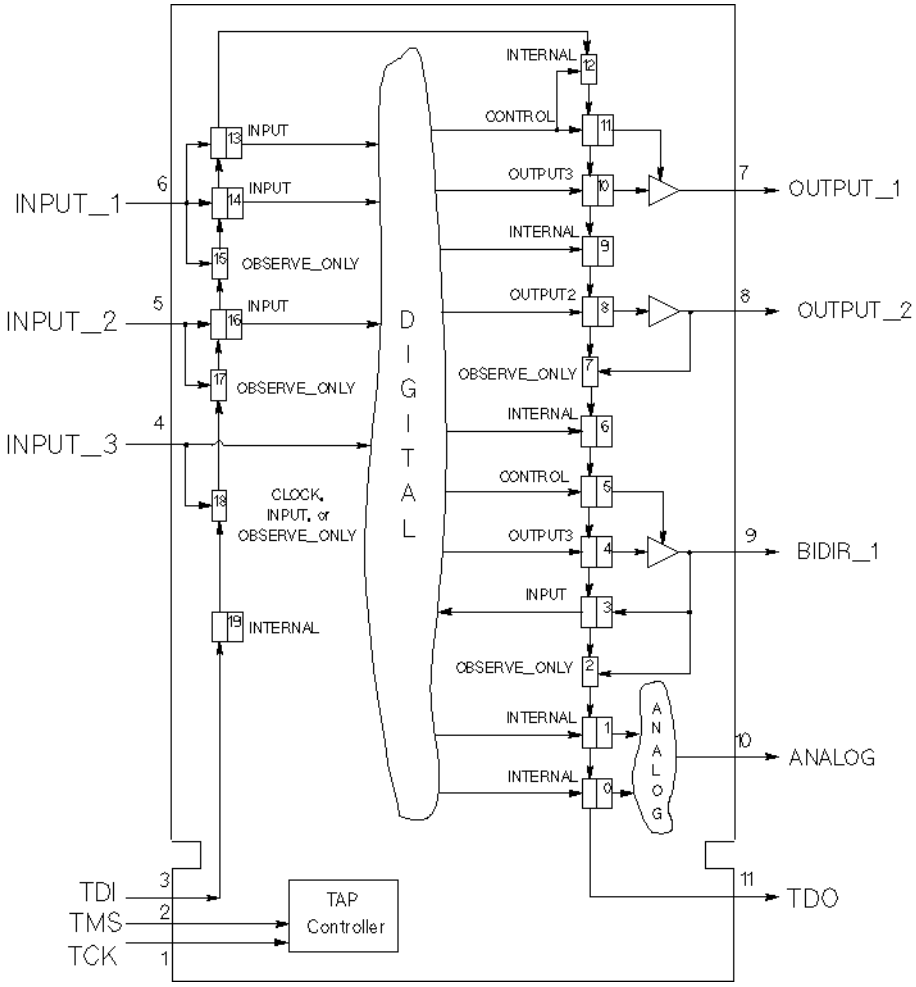


Figure B.8—A component that illustrates several OBSERVE_ONLY and INTERNAL cells

Cells 0 and 1 are **INTERNAL** cells between the digital system logic and the analog system functions. Note that cells 0 and 1 are not associated with pin 10.

Cells 6, 9, and 12 are cells that are observing signals from the system logic, are not associated with system pins, and are described as **INTERNAL** cells.

Cell 19 is an extra cell in the boundary-scan register. It is not observing the system logic or a system pin and is described as an **INTERNAL** cell.

The definition of the boundary-scan register for Figure B.8 is as follows:

```
attribute BOUNDARY_LENGTH of Figure_B8: entity is 20;
attribute BOUNDARY_REGISTER of Figure_B8: entity is
--
--num cell  port      function   safe [ccell disval rslt]
--
" 0  (BC_1, *,        internal,     0),               "&
" 1  (BC_1, *,        internal,     1),               "&
" 2  (BC_0, BIDIR_1,  observe_only, X),               "&
```



```
" 3  (BC_1, BIDIR_1,  input,        X),                "&
" 4  (BC_1, BIDIR_1,  output3,      0,  5,    0,    Z), "&
" 5  (BC_1, *,        control,      0),                "&
" 6  (BC_0, *,        internal,     X),                "&
" 7  (BC_0, OUTPUT_2, observe_only, X),                "&
" 8  (BC_1, OUTPUT_2, output2,      1),                "&
" 9  (BC_0, *,        internal,     X),                "&
"10  (BC_1, OUTPUT_1, output3,      0,  11,   0,    Z), "&
"11  (BC_1, *,        control,      0),                "&
"12  (BC_0, *,        internal,     X),                "&
"13  (BC_1, INPUT_1,  input,        X),                "&
"14  (BC_1, INPUT_1,  input,        X),                "&
"15  (BC_0, INPUT_1,  observe_only, X),                "&
"16  (BC_1, INPUT_2,  input,        X),                "&
"17  (BC_0, INPUT_2,  observe_only, X),                "&
"18  (BC_0, INPUT_3,  observe_only, X),                "&
"19  (BC_0, *,        internal,     X)                 ";
```

B.11.1.3 Merged cells

The component shown in Figure B.9 will be used to illustrate a boundary-scan register description in which special cases are handled. These special cases arise because this standard allows boundary-scan register cells to be merged when the system logic between them is null (see, for example, Figure 11-43 and Figure 11-44). Cells may be merged if the “logic” between them is simply a data path, such as a wire or a buffer. When the merging is done, the resulting cell must obey a combination of the rules of the merged cells.

The definition of the boundary-scan register for Figure B.9 is as follows:

```
attribute BOUNDARY_LENGTH of Figure_B9: entity is 10;
attribute BOUNDARY_REGISTER of Figure_B9: entity is
--
--num cell port      function   safe [ccell disval rslt]
--
" 0  (BC_1, *,        control,      0),                "&
" 1  (BC_1, OUT2,     output2,      1,  1,    1,    Weak1),"&
" 2  (BC_7, BIDIR1,   bidir,        X,  3,    0,    Z), "&
" 3  (BC_2, *,        control,      0),                "&
" 4  (BC_1, *,        control,      0),                "&
" 5  (BC_1, BIDIR3,   input,        X),                "&
" 5  (BC_1, BIDIR2,   output3,      X,  7,    1,    Z), "&
" 6  (BC_1, BIDIR2,   input,        X),                "&
" 6  (BC_1, BIDIR3,   output3,      X,  4,    0,    Z), "&
" 7  (BC_1, *,        control,      1),                "&
" 8  (BC_1, IN2,      input,        X),                "&
" 9  (BC_1, IN1,      input,        X),                "&
" 9  (BC_1, OUT1,     output3,      X,  0,    0,    Z)  ";
```

Cells 0, 4, and 7 are control cells located between the system logic and the enable for signals `OUT1`, `BIDIR2`, and `BIDIR3`. Notice that values are assigned to the “safe” subfields for these cells to cause the associated drivers to disable.

Cell 3 is the control for the reversible cell (see Figure 11-37c) used on the bidirectional signal `BIDIR1`. Notice that its “safe” subfield is given the value that causes `BIDIR1` to be an input.

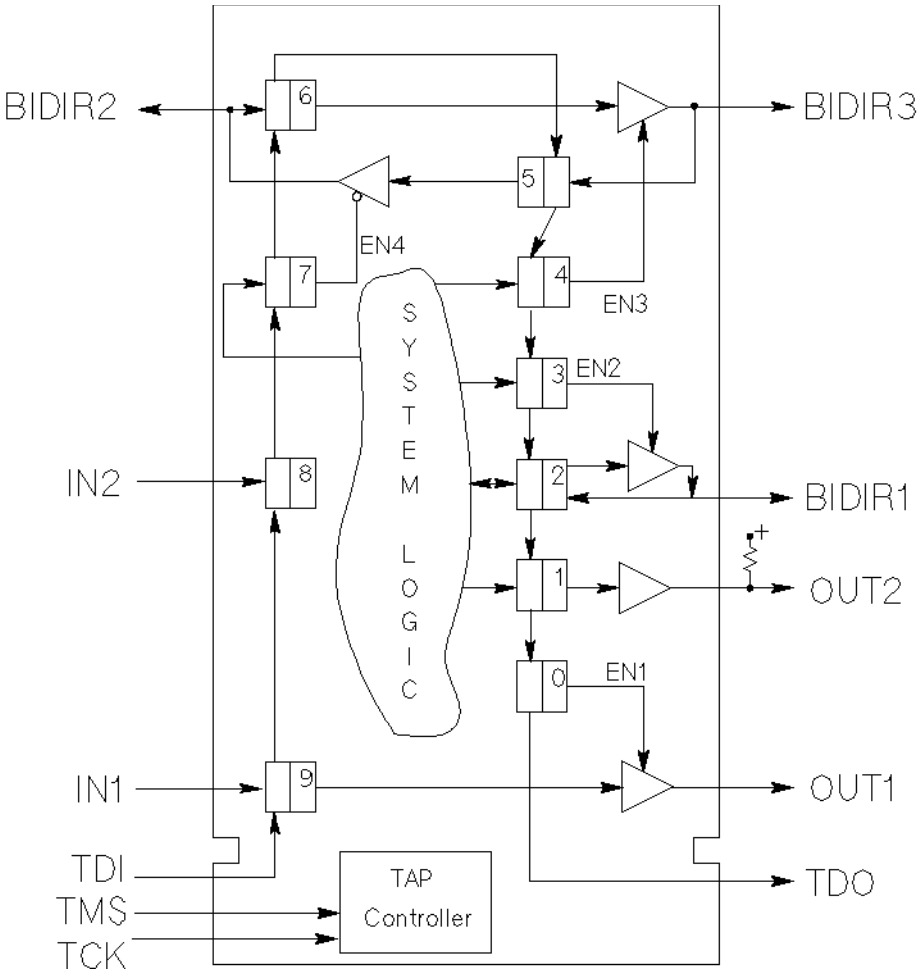


Figure B.9—A component that illustrates several merged cells

Cell 1 is a 2-state output data cell. Note that it has the three extra fields, indicating that it controls its own open-collector, asymmetrical driver. By placing a “1” in cell 1, the driver at `OUT2` can be set to the inactive drive state, in which it will output the “**WEAK1**” state.

Cell 2 is the reversible cell of Figure 11-37d. This cell serves either as an input (if the control cell has turned off the output driver, meaning cell 3 produces a “0”) or as the data source for the driver (if the output is enabled).

Note that the structures for `BIDIR2` and `BIDIR3` (see Figure 11-36) would allow observation of the driver, thus allowing a simple consistency check.

Cell 5 (and similarly cells 6 and 9) has merged behavior—it serves as the input receiver for `BIDIR3` and as the data source for `BIDIR2`. As a result, the cell has two lines of description in the boundary-scan register definition. The first gives its behavior as an input cell, while the second describes its characteristics as an output cell. Note that cell **BC_1** used in this capacity must support both **INPUT** and **OUTPUT3** functions. This is reflected in the definition of **BC_1**, where both functions exist for all instructions.

The example illustrated by Figure B.9 is extreme and includes several unusual cases. Most component implementations will be quite simple and routine, as illustrated by the example component description in Clause B.12.

B.12 A typical application of BSDL

The following example is for the Texas Instruments SN74BCT8374 Octal D Flip-Flop (see Figure B.10) using version STD_1149_1_2001.

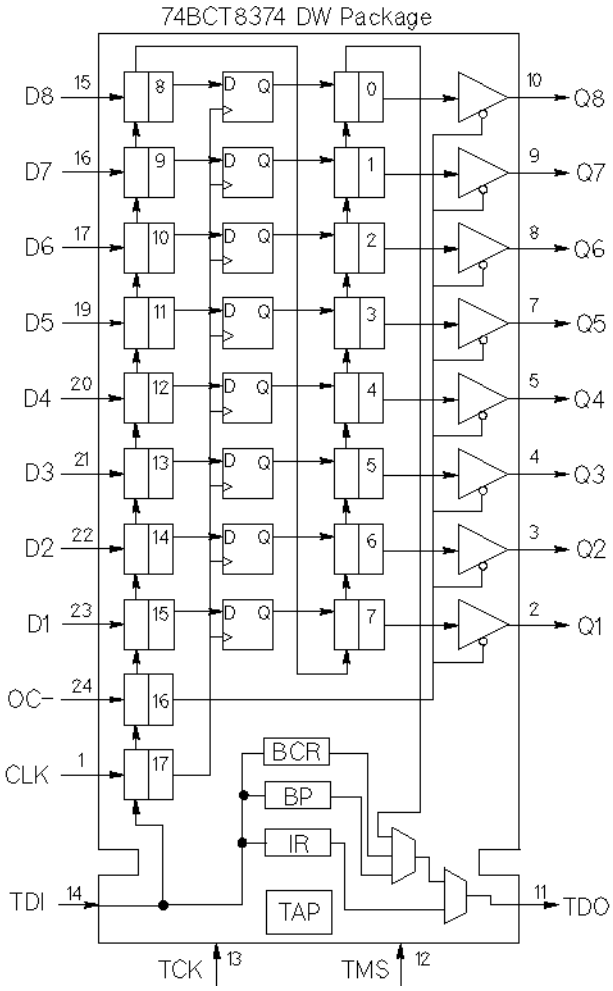


Figure B.10—Texas Instruments SN74BCT8374

```
entity ttl74bct8374 is

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

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

  --Get IEEE Std 1149.1-2001 attributes and definitions
  use STD_1149_1_2001.all;
```

```
  attribute COMPONENT_CONFORMANCE of ttl74bct8374: entity is
     "STD_1149_1_2001";

  attribute PIN_MAP of ttl74bct8374: entity is PHYSICAL_PIN_MAP;

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

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

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

    attribute INSTRUCTION_LENGTH of ttl74bct8374 : entity is 8;

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

    attribute INSTRUCTION_CAPTURE of ttl74bct8374 : entity is
       "10000001";

    attribute REGISTER_ACCESS of ttl74bct8374 : entity is
       "BOUNDARY (READBN, READBT, CELLTST)," &
       "BYPASS (TOPHIP, RUNT)," &
       "BCR[2] (SCANCN, SCANCT)";   -- 2-bit boundary control register

    attribute BOUNDARY_LENGTH of ttl74bct8374 : entity is 18;

    attribute BOUNDARY_REGISTER of ttl74bct8374 : entity is
      -- num cell  port      function   safe [ccell disval rslt]
        "17 (BC_1, CLK,      input,        X)," &
        "16 (BC_1, OC_NEG,   input,        X), " &-- Merged input/
        "16 (BC_1, *,        control,      1), " &--        control
        "15 (BC_1, D(1),     input,        X)," &
        "14 (BC_1, D(2),     input,        X)," &
        "13 (BC_1, D(3),     input,        X)," &
        "12 (BC_1, D(4),     input,        X)," &
```

```
        "11 (BC_1, D(5),     input,        X)," &
        "10 (BC_1, D(6),     input,        X)," &
        " 9 (BC_1, D(7),     input,        X)," &
        " 8 (BC_1, D(8),     input,        X)," &
                       -- cell 16 @ 1 -> Hi-Z
        " 7 (BC_1, Q(1),     output3,      X,  16,   1,    Z)," &
        " 6 (BC_1, Q(2),     output3,      X,  16,   1,    Z)," &
        " 5 (BC_1, Q(3),     output3,      X,  16,   1,    Z)," &
        " 4 (BC_1, Q(4),     output3,      X,  16,   1,    Z)," &
        " 3 (BC_1, Q(5),     output3,      X,  16,   1,    Z)," &
        " 2 (BC_1, Q(6),     output3,      X,  16,   1,    Z)," &
        " 1 (BC_1, Q(7),     output3,      X,  16,   1,    Z)," &
        " 0 (BC_1, Q(8),     output3,      X,  16,   1,    Z)";

end ttl74bct8374;
```

B.13 The 1990 version of BSDL

The 1990 version of BSDL is a de facto industry standard. The information presented in this clause is provided as a courtesy to tool implementers who wish to support BSDL descriptions written before this 2001 version was defined. This form of BSDL is a subset of the 2001 version except where noted below. New BSDL descriptions should not use the 1990 form.

In the 1990 version of BSDL, the following syntactic elements are not supported:

<component conformance statement>
<grouped port identification>
<compliance enable description>
<runbist description>
<intest description>
<BSDL extensions>

In the 1990 version, **KEEPER** and **PRELOAD** were not recognized as reserved words. Also, in the 1990 version, cell types **BC_0**, **BC_7**, **BC_8**, **BC_9**, and **BC_10** need to be specified in a user-supplied VHDL package.

B.13.1 The 1990 Standard VHDL Package STD_1149_1_1990

The following shall be the complete content of Standard VHDL Package STD_1149_1_1990. Note that both the VHDL package and the VHDL package body are shown. This information defines the basis of BSDL and typically would be write-protected by a system administrator. An explanation of the cell definitions (e.g., **BC_1**, **BC_2**, etc.) in the package body is given in B.10.2. BSDL files that use <standard VHDL package identifier> **STD_1149_1_1990** shall be processed using this Standard VHDL Package.

NOTES

1—The figure references are to the 1990 edition of IEEE Std 1149.1.

2 —Where figures are cited, the suffix “c” is used to denote a control cell. The suffix “d” denotes a data cell.

```
package STD_1149_1_1990 is

-- Give pin mapping declarations

attribute PIN_MAP : string;
```

```
subtype PIN_MAP_STRING is string;

-- Give TAP control declarations

type CLOCK_LEVEL is (LOW, BOTH);
type CLOCK_INFO  is record
  FREQ : real;
  LEVEL: CLOCK_LEVEL;
end record;

attribute  TAP_SCAN_IN   : boolean;
attribute  TAP_SCAN_OUT  : boolean;
attribute  TAP_SCAN_CLOCK: CLOCK_INFO;
attribute  TAP_SCAN_MODE : boolean;
attribute  TAP_SCAN_RESET: boolean;

-- Give instruction register declarations

attribute  INSTRUCTION_LENGTH : integer;
attribute  INSTRUCTION_OPCODE : string;
attribute  INSTRUCTION_CAPTURE : string;
attribute  INSTRUCTION_DISABLE : string;
attribute  INSTRUCTION_GUARD : string;
attribute  INSTRUCTION_PRIVATE : string;
attribute  INSTRUCTION_USAGE : string;
attribute  INSTRUCTION_SEQUENCE : string;

-- Give ID and USER code declarations

type ID_BITS is ('0', '1', 'x', 'X');
type ID_STRING is array (31 downto 0) of ID_BITS;
attribute IDCODE_REGISTER  :  ID_STRING;
attribute USERCODE_REGISTER:  ID_STRING;

-- Give register declarations

attribute REGISTER_ACCESS : string;

-- Give boundary cell declarations

type BSCAN_INST is (EXTEST, SAMPLE, INTEST, RUNBIST);
type CELL_TYPE is (INPUT, INTERNAL, CLOCK,
                   CONTROL, CONTROLR, OUTPUT2,
                   OUTPUT3, BIDIR_IN, BIDIR_OUT);
type CAP_DATA is (PI, PO, UPD, CAP, X, ZERO, ONE);
type CELL_DATA is record
  CT : CELL_TYPE;
  I  : BSCAN_INST;
  CD : CAP_DATA;
end record;
type CELL_INFO is array (positive range <>) of CELL_DATA;

-- Boundary cell deferred constants (see package body)
```

```
constant BC_1  : CELL_INFO;
constant BC_2  : CELL_INFO;
constant BC_3  : CELL_INFO;
constant BC_4  : CELL_INFO;
constant BC_5  : CELL_INFO;
constant BC_6  : CELL_INFO;

-- Boundary register declarations

attribute BOUNDARY_CELLS : string;
attribute BOUNDARY_LENGTH : integer;
attribute BOUNDARY_REGISTER : string;

-- Miscellaneous

attribute DESIGN_WARNING : string;

end STD_1149_1_1990;  -- End of IEEE Std 1149.1-1990 Package


package body STD_1149_1_1990 is   -- Standard boundary cells

-- Description for f10-12, f10-16, f10-18c, f10-18d, f10-21c

constant BC_1 : CELL_INFO :=
 ((INPUT,   EXTEST,  PI),  (OUTPUT2,  EXTEST,  PI),
  (INPUT,   SAMPLE,  PI),  (OUTPUT2,  SAMPLE,  PI),
  (INPUT,   INTEST,  PI),  (OUTPUT2,  INTEST,  PI),
  (INPUT,   RUNBIST, PI),  (OUTPUT2,  RUNBIST, PI),
  (OUTPUT3, EXTEST,  PI),  (INTERNAL, EXTEST,  PI),
  (OUTPUT3, SAMPLE,  PI),  (INTERNAL, SAMPLE,  PI),
  (OUTPUT3, INTEST,  PI),  (INTERNAL, INTEST,  PI),
  (OUTPUT3, RUNBIST, PI),  (INTERNAL, RUNBIST, PI),
  (CONTROL, EXTEST,  PI),  (CONTROLR, EXTEST,  PI),
  (CONTROL, SAMPLE,  PI),  (CONTROLR, SAMPLE,  PI),
  (CONTROL, INTEST,  PI),  (CONTROLR, INTEST,  PI),
  (CONTROL, RUNBIST, PI),  (CONTROLR, RUNBIST, PI) );

-- Description for f10-8, f10-17, f10-19c, f10-19d, f10-22c

constant BC_2 : CELL_INFO :=
 ((INPUT,   EXTEST,  PI),  (OUTPUT2, EXTEST,   UPD),
  (INPUT,   SAMPLE,  PI),  (OUTPUT2, SAMPLE,   PI),
  (INPUT,   INTEST,  UPD),  -- Intest on output2 not supported
  (INPUT,   RUNBIST, UPD), (OUTPUT2, RUNBIST,  UPD),
  (OUTPUT3, EXTEST,  UPD), (INTERNAL, EXTEST,  PI),
  (OUTPUT3, SAMPLE,  PI),  (INTERNAL, SAMPLE,  PI),
  (OUTPUT3, INTEST,  PI),  (INTERNAL, INTEST,  UPD),
  (OUTPUT3, RUNBIST, PI),  (INTERNAL, RUNBIST, UPD),
  (CONTROL, EXTEST,  UPD), (CONTROLR, EXTEST,  UPD),
  (CONTROL, SAMPLE,  PI),  (CONTROLR, SAMPLE,  PI),
  (CONTROL, INTEST,  PI),  (CONTROLR, INTEST,  PI),
  (CONTROL, RUNBIST, PI),  (CONTROLR, RUNBIST, PI) );
```

```
-- Description for f10-9

constant BC_3 : CELL_INFO :=
 ((INPUT, EXTEST,  PI),    (INTERNAL, EXTEST,  PI),
  (INPUT, SAMPLE,  PI),    (INTERNAL, SAMPLE,  PI),
  (INPUT, INTEST,  PI),    (INTERNAL, INTEST,  PI),
  (INPUT, RUNBIST, PI),    (INTERNAL, RUNBIST, PI) );

-- Description for f10-10, f10-11

constant BC_4 : CELL_INFO :=
 ((INPUT, EXTEST,  PI),  -- Intest on input not supported
  (INPUT, SAMPLE,  PI),  -- Runbist on input not supported
  (CLOCK, EXTEST,  PI),   (INTERNAL, EXTEST,  PI),
  (CLOCK, SAMPLE,  PI),   (INTERNAL, SAMPLE,  PI),
  (CLOCK, INTEST,  PI),   (INTERNAL, INTEST,  PI),
  (CLOCK, RUNBIST, PI),   (INTERNAL, RUNBIST, PI) );

-- Description for f10-20c, a combined input/control

constant BC_5 : CELL_INFO :=
 ((INPUT, EXTEST,  PI),   (CONTROL, EXTEST,  PI),
  (INPUT, SAMPLE,  PI),   (CONTROL, SAMPLE,  PI),
  (INPUT, INTEST,  UPD),  (CONTROL, INTEST,  UPD),
  (INPUT, RUNBIST, PI),   (CONTROL, RUNBIST, PI) );

-- Description for f10-22d, a reversible cell

constant BC_6 : CELL_INFO :=
 ((BIDIR_IN, EXTEST,  PI),  (BIDIR_OUT, EXTEST,  UPD),
  (BIDIR_IN, SAMPLE,  PI),  (BIDIR_OUT, SAMPLE,  PI),
  (BIDIR_IN, INTEST,  UPD), (BIDIR_OUT, INTEST,  PI),
  (BIDIR_IN, RUNBIST, UPD), (BIDIR_OUT, RUNBIST, PI) );

end STD_1149_1_1990;  -- End of 1990 Package Body
```

B.13.2 A typical application of BSDL, 1990 version

The following example is for the Texas Instruments SN74BCT8374 Octal D Flip-Flop using version STD_1149_1_1990.

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

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

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

  attribute PIN_MAP of ttl74bct8374 : entity is PHYSICAL_PIN_MAP;

  constant DW:   PIN_MAP_STRING:="CLK:1, Q:(2,3,4,5,7,8,9,10), " &
         "D:(23,22,21,20,19,17,16,15)," &
```

```
          "GND:6, VCC:18, OC_NEG:24, TDO:11, TMS:12, TCK:13, TDI:14";

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

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

    attribute INSTRUCTION_LENGTH of ttl74bct8374 : entity is 8;

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

    attribute INSTRUCTION_CAPTURE of ttl74bct8374 : entity is "10000001";
    attribute INSTRUCTION_DISABLE of ttl74bct8374 : entity is "TRIBYP";
                                        -- Now obsolete, see note below.
    attribute INSTRUCTION_GUARD   of ttl74bct8374 : entity is "SETBYP";
                                        -- Now obsolete, see note below.
--NOTE-The INSTRUCTION_SEQUENCE and INSTRUCTION_USAGE attributes would
--appear here, but they are now obsolete.

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

    attribute BOUNDARY_CELLS of ttl74bct8374 : entity is "BC_1";
                                        -- Now obsolete, see note below.
    attribute BOUNDARY_LENGTH of ttl74bct8374 : entity is 18;

--NOTE-The "attribute INSTRUCTION_DISABLE of ttl74bct8374:entity is
--"TRIBYP";" statement is the same as the HIGHZ instruction defined in
--this standard and the "attribute INSTRUCTION_GUARD of ttl74bct8374:
--entity is "SETBYP";" statement is the same as CLAMP instruction. The
--"attribute BOUNDARY_CELLS of ttl74bct8374:entity is "BC_1";" statement
--has been removed from later versions of BSDL, since the cells being
--used can be identified while processing the BOUNDARY_REGISTER
--attribute.
```

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

end ttl74bct8374;
```

B.13.3 Obsolete syntax

Several attributes were enumerated in the 1990 version of BSDL that were all string-valued. They shall be obsolete in this 2001 version (and the earlier 1994 version) of BSDL. The **INSTRUCTION_GUARD** and **INSTRUCTION_DISABLE** attributes were made obsolete by the standardization of the *CLAMP* and *HIGHZ* instructions. The **BOUNDARY_CELLS** attribute was found to be unnecessary. With regard to the **INSTRUCTION_SEQUENCE** and **INSTRUCTION_USAGE** attributes, agreement was never reached on either their scope or the applications they were intended to handle. These two attributes have been dropped.

All these obsolete attributes had string values, and so a tool intended to ignore statements defining the attributes can be designed to ignore string-valued attributes with the obsolete names. Their syntax shall be provided in B.13.3.1 to assist users in the development of tools that can read both the version of BSDL documented in this annex and the earlier draft version of the language.

B.13.3.1 Syntax

attribute <obsoleted attribute name> **of** <component name> **: entity is** <obsoleted string> **;**

<obsoleted attribute name>::= **INSTRUCTION_GUARD** | **INSTRUCTION_DISABLE** | **INSTRUCTION_SEQUENCE** | **INSTRUCTION_USAGE** | **BOUNDARY_CELLS**

<obsoleted string>::= <string>

B.13.4 Miscellaneous points on the 1990 version

In parsing a 1990 version of BSDL, a violation of the condition described in semantic check B.8.14.4 p) shall result in the issuance of a warning message by parsing software. This acknowledges that the 1990 edition of this standard allowed a control cell to fan out to more than one driver, and each driver could be enabled with an independent choice of control value. Semantic check B.8.14.4 p) reflects the strengthening

of this standard to require specifically that all drivers controlled by a single control cell shall be disabled by the same value.

The VHDL package body shown in B.13.1 has a fourth value of <capture instruction>, with a value of RUNBIST shown in the <capture descriptor> elements (see B.10.2). This value has been removed in this 2001 version of BSDL (and the earlier 1994 version), since this version now provides for RUNBIST description (see B.8.15).

In the 1990 version of BSDL, the device ID register was named **IDCODE** in the **REGISTER_ACCESS** attribute. In this annex, this name has been changed to **DEVICE_ID** to match the definition used elsewhere in this standard.

In the 1990 version of BSDL, the following values of given BSDL syntactical elements did not exist:

— <cell context> value **OBSERVE_ONLY**
— <disable result> value **KEEPER**
— <function> value **OBSERVE_ONLY**
— <instruction name> value **PRELOAD**

B.14 The 1994 version of BSDL

The 1994 version of BSDL was defined by IEEE Std 1149.1b-1994. The information presented in this clause is provided to help tool implementers support BSDL descriptions written before this 2001 version was defined. This form of BSDL is a subset of the 2001 version except where noted below. New BSDL descriptions should not use the 1994 form.

In the 1994 version, **KEEPER** and **PRELOAD** were not recognized as a reserved words. Also, in the 1994 version, cell types **BC_8**, **BC_9**, and **BC_10** need to be specified in a user-supplied VHDL package.

B.14.1 The Standard VHDL Package STD_1149_1_1994

The following shall be the complete content of Standard VHDL Package STD_1149_1_1994. Note that both the VHDL package and the VHDL package body are shown. This information shall define the basis of BSDL and typically would be write-protected by a system administrator. An explanation of the cell definitions (e.g., **BC_1**, **BC_2**, etc.) in the package body is given in B.10.2. BSDL descriptions that use <standard VHDL package identifier> **STD_1149_1_1994** shall be processed using this Standard VHDL Package.

NOTE—The figure references are to the 1993 edition of IEEE Std 1149.1.

```
package STD_1149_1_1994 is

-- Give component conformance declaration

attribute COMPONENT_CONFORMANCE : string;

-- Give pin mapping declarations

attribute PIN_MAP : string;
subtype PIN_MAP_STRING is string;

-- Give TAP control declarations
```

```
type CLOCK_LEVEL is (LOW, BOTH);
type CLOCK_INFO  is record
  FREQ : real;
  LEVEL: CLOCK_LEVEL;
end record;

attribute  TAP_SCAN_IN   : boolean;
attribute  TAP_SCAN_OUT  : boolean;
attribute  TAP_SCAN_CLOCK: CLOCK_INFO;
attribute  TAP_SCAN_MODE : boolean;
attribute  TAP_SCAN_RESET: boolean;

-- Give instruction register declarations

attribute  INSTRUCTION_LENGTH : integer;
attribute  INSTRUCTION_OPCODE : string;
attribute  INSTRUCTION_CAPTURE : string;
attribute  INSTRUCTION_PRIVATE : string;

-- Give ID and USER code declarations

type ID_BITS is ('0', '1', 'x', 'X');
type ID_STRING is array (31 downto 0) of ID_BITS;
attribute IDCODE_REGISTER  :  ID_STRING;
attribute USERCODE_REGISTER:  ID_STRING;

-- Give register declarations

attribute REGISTER_ACCESS : string;

-- Give boundary cell declarations

type BSCAN_INST is (EXTEST, SAMPLE, INTEST);
type CELL_TYPE is (INPUT, INTERNAL, CLOCK, OBSERVE_ONLY,
                   CONTROL, CONTROLR, OUTPUT2,
                   OUTPUT3, BIDIR_IN, BIDIR_OUT);
type CAP_DATA is (PI, PO, UPD, CAP, X, ZERO, ONE);
type CELL_DATA is record
  CT : CELL_TYPE;
  I  : BSCAN_INST;
  CD : CAP_DATA;
end record;
type CELL_INFO is array (positive range <>) of CELL_DATA;

-- Boundary cell deferred constants (see package body)

constant BC_0  : CELL_INFO;
constant BC_1  : CELL_INFO;
constant BC_2  : CELL_INFO;
constant BC_3  : CELL_INFO;
constant BC_4  : CELL_INFO;
constant BC_5  : CELL_INFO;
constant BC_6  : CELL_INFO;
constant BC_7  : CELL_INFO;
```

```
-- Boundary register declarations

attribute BOUNDARY_LENGTH : integer;
attribute BOUNDARY_REGISTER : string;

-- Miscellaneous

attribute PORT_GROUPING : string;
attribute RUNBIST_EXECUTION : string;
attribute INTEST_EXECUTION : string;
subtype BSDL_EXTENSION is string;
attribute COMPLIANCE_PATTERNS : string;
attribute DESIGN_WARNING : string;

end STD_1149_1_1994;  -- End of 1149.1-1994 Package

package body STD_1149_1_1994 is   -- Standard boundary cells

-- Generic cell capturing minimum allowed data

constant BC_0 : CELL_INFO :=

 ((INPUT,   EXTEST,  PI),      (OUTPUT2,  EXTEST,  X),
  (INPUT,   SAMPLE,  PI),      (OUTPUT2,  SAMPLE,  PI),
  (INPUT,   INTEST,  X),       (OUTPUT2,  INTEST,  PI),
  (OUTPUT3, EXTEST,  X),       (INTERNAL, EXTEST,  X),
  (OUTPUT3, SAMPLE,  PI),      (INTERNAL, SAMPLE,  X),
  (OUTPUT3, INTEST,  PI),      (INTERNAL, INTEST,  X),
  (CONTROL, EXTEST,  X),       (CONTROLR, EXTEST,  X),
  (CONTROL, SAMPLE,  PI),      (CONTROLR, SAMPLE,  PI),
  (CONTROL, INTEST,  PI),      (CONTROLR, INTEST,  PI),
  (BIDIR_IN,EXTEST,  PI),      (BIDIR_OUT, EXTEST, X ),
  (BIDIR_IN,SAMPLE,  PI),      (BIDIR_OUT, SAMPLE, PI),
  (BIDIR_IN,INTEST,  X ),      (BIDIR_OUT, INTEST, PI),
  (OBSERVE_ONLY, SAMPLE, PI),  (OBSERVE_ONLY, EXTEST, PI) );

-- Description for f10-18, f10-29, f10-31c, f10-31d, f10-33c, f10-41d

constant BC_1 : CELL_INFO :=
 ((INPUT,   EXTEST,  PI),  (OUTPUT2,  EXTEST,  PI),
  (INPUT,   SAMPLE,  PI),  (OUTPUT2,  SAMPLE,  PI),
  (INPUT,   INTEST,  PI),  (OUTPUT2,  INTEST,  PI),
  (OUTPUT3, EXTEST,  PI),  (INTERNAL, EXTEST,  PI),
  (OUTPUT3, SAMPLE,  PI),  (INTERNAL, SAMPLE,  PI),
  (OUTPUT3, INTEST,  PI),  (INTERNAL, INTEST,  PI),
  (CONTROL, EXTEST,  PI),  (CONTROLR, EXTEST,  PI),
  (CONTROL, SAMPLE,  PI),  (CONTROLR, SAMPLE,  PI),
  (CONTROL, INTEST,  PI),  (CONTROLR, INTEST,  PI) );

-- Description for f10-14, f10-30, f10-32c, f10-32d, f10-35c

constant BC_2 : CELL_INFO :=
 ((INPUT,   EXTEST,  PI),  (OUTPUT2, EXTEST,   UPD),
  (INPUT,   SAMPLE,  PI),  (OUTPUT2, SAMPLE,   PI),
```

```
  (INPUT,   INTEST,  UPD),  -- Intest on output2 not supported
  (OUTPUT3, EXTEST,  UPD), (INTERNAL, EXTEST,  PI),
  (OUTPUT3, SAMPLE,  PI),  (INTERNAL, SAMPLE,  PI),
  (OUTPUT3, INTEST,  PI),  (INTERNAL, INTEST,  UPD),
  (CONTROL, EXTEST,  UPD), (CONTROLR, EXTEST,  UPD),
  (CONTROL, SAMPLE,  PI),  (CONTROLR, SAMPLE,  PI),
  (CONTROL, INTEST,  PI),  (CONTROLR, INTEST,  PI) );

-- Description for f10-15

constant BC_3 : CELL_INFO :=
 ((INPUT, EXTEST,  PI),    (INTERNAL, EXTEST,  PI),
  (INPUT, SAMPLE,  PI),    (INTERNAL, SAMPLE,  PI),
  (INPUT, INTEST,  PI),    (INTERNAL, INTEST,  PI) );

-- Description for f10-16, f10-17

constant BC_4 : CELL_INFO :=
 ((INPUT, EXTEST,  PI),        -- Intest on input not supported
  (INPUT, SAMPLE,  PI),
  (OBSERVE_ONLY, EXTEST, PI),
  (OBSERVE_ONLY, SAMPLE, PI),  -- Intest on observe_only not supported
  (CLOCK, EXTEST,  PI),   (INTERNAL, EXTEST,  PI),
  (CLOCK, SAMPLE,  PI),   (INTERNAL, SAMPLE,  PI),
  (CLOCK, INTEST,  PI),   (INTERNAL, INTEST,  PI) );

-- Description for f10-41c, a combined input/control

constant BC_5 : CELL_INFO :=
 ((INPUT, EXTEST,  PI),   (CONTROL, EXTEST,  PI),
  (INPUT, SAMPLE,  PI),   (CONTROL, SAMPLE,  PI),
  (INPUT, INTEST,  UPD),  (CONTROL, INTEST,  UPD) );

-- Description for f10-35d, a reversible cell
-- !! Not recommended; replaced by BC_7 below !!

constant BC_6 : CELL_INFO :=

 ((BIDIR_IN, EXTEST,  PI),  (BIDIR_OUT, EXTEST,  UPD),
  (BIDIR_IN, SAMPLE,  PI),  (BIDIR_OUT, SAMPLE,  PI),
  (BIDIR_IN, INTEST,  UPD), (BIDIR_OUT, INTEST,  PI) );

-- Description for f10-34d, self monitor reversible
-- !! Recommended over cell BC_6 !!

constant BC_7 : CELL_INFO :=
 ((BIDIR_IN, EXTEST,  PI),  (BIDIR_OUT, EXTEST,  PO),
  (BIDIR_IN, SAMPLE,  PI),  (BIDIR_OUT, SAMPLE,  PI),
  (BIDIR_IN, INTEST,  UPD), (BIDIR_OUT, INTEST,  PI) );

end STD_1149_1_1994;  -- End of IEEE Std 1149.1-1994 Package Body
```

B.14.2 Miscellaneous points on 1994 version

In the 1994 version of BSDL, the following values of given BSDL syntactical elements did not exist:

— <disable result> value **KEEPER**
— <instruction name> value **PRELOAD**