

**1532™**

# **IEEE Standard for In-System Configuration of Programmable Devices**

**IEEE Computer Society**

Sponsored by the  
Test Technology Technical Council



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

10 January 2003

Print: SH95071  
PDF: SS95071

# IEEE Standard for In-System Configuration of Programmable Devices

Sponsor

**Test Technology Technical Council**  
of the  
**IEEE Computer Society**

Approved 11 December 2002

**IEEE-SA Standards Board**

**Abstract:** The communication protocol described by IEEE Std 1149.1™-2001 (Standard Test Access Port and Boundary-Scan Architecture) has been adopted by this standard for providing standardized programming access and methodology for programmable integrated circuit devices. Devices that implement this standard will first be compliant with IEEE Std 1149.1-2001, which is used for testing purposes. A device, or set of devices, implementing this standard can be programmed (written), read back, erased, and verified, singly or concurrently, with a standardized set of resources. Sample implementation and application details (which are not part of this standard) are included for illustrative purposes.

**Keywords:** complex programmable logic device (CPLD), erasable programmable read-only memory (EPROM), field programmable gate array (FPGA), Flash RAM, in-system (or in situ) configuration (ISC), programmable device

---

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

Copyright © 2003 by the Institute of Electrical and Electronics Engineers, Inc.  
All rights reserved. Published 10 January 2003. Printed in the United States of America.

Print: ISBN 0-7381-3507-0 SH95071  
PDF: ISBN 0-7381-3508-9 SS95071

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

**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.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

# Introduction

(This introduction is not part of IEEE Std 1532-2002, IEEE Standard for In-System Configuration of Programmable Devices.)

The development of this standard began with a preliminary meeting held 19 April 1996. There, a group of Programmable Device vendors and users discussed the possibility of standardizing the programming process for these devices. Subsequent meetings were held regularly and agreement was developed for basing the programming process upon the IEEE Std 1149.1 communication protocol, because many of these devices were expected to support IEEE Std 1149.1 for testing purposes. This group adopted as its mission:

*To define, document, and promote the use of a standardized process and methodology for implementing programming capabilities within programmable integrated circuit devices, utilizing (and compatible with) the IEEE Std 1149.1 communication protocol. This standard would allow the programming of one or more compliant devices concurrently, while mounted on a board or embedded in a system, known as “In-System Configuration.” Concurrent programming may often result in significant programming time efficiencies. The In-System feature would address the need to configure or reconfigure, read back, verify or erase programmable devices after they have been installed by a manufacturing process. This eliminates handling damage and the need for manufacturing steps and inventory management related to preprogrammed devices.*

This work received the support of the IEEE Std 1149.1 Working Group which ultimately urged that this effort become its own standard within the Test Technology Technical Committee of the IEEE Computer Society, in July of 1998. The initially approved version of this standard described the necessary hardware elements for compliance. By making this available quickly and early, the development of compliant silicon was facilitated and its availability accelerated. The next revision of this standard described the software elements necessary for full implementation of this standard. The software elements described include extensions to the BSDL file that describe the programming algorithm and hardware implementation of this standard. Additionally a new data file format has been described. This revision further extends the standard by adding support for devices that require adaptive programming algorithms along with some slight modifications to the initial standard that will help to better meet user’s needs.

Following is a list of participants in the In-System Configuration Working Group of the Test Technology Technical Council. Voting members at the time of publication are marked with an asterisk (\*).

**Neil G. Jacobson, *Chair***

**David Bonnet, *Vice-Chair***

**Dennis Lia, *Secretary***

**Mark Moyer, *Editor***

**Kenneth P. Parker, *Editor Emeritus***

|                       |                  |                    |
|-----------------------|------------------|--------------------|
| Turgut Abacioglu      | P. D. Felen      | David Paul         |
| John Andrews          | Judy Chen Feng   | Bruce Peterson     |
| Don Arnold            | Dan Gravitz      | Etienne Racine     |
| Bill Aronson          | Jack Griffin*    | Bapcha Ramamurthy  |
| Marco Barahmand       | Kurt Guntheroth  | Doug Raymond       |
| Jason Baumbach*       | Peter Hansen     | Jacky Renaux       |
| Glenn Baxter          | Steve Harrison   | Robert Russell     |
| Menachem Blasberg     | Bernie Haverkamp | David Lee Rutledge |
| Terry Borroz          | Alan Herrmann*   | Bobbie Saville     |
| Stephen Brown         | Brad Ishihara*   | Sanjeev Sharma*    |
| Ian Bryant            | John Joy         | Steve Stark        |
| Steven Butkovich      | Raj Kamdar       | Chung Sun          |
| Tapan Chakraborty     | Doug Kay         | Jerry Talen        |
| Sung Chung*           | Arthur Khu       | Howard Tang*       |
| C. J. Clark*          | Peter Larsen     | Elke Tiemeyer      |
| Ray Dellecker *       | Bertrand Leigh   | Douglas Way*       |
| Yadvindra Dhami       | Frank Lien       | Thomas Wenzel      |
| Wim Driessen*         | Keith Loftstrom  | Mike Westermeier   |
| Alon Drory            | Patrick McHugh   | Kevin Wible        |
| Ted Eaton*            | G. Meijerink     | Bill Wilkie        |
| Rob Eccles            | Matthew Murphy   | Carly Yellin       |
| Peter van den Eijnden | Gary O'Brien     | Dan Young          |
| Bernardo Elayda       |                  | Piotr Zbysinski    |

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

|                   |                       |                      |
|-------------------|-----------------------|----------------------|
| Morris Balamut    | Frans De Jong         | Yinghua Min          |
| Roger Bennetts    | Heiko Ehrenberg       | Mark Moyer           |
| David Bonnett     | Peter van den Eijnden | Benoit Nadeau-Dostie |
| Terry Borroz      | Kazumi Hatayama       | Bruce Peterson       |
| Tapan Chakraborty | Neil G. Jacobson      | Mike Ricchetti       |
| Keith Chow        | Adam Ley              | Gordon Robinson      |
| C. J. Clark       | Dennis Lia            | Robert Russell       |
| Luis E. Cordova   |                       | Douglas Way          |

When the IEEE-SA Standards Board approved this standard on 11 December 2002, it had the following membership:

**James T. Carlo**, *Chair*  
**James H. Gurney**, *Vice Chair*  
**Judith Gorman**, *Secretary*

Sid Bennett  
H. Stephen Berger  
Clyde R. Camp  
Richard DeBlasio  
Harold E. Epstein  
Julian Forster\*  
Howard M. Frazier

Toshio Fukuda  
Arnold M. Greenspan  
Raymond Hapeman  
Donald M. Heirman  
Richard H. Hulett  
Lowell G. Johnson  
Joseph L. Koepfinger\*  
Peter H. Lips

Nader Mehravari  
Daleep C. Mohla  
William J. Moylan  
Malcolm V. Thaden  
Geoffrey O. Thompson  
Howard L. Wolfman  
Don Wright

\*Member Emeritus

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

Alan Cookson, *NIST Representative*  
Satish K. Aggarwal, *NRC Representative*

Noelle D. Humenick  
*IEEE Standards Project Editor*

# Contents

|      |                                                                                    |    |
|------|------------------------------------------------------------------------------------|----|
| 1.   | Overview .....                                                                     | 1  |
| 1.1  | Scope .....                                                                        | 1  |
| 1.2  | Organization of the standard .....                                                 | 1  |
| 1.3  | Context .....                                                                      | 2  |
| 1.4  | Objectives .....                                                                   | 2  |
| 2.   | References .....                                                                   | 3  |
| 3.   | Definitions and acronyms .....                                                     | 3  |
| 3.1  | Definitions .....                                                                  | 3  |
| 3.2  | Acronyms .....                                                                     | 9  |
| 4.   | System modal states .....                                                          | 9  |
| 4.1  | Definition of system modal states .....                                            | 10 |
| 4.2  | Description of system modal state transitions .....                                | 12 |
| 4.3  | System modal state transition timing .....                                         | 15 |
| 4.4  | System I/O pin behavior .....                                                      | 18 |
| 5.   | ISC architecture .....                                                             | 22 |
| 5.1  | Overview .....                                                                     | 22 |
| 5.2  | Specification .....                                                                | 35 |
| 5.3  | Description .....                                                                  | 38 |
| 6.   | ISC Instructions .....                                                             | 39 |
| 6.1  | General properties of ISC instructions .....                                       | 39 |
| 6.2  | Mandatory ISC instructions .....                                                   | 46 |
| 6.3  | Optional ISC programming instructions .....                                        | 52 |
| 6.4  | Optional ISC program control/security instructions .....                           | 59 |
| 6.5  | Optional address/data access instructions .....                                    | 63 |
| 6.6  | Requirements for IEEE Std 1149.1-2001 masters and ISC programming algorithms ..... | 68 |
| 6.7  | Adding new instructions for ISC programming support .....                          | 70 |
| 7.   | Data registers .....                                                               | 70 |
| 7.1  | Device identification register .....                                               | 71 |
| 7.2  | Bypass register .....                                                              | 72 |
| 7.3  | Boundary scan register .....                                                       | 72 |
| 7.4  | ISC_Default .....                                                                  | 72 |
| 7.5  | ISC_Config .....                                                                   | 73 |
| 7.6  | ISC_PData .....                                                                    | 74 |
| 7.7  | ISC_RData .....                                                                    | 74 |
| 7.8  | ISC_Sector .....                                                                   | 74 |
| 7.9  | ISC_Info .....                                                                     | 74 |
| 7.10 | ISC_Data .....                                                                     | 75 |
| 7.11 | ISC_Address .....                                                                  | 75 |
| 7.12 | ISC_Inc .....                                                                      | 75 |

|     |                                                          |     |
|-----|----------------------------------------------------------|-----|
| 8.  | Conformance and documentation requirements .....         | 75  |
| 8.1 | Conformance .....                                        | 75  |
| 8.2 | Documentation .....                                      | 76  |
| 8.3 | BSDL extension for ISC description (STD_1532_2002) ..... | 78  |
| 8.4 | ISC extension structure .....                            | 79  |
| 8.5 | ISC BSDL attribute definitions .....                     | 81  |
| 9.  | Data file format .....                                   | 110 |
| 9.1 | Data file syntax specification .....                     | 110 |
| 9.2 | Description .....                                        | 113 |
| 9.3 | Mapping data onto a flow .....                           | 114 |
| 9.4 | Merging flows for concurrent programming .....           | 115 |
| 9.5 | Data file transmission CRC calculation .....             | 117 |
|     | Annex A (informative) Use of deviced features .....      | 119 |
|     | Annex B (informative) ISC programming flows .....        | 123 |
|     | Annex C (informative) Example ISC BSDL/data file .....   | 134 |

# IEEE Standard for In-System Configuration of Programmable Devices

## 1. Overview

### 1.1 Scope

This standard describes a series of mandatory and optional programming instructions and associated data registers that define a standard methodology for accessing and configuring programmable devices. These additional registers and instructions extend the capabilities of devices that comply with IEEE Std 1149.1<sup>TM</sup>-2001 such that the IEEE Std 1149.1-2001 Test Access Port may be used for configuration activities. A data description format and extensions to Boundary-Scan Description Language (BSDL) are also specified that provide for the development of standardized automation tools for device programming.

### 1.2 Organization of the standard

This standard is divided into nine clauses and three annexes, as follows:

- Clause 1 describes the scope, context, objectives, and organization of this standard.
- Clause 2 provides references to other IEEE documentation needed to understand this document.
- Clause 3 defines terms and acronyms used in this standard.
- Clause 4 defines the variations of system state that are unique to programmable logic devices.
- Clause 5 describes the general architecture of an in-system configurable (ISC) device.
- Clause 6 describes the ISC instruction set.
- Clause 7 describes the ISC register set and the operation of each register.
- Clause 8 describes conformance and documentation requirements.
- Clause 9 provides a data file format to be used to describe data to be downloaded to an ISC device.
- Annex A provides a device designer with a rationale for implementing various features of this standard.
- Annex B describes programming flows for combinations of devices compliant with this standard.
- Annex C gives a coordinated example of a BSDL file and data file for a hypothetical ISC device.

### 1.3 Context

Figure 1 shows a printed circuit board containing many types of devices. Of these, some could be compliant with IEEE Std 1149.1-2001 for the support of testing activities. Three devices labeled “ISC” are in-system configurable, meaning they are programmable devices that may not have their programming states loaded into them before the board has been assembled. These devices may be programmed after they have been attached to the board, via a physical and logical protocol. The physical protocol for programming utilizes the IEEE Std 1149.1-2001 Test Access Port (TAP) pins, which allows a multiplicity of devices (both ISC and conventional) to be accessed serially with four or five wires. The logical protocol is based upon new instructions added to the IEEE Std 1149.1-2001 instruction set and is the subject of this standard document.

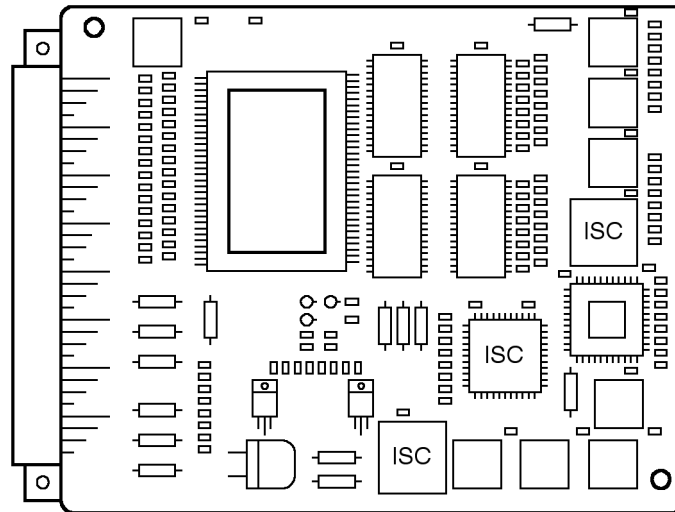


**Figure 1—A printed circuit board containing a variety of components, including three ISC integrated circuits**

The ISC devices also contain testability circuitry compliant with IEEE Std 1149.1-2001. This allows them to participate in manufacturing tests that detect and diagnose faults such as open solder joints and shorts either before or after they are programmed.

### 1.4 Objectives

The objective of this standard is to enhance user access to IEEE Std 1149.1-2001-based programmable devices. Since such devices are built on the foundation of the IEEE Std 1149.1-2001 Test Access Port and TAP Controller state machine, they shall already comply with that standard.

Devices that adhere to this standard will realize significant savings in software, product development, and manufacturing costs related to ISC. New tools will not need to be developed to support new vendor parts with ISC features as they arrive in the marketplace. These parts can be integrated immediately into existing systems and tools.

In addition, tools based on this standard may be developed upon the existing IEEE Std 1149.1-2001-based tool sets and knowledge base, rapidly incorporating the features of this standard.

This standard also allows for multiple devices from independent vendors to be programmed concurrently. Development time and manufacturing costs (in terms of programming time) can thereby be significantly reduced, even in the multivendor environment. In the past, one family of devices from a single vendor may have been incompatible with another family from that same vendor, with respect to the programming process.

The standard also serves board and system designers by providing a consistent framework for system initialization, stabilization, and event initiation, which is important for robust system design.

The standard will realize the following additional benefits:

- It will serve as the development model for new entrants to the ISC industry and guarantee their immediate access to all the latest tools, development systems, and test and manufacturing flows and processes.
- It will facilitate innovations potentially including applications in reconfigurable computing, emulation, diagnostics, and field upgrading that would be usable across vendors.
- It will encourage tool vendors targeting the programmable device marketplace that there will be sufficient return on their investment to warrant tool development.

## 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.

IEEE Std 1149.1-2001, IEEE Standard Test Access Port and Boundary-Scan Architecture.<sup>1,2</sup>

## 3. Definitions and acronyms

### 3.1 Definitions

For the purposes of this standard, the following terms and definitions apply. See also IEEE Std 1149.1-2001 for definitions of terms used in this document.

**3.1.1 adaptive programming (algorithm):** This term may be used when the minimum burn time may vary among identical devices. To optimize the overall programming time, the device can be queried for the status of the programming operation, which will be given by the optional “Operation Success Indication.” If the programming operation has been successfully completed, then the next step in the programming process can commence. If the device has not completed its programming operation to satisfaction, then it will continue until completion or until the maximum time has been met. This process will require the looping of one or more instructions for successful completion. *Contrast: deterministic programming.*

NOTE—See 5.1.10.

**3.1.2 addressing:** Programmable devices usually contain one or several arrays of memory locations. Each memory location has an address, which must be provided when reading or programming memory content.

**3.1.3 asymmetrically writeable device:** A device with program memory that can only be written to one state. For example, a “1” may be written into a location containing a “0”, but not vice-versa. Such devices must be erased to the blank state before new program data can be written. *Contrast: symmetrically writeable device.*

<sup>1</sup>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/>).

<sup>2</sup>The IEEE standards or products referred to in Clause 2 are trademarks owned by the Institute of Electrical and Electronics Engineers, Incorporated.

**3.1.4 blank:** Programmable devices usually contain one or several arrays of memory locations. The data placed in these locations constitute the device program. Many devices are asymmetrical with respect to writing data into these locations. For example, a “1” may be written into a location containing a “0”, but not vice-versa. Thus, a “blank” device is filled with a field of overwriteable states. For symmetrically writeable devices, the “blank” state is usually defined as a field of states that implement no programmable functionality. *See also:* **device program**.

**3.1.5 burn time:** An interval of time during which a set (or subset) of device programming bits is presented to on-chip programming functions, which become the definition of the logic functions the device will subsequently implement. In some older technologies, tiny on-chip fuses are vaporized (hence the term *burn*), which requires a minimum time to achieve. Burn time is technology dependent and may be nearly instantaneous in some devices or require many milliseconds in others. Synonyms include bake time, cook time, and soak time.

**3.1.6 chain:** Two or more IEEE Std 1149.1-2001-compliant devices that share common test clock (TCK) and test mode select (TMS) signals and are linked together with the test data output (TDO) signal of one device connected to the test data input (TDI) signal of a subsequent device. The overall chain may be accessed by a 4 or 5-wire test access port (TAP). *See also:* **test access port (TAP)**.

**3.1.7 checksum:** This integer is created by a mathematical process that operates on a stream of data. The integer for this data is then published so that it can be compared to subsequently recalculated integers for that same data. If the data have not been changed or disturbed, the published integer should match a newly calculated integer. The mathematical process is often based upon binary addition (to some modulus) or logical Exclusive OR of bytes or words. There are well-known flaws with such processes that can allow damaged data streams to masquerade as good by having the same checksum. *See also:* **CRC cyclic redundant codes**.

**3.1.8 completion:** In the context of an ISC instruction, this process occurs when the sequence of activities performed during execution has finished. However, the requirements for execution may not be met, or error conditions may be encountered such that successful completion is not possible. These events constitute unsuccessful completion. *See also:* **Execute**.

**3.1.9 complex programmable logic device (CPLD):** A high-capacity programmable logic device (PLD) usually provided in packages with 40 or more pins, essentially an array of programmable logic blocks interconnected by a routing matrix.

**3.1.10 concurrentization:** The process of making two or more devices work concurrently.

**3.1.11 concurrent operation:** The ability to conduct programmable device operations in two or more devices in the same IEEE Std 1149.1-2001 chain concurrently in time. During this process, all devices in a chain operate continuously in the *Run-Test/Idle* TAP Controller state. In particular, the burn times of the devices can be made to elapse concurrently.

**3.1.12 CRC cyclic redundant codes:** This code can be used as an improved checksum process. Such a code (properly constructed) is much less prone to aliasing a bad data stream with a good stream because it encodes the positional information of 1s and 0s in a stream as well as their frequency. (*See also:* **checksum**.)

**3.1.13 deterministic programming (algorithm):** A device implements this algorithm when the minimum required burn time to satisfy programming operations is documented in the BSDL file and does not change for like devices. *Contrast:* **adaptive programming**.

**3.1.14 device program:** A sequence of bits that, when loaded into a programmable device via a protocol, subsequently define the functions it will implement. *See also:* **volatile program** and **nonvolatile program**.

**3.1.15 displaced:** An instruction in the IEEE Std 1149.1-2001 instruction register is displaced when another instruction is shifted in and then a falling edge of TCK in the *Update-IR* TAP Controller state occurs. Note that any instruction besides IDCODE (the default instruction jammed into the IEEE Std 1149.1-2001 instruction register) will be displaced immediately if the TAP Controller enters the *Test-Logic-Reset* TAP Controller state.

**3.1.16 duration:** In the context of an ISC instruction, a period of time that starts when a TAP Controller first enters the *Run-Test/Idle* TAP Controller state. This period of time is given for each ISC instruction, which specifies how much time (typically a minimum) the action(s) performed by that instruction will take to complete.

**3.1.17 dynamic concurrentization:** This process merges the programming algorithms of multiple devices at run time and is based on the intermediate results of the programming operations. With this process, decisions regarding when information is shifted into or out of a device or when operations are executed are made immediately before the action takes place. *Contrast:* **static concurrentization**.

**3.1.18 electrically erasable programmable read-only memory (EEPROM):** Programmable read-only memories have evolved from mask-programmable devices to devices that were erasable by exposure to ultraviolet light (EPROM) to devices that can be electrically erased.

**3.1.19 enabled protection mechanism:** ISC devices may be implemented with the ability to protect the memory content from erasure, reading or writing, in various combinations. When any of these mechanisms is available, it can be enabled, thus preventing the action it can protect against. *See also:* **protect**.

NOTE—See 5.1.11.

**3.1.20 erasable programmable logic device (EPLD):** At first an EPROM-based device, the term now includes flash and EEPROM-based devices as well.

**3.1.21 erase:** An operation that removes any previously installed device program by changing programmed bits to their unprogrammed state. Erasure may be implemented in several ways. For example, there may be a “bulk erase” function that erases the entire device program in one action. There may be selective, or addressed erasure that only erases a specified “word” of the program. Between these two extremes, there may be a “sector” erase feature that erases a group of words. This operation may be inhibited if the device contains a security feature enabled by a Protect operation.

**3.1.22 execute:** In the context of an ISC instruction, execution occurs after the TAP Controller has proceeded to the *Run-Test/Idle* TAP Controller state. In this state, the instruction performs a specific function that may contain multiple steps or a set of activities. After a minimum amount of time (or clocking of TCK) has occurred, the instruction completes execution. Additional time (or clocking of TCK) beyond this minimum requirement does not affect the results of the instruction’s execution.

**3.1.23 explicit addressing:** Address information is provided by an external algorithm when operating on memory content.

**3.1.24 field-programmable gate array (FPGA):** A programmable alternative to a masked gate array using a channeled programmable interconnect matrix to join blocks of configurable logic. Physical characteristics such as line length and number of fuses are not known until routing. This may lead to some timing unpredictability. The advantage comes in logic density and flexibility.

**3.1.25 fixed system pin:** A system pin with an I/O function that is independent of configuration information programmed into the device and has not been designated as an ISC system pin by the device designer.

**3.1.26 flash random access memory (Flash RAM):** A memory technology similar to EEPROM but implemented with a single transistor rather than two.

**3.1.27 flow:** A logical sequence of ISC instructions that performs a basic programming function such as erasing programmable memory, verifying it, or programming it.

**3.1.28 generic array logic (GAL):** An earlier programmable logic technology still widely used, featuring a fixed OR array and programmable AND array.

**3.1.29 implicit addressing:** Address information is provided by an internally managed process when operating on memory content. For example, successive memory locations may be sequentially accessed by an “auto-increment” addressing process supported by the device such that an external algorithm does not have to be aware of how memory locations are organized.

**3.1.30 in-system configuration (ISC):** The ability to load, read back, or erase programming bits in a programmable logic device after it has been attached to a printed circuit board. *See also:* **device program.**

**3.1.31 ISC\_Disable\_Erase signal:** An optional signal used to indicate that program data erasure is not allowed.

NOTE—See 5.1.11.3.

**3.1.32 ISC\_Disable\_Program signal:** An optional signal used to indicate that program data writing is not allowed.

NOTE—See 5.1.11.2.

**3.1.33 ISC\_Disable\_Read signal:** An optional signal used to indicate that program data reading is not allowed.

NOTE—See 5.1.11.1.

**3.1.34 ISC\_Done signal:** A signal that is used to indicate that a device program has been successfully written to the satisfaction of an external algorithm. When this signal is cleared, the device is considered to be unprogrammed, even if physically, the device does contain programmed bits. When this signal is set, the device is considered to be programmed and ready for operation. In volatile devices, this signal is automatically cleared at power-up. In nonvolatile devices, this signal maintains its state during nonpowered intervals. This signal can be set by programming operations described in this standard, and it may be protected by a security function implemented within the device.

**3.1.35 ISC\_Enabled signal:** A signal that is used to indicate that a device has been successfully conditioned for programming activity. This signal is cleared during power-up for both volatile and nonvolatile devices.

NOTE—See Clause 4.

**3.1.36 ISC system pin:** A system pin with an I/O function that is defined by configuration information programmed into the device, or has been designated as an ISC system pin by the device designer.

**3.1.37 load:** In the context of an instruction, an instruction is loaded when opcode bits present in the IEEE Std 1149.1-2001 instruction shift register are updated into the parallel-hold portion of the register by passing through the *Update-IR* TAP Controller state. At the time an instruction is loaded, its data register is selected for data shifting. *Contrast:* **execute.**

**3.1.38 mission logic:** *See:* **system logic.**

**3.1.39 mission pin:** *See: system pin.*

**3.1.40 noninvasive instruction:** Instructions from IEEE Std 1149.1-2001 (including BYPASS, SAMPLE, PRELOAD, IDCODE, and USERCODE) that maintain the device in system mode rather than in test mode. Thus, these instructions do not interfere with the system behavior of the device. *Contrast: pin-permission instruction.*

**3.1.41 nontest instruction:** Instructions that bring the device out of test mode and put it into unprogrammed, operational, or ISC accessed mode. These instructions can be, but are not limited to, any of the noninvasive instructions or any of the instructions defined in this standard. *Contrast: test instruction.*

**3.1.42 nonvolatile program:** Device programming that is retained by a device even when power is removed and later restored.

**3.1.43 nullified:** Proper ISC instruction operation is predicated on the device being in the ISC accessed system modal state while it is being executed. If this condition is not met, the instruction's operation is nullified. A nullified ISC instruction will still place a defined register in the shift path and shift bits through it as required by IEEE Std 1149.1-2001, but the ISC actions defined for the instruction by this standard will not be completed and the result of its execution is undefined.

NOTE—See Clause 4.

**3.1.44 optimize:** ISC programming activities can be optimized for time efficiency. There are two forms of optimization. First, if a device has a specific instruction loaded, it may be redundant to reload this instruction. This is called “IR optimization.” Second, when a register is targeted between TDI and TDO, “DR optimization” can be employed by shifting out captured data while new update data are being shifted in, rather than first shifting out captured data and then shifting in new update data. When a chain of devices is operated, optimization can be performed, but each device in the chain must be considered. For example, if two of three devices in subsequent activities will be executing the same instruction but the third will need a new instruction loaded, then IR optimization is not possible because loading the third device's instruction forces the reload of the first two.

**3.1.45 pin-permission instruction:** Instructions from IEEE Std 1149.1-2001 (including EXTEST, INTEST, CLAMP, HIGHZ, and RUNBIST) and instructions from this standard that place the device in test or programming modes. These instructions take over control of the system I/O pins. *Contrast: noninvasive instruction.*

**3.1.46 program:** An operation that installs externally supplied device programming bits (or a portion thereof) within a programmable device. This operation may be inhibited if the device contains a security feature enabled by a Protect operation.

**3.1.47 programmable array logic (PAL):** An earlier programmable logic technology still widely used, featuring a fixed OR array and programmable AND array.

**3.1.48 programmable logic devices:** This family of devices includes complex programmable logic device (CPLD), electrically erasable programmable read-only memory (EEPROM), erasable programmable logic device (EPLD), field-programmable gate array (FPGA), flash random access memory, and programmable array logic.

**3.1.49 protect:** Traditionally, an operation that prevents a subsequent read operation from being able to read out a device program. This is a security feature used to prevent copying or deciphering of a device program. In some implementations, writing one or more “security” bits located somewhere in the device program enables protection. Once set, these bits block subsequent read operations. Security bits are usually only

cleared by erasing the device program. More recently, this operation has been expanded to include the prevention of unwanted programming or erasure operations.

**3.1.50 read:** An operation that allows a device program to be read back out from a programmed device. This operation may be inhibited if the device contains a security feature enabled by a protect operation.

**3.1.51 signal:** A named signal is used to denote a binary condition used to choose between two outcomes. It is deliberately “implementation-vague” so as not to constrain a designer’s freedom. For example, a signal may be implemented as a bit in a register, or derived as a function of several bits, or as a one-hot decode from an instruction decoder, as fits the designer’s needs. This should not be confused with I/O signals on device pins that may be high, low, or disabled.

**3.1.52 simple programmable logic device (SPLD):** A family of programmable logic devices, including programmable array logic (PAL), generic array logic (GAL), and the erasable programmable logic device (EPLD).

**3.1.53 static concurrentization:** This process merges the programming algorithms prior to any operation being executed on the chain of devices. All decisions regarding optimization of the programming algorithms have been made before run time so that the run-time for the flow is completely predictable. *Contrast:* **dynamic concurrentization.**

**3.1.54 symmetrically writeable device:** A device with program memory that can be written to either state. These devices are often Static RAM-based. *Contrast:* **asymmetrically writeable device.**

**3.1.55 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. In the case of programmable devices, the system logic may not be defined until after the device is programmed. *See also:* **device program.** *Syn.:* **mission logic.**

**3.1.56 system modal states:** IEEE Std 1149.1-2001 defines two modes, “system” mode and “test” mode. Devices that conform to this standard have a system mode that is divided into four modal states. This reflects the fact that programmable devices, at various times in their operation, may (A) be in “Operational” mode containing system logic, may (B) be in an “Unprogrammed” mode with blank logic configuration memory, may (C) be in “ISC Accessed” mode for programming or accessing logic configuration memory, or may (D) be in “ISC Complete” mode when accessing configuration memory is complete before returning to the operational mode.

NOTE—See Clause 4.

**3.1.57 system pin:** A component pin that feeds, or is fed from, the on-chip system logic. In the case of programmable devices, the system logic may not be defined until after the device is programmed. In devices that contain conventional logic in addition to programmable logic, system pins may be classified as ISC system pins and fixed system pins. *Syn.:* **mission pin.** *See also:* **device program.**

NOTE—See 3.1.25 and 3.1.36.

**3.1.58 TAP Controller:** A 16-state finite state machine used to control the activities of the test access port and the test logic within the device. *See also:* **Test Access Port (TAP).**

**3.1.59 test access port (TAP):** Defined by IEEE Std 1149.1-2001. This consists of four (optionally five) device pins dedicated for use by IEEE Std 1149.1-2001 and programming usage defined by this standard. TAP pins are not system pins.

**3.1.60 test instruction:** Instructions that include any pin-permission instruction that puts the device into test mode but not into any of the programming modes.

**3.1.61 verify:** An operation similar to reading that compares an installed device program (or a portion thereof) to a program supplied externally. If a discrepancy is found, an error indication is given. This operation may be inhibited if the device contains a security feature enabled by a protect operation.

**3.1.62 volatile program:** Device programming that is lost when a device has power removed from it.

**3.1.63 write:** *See:* **program.**

## 3.2 Acronyms

The following acronyms are used within this standard.

|        |                                                                                            |
|--------|--------------------------------------------------------------------------------------------|
| BSDL   | Boundary-Scan Description Language (see IEEE Std 1149.1-2001)                              |
| BSR    | Boundary-Scan Register                                                                     |
| CPLD   | Complex Programmable Logic Device                                                          |
| EEPROM | Electrically Erasable Programmable Read-Only Memory                                        |
| EPLD   | Electrically Programmable Logic Device                                                     |
| EPROM  | Erasable Programmable Read-Only Memory                                                     |
| FPGA   | Field-Programmable Gate Array                                                              |
| GAL    | Generic Array Logic                                                                        |
| IC     | Integrated Circuit                                                                         |
| I/O    | Input/Output                                                                               |
| IOM    | Input/Output Module                                                                        |
| ISC    | In-System Configuration                                                                    |
| OST    | Operation Success Test                                                                     |
| PAL    | Programmable Array Logic                                                                   |
| PROM   | Programmable Read-Only Memory                                                              |
| SRAM   | Static Random Access Memory                                                                |
| TAP    | IEEE Std 1149.1-2001 Test Access Port                                                      |
| TCK    | The Test Clock input pin contained in the IEEE Std 1149.1-2001 TAP                         |
| TDI    | The Test Data Input pin contained in the IEEE Std 1149.1-2001 TAP                          |
| TDO    | The Test Data Output pin contained in the IEEE Std 1149.1-2001 TAP                         |
| TLR    | Test-Logic-Reset, the reset state of the IEEE Std 1149.1-2001 TAP Controller state diagram |
| TMS    | The Test Mode Select input pin contained in the IEEE Std 1149.1-2001 TAP                   |
| TRST*  | The Test Reset input pin contained in the IEEE Std 1149.1-2001 TAP (this pin is optional)  |

NOTE—Following the convention used in IEEE Std 1149.1-2001, pins or signals with a trailing asterisk (“\*”) in their names are active low true signals.

## 4. System modal states

A conventional (i.e., nonprogrammable) device compliant with IEEE Std 1149.1-2001 has a “system” mode and a “test” mode. In system mode, the device performs its intended mission even while executing “noninvasive” nontest instructions. In test mode, test instructions control the device’s system I/O pins for testing purposes. A compliant IEEE Std 1149.1-2001 device passes from system mode to test mode when certain IEEE Std 1149.1-2001 instructions become active, as shown in Figure 2.

Devices that are compliant with IEEE Std 1149.1-2001 power-up in system mode. Noninvasive nontest IEEE Std 1149.1-2001 instructions include BYPASS, PRELOAD, SAMPLE, IDCODE, and USERCODE.

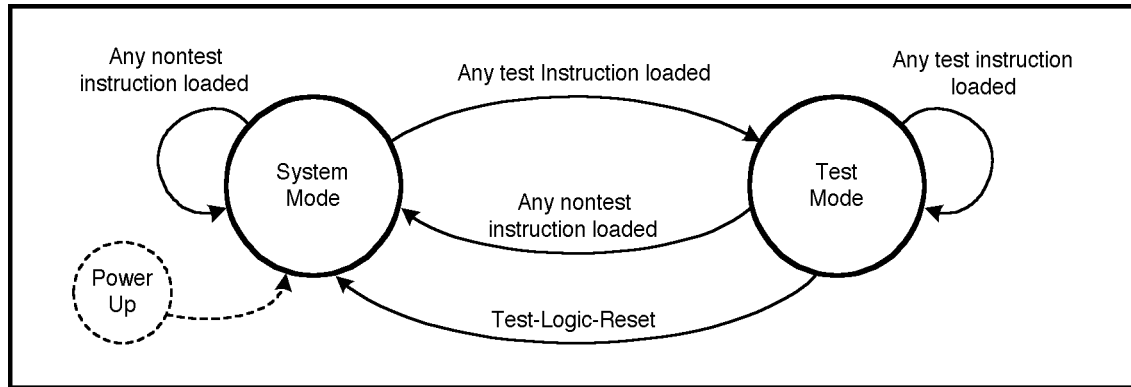


**Figure 2—Transitions from system mode to test mode for a conventional device compliant with IEEE Std 1149.1-2001**

Test instructions include EXTEST, INTEST, CLAMP, HIGHZ, and RUNBIST. Entering the Test-Logic-Reset TAP Controller state will also force the device into system mode.

#### NOTES

1—After a device has been in test mode, it may return to system mode but be unable to resume logical operation in concert with other devices it is connected to because while in test mode, all communication with these devices was severed. It will likely be necessary to issue a system reset to restore proper system operation.

2—User-defined instructions may create new modes of operation that are not depicted in Figure 2.

Unlike conventional integrated circuit devices, a programmable logic device may have many different user-defined operation behaviors depending on the pattern that is currently programmed into the device. To precisely define device behavior before, during, and after programming, the system mode of an ISC device has four different “system modal states”, all comprising the “system” mode of the device as defined by IEEE Std 1149.1-2001. (These are defined in 4.1.) The four system modal states define the “system” behavior of system I/O pins when noninvasive instructions from IEEE Std 1149.1-2001 are loaded. (Test instructions already have I/O behavior specified by IEEE Std 1149.1-2001, and ISC instructions defined herein also have prescribed I/O behavior.)

NOTE—See 5.1.1 for a classification of system pins. For the discussion in this clause, assume that all system I/O pins are associated with programmable logic.

Well-defined and predictable I/O behavior is of paramount importance to designers of boards and systems who must ensure that the power-up behavior and system initialization processes of their designs are robust. This is of particular concern where a design is programmable. It must exhibit robust behavior before, during, and after the programming process.

## 4.1 Definition of system modal states

The four system modal states (see Figure 3) of a programmable device conforming to this standard are as follows:

- **Unprogrammed:** In this system modal state, an ISC device may be blank (see 3.1.4) or incompletely programmed. In this system modal state and while a noninvasive IEEE Std 1149.1-2001 instruction (see 3.1.40) is loaded, the device’s system I/O pins are disabled such that they cannot contend with output drivers that may exist in other devices connected to the ISC device. Any ISC instruction (excepting ISC\_ENABLE) will have its ISC operation nullified (see rule b) of 6.1.1) if executed in

this state. Upon power-up, volatile devices, blank devices, and incompletely programmed nonvolatile devices should reside in this state.

- **ISC Accessed:** In this system modal state, ISC instructions may be used (when permitted) to read, write, verify, protect, or erase the device. In this system modal state and while a noninvasive IEEE Std 1149.1-2001 instruction is loaded, the device's system I/O pins are either disabled or specifically controlled by the content of the device's Boundary-Scan Register as determined in rule d) of 6.1.1.
- **ISC Complete:** This system modal state exists so an external algorithm can control transitions to the Unprogrammed or Operational system modal state after ISC operations are complete. This system modal state is entered as a result of executing the ISC\_DISABLE instruction. This system modal state is temporary; if any instruction other than ISC\_DISABLE is loaded into the instruction register (at the falling edge of TCK in the Update-IR TAP Controller state), this system modal state is exited. Thus, this system modal state cannot exist when a noninvasive IEEE Std 1149.1-2001 instruction is present in the instruction register.
- **Operational:** In this system modal state, the device is ready for its “operational” mode defined by its programming. In this system modal state and while a noninvasive IEEE Std 1149.1-2001 instruction is loaded, the device's system I/O pins take on their programmed system nature of inputs, outputs, bidirectionals, or unused. Any ISC instruction (excepting ISC\_ENABLE) will have its operation nullified (see rule b) of 6.1.1) if executed in this system modal state. Successfully programmed nonvolatile devices, upon power-up, should reside in this system modal state.

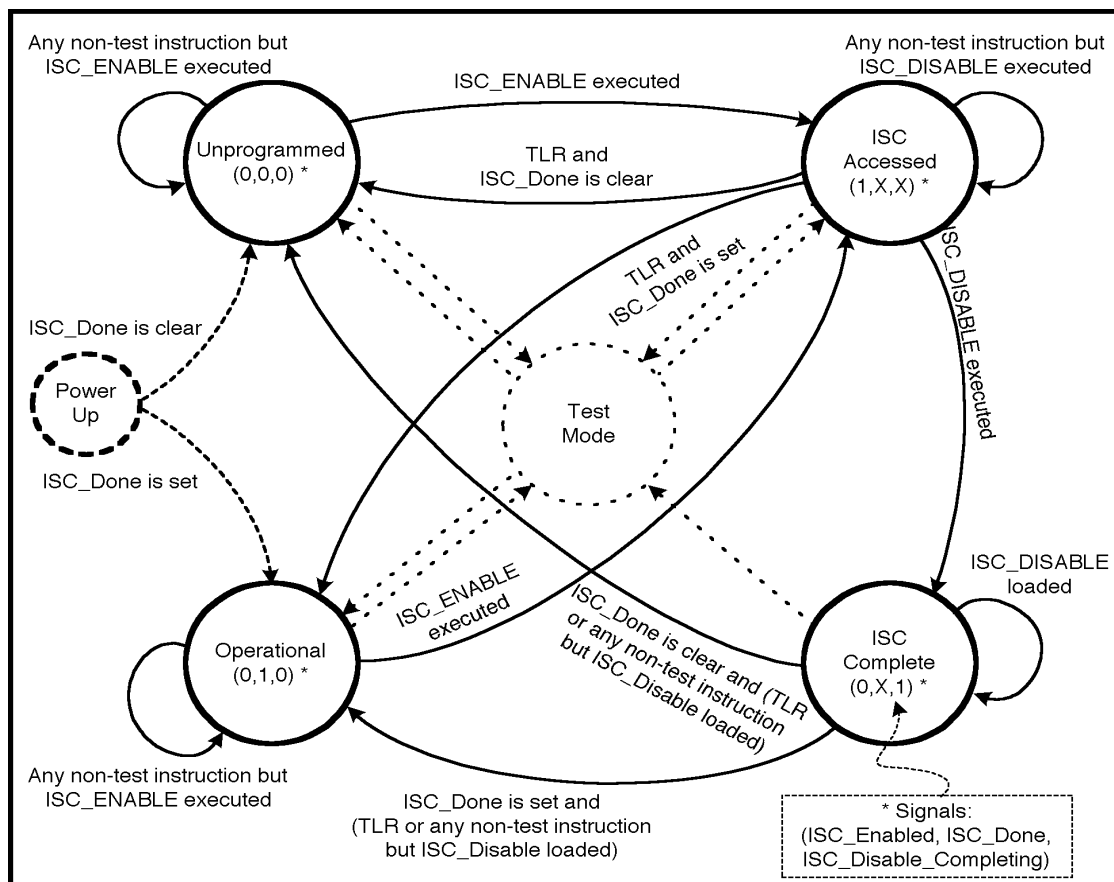


**Figure 3—Four system modal states that comprise the “system mode” of operation for an ISC device and transitions. (A description of the transitions to and from test mode appears in Figure 4. The three signals are described in 4.2 and 4.3.)**

In all system modal states, all public IEEE Std 1149.1-2001 instructions are available and supported per IEEE Std 1149.1-2001. This means that IEEE Std 1149.1-2001 chain integrity or infrastructure testing can be accomplished at any time regardless of the system modal state of any ISC device.

#### NOTES

1—Certain operations conducted in test mode (for example, EXTEST-based interconnection tests) may not be electrically compatible with programming activities (or vice versa) conducted in the ISC Accessed system modal state due to noise or power supply considerations. However, it may be quite reasonable to read out a device USERCODE from the ISC Accessed system\_modal state and later return to an ISC instruction.

2—There may be restrictions that should be observed for a given ISC device regarding the interruption of a programming flow in order to execute a non-ISC instruction.

It is important to realize that a noninvasive IEEE Std 1149.1-2001 instruction such as BYPASS may demonstrate up to three distinct system I/O behaviors depending on the system modal state of the device, as follows:

- a) All pins disabled (Unprogrammed system modal state)
- b) All pins either disabled or specifically controlled by the content of the device's Boundary-Scan Register as determined in rule d) of 6.1.1 (ISC Accessed system modal state)
- c) All pins behaving in their programmed mission mode (Operational system modal state).

This modal behavior conforms to IEEE Std 1149.1-2001 because of the definition of system modal states given by this standard. The plurality of system I/O pin behaviors allows a device to move from executing an ISC instruction to executing a noninvasive IEEE Std 1149.1-2001 instruction with a seamless and predictable system I/O behavior.

NOTE—It can also lead to a potentially surprising result. For example, when a device is in the ISC Accessed system modal state and its pins are controlled by the Boundary-Scan Register, the PRELOAD instruction may be used to change the content of the Boundary-Scan Register. This change, upon the falling edge of TCK in the Update-DR TAP Controller state, will appear at the device I/O pins, simulating the operation of EXTEST. This is because the system behavior of this device in the ISC Accessed system modal state is defined to have the property of the system I/Os being controlled by the Boundary-Scan Register.

## 4.2 Description of system modal state transitions

Figure 3 shows the four system modal states for an ISC device and how transitions between these states occur. (Note that Figure 3 also shows transitions to test mode. These are further described in Figure 4.) System modal state transitions are governed by the following three factors:

- The instruction loaded into the IEEE Std 1149.1-2001 instruction register. This instruction becomes effective on the falling edge of TCK in the *Update-IR* TAP Controller state. ISC instructions will define system I/O behavior at this time, but they also require some time spent in the *Run-Test/Idle* TAP Controller state for their operations to complete.
- The state of an internal “ISC\_Done” signal. (In volatile devices, this signal is cleared at power-up. In nonvolatile devices, this signal is retained during loss of power.) The ISC\_Done signal is set when the device is programmed to the satisfaction of an external ISC programming algorithm. This indicates the device is no longer blank. The ISC\_Done signal is set one of three ways; by executing the ISC\_DISABLE instruction to completion with a specific data pattern set in the ISC\_Config register, by execution of the (optional) ISC\_PROGRAM\_DONE instruction, or by programming one or more specific bits in the program memory array. The ISC\_Done signal also determines the functionality of the device system I/O pins on power-up. If the ISC\_Done signal is clear, the device I/O pins are disabled.

NOTE—The ISC\_Done signal may be subject to protection by a security feature that may prevent changes to its state.

- The state of an internal volatile signal called the “ISC\_Enabled” signal. This signal is set by the execution of the ISC\_ENABLE instruction and may be cleared by subsequent events. In both volatile and nonvolatile devices, this signal is cleared at power-up.

Two of these factors (the ISC\_DISABLE instruction is loaded and ISC\_Enabled is set) are utilized to derive a third signal named “ISC\_Disable\_Completing.” This signal is used to indicate that the ISC Complete system modal state has been entered legally, that is, from the ISC Accessed system modal state by action of the ISC\_DISABLE instruction. The derivation of ISC\_Disable\_Completing is conceptualized in 4.3, although any number of other derivations are possible.

The Unprogrammed system modal state is entered upon power up if the ISC\_Done signal is cleared. The only way to exit the Unprogrammed system modal state is by loading and executing the ISC\_ENABLE instruction (see 6.2.1), which sets the ISC\_Enabled signal upon successful completion. Setting the ISC\_Enabled signal causes a transition to the ISC Accessed system modal state. If the ISC\_ENABLE instruction is not successfully completed, the ISC\_Enabled signal is not set and the device stays in the Unprogrammed system modal state.

NOTE—If while in the Unprogrammed or Operational system modal state an ISC instruction other than ISC\_ENABLE is loaded, it will not operate to fulfill its intended mission (see rule b) of 6.1.1); that is, its operation will be nullified.

Once the ISC Accessed system modal state is entered (that is, the ISC\_Enabled signal is set), all other ISC instructions are enabled to operate as documented in Clause 6. The preferred way to exit the ISC Accessed system modal state is by executing the ISC\_DISABLE instruction, which clears the ISC\_Enabled signal (and potentially sets the ISC\_Done signal) and causes a transition to the ISC Complete system modal state. If the Test-Logic-Reset TAP Controller state is entered while in the ISC Accessed system modal state, then the currently loaded instruction is aborted and either the Unprogrammed system modal state is entered if ISC\_Done is clear, or the Operational system modal state is entered if ISC\_Done is set. In typical ISC programming flows, a sequence of operations, such as erasing, writing, reading, and protecting would all be done successively within the ISC Accessed system modal state.

## NOTES

1—If the ISC Accessed system modal state is left abnormally (by traveling to the *Test-Logic-Reset* TAP Controller State) while the ISC\_Done signal is set, it is possible that an improperly programmed device could be set to the Operational system modal state. It is the responsibility of external controlling algorithms to avoid this. In some technologies, it is possible that the ISC\_Done signal can only be cleared by an erase operation. In such devices, it will be important to construct the erase operation to clear the ISC\_Done signal early in the erasure process. This reduces the chance that terminating the erasure process before completion will leave the ISC\_Done signal set.

2—The ISC\_Disable\_Completing signal is also generated by the loading of ISC\_DISABLE as seen in 4.3.

In the ISC Complete system modal state, the device is ready to transition to either the Operational or Unprogrammed system modal states, depending on the state of the ISC\_Done signal. The transition from ISC Complete is accomplished by loading any instruction other than ISC\_DISABLE into the instruction register. One way to do this is to cause the TAP Controller to enter the *Test-Logic-Reset* TAP Controller state, which forces the IDCODE instruction into the instruction register, which in turn forces a transition to either the Unprogrammed or the Operational system modal state depending on the state of the ISC\_Done signal. However, if several ISC devices in a chain are in the ISC Complete system modal state, then it may be desirable to sequence their transition to the Operational system modal state in a particular order. Loading one ISC device with BYPASS while reloading the others with ISC\_DISABLE, and so progressing until all devices are loaded with BYPASS would accomplish this objective.

Once a device enters the Operational system modal state from the ISC Complete system modal state, it is fully programmed and capable of performing its mission function. System I/O pins are subject to the

instruction that was used to displace the ISC\_DISABLE instruction. If it is a test instruction (like EXTEST), then that instruction controls the system I/O pins. If the instruction was an ISC instruction (like ISC\_ENABLE), then it will control the system I/O pins per rule d) of 6.1.1. If the new instruction was a noninvasive IEEE Std 1149.1-2001 instruction (like BYPASS), the system I/O pins are governed by the programming of the device and take on their intended system function. The only way to leave the Operational system modal state (excepting a transition into test mode) is to load and execute the ISC\_ENABLE instruction, thus returning to the ISC Accessed system modal state. This might be done to reprogram, verify, or erase a device's programming at some future point in time.

## NOTES

1—If while in the Operational system modal state an ISC instruction other than ISC\_ENABLE is loaded, it will not operate to fulfill its intended mission (see rule b) of 6.1.1); that is, its operation will be nullified.

2—If the Operational system modal state is entered abnormally (in response to the *Test-Logic-Reset* TAP Controller state), the device may not be capable of performing its mission function. If this abnormal transition is performed, it is the responsibility of the programming algorithm to subsequently erase the ISC\_Done signal so that the device no longer appears to be properly programmed.

Transitions to and from test mode in Figure 3 were shown with unlabeled dotted lines to avoid clutter. Figure 4 shows how transitions between the system modal states and test mode occur. From any system modal state, the transition to test mode occurs when a test instruction (EXTEST, INTEST, CLAMP, HIGHZ, or RUNBIST) is loaded.

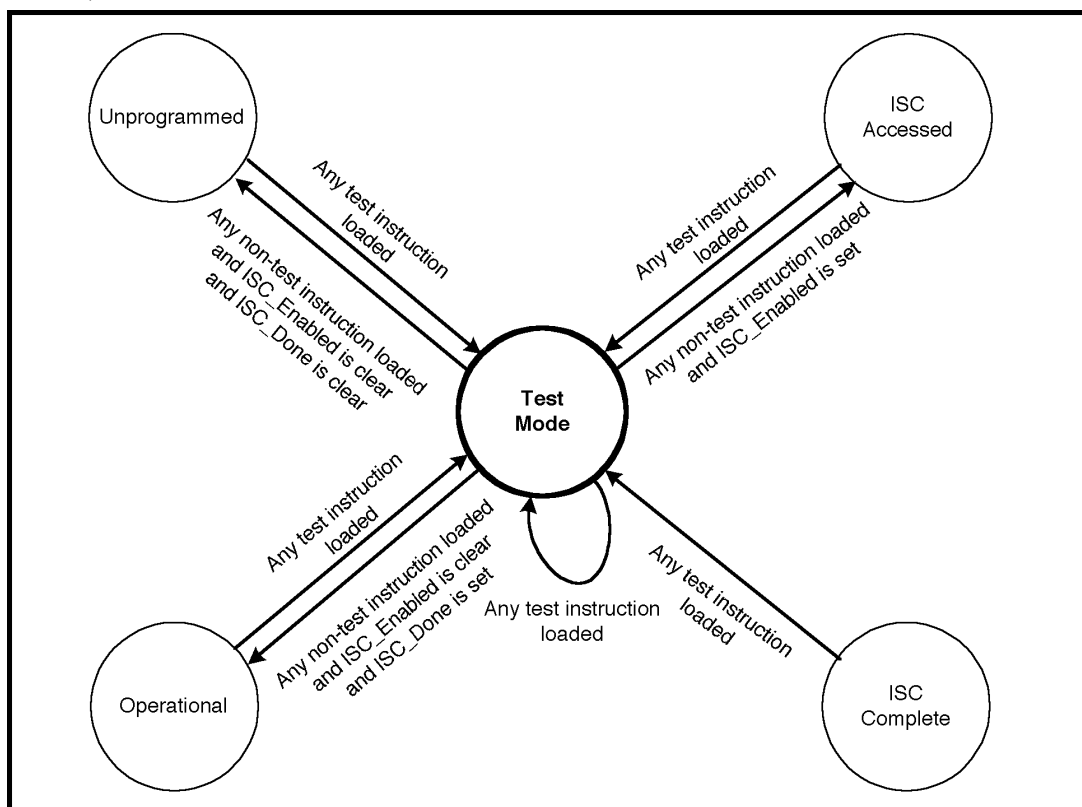


**Figure 4—Transitions to and from system modal states and test mode. (A description of the transitions among the four modal states appears in Figure 3.)**

There are three ways to return to a system modal state from test mode. All of these require that a nontest instruction (BYPASS, PRELOAD, SAMPLE, IDCODE, USERCODE) or an ISC instruction described by this standard be loaded and are qualified by the states of the ISC\_Enabled and ISC\_Done signals.

The design of a particular ISC device may not be able to support all the transitions shown in Figure 4. For example, depending on implementation technology, when in the ISC Accessed system modal state, an ISC device may generate high programming voltages internally. While these voltages are present, it may be difficult to ensure correct operation of the device if it also makes transitions on system I/O pins. In all cases, the designer is required to ensure that the IEEE Std 1149.1-2001 instructions operate as specified (see permission s) of 5.2 and related notes) and to document instruction sequences that should be avoided. (See 8.2.)

### 4.3 System modal state transition timing

Transitions on output pins and between system modal states occur at prescribed times shown in Figure 5. (This drawing does not show time to scale, but represents the ordering of events.)

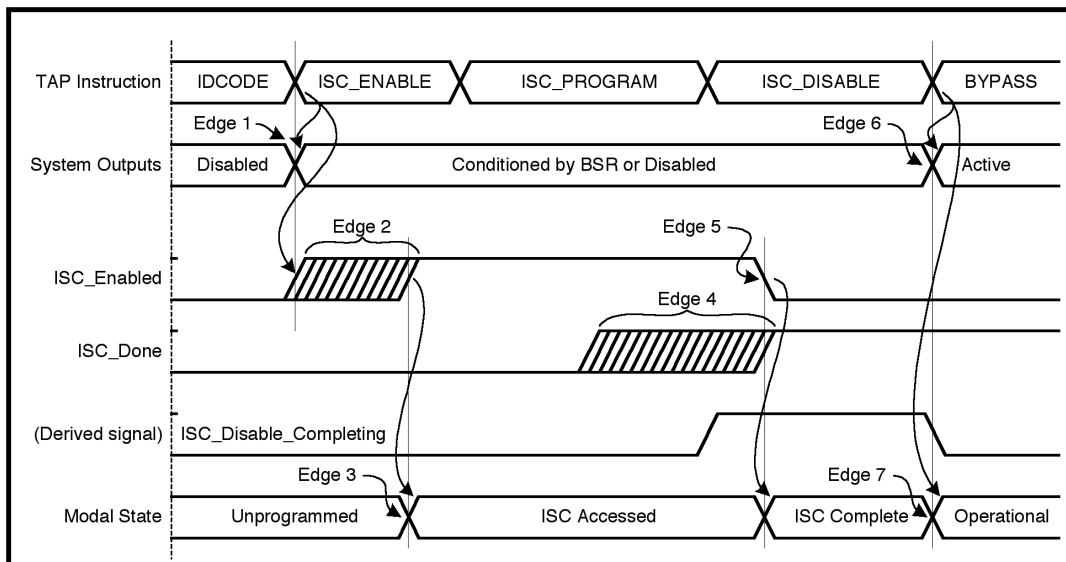


**Figure 5—Waveforms (not to scale) for signals that control system outputs and system modal state transitions**

NOTE—In Figure 5, the convention is used that “set” is high and “clear” is low.

There are seven edges of interest shown in Figure 5. They are as follows:

- Edge 1: System I/O pins that were disabled take on the state given by rule d) of 6.1.1 when the instruction changes from IDCODE to ISC\_ENABLE (triggered by the falling edge of TCK in the *Update-IR* TAP Controller state).
- Edge 2: The ISC\_Enabled signal is asserted somewhere between the time the instruction changes from IDCODE to ISC\_ENABLE (triggered by the falling edge of TCK in the *Update-IR* TAP Controller state) and the last rising edge of TCK in the *Run-Test/Idle* TAP Controller state that satisfies the minimum duration specification (see rule i) of 6.1.1) for ISC\_ENABLE.
- Edge 3: The system modal state changes from Unprogrammed to ISC Accessed when the ISC\_Enabled signal is asserted.
- Edge 4: The ISC\_Done signal is set either by programming (with the ISC\_PROGRAM instruction) one or more bits in the program data area, by the execution of the ISC\_DISABLE instruction, or by the execution of the ISC\_PROGRAM\_DONE instruction. The ISC\_Done signal is set no later than

the last rising edge of TCK in the *Run-Test/Idle* Tap Controller state that satisfies the minimum duration specification [see rule i) of 6.1.1 or burn time, rule c) of 6.4.1.1 and rule d) of 6.2.3.1) as appropriate] for the instruction setting *ISC\_Done*.

- Edge 5: The *ISC\_Enabled* signal is cleared during execution of the *ISC\_DISABLE* instruction. This occurs on the last rising edge of TCK in the *Run-Test/Idle* Tap Controller state that satisfies the minimum duration specification (see rule i) of 6.1.1) for *ISC\_DISABLE*. This causes the system modal state to change from *ISC Accessed* to *ISC Complete*.

NOTE—In a preferred embodiment, the *ISC\_Done* and *ISC\_Enabled* signals (at edges 4 and 5) transition at the same time. If *ISC\_Done* should transition before *ISC\_Enabled*, it is possible for a transition to the *Test-Logic-Reset* TAP Controller state to occur between them, resulting in possible confusion about what *ISC\_Done* means. (It will be the responsibility of external programming algorithms to avoid this confusion.) Simultaneous transition of these two signals requires that the *ISC\_Done* signal be controlled by the *ISC\_DISABLE* instruction.

- Edge 6: When any non-ISC instruction is loaded into the IEEE Std 1149.1-2001 instruction register (upon the falling edge of TCK in the *Update-IR* TAP Controller state), the system outputs take on their programmed nature.
- Edge 7: When any instruction other than *ISC\_DISABLE* is loaded into the IEEE Std 1149.1-2001 instruction register (upon the falling edge of TCK in the *Update-IR* TAP Controller state), the system modal state changes from *ISC Complete* to *Operational*.

The case where *ISC\_DISABLE* is executed and then *ISC\_ENABLE* is immediately executed is shown in Figure 6. When this occurs, the system modal state changes briefly to *Operational*, but because the IEEE Std 1149.1-2001 instruction register still contains an ISC instruction, the system I/O pins remain in the state governed by rule d) of 6.1.1. Once the *ISC\_ENABLE* instruction completes, the device is again in the *ISC Accessed* system modal state at Edge 8. Thus, if several ISC operations are executed back-to-back (for example, programming, followed by read-back), the device I/O pins will not leave their quiescent state even when these ISC operations are bounded by *ISC\_ENABLE* and *ISC\_DISABLE* instructions.

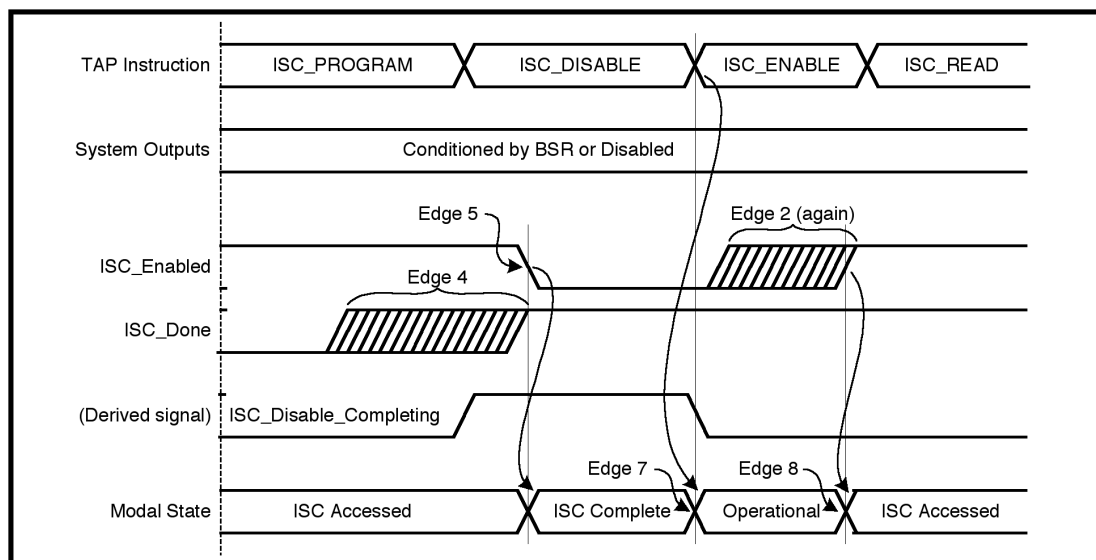


**Figure 6—Waveforms observed when *ISC\_ENABLE* is executed upon completion of *ISC\_DISABLE***

The derived signal *ISC\_Disable\_Completing*, first referenced as a control signal in Figure 3 and as its waveforms appear in Figure 5 and Figure 6, may at first appear to be a signal from the instruction decoder (call it *ISC\_Disable\_Decoded*) that is set whenever the *ISC\_DISABLE* instruction is loaded in the IEEE Std 1149.1-2001 instruction register. However, it must be a function of both *ISC\_Disable\_Decoded*

and ISC\_Enabled as shown in Figure 7. This reflects the fact that ISC\_Disable\_Completing indicates that the ISC\_DISABLE instruction was loaded properly while the device was in the ISC Accessed system modal state (that is, the ISC\_Enabled signal is set).

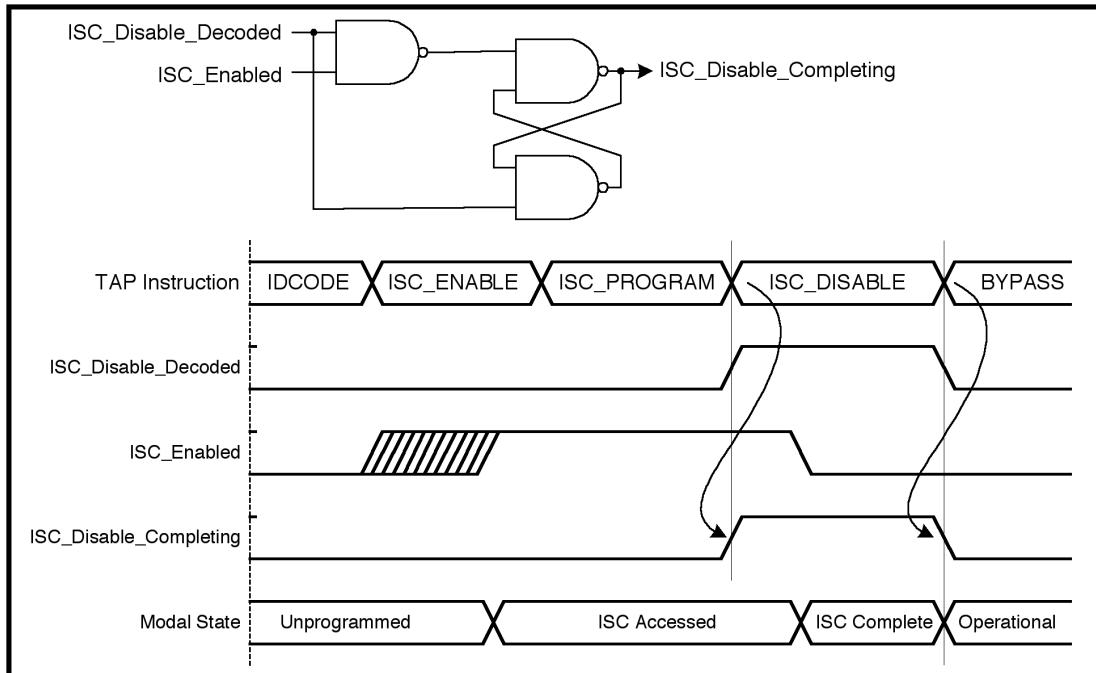


**Figure 7—Derivation of the ISC\_Disable\_Completing signal. The circuit shown is conceptual as there are other implementations with the same result**

The case where the ISC\_DISABLE instruction is loaded when the device is not in the ISC Accessed system modal state is shown in Figure 8. When this occurs, the system modal state does not change.

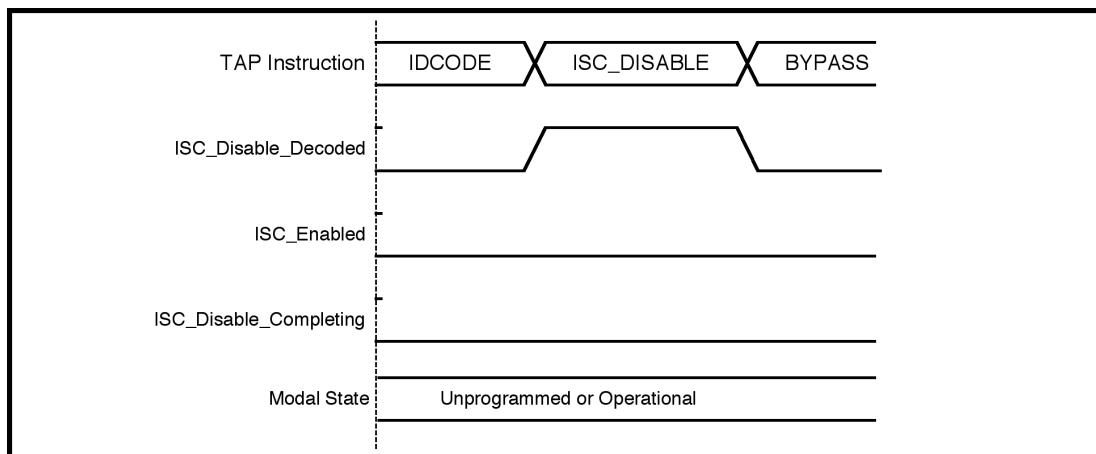


**Figure 8—The system modal state remains unchanged, and the ISC\_Disable\_Completing signal remains cleared when the ISC\_DISABLE instruction is not properly prefaced with the ISC\_ENABLE instruction**

## 4.4 System I/O pin behavior

System I/O pins of a device compliant with rules given herein have behavior summarized as follows:

- If an ISC instruction defined by this standard (see Clause 6) is loaded in the instruction register, the system I/O pins are controlled per rule d) of 6.1.1 regardless of what system modal state the device may be in.

NOTE—An instruction is loaded in the instruction register on the falling edge of TCK in the *Update-IR* TAP Controller state.

- If an IEEE Std 1149.1-2001 test mode instruction (EXTEST, INTEST, RUNBIST, CLAMP, HIGHZ) is loaded in the instruction register, the system I/O pins are controlled as defined by that standard regardless of what system modal state the device may have been in before the test mode instruction was loaded.
- If an IEEE Std 1149.1-2001 nontest mode instruction (BYPASS, IDCODE, USERCODE, SAMPLE, PRELOAD) is loaded in the instruction register, the system I/O pin behavior is determined by the system modal state of the device when the instruction is loaded.

NOTE—The system modal state of the device cannot be changed by the execution of non-ISC instructions. The system modal state may, however, change from ISC Complete to Operational or Unprogrammed upon loading any instruction other than ISC\_DISABLE. The system modal state of the device can be changed by passing to the *Test-Logic-Reset* TAP Controller state, potentially resulting in a change of system I/O pin behavior.

It is when an IEEE Std 1149.1-2001 nontest mode instruction is loaded in an ISC device that there is a choice of behaviors for the system I/O pins. These are summarized as follows:

- If the device is in the Unprogrammed system modal state, the system I/O pins are disabled.
- If the device is in the ISC Accessed system modal state, the system I/O pins are governed by rule d) of 6.1.1.
- If the device is in the Operational system modal state, the system I/O pins operate as determined by the programming of the device.

NOTE—If at the point of time an IEEE Std 1149.1-2001 nontest instruction was loaded into the device it was in the ISC Complete system modal state, the device transitions to the Operational or Unprogrammed system modal state because the ISC\_DISABLE instruction has been displaced.

When a device complying with this standard is in the Operational system modal state, it behaves as if it was a conventional device that contains both IEEE Std 1149.1-2001 testability circuitry and mission circuitry as shown in Figure 9. The typical programmable device contains a series of Input-Output Modules (IOMs) that contain Boundary-Scan Register cells and a three-state output driver. The Boundary-Scan Register cells shown have the required Capture Flip-flop (C) and an Update Latch (U). The IOM depicted is fully bidirectional.

NOTE—The IOM shown in Figure 9 can be implemented more efficiently with a single bidirectional data cell structure (see IEEE Std 1149.1-2001) rather than the two-cell data structure shown here for the sake of simplicity. This will eliminate 33% of the cells, saving silicon area as well as test data shift time and test program storage space.

When a device is configured, a given system I/O pin may be fully bidirectional, or limited to being an input, output, or unconnected pin. This limiting may be accomplished by tying the control cell input (the lowest cell in the IOM in Figure 9) to a fixed “0” or “1”. When a pin-permission instruction such as EXTEST is loaded into the instruction register, the driver data and control signals are derived from the Update latches (U) as IEEE Std 1149.1-2001 requires.

NOTE—If a pin’s programming is “unused,” it should at the very least still be monitored by the input Boundary-Scan Register cell of the IOM. The IOM could be treated as fully bidirectional, however, to allow a testing procedure to

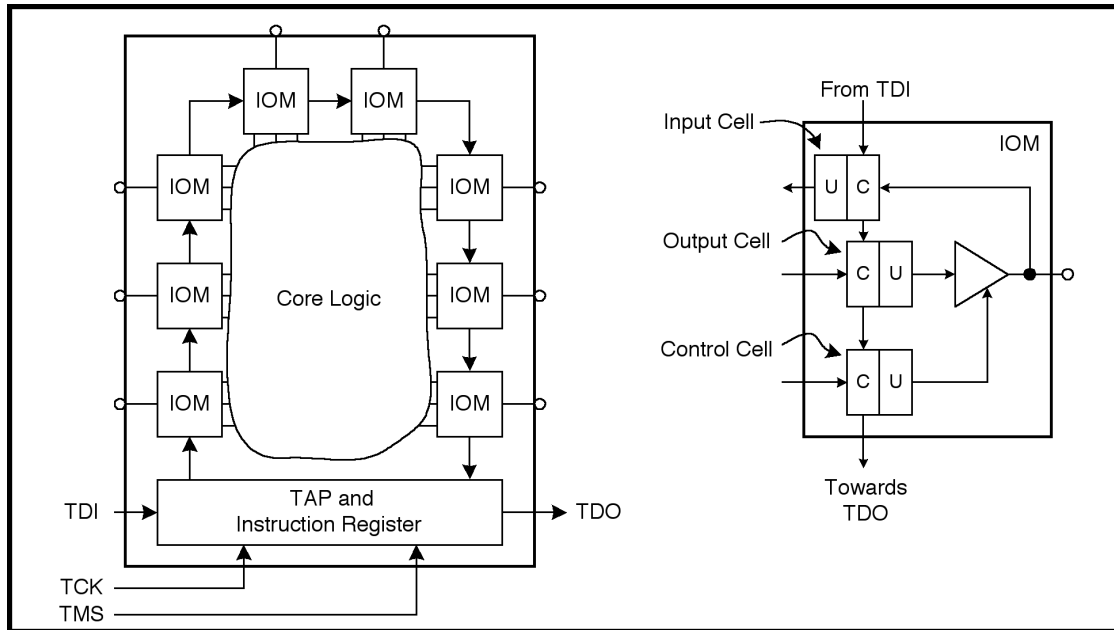


**Figure 9—A programmed device and its connection to the Boundary-Scan Register**

discover whether shorts exist from “unused” pins to other pins. This would allow the testing procedure to establish an identity (using the IOM driver) for the unused pin that the IOM receiver would verify. Testing even the unused pins for shorts will help discover manufacturing process problems and ensure the integrity of the system in the event the programming is subsequently changed such that unused pins are later used.

When a programmable device is in the Unprogrammed system modal state and an IEEE Std 1149.1-2001 nontest instruction is loaded, then there is, by definition, no core logic as shown in Figure 10. In order to meet the requirement (see rule e) of 6.1.1) that a device in the Unprogrammed system modal state have all of its drivers inactive, the control cell in the IOM should produce a logic level that will disable the driver, a “0” in this case. When a programmable device is in the ISC Accessed or ISC Complete system modal states, then the device drivers should either be controlled by the Boundary-Scan Register or be inactive as governed by rule d) of 6.1.1. Example data cell designs are shown in Figure 11. Two example control cell designs that meet the two options for rule c) of 6.1.1 are shown in Figure 12 and Figure 13.

The data cell design shown in Figure 11(A), used when the device is intended to exhibit the HIGHZ-like behavior (see rule d) of 6.1.1) is identical to a “generic” Boundary-Scan Register data cell design from IEEE Std 1149.1-2001, which may be used for inputs or outputs. It does not make use of any of the specialized ISC signals defined by this standard. The “Mode” signal is generated by the instruction decoder as described in the table within Figure 11. Note that Mode changes on the falling edge of TCK in the *Update-IR* TAP Controller state.

NOTE—The cell shown in Figure 11(A) is known as “BC\_1” in BSDL. (See IEEE 1149.1-2001, B.9 “The Standard VHDL Package STD\_1149\_1\_2001” and B.10.2 “Cell description constants” for information on describing Boundary-Scan Register cells.)

The data cell design shown in Figure 11(B) uses ISC\_Enabled to ensure that when the device is intended to exhibit the CLAMP-like behavior (see rule d) of 6.1.1), it will pass the Boundary-Scan Register content to the pin.

NOTE—With respect to the BSDL description (see IEEE Std 1149.1-2001) of the data cell design depicted in Figure 11(B), while materially different in the output multiplexing structure from the cell design called BC\_1 in BSDL, this design is still equivalent to BC\_1. This is because the BC\_1 structure describes the capture behavior of the cell and not the multiplexing structure on its output.

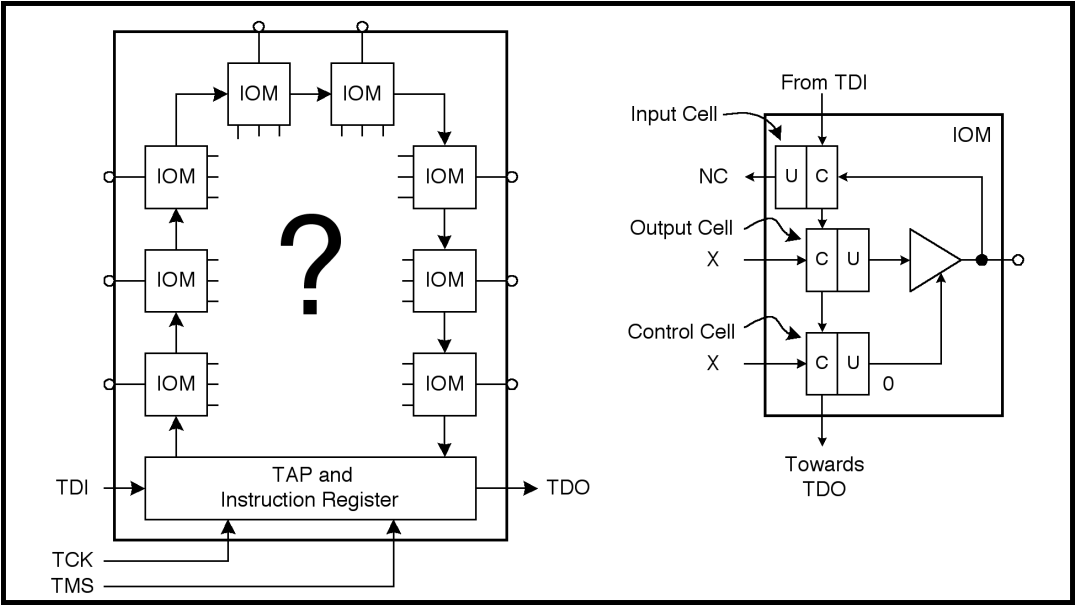


Figure 10—An unprogrammed device and how the unconnected IOM signals should be

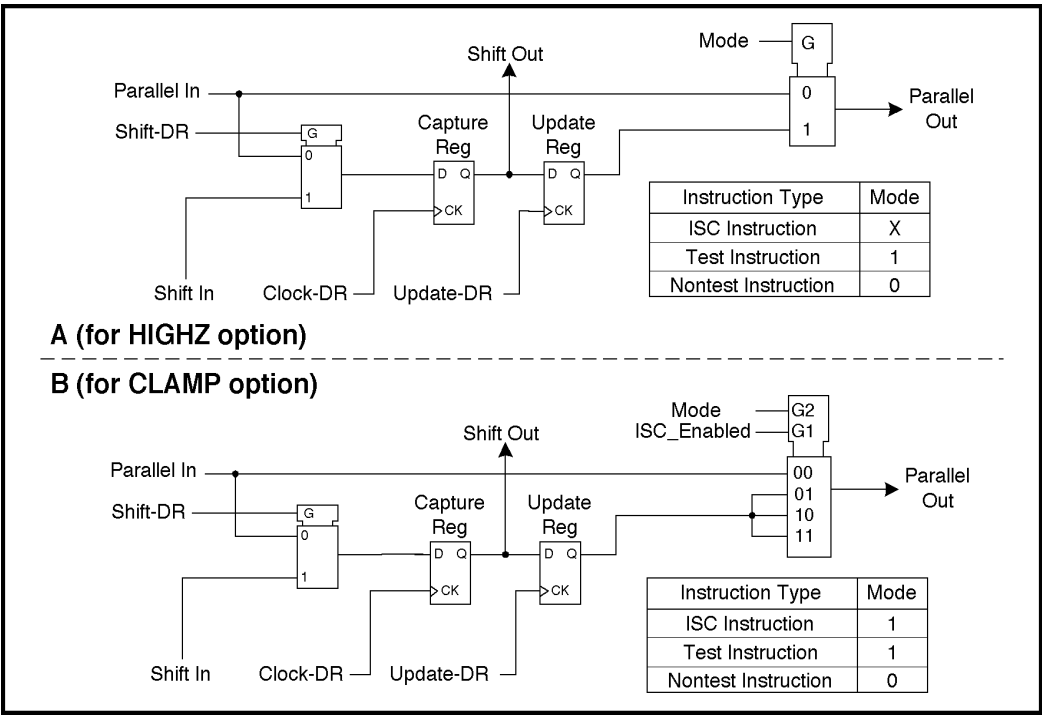


Figure 11—Possible data cell designs that meet the requirements of this standard

The control cell design incorporates ISC signals in its design to govern when an IOM driver is enabled or disabled, and where its source of control data comes from.

Figure 12 shows a possible control cell design that will disable the system I/O pins whenever an ISC instruction is loaded, or whenever a nontest instruction is loaded and the device is in the ISC Accessed system modal state. Loading a test mode instruction will allow the Boundary-Scan Register to control the

state of the driver enable. The “Mode1” and “Mode2” signals are generated by the instruction decoder as described in the table within Figure 12. Note that mode signals change on the falling edge of TCK in the *Update-IR* TAP Controller state, but that the ISC\_Enabled and ISC\_Done signals change at other times governed by the execution of certain ISC instructions. Thus, these signals cannot change while an ISC instruction is not loaded.

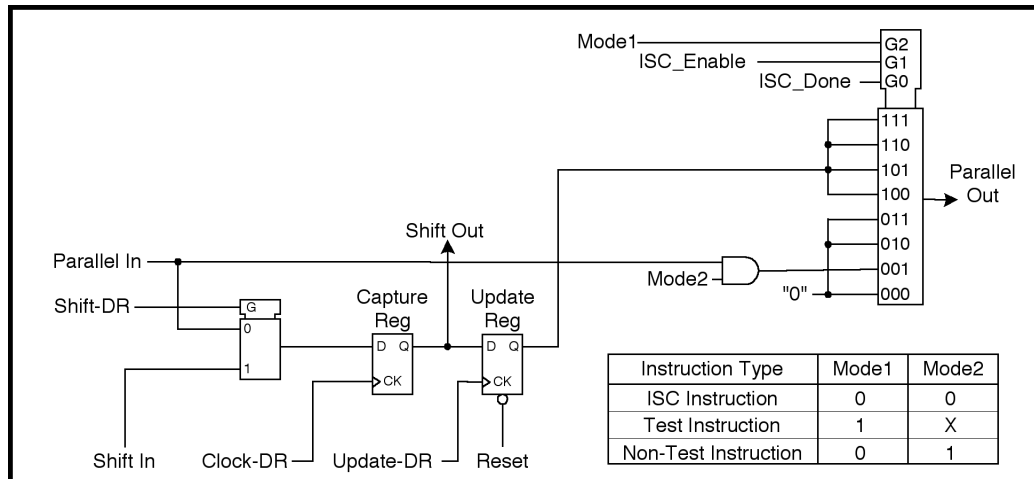


**Figure 12—A possible (unoptimized) control cell design that satisfies rule d) of 6.1.1 by disabling the system I/O pins**

The control cell design in Figure 13 is similar except that it uses the Boundary-Scan Register to control the output driver enable whenever an ISC (or test mode) instruction is loaded, as well as when a nontest instruction is loaded and the device in the ISC Accessed system modal state. This design has only one “Mode” signal which is derived per the table in Figure 13.

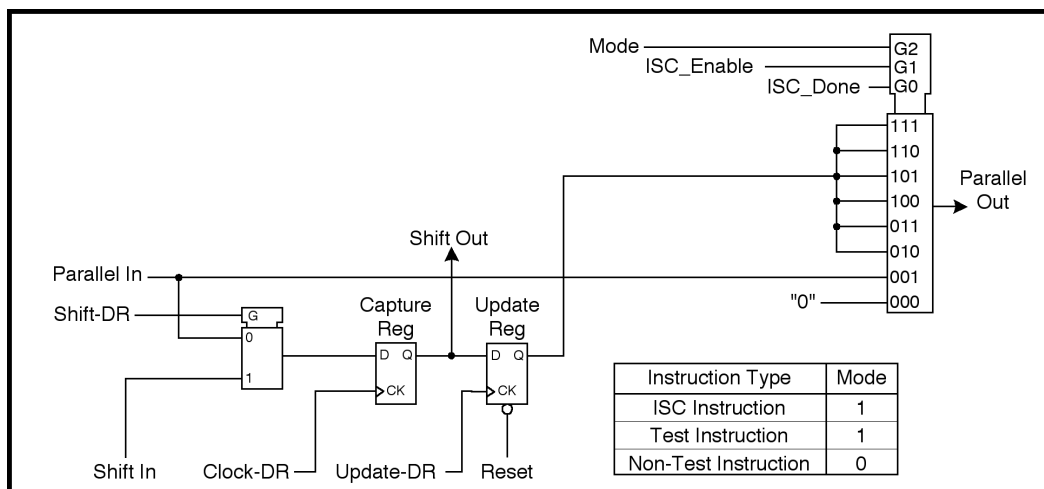


**Figure 13—A possible (unoptimized) control cell design that satisfies rule d) of 6.1.1 by using the Boundary-Scan Register to control the system I/O pin state**

Both of the designs in Figure 12 and Figure 13 have been shown in an unoptimized form for clarity. A designer would probably spend some effort on optimizing the speed of a single path (Parallel In to Parallel Out). All other paths are low speed and may be compromised accordingly.

NOTE—With respect to the BSDL description (see IEEE Std 1149.1-2001) of the control cell designs depicted in Figure 12 and Figure 13, while materially different in the output multiplexing structure from the cell design called BC\_1 in BSDL, these designs are still equivalent to BC\_1. This is because the BC\_1 structure describes the capture behavior of the cell and not the multiplexing structure on its output.

## 5. ISC architecture

### 5.1 Overview

#### 5.1.1 Classification of system pins

This standard recognizes that multiple functions may be placed upon a single integrated circuit die and that some of these may be programmable logic functions addressable by this standard and others may be conventional (nonprogrammable) electronic structures. In some cases, it may be important to differentiate System I/O pins based on recognition that they are connected to programmable functionality or fixed functionality.

The differentiation of System I/O pins is based on whether their I/O functions (input, output, bidirectional) are programmable by the ISC programming process. This excludes other types of pin configuration capability that could exist for pins that may determine parametric factors such as logic voltage levels, drive currents, pull-up or pull-down functions, and so on, that any pin may also possess.

##### 5.1.1.1 In-system configurable system pins

ISC system pins are those pins whose I/O functions (input, output, or bidirectional) are established by configuration information installed by the ISC programming process. Further, additional I/O pins chosen by the device designer may also be designated as ISC system pins. All ISC system pin behavior is determined by the system modal state as discussed in 4.4. Except where specifically noted in this document, the term “system pin” will be synonymous with “ISC system pin.”

##### 5.1.1.2 Fixed system pins

Fixed system pins are those whose I/O functions (input, output, or bidirectional) are not determined by configuration information installed by the ISC programming process, and have not been designated as ISC system pins by the device designer. Fixed system pin behavior is completely specified by IEEE Std 1149.1-2001. This, along with the fact that ISC instructions operate within system modal states (see 4.1) implies that fixed system pins may be fully operational while ISC programming activities are in progress. To reiterate, fixed system pins are in system mode when ISC instructions are being executed.

NOTE—Thus, the designation of fixed and ISC system pins may be predicated solely on their relationship with the programmable portion of a device, or may be arbitrarily designated as ISC system pins at the discretion of the device designer. This allows the designer to determine the I/O behavior of any pin on the device while it is being programmed. This has important consequences as the following example shows.

An example of a device with both fixed and ISC system pins is shown in Figure 14. In this device, a microcontroller and a programmable logic device have been integrated onto the same IC substrate. When first powered up (and with no programming present within the programmable section), the ISC system pins are disabled and the fixed pins are active, allowing the microcontroller to perform its function. Subsequently, through the TAP, the programmable portion of the device can be configured. While this is going on, the ISC system pins are either disabled or fixed by a pattern held in the Boundary-Scan Register. The fixed pins, at the same time, are free to perform system functions. Finally, when the ISC portion of the device is fully configured, the ISC system pins take on their system function when the ISC\_DISABLE instruction is

displaced. Notice that the controller that actually programmed the ISC portion of the device may have been the microcontroller on the same substrate, along with other external circuitry not shown.

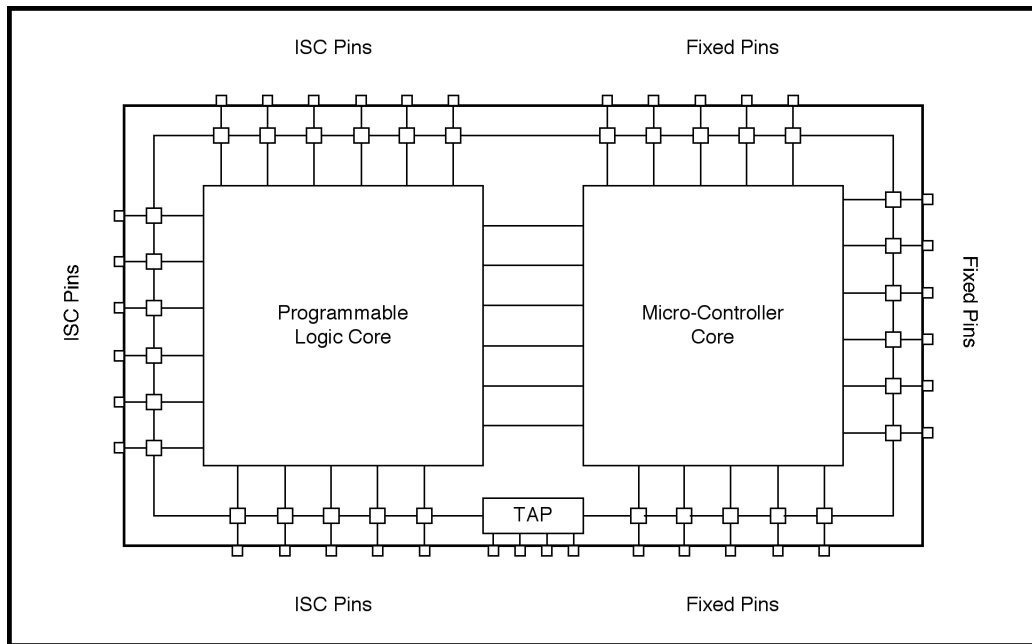


**Figure 14—A device containing both programmable and nonprogrammable logic, with associated pins**

#### NOTES

1—In Figure 14, there are connections between the programmable and nonprogrammable cores. These may or may not have Boundary-Scan Register cells at the discretion of the designer, but in either case, when the programmable core is unprogrammed, the nonprogrammable logic should perform appropriately.

2—The designer of a device containing less intelligent nonprogrammable logic could, if desired, design all I/O pins as ISC system pins. Then, all pins would be disabled or fixed at static states during the programming process.

#### 5.1.2 The test access port (TAP)

A device compliant with this standard will possess a Test Access Port containing four (optionally five) pins labeled Test Clock (TCK), Test Mode Select (TMS), Test Data Input (TDI), and Test Data Output (TDO). The optional fifth pin is Test Reset (TRST\*), which is active low. These pins are dedicated and may not be shared with other functions.

#### 5.1.3 IEEE Std 1149.1-2001 compliance enable pins

IEEE Std 1149.1-2001 contains, in section 4.8, “Subordination of This Standard Within a Higher Level Test Strategy,” the concept of compliance enable pins. These are one or more pins that must be statically conditioned with a set of logic states before the IEEE Std 1149.1-2001 facilities within the device are guaranteed to be compliant. Further, these pins must be maintained at those states for the duration of any IEEE Std 1149.1-2001-based activities. Such pins may exist on devices that comply with this standard and are thus required to be conditioned before and during all ISC activities.

NOTE—Signals associated with compliance enable pins are not included in the Boundary-Scan Register, per IEEE Std 1149.1-2001.

#### 5.1.4 Programmable voltage pins

For systems requiring overvoltage programming, allowance may be made for one or more static programming voltage pins used to supply energy for the physical programming process (see t) of 5.2). ISC devices shall interpret all instructions in the same manner, and the programming voltage pin(s) may not be used to signal additional data or control functions.

#### 5.1.5 The TAP controller and instruction unit

The TAP is used to drive a 16-state finite state machine called the TAP Controller, completely specified by IEEE Std 1149.1-2001. This state machine controls all aspects of the test logic and the ISC programming process.

Each compliant device contains an instruction register and instruction decoder. This unit, along with signals derived from the TAP, is used to sequence activities within the test and ISC logic.

#### 5.1.6 Mandatory instruction from IEEE Std 1149.1-2001

The mandatory instructions from IEEE Std 1149.1-2001 are required; SAMPLE, PRELOAD, BYPASS, and EXTEST. Programmable devices are also required to implement the IDCODE and USERCODE instructions. See 6.2 for additional mandatory instructions specified by this standard.

#### 5.1.7 Data registers

All devices adhering to this standard are required to implement the Bypass Register, the Boundary-Scan Register, and the Device Identification Register from IEEE Std 1149.1-2001. In addition, specific registers targeted by ISC instructions documented in this standard are also required. (See Clause 7.)

#### 5.1.8 System modal state control signals

As discussed in Clause 4, the system state of a programmable device conforming to this standard has four identified system modal states. Two control signals are required to identify which of these modal states the device currently resides within.

##### 5.1.8.1 ISC\_Done

This signal, which is nonvolatile in programmable devices that are nonvolatile, is used to indicate that a device has been programmed to the satisfaction of an external programming algorithm. In volatile devices, this signal is initially clear upon power-up.

##### 5.1.8.2 ISC\_Enabled

This volatile signal, initially clear when a programmable device is first powered, is set by the successful execution of the ISC\_ENABLE instruction (see j) of 5.2). This signal is cleared by the execution of the ISC\_DISABLE instruction or by passage of the TAP Controller to the *Test-Logic-Reset* TAP Controller state (see l) of 5.2). This signal must be set for other ISC instructions to operate correctly.

#### 5.1.9 Optional status indication

An optional status indication scheme is defined by this standard. If this status indication scheme is implemented in a device, it shall adhere to rules given in 5.2.

NOTE—A designer is free to implement a private status indication scheme if desired, but tools designed to work with this standard will not be able to determine status unless specifically modified to recognize this private functionality.

As described in 6.1, each ISC instruction causes events to occur that are initiated by entering the *Run-Test/Idle* TAP Controller state and remaining there for some minimum duration. It is possible that errors could be sensed by the circuitry responding to an instruction that would be useful information for an external programming algorithm. This algorithm could branch to appropriate exception handling routines when it observed errors reported by the status indication.

As all ISC instructions place some data register between TDI and TDO, the data register may be loaded with a status indication field in the *Capture-DR* TAP Controller state. This field is (at minimum) the least two significant bits of the register and indicates the status of the ISC instruction functionality that was last executed in the *Run-Test/Idle* TAP Controller state.

NOTE—If the status indication field is implemented, then all ISC data registers are required to be a minimum of two bits long. As seen in Clause 6 and Clause 7, some ISC data registers may be defaulted to the Bypass Register. This default is not compatible with the implementation of status reporting since the Bypass Register is only one bit long.

Figure 15 gives a structure for the registers that may at some time appear between TDI and TDO. Status reporting can be added by appending a two-bit (minimum) register onto the ISC data registers as shown in Figure 16.

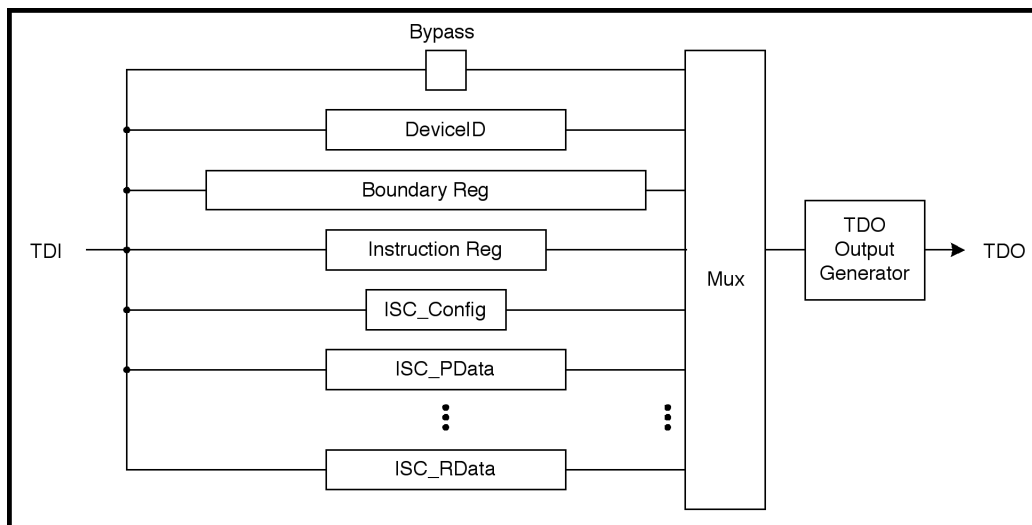


**Figure 15—Structure of registers placed between TDI and TDO when there is no status reporting**

Simply appending a status field to each register lengthens each by two (or more) bits. This simple model is consistent with the rules in IEEE Std 1149.1-2001 concerning the concatenation of registers: The concatenated register paths (between TDI and TDO) must have a single unique name. Thus, there is no “Status” register addressable by some instruction, only the normal ISC register set extended by two (or more) bits. In a more sophisticated implementation, a designer could reuse the least significant bits of most ISC data registers as status bits that are captured in the *Capture-DR* TAP Controller state. Note that registers, such as ISC\_RData (see 7.7), must be extended in length to prevent the status bits from overwriting data that is intended to be read back.

The status field indicates the execution status of the current instruction if the TAP Controller enters the *Capture-DR* TAP Controller state after the *Run-Test/Idle* TAP Controller state. The status of the previously executed ISC instruction can be observed when the TAP Controller enters the *Capture-DR* state after loading a new ISC instruction, but before entering the *Run-Test/Idle* TAP Controller state.

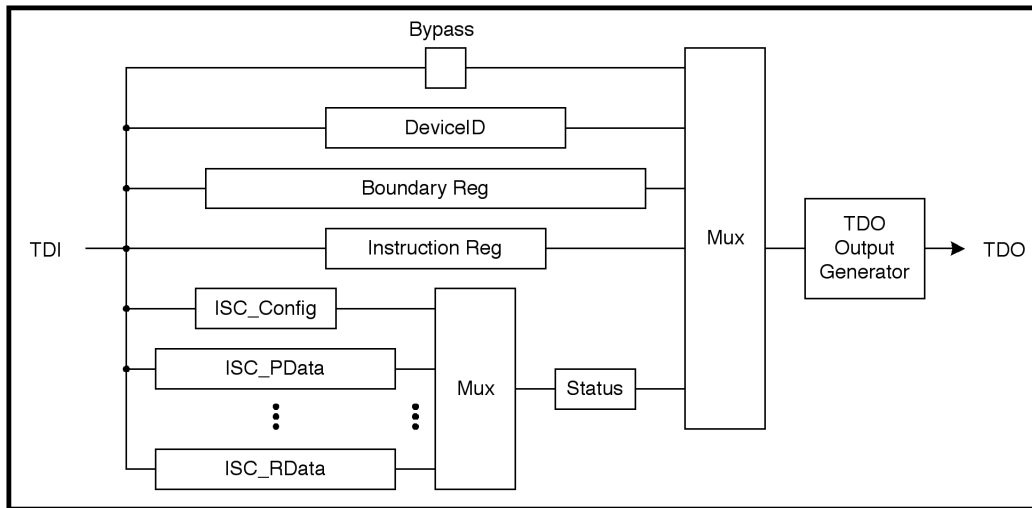


**Figure 16—Insertion of a status reporting register at the least significant two bits of the ISC data registers**

NOTE—The ISC\_NOOP instruction may be used as a status query instruction. However, it will return status of the last executed ISC instruction. If ISC\_NOOP is executed a second time, then the status shifted out will be that of the previous ISC\_NOOP execution.

Status bits may be manipulated at other times by instructions during their execution. For example, upon first entry into the *Run-Test/Idle* TAP Controller state, instructions may load the status register with an error code that indicates the required duration in the *Run-Test/Idle* TAP Controller state has not expired. After the duration has expired (but before leaving the *Run-Test/Idle* TAP Controller state), the status code may be updated to a non-error state. If the instruction is not allowed to complete its required duration (by leaving the *Run-Test/Idle* TAP Controller state early), the content of the status field subsequently captured will indicate an error.

The ability to observe instruction execution status allows external programming algorithms to respond to errors more closely to the time when they occur. This can allow a programming process to immediately abort or take some remedial action. If status indication is not implemented, an algorithm can still make basic checks, but a substantial amount of time and activity may pass between the occurrence of the error and when it is detected. For example, an ISC\_PROGRAM instruction may sense the failure of a data packet to be programmed and present this to an algorithm via the status field. If status is unavailable (or not checked), the algorithm may continue programming all subsequent locations (which may also be failing) and not see any error until a verification process is then executed. Notice that if verification has been made impossible by the activation of a read-protection security mechanism, then the error(s) could not be detected.

The status register may be constructed as shown in Figure 17. At a minimum, it contains a two-bit error code that is used to indicate the status of an executed instruction (see 5.2 and 6.1.1). Optionally, it may also contain an N-bit field containing additional information subcodes defined by the device designer to convey the cause of an error. Two examples are (1) the minimum duration for time spent in the *Run-Test/Idle* TAP Controller state was not met (see rule i) of 6.1.1), and (2) a security violation prevented the instruction from operating (e.g., read security has blocked ISC\_READ instruction execution; see 5.1.11.1 and 6.3.1).

#### 5.1.10 Optional operation success indication

An optional operation success indication scheme is defined by this standard. If the operation success indication scheme is implemented it shall conform to the rules set forth in 5.2. This indication scheme is only required for those devices that implement an adaptive programming algorithm.

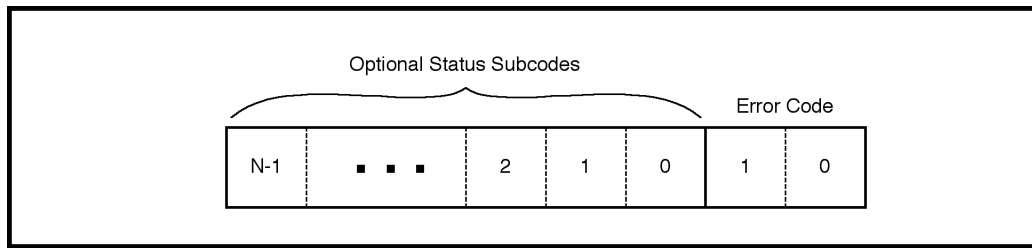


**Figure 17—Status register with optional subcode bits**

Devices that require the use of adaptive programming algorithms will require a mechanism to indicate the successful completion of certain operations. Some operations do not have a set minimum amount of time, and it will be required that the device notify the programming hardware when the operation is successful and can be terminated. The operation success indication mechanism allows an external programming algorithm to execute an operation for a given amount of time and at the end of that time shift out the operation success indication bit(s) (see 8.5.7.7). If the operation was completed successfully, the external programming algorithm can continue on to the next step in the programming process. If the operation was not successful and the maximum number of iterations has not been met, then the operation will be run again until either it is successful or until the maximum number of iterations is met. There may also be defined a minimum number of iterations the operation shall be executed before the operation success indication is evaluated (see 6.1.1). This will allow tools that implement dynamic concurrentization to maximize the amount of concurrency with devices that implement deterministic programming algorithms. The designer must make a tradeoff between increased concurrency with devices that implement deterministic programming algorithms or potentially improved programming times for dynamic concurrentization algorithms when specifying a minimum number of loops. If only a few loops are required, then a dynamic concurrentization algorithm will be able to exit out of the loop upon success sooner than if a greater number of loops are required. However, the fewer the number of required loops, the more difficult it is for a dynamic concurrentization algorithm to achieve concurrency between adaptive and deterministic devices.

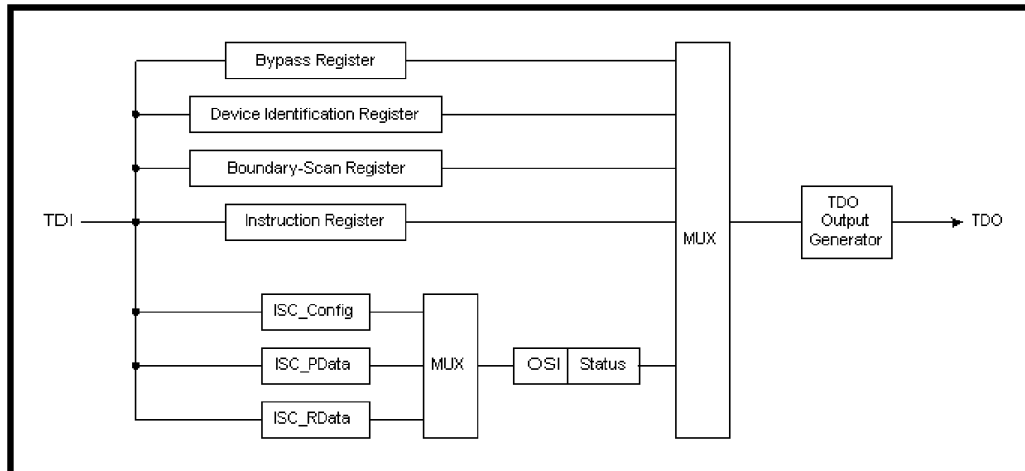
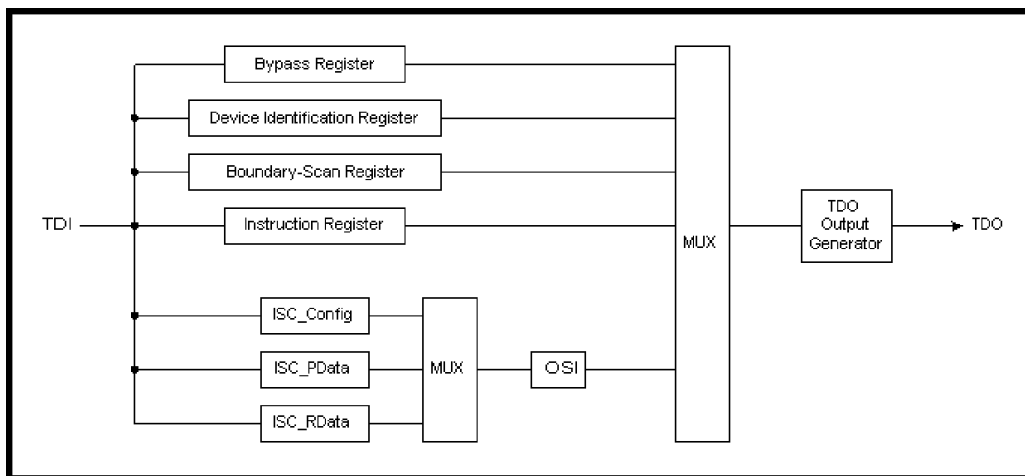
There are no restrictions placed on the number of bits used in the operation success indication nor on the values used to indicate success. The indication that an adaptive programming algorithm has been implemented is given by the use of the loop and OST keywords in the BSDL file (see 8.5.7). The “operation success” pattern will be defined as a part of the flow where it will be monitored.

#### 5.1.10.1 Implementing the operation success indication

There are multiple approaches to implementing the operation success indication. These approaches include, but are not limited to, implementing the operation success indication with the status register or implementing it with another register or set of registers. If the optional status register (see 5.1.9) has been implemented, and the operation success indication is included as a part of that register, then the operation success indication shall be included as a part of the subcode bits. If this approach is taken, it will affect the shift length of all ISC data registers, regardless of whether or not they require the operation success indication. An example of this is shown in Figure 18.

Alternatively, the operation success indication could be appended to the register(s) (such as ISC\_PData or ISC\_RData) that may be targeted by an instruction that would require the operation success indication. In this instance, the only shift operations that would be affected are those that require the operation success indication. This is demonstrated in Figure 19.

Alternatively, the operation success indication may be implemented as a stand alone register with an instruction used to access this register. If this approach is taken, the instruction shall follow the rules set forth in 6.1.1.

**Figure 18—Operation success indication with status****Figure 19—Operation success indication on select registers**

#### 5.1.10.2 Monitoring the operation success indication bits

Regardless of how the operation success indication is implemented, it will be evaluated each time an action or sequence of actions is executed. The flow provided in the BSDL file will provide two, and possibly three, pieces of information necessary to evaluate the success of the operation. It will provide the success requirements, a maximum number of times the success requirements should be evaluated before declaring an error, and possibly a minimum number of times the action(s) should be executed before evaluating the success of the operation. If an operation has been evaluated and success has not been indicated, then an error should not be declared unless the maximum number of iterations has been met. If the maximum number of iterations has been met and success has not been indicated, then it should be considered that the operation has failed. When an error is declared, it should be handled based on whether or not the `Exit_On_Error` tag has been associated with the flow used to describe the adaptive programming loop (see 8.5.7.2). If the tag is associated with the flow, then all programming operations should stop at the end of the data register or instruction register scan that generated the error.

In the case that the operation success indication has been implemented as subcodes in the status bits, it is noted that the functionality of the operation success indication bits is completely independent of the status bits as described in 5.1.9. The operation success indication notifies a programming tool when success has been achieved, whereas the status bits notify the programming tool when there has been an error that needs to be addressed such as an incorrect load of the instruction register. It should not be interpreted as an error if the operation success indication does not indicate that success has been achieved. The only time an error should be declared is when the status bits indicated that there has been an error or when the maximum loop count has been reached and the operation success indication does not report success.

### 5.1.11 Optional security provisions

An optional security mechanism is provided by this standard. Security features, when activated, prevent unwanted activities such as reading out programming data, changing programming data, or unauthorized erasure. This security mechanism is intended to work in conjunction with the optional `ISC_PROGRAM_SECURITY` instruction (see 6.4.3) or with `ISC_PROGRAM`. This security mechanism is very flexible and contains several options that allow it to implement all security features known to be in use as well as a richer set of capabilities.

The main features of the security mechanism defined herein are as follows:

- a) It allows up to three discrete forms of protection: read-protection, program-protection, and erasure-protection.
- b) It provides for the implementation of a “key” that will allow a user to “unlock” protection. A user can generate the key rather than it being hard-wired by the IC vendor.
- c) A “permanent” lock feature may be added that, when activated, will prevent any subsequent modification of security.
- d) The key may be encrypted by an on-chip function before it is stored such that knowledge of the original key is not sufficient to unlock security.
- e) The architecture of this security mechanism may be reduced in complexity to offer only the most basic read-protection at very low cost, emulating those capabilities used in past devices.

The security mechanism is based upon an Internal Protection Control register (the IPC register) that is nonvolatile in programmable devices that are nonvolatile. This register contains one, two, or three discrete bits used to control the form(s) of security, an additional (optional) N-bit key field and optional space for other functions (such as optional sector selection) as needed by the designer.

Figure 20 shows a general architecture of this security mechanism with all three forms of security implemented independently. The IPC register is divided into a three-bit field for function disable signals and an N-bit key field. This register is programmed by `ISC_PROGRAM_SECURITY` (see 6.4.3), which transfers the contents of the `ISC_PData` register (see 7.6) into the IPC register. This transfer occurs upon the successful completion of the instruction and may be conditioned by some additional factors. In Figure 20, the transfer is allowed when the key bits in the IPC register are all-zero or match those in `ISC_Pdata`, and the key bits in IPC are not all-ones. The “all-ones” key is used as a permanent lock; once the all-ones key is installed (and erasure has been disabled), there is no way to update the IPC register in the future. This permanently establishes the states of the three security signals. (Note that it is assumed that the initial state of the IPC for a blank device is all-zero.) The all-zero code may be thought of as “no security.” The all-zero code is the only code that can be directly overwritten. Once a key (containing both 1s and 0s) has been entered, it cannot be changed directly to another key. Note that `ISC_ERASE` (if not disabled) may be used to zero out the IPC register. Thus, erasure is the only means for changing a nonzero security key.

This architecture may be simplified by removing one or two of the security signals if they are not desired, by sharing any combination of security signals, by removing the all-ones permanent lock feature, or by removing the key field altogether. The architecture may implement stronger security by encoding the key while it is being transferred such that the original key will not match the stored key. This encoding may be

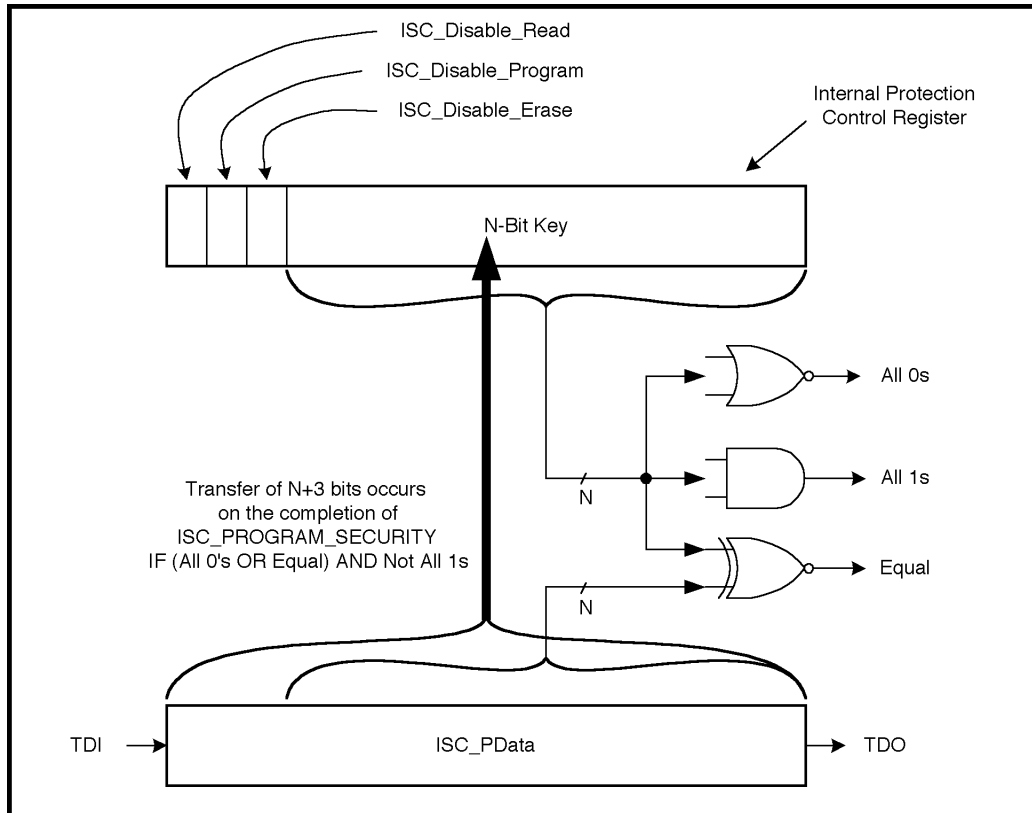


**Figure 20—Architecture of the optional security mechanism**

made entirely general by making it a function of other bits previously stored elsewhere in program memory. This range of implementation options allows any level of security, from none to arbitrarily strong keyed security to permanent locking. Further, all known security systems previously developed are subsets of this general capability.

#### 5.1.11.1 ISC\_Disable\_Read

This optional signal, which is nonvolatile in programmable devices that are nonvolatile, is used to indicate to the ISC\_READ instruction (see 6.3.1) that the device programming data is secured and should not be read out. This signal may be controlled by an independent bit of memory, or it may share a bit with one or both of the other disable signals. If the optional status indication capability (see 5.1.9) is implemented, then the fact that readback has been inhibited can be indicated. Read-protection is often used to inhibit malicious or inadvertent exposure of system design details.

#### 5.1.11.2 ISC\_Disable\_Program

This optional signal, which is nonvolatile in programmable devices that are nonvolatile, is used to indicate to the ISC\_PROGRAM instruction (see 6.2.3) that the device programming data is secured and should not be overwritten. This signal may be controlled by an independent bit of memory, or it may share a bit with one or both of the other disable signals. If the optional status indication capability (see 5.1.9) is implemented, then the fact that programming has been inhibited can be indicated. Program-protection may be used to inhibit malicious or inadvertent overwriting of existing program data.

### 5.1.11.3 ISC\_Disable\_Erase

This optional signal, which is nonvolatile in programmable devices that are nonvolatile, is used to indicate to the ISC\_ERASE instruction (see 6.3.2) that the device programming data is secured and should not be erased. This signal may be controlled by an independent bit of memory, or it may share a bit with one or both of the other disable signals. If the optional status indication capability (see 5.1.9) is implemented, then the fact that erasure has been inhibited can be indicated. Erase-protection may be used to inhibit malicious or inadvertent erasure of existing program data.

### 5.1.12 Programmable memory array access

A programmable logic device contains a memory array that holds the configuration data that determines the programmed function of the device. A basic model is shown in Figure 21. Note that it is composed of a number of rows of data, some of which may contain blank or unused bits. An address and data register are used to access the array. The data register must be wide enough to accommodate the longest row of data. When rows containing unused locations are accessed, corresponding locations in the data register are “don’t care” locations.

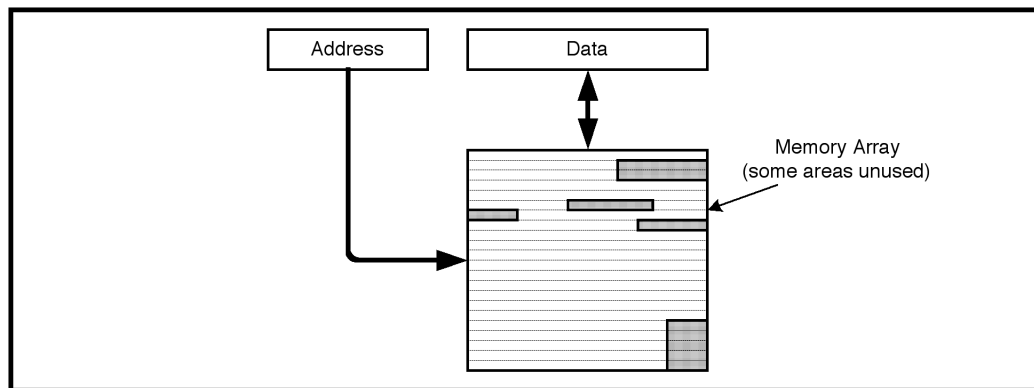


**Figure 21 — Basic model of a PLD configuration memory**

The rules in this standard are designed to precisely and unambiguously define an access methodology for In-System Configuration of programmable logic devices, but at the same time, offer designers a high degree of freedom in the design of the internal programmable structures. Thus, a device designer has many options for delivering address and data information to the internal memory array(s). Figure 22 shows a very basic access concept, where the address and data registers are linked together as one shiftable ISC register. Address and data can be shifted in or out of these two registers via the TDI/TDO shift path, thus setting up a program or read cycle to be executed by ISC instructions. The combination of address and data registers can be labeled the ISC\_PData and ISC\_RData registers as is allowed by IEEE Std 1149.1-2001. (See Clause 7 for a discussion of ISC data registers.) Whenever an operation on the memory array is done, a full scan of address and data is required.

A different memory access scheme is shown in Figure 23. Here, the address register can be independently accessed (see the ISC\_ADDRESS\_SHIFT instruction, 6.5.4) to initialize an address. The data register can be accessed directly by the ISC\_PROGRAM instruction (6.2.3) or ISC\_READ instruction (6.3.1). This scheme still requires that address and data both be shifted into the device to program a single location.

A more sophisticated approach simply adds address generation logic (see Figure 24). This logic may perform an auto-increment function, even skipping unused addresses in a fragmented address space. The triggering of this function may be tied to the programming and reading instructions to coordinate the programming of successive memory locations without the need for loading the address register for each.

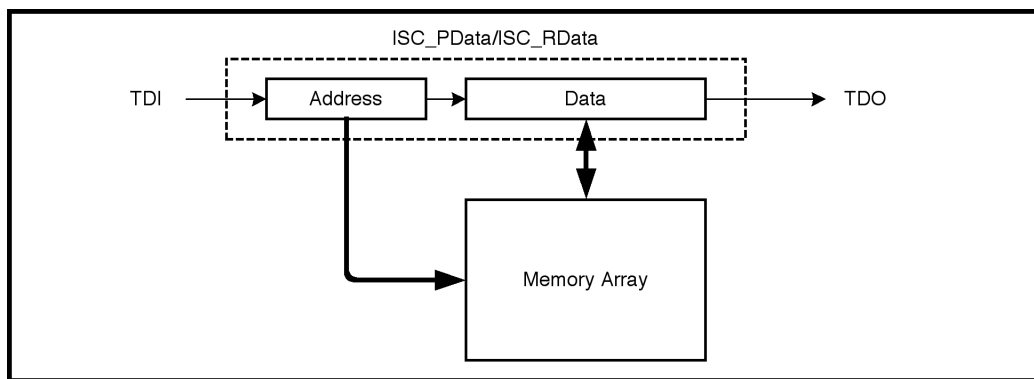


**Figure 22—Basic Programmable Memory Array access**

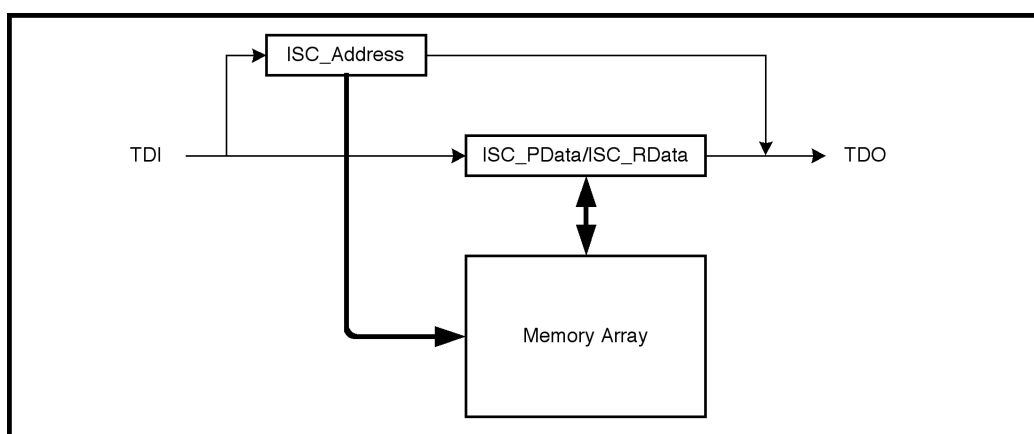


**Figure 23—A structure with individual access to address and data**

This may greatly reduce the time and amount of data shifting (and storage) needed for linearly accessing long memories.

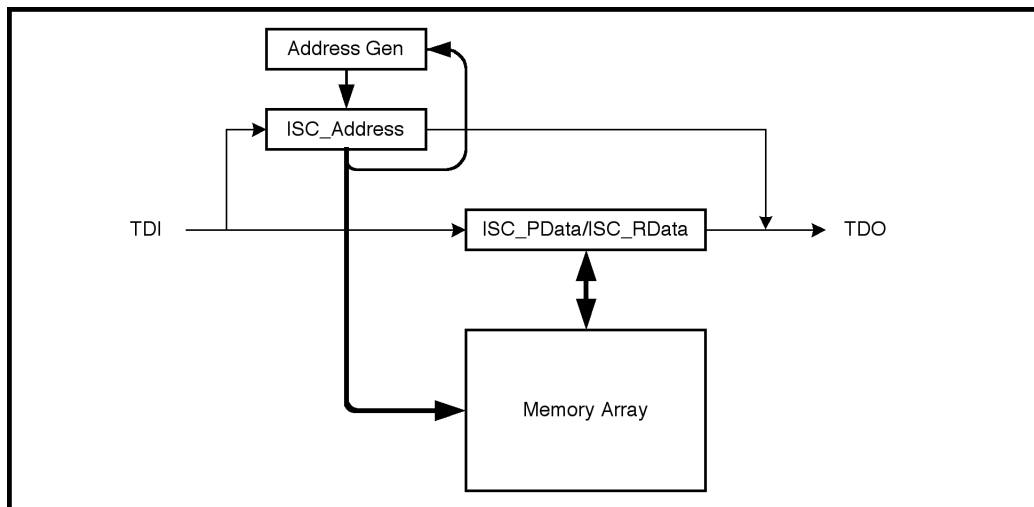


**Figure 24—Memory array access with independently controlled address and data registers**

It is also possible that the memory data for a programmable device is stored in more than one array, as shown in Figure 25. In this situation, there are two memory arrays with different lengths, widths, and even different burn times. By selecting an address scheme, such as having the high-order address line select between the two arrays, it is possible to use the same structure seen earlier for accessing memory. Memory can be accessed linearly as one block, although a change in duration in the *Run-Test/Idle* TAP Controller state may be needed when the array boundary is crossed.

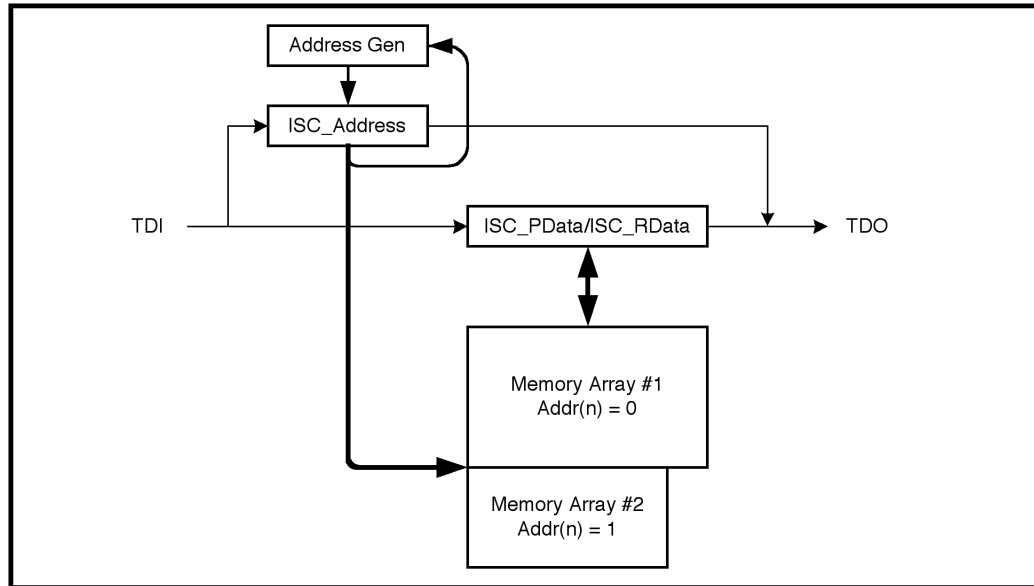


**Figure 25—Accessing two memory arrays with different dimensions**

A similar situation is shown in Figure 26 where a device contains two quite different arrays of memory, one being volatile and the other nonvolatile. The selection between them is controlled by a bit in the *ISC\_Config* register (which may also control other important parameters as well). In this case, the *ISC\_ENABLE* instruction effectively chooses which array will be accessed. Such a device on a board may have its nonvolatile content supplied once during board manufacture, and the volatile content refreshed while online in the field. Proper execution of *ISC\_ENABLE* will prevent programming of the volatile content from interfering with the nonvolatile content.

Figure 27 shows a scheme where the address register is not directly accessible from the outside, but is controlled by a short (perhaps one-bit) command field appended to the data register. One command could initialize the address register, and a second could increment it. While this again shortens the volume of data that must be scanned into the device, it has less addressing flexibility in that the start of a memory operation cannot be set to an arbitrary location.

The inventiveness of a device designer has few constraints imposed by this standard. At least five access approaches may be utilized, as follows:

- a) The address may be directly loaded into the *ISC\_Address* register (see 7.11) by previous use of the *ISC\_ADDRESS\_SHIFT* instruction (see 6.5.4). This is called explicit addressing.
- b) The address may be auto-initialized to the start of the array by the act of loading an *ISC* instruction destined to access memory. Subsequent accesses may automatically increment the addressing mechanism to the next valid entry. This is called implicit addressing.
- c) An address may be shifted into the *ISC\_RData* or *ISC\_PData* registers in the *Shift-DR* TAP Controller state where, upon passing the *Update-DR* TAP Controller state, it is then loaded into an addressing register. (For example, see permissions e) of 6.3.1.1 and g) of 6.3.1.1.)

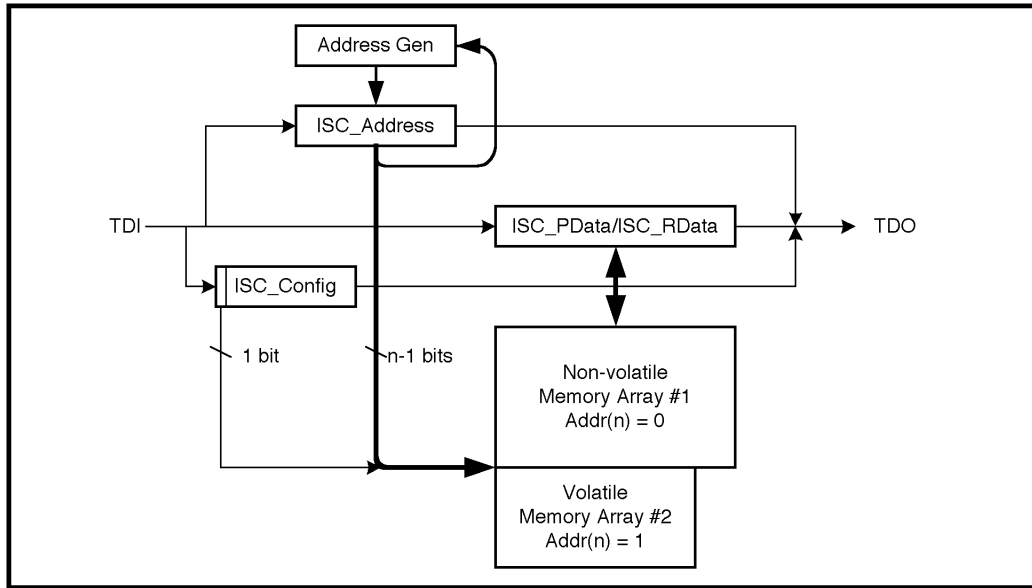


**Figure 26—A programmable device with two embedded technologies**

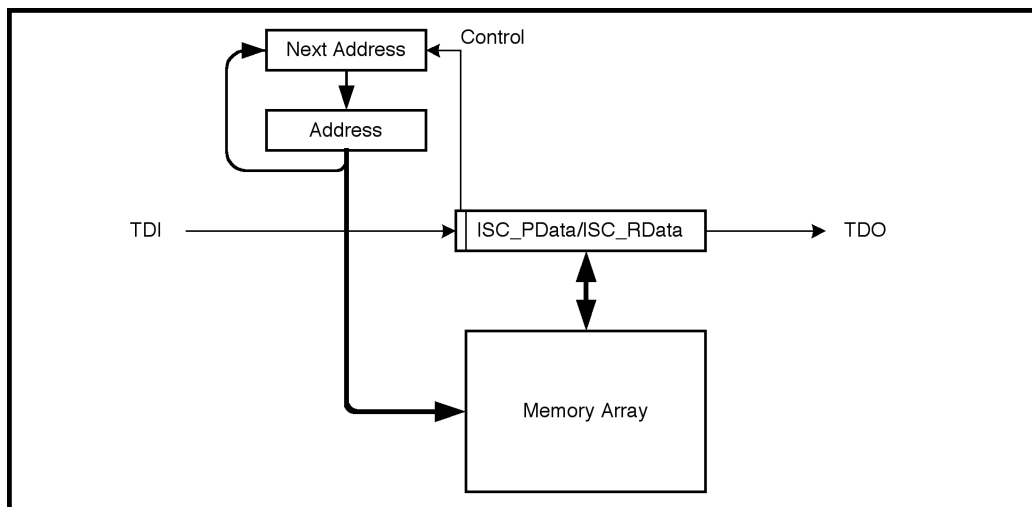


**Figure 27—An approach where there is no direct access to the addressing structure**

- d) Address control commands may be shifted into the ISC\_RData register in the *Shift-DR* TAP Controller state where, upon passing the *Update-DR* TAP Controller state, the commands control an addressing register. (For example, see permissions e) of 6.3.1.1 and g) of 6.3.1.1.) These commands may cause actions, such as initializing the address, incrementing it, decrementing it, shifting it, adding an offset, and so on.
- e) A combination of these methods may be implemented.

The actual technology of the addressing method is not specified by this standard, again allowing the designer freedom to use binary addressing, one-hot addressing, address encoding, and so on.

## 5.2 Specification

### Rules

- a) A component conforming to this standard shall first conform to all mandatory provisions of IEEE Std 1149.1-2001.
- b) The component shall conform to all rules within IEEE Std 1149.1-2001 that are specified for programmable devices.

NOTE—Rule b) assures that the normally optional IDCODE and USERCODE instructions are understood to be mandatory for devices conforming to this standard.

- c) The system mode (described by IEEE Std 1149.1-2001) of a component conforming to this standard shall be organized as four system modal states, Unprogrammed, Operational, ISC Accessed, and ISC Complete, as given in Clause 4.

NOTE—System modal states are determined by the ISC\_Done and ISC\_Enabled signals (along with a derived signal ISC\_Disable\_Completing) as shown in Figure 3. Further rules given below govern the derivation and timing of these signals.

- d) The component shall contain an ISC\_Done signal that is nonvolatile in nonvolatile programmable devices and that is cleared during power-up in volatile programmable devices.
- e) When the ISC\_Done signal is not protected by an enabled protection mechanism, the ISC\_Done signal shall be set by exactly one of three choices of implementation:
  - 1) By successful completion of the ISC\_DISABLE instruction when a specified data pattern is in place in the ISC\_Config register. (See 6.2.2.)
  - 2) By successful completion of the (optional) ISC\_PROGRAM\_DONE instruction. (See 6.4.1.)
  - 3) By successfully writing a specified bit or set of bits to the program memory using the ISC\_PROGRAM instruction. (See 6.2.3.)

### NOTES

1—It is intended that ISC\_Done should be set upon completion of programming and verifying the program.

2—Choice 1) above is the preferred implementation, with ISC\_Done being set as closely to the clearing of the ISC\_Enabled signal as possible. (See the discussion in 4.3.)

3—As noted in choice 3) above, the ISC\_Done signal may be a function of a set of bits in program memory. While the “AND” of these bits might be one obvious function that derives ISC\_Done, this standard does not mandate a function.

- f) When the ISC\_Done signal is not protected by an enabled protection mechanism, the ISC\_Done signal shall be cleared by one or more of three choices of action:
  - 1) By successful completion of the ISC\_ERASE instruction. (See 6.3.2.)
  - 2) By successful completion of the (optional) ISC\_ERASE\_DONE instruction. (See 6.4.2.)
  - 3) By successfully writing a specified bit or set of bits to the program memory using the ISC\_PROGRAM instruction. (See 6.2.3.)

NOTE—This last option 3) can only be supported in programmable devices that have symmetrical programming capability; that is, they can program both “1” and “0” bits in memory locations.

- g) Either the current state of the ISC\_Done signal (“1” indicates “set,” “0” indicates “cleared”) or the state that existed prior to entering the ISC Accessed system modal state shall be visible in bit two of the instruction register at the rising edge of TCK in the *Capture-IR* TAP Controller state.

## NOTES

1—Bits one and zero of the instruction register are already committed to capturing a “01” pattern by IEEE Std 1149.1-2001. (See IEEE Std 1149.1-2001, Table 7-1 in 7.2, Instruction Register Operation.)

2—This rule guarantees that the ISC\_Done signal may be interrogated upon loading any instruction defined by IEEE Std 1149.1-2001 or this standard. The programmed state of the device may then be determined without the need for entering the ISC Accessed system modal state.

3—Because of the new instructions mandated by this standard, the minimum length of the instruction register already exceeds two bits, and hence, this rule does not itself expand the length of this register.

- h) An updated state of the ISC\_Done signal (“1” indicates “set,” “0” indicates “cleared”) shall be visible in bit two of the instruction register no later than upon successful completion of ISC\_DISABLE. (See 6.2.2.)

NOTE—This rule defines when a recently updated state of the ISC\_Done signal must be observable in the instruction register when shifting out captured instruction register bits. In some implementations, there may be unusual voltages present at the time ISC\_Done signal is updated that make the immediate transfer of the ISC\_Done signal value to the instruction register impractical. Toolmakers must not rely on seeing an updated ISC\_Done signal state before the completion of ISC\_DISABLE.

- i) The component shall contain an ISC\_Enabled signal that is cleared upon device power-up.
- j) The ISC\_Enabled signal shall only be set by the successful completion of the ISC\_ENABLE instruction. (See 6.2.1.)
- k) The ISC\_Enabled signal shall be cleared by either of two events:
  - 1) By successful completion of the ISC\_DISABLE instruction. (See 6.2.2.)
  - 2) By the TAP Controller moving to the *Test-Logic-Reset* TAP Controller state.

## NOTES

1—The TAP Controller may enter the *Test-Logic-Reset* TAP Controller state asynchronously in response to the assertion of the TRST\* TAP signal.

2—The ISC\_Enabled signal may also be cleared by an event specified in permission (s) below.

- l) If, while a device is in the ISC Accessed system modal state an IEEE Std 1149.1-2001 pin-permission instruction (EXTEST, INTEST, CLAMP, HIGHZ, or RUNBIST) is loaded, the device shall enter test mode and all rules required for these instructions by IEEE Std 1149.1-2001 take effect immediately.

NOTE—Data residing in the boundary-scan register may be utilized by these instructions.

- m) If a device was in the ISC Accessed system modal state and one or more IEEE Std 1149.1-2001 pin-permission instructions (EXTEST, INTEST, CLAMP, HIGHZ, or RUNBIST) have subsequently been executed, loading an ISC instruction shall (with the exception given in permission s) below) return the device to the ISC Accessed system modal state.

NOTE—Data residing in the boundary-scan register utilized by the IEEE Std 1149.1-2001 instruction(s) may be utilized by an ISC instruction that is loaded per rule d) of 6.1.1.

- n) If recommendation v) below is implemented, the least significant two (or more) bits of all ISC data registers shall capture a code that indicates the status of the most recently loaded ISC instruction that executed in the *Run-Test/Idle* TAP Controller state. These bits include:
  - 1) A mandatory two-bit error code field in bit positions 0 and 1.
  - 2) An optional N-bit status subcode field (starting at bit 2).

## NOTES

1—The status error codes for instructions are assigned in Clause 6. ISC data registers are described in Clause 7. Subcodes are discussed in 5.1.9.

2—If no ISC instruction has been executed in the *Run-Test/Idle* TAP Controller state since device power-up, the status bits are not defined. It is expected that an external algorithm will know when status bits are defined.

3—If the execution of non-ISC instructions are intermixed with ISC instructions, this will have no effect on the status field observed the next time an ISC instruction is executed.

4—If the currently loaded ISC instruction has not yet executed in the *Run-Test/Idle* TAP Controller state, then the status captured in the *Capture-DR* TAP Controller state will be that of the most recently loaded ISC instruction that did execute in the *Run-Test/Idle* TAP Controller state.

- o) If recommendation v) is implemented, the status field of the targeted ISC data register shall be loaded no later than the rising edge of TCK in the *Capture-DR* TAP Controller state.

NOTE—This rule explains when the status field of the data register is to be captured in preparation for shifting. The status bits may be manipulated at other times by instructions while they are executing. For example, instructions may load the status field with an error code that indicates the required duration in the *Run-Test/Idle* TAP Controller state has not expired upon first entering this state. After the duration has expired (but before leaving the *Run-Test/Idle* TAP Controller state), the status code may be updated to a nonerror state. If the instruction is not allowed to complete its required duration (by leaving the *Run-Test/Idle* TAP Controller state early), the content of the status field subsequently captured will indicate an error.

- p) No instruction specified by IEEE Std 1149.1-2001 shall have any effect upon ISC data registers specified by this standard.

NOTE—In particular, the address, data, status, and protection mechanisms implemented in accordance with the rules of this standard shall see no side effects as a result of executing any instruction defined by IEEE Std 1149.1-2001.

- q) If the optional security capability described in this standard is implemented, it shall generate one or more of the three signals: *ISC\_Disable\_Read*, *ISC\_Disable\_Program*, and *ISC\_Disable\_Erase*.

## NOTES

1—The optional security capability is described in 5.1.11.

2—This rule does not prohibit the implementation of a proprietary security capability, but tools created to support this standard may not be able to access and control this capability without special modification.

- r) The operation success indication shall be implemented on devices that require an adaptive programming algorithm.

## NOTES

1—This rule does not specify a specific implementation for the indication scheme, only that the operation success indication exists and is accessible to an external programming algorithm. The operation success indication may be included as a part of the optional status indication, but that is not a requirement. If the operation success indication bits are included as a part of the optional status indication field, then their operation will be completely independent of the status bits.

2—The operation success indication facility is described in 5.1.10.