IEEE Std 1500™-2005

IEEE Standards

1500™

IEEE Standard Testability Method for Embedded Core-based Integrated Circuits

IEEE Computer Society

Sponsored by the
Test Technology Technical Council

IEEE

3 Park Avenue, New York, NY10016-5997, USA

29 August 2005

Print: SH95335
PDF: SS95335

Recognized as an
American National Standard (ANSI)

IEEE Std 1500™-2005

IEEE Standard Testability Method for Embedded Core-based Integrated Circuits

Sponsor

Test Technology Technical Council
of the
IEEE Computer Society

Approved 30 June 2005

American National Standards Institute

Approved 20 March 2005

IEEE-SA Standards Board

Abstract: This standard defines a mechanism for the test of core designs within a system on chip (SoC). This mechanism constitutes a hardware architecture and leverages the core test language (CTL) to facilitate communication between core designers and core integrators.
Keywords: core test, embedded core test, IP test, test reuse

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

Print: ISBN 0-7381-4693-5 SH95335
PDF: ISBN 0-7381-4694-3 SS95335

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. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE.

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

Piscataway, NJ 08854

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.

Introduction

This introduction is not part of IEEE Std 1500-2005, IEEE Standard Testability Method for Embedded Core-based Integrated Circuits.

IEEE Std 1500 is a scalable standard architecture for enabling test reuse and integration for embedded cores and associated circuitry. It foregoes addressing analog circuits and focuses on facilitating efficient test of digital aspects of systems on chip (SoCs). IEEE Std 1500 has serial and parallel test access mechanisms (TAMs) and a rich set of instructions suitable for testing cores, SoC interconnect, and circuitry. In addition, IEEE Std 1500 defines features that enable core isolation and protection. IEEE Std 1500 will reduce test cost through improved automation, promote good design-for-test (DFT) technique, and improve test quality through improved access.

Core test language (CTL) is the official mechanism for describing IEEE 1500 wrappers and test data associated with cores. CTL is defined in IEEE P1450.6 ™[a] and was originally begun as part of the development of IEEE Std 1500.

IEEE Std 1500 was broadly influenced by the past work of the IEEE Std 1149.1™ Working Group and has several members from that group. IEEE Std 1149.1 and IEEE Std 1500 have similar goals at different levels of integration. IEEE Std 1149.1 describes a wrapper architecture and access mechanism designed for the purpose of testing components of a board whereas IEEE Std 1500 has a similar structure targeted towards testing cores in an SoC.

IEEE Std 1500 has been a continuous effort for its participants due to the goal of resolving the needs of reconciling and accommodating disparate test strategies and motives. The greatest effort has been put into supporting as many requirements as possible while still producing a cohesive and consistent standard.

Objective of the IEEE 1500 effort

The Embedded Core Test Working Group was approved in 1997 with the charter to develop a standard test method for integrated circuits (ICs) containing embedded cores, i.e., reusable megacells. That method would be independent of the underlying functionality of the IC or its individual embedded cores. The method will create the necessary testability requirements for detection and diagnosis of such ICs, while allowing for ease of interoperability of cores originated from distinct sources. This method will be usable for all classes of digital cores including hierarchical ones (subclause 15.1 discusses hierarchical core-wrapper configurations).

In order to satisfy that charter, the Embedded Core Test Working Group was organized into several task forces:

Core Test Language

Scalable Architecture

Compliance Definition/Information Model

Terminology/Glossary

Edition

Mergeable Cores Test

Benchmarking

Industry & Media Relations

[a]Information on references can be found in Clause 2.

Achievements

Since its inception, the Embedded Core Test Working Group has produced eight drafts of the preliminary standard, considering all aspects of core-based test. Due concern has been given to ensuring that a broad spectrum of users will be satisfied through flexibility. Both serial and parallel TAMs were developed. A definition for core wrappers was created, and a set of instructions developed. The CTL was begun, and an information model and compliance definition using that language were developed.

Notice to users

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: http://standards.ieee.org/reading/ieee/updates/errata/index.html. Users are encouraged to check this URL for errata periodically.

Interpretations

Current interpretations can be accessed at the following URL: http://standards.ieee.org/reading/ieee/interp/index.html.

Patents

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 or patent applications for which a license may be required to implement an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Participants

At the time this standard was prepared, the Embedded Core Test Working Group had the following membership:

Yervant Zorian, *Chair*
Karim Arabi, *Vice Chair, Compliance Definition*
Francisco da Silva, *Vice Chair, Edition*
Lee Whetsel, V*ice Chair, Scalable Architecture*
Sudipta Bhawmik, *Secretary*

Luis Basto
Dwayne Burek
Vivek Chickermane
Wu-Tung Cheng
Mike Collins
Bulent Dervisoglu
Jason Doege
Grady Giles
Alan Hales
Rohit Kapur
Brion Keller
Erik Jan Marinissen
Mike Mateja
Teresa McLaurin
Fidel Muradali
Mike Ricchetti
Paul Soong
Jon Udell
Tom Waayers

Past members include the following:

Saman Adham
James Beausang
Debashis Bhattacharya
Tapan Chakraborty
Chen-Huan Chiang
CJ Clark
Adam Cron
Al Crouch
Scott Davidson
Ted Eaton
Joan Figueras
Pradipta Ghosh
Sanjay Gupta
Andy Halliday
Peter Harrod
Douglas Kay
Bernd Koenemann
Maurice Lousberg
Samy Makar
Meryem Marzouki
Jim Monzel
Nilanjan Mukherjee
Date Noorlag
Franc Novak
Adam Osseiran
Chris Papachristou
Srinivas Patil
Kim Petersen
Paolo Prinetto
Janusz Rajski
Rochit Rajsuman
Paul Reuter
Gordon Robinson
Todd Rockoff
Eddie Rodriguez
Samvel Shoukourian
Rajagopalan Srinivasan
Tony Taylor
Ted Vaida
Prab Varma
Ken Wagner
Michael Wahl
Ron Walther
Tom W. Williams
Cheng-Wen Wu
Shianling Wu
Hans-Joachim Wunderlich
Sitaram Yadavalli
Avetik Yessayan
Greg Young
Alex Zamfirescu
Kamran Zarrineh

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

Ken-ichi Anzou
Luis Basto
Roger Bennetts
Sudipta Bhawmik
Dave Bonnett
Keith Chow
Antonio M. Cicu
Luis Cordova
Francisco da Silva
Dave Dowding
William Eklow
Grady Giles
Sandeep Goel
Alan Hales
Peter Harrod
Mitsuaki Ishikawa
Neil Jacobson
Rohit Kapur
Jake Karrfalt
Brion Keller
Adam Ley
Maurice Lousberg
Yuhai Ma
Ryan Madron
Erik Jan Marinissen
Denis Martin
Gregory Maston
Yinghua Min
Mehdi Mohtashemi
James Monzel
Richard Morren
Benoit Nadeau-Dostie
Charles Ngethe
Franc Novak
Steven Oostdijk
Adam Osseiran
Klaus Rapf
Paul Reuter
Mike Ricchetti
Gordon Robinson
Srinivasa Vemuru
Tom Waayers
Gregg Wilder
T. W. Williams
Li Zhang
Yervant Zorian

When the IEEE-SA Standards Board approved this standard on 20 March 2005, it had the following membership:

Steve M. Mills, *Chair*
Richard H. Hulett, *Vice Chair*
Judith Gorman, *Secretary*

Mark D. Bowman
Dennis B. Brophy
Joseph Bruder
Richard Cox
Bob Davis
Julian Forster*
Joanna N. Guenin
Mark S. Halpin
Raymond Hapeman
William B. Hopf
Lowell G. Johnson
Herman Koch
Joseph L. Koepfinger*
David J. Law
Daleep C. Mohla
Paul Nikolich
T. W. Olsen
Glenn Parsons
Ronald C. Petersen
Gary S. Robinson
Frank Stone
Malcolm V. Thaden
Richard L. Townsend
Joe D. Watson
Howard L. Wolfman

*Member Emeritus

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

Satish K. Aggarwal, *NRC Representative*
Richard DeBlasio, *DOE Representative*
Alan H. Cookson, *NIST Representative*

Michelle D. Turner
IEEE Standards Project Editor

Contents

IEEE Standard Testability Method for Embedded Core-based Integrated Circuits

1. Overview

IEEE Std 1500™ defines a scalable architecture for independent, modular test development and test application for embedded design blocks and also enables test of the external logic surrounding these cores. Modular testing is typically a requirement for embedded nonlogic blocks, such as memories, and for embedded predesigned nonmergeable intellectual property (IP) cores. In addition, the IEEE 1500 architecture can also be used to partition large design blocks into smaller blocks of more manageable size and to facilitate test reuse for blocks that are reused from one system-on-chip (SoC) design to the next.

The IEEE 1500 architecture comprises hardware requirements, through the definition of a standardized core wrapper, and uses a test-specific language to communicate information between core providers and core users. This language is the IEEE P1450.6™ [1] core test language (CTL). Although IEEE Std 1500 limited itself to test aspects internal to nonmergeable cores, careful consideration was given to the interoperability of such cores, resulting in plug-and-play (PnP) requirement definitions. SoC-specific issues such as those related to the design of test access mechanisms (TAMs) are excluded from this standard and assumed to be addressed by the SoC designer.

IEEE Std 1500 specifically focuses on defining test requirements for unidirectional non-tristate digital terminals, as these represent a minimum and mandatory set of requirements upon which the more complex bidirectional terminals are based. It is, therefore, implied that support for bidirectional or tristatable terminals is provided only to the extent that the individual unidirectional terminals, i.e., the bidirectional or tristatable terminal, are available for IEEE 1500 wrapper insertion. In addition, the hardware architecture defined in this standard anticipates a synchronous wrapper design methodology.

While IEEE Std 1500 does not discuss chip-level design, the architecture defined in this standard does not prevent interfacing with IEEE 1149.1™-based standards. An example of this interface is provided in Annex C for the reader's benefit.

All rules described in this standard apply to the case where the IEEE 1500 wrapper is enabled (the wrapper logic actively participates in the test of the core) except rules specific to the Wrapper Disabled state of the IEEE 1500 wrapper. In Wrapper Disabled state, the IEEE 1500 wrapper is disabled, allowing functional

[1] Information on references can be found in Clause 2.

operation of the wrapped core. IEEE P1450.6 constructs were added to this standard, where appropriate, to further guide the reader. It is anticipated that the reader will refer to these CTL constructs documented in IEEE P1450.6. Additional discussion that complements the body of this standard are presented in annex clauses:

— Annex A contains the legend for IEEE 1500 wrapper cells.

— Annex B shows examples of IEEE 1500 wrapper cells.

— Annex C presents similarities between IEEE Std 1500 and IEEE Std 1149.1 and discusses an example interface between IEEE Std 1500 and IEEE Std 1149.1.

1.1 Scope

IEEE Std 1500 has developed a standard design-for-testability method for integrated circuits (ICs) containing embedded nonmergeable cores. This method is independent of the underlying functionality of the IC or its individual embedded cores. The method creates the necessary requirements for the test of such ICs, while allowing for ease of interoperability of cores that may have originated from different sources.

1.2 Purpose

The aim of IEEE Std 1500 is to provide a consistent scalable solution to the test reuse challenges specific to the reuse of nonmergeable cores, while preserving the IP aspects that are often associated with these cores. This objective is achieved through provision of a core-centric methodology that enables successful integration of cores into SoCs.

IEEE Std 1500 provides a bridge between core providers and core users and also facilitates the automation of test data transfer and reuse between these two entities via the use of the IEEE P1450.6 CTL. This automation relies on information requirements (the information model) placed on the core provider to ensure that the core can be successfully integrated by the core user. The result is shorter time to market for core providers and core users.

The data transfer and reuse from the core provider to the core user are based on the premise that the core test data are left unchanged, while the test protocol is adapted from the IEEE 1500 hardware interface to the SoC.

2. Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

IEEE Std 1149.1, IEEE Standard Test Access Port and Boundary-Scan Architecture.[2]

IEEE P1450.6, Draft Standard for Standard Test Interface Language (STIL) for Digital Test Vector Data—Core Test Language (CTL), http://grouper.ieee.org/groups/ctl/.

[2]IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, Piscataway, NJ 08854, USA (http://standards.ieee.org/).

3. Definitions, acronyms, and abbreviations

This clause lists some definitions of terms that have been used throughout this standard. In addition, a list of acronyms and abbreviations is also provided. The criteria for differentiating various terms are based on the following rules:

a) General terminology in the scope of electrical engineering and/or test technology, which do not influence the definition of this standard itself.
   1) These terms are used in this standard without further explanation. It is assumed that the readership has sufficient background knowledge to understand these terms.
   2) Some of these terms may already be defined in *The Authoritative Dictionary of IEEE Standards Terms*. Therefore, someone looking for a particular definition could always consult *The Authoritative Dictionary* to verify the meaning of the term.

b) Terms that are specific to this standard
   1) These terms are defined in this clause.
   2) General definitions are valid throughout this standard.
   3) Local definitions are relevant only to a specific clause of this standard.

Effort has been made to achieve consistent usage of all terms (e.g., wrapper, TAM).

3.1 Definitions

3.1.1 access mechanisms: Mechanisms by which signals may be propagated to and from a core, from either embedded circuitry or from the primary inputs and outputs of the system chip. There are two types of access mechanisms:

(a) Functional access: The mechanism for moving stimuli to and observing responses from a core or user-defined logic (UDL) during functional operation or normal mode.

b) Test access: The mechanism for moving stimuli to and observing responses from a core or UDL during nonfunctional operation or test mode.

3.1.2 auxiliary clock (AUXCK): A functional clock that may be used in conjunction with wrapper clock (WRCK) during core test for capturing, shifting, updating, and optionally transferring test data in a wrapper.

3.1.3 bypass: As applied to core wrappers, an abbreviated sequential path connecting a wrapper serial input (WSI) to a wrapper serial output (WSO).

3.1.4 captureWR: A wrapper terminal used to enable and control a Capture operation in the selected IEEE 1500 wrapper register (WR).

3.1.5 cell functional input (CFI): For input wrapper cells, the cell's input, which is connected to a wrapper functional input (WFI); for output wrapper cells, the cell's input, which is connected to a core output.

NOTE—See CFI pin in Figure 16.

3.1.6 cell functional output (CFO): For input wrapper cells, the cell's output, which is connected to a core input; for output wrapper cells, the cell's output, which is connected to a wrapper functional output (WFO).

NOTE—See CFO pin in Figure 16.

3.1.7 cell test input (CTI): A wrapper boundary register (WBR) cell's test data input.

3.1.8 cell test output (CTO): A wrapper boundary register (WBR) cell's test data output.

3.1.9 control: The process of applying test pattern stimuli.

3.1.10 core: Predesigned circuit block that can be tested as an individual unit.

3.1.11 core data register (CDR): Optional data register that belongs to a core being wrapped.

3.1.12 core input: An input terminal of an unwrapped core.

3.1.13 core integrator: An entity that incorporates one or more cores into an system on chip (SoC).

3.1.14 core isolation: A test mode feature preventing core-to-core or core-to-UDL (i.e., user-defined logic) interaction.

3.1.15 core output: An output terminal of an unwrapped core.

3.1.16 core provider: An entity that designs cores that can be reused in other designs.

3.1.17 core test: A test methodology that is applied to an embedded core.

3.1.18 core test language (CTL): A standard language for core suppliers to provide test data that can be used to test a core once it is integrated into a system on chip (SoC). The language presents a format to describe test and support data so that the core can be effectively integrated, reused and tested.

NOTE—See IEEE P1450.6 reference documentation.

3.1.19 dedicated shift path: A shift path comprising storage elements that do not participate in functional operation.

3.1.20 dedicated wrapper (cell): A wrapper style that does not share hardware with core functionality. This style allows certain test operations to occur concurrently and transparently during functional operation. This definition could apply to individual cells.

3.1.21 external safe state: A configuration of safe state in which the outputs of a core are in a state that prevents them from interfering with a block of logic outside the core. *See also* **internal safe state**; **safe state**.

3.1.22 firm core: A predesigned block of functional logic such as a macro, megacell, or memory that has a process technology-dependent netlist representation and may be amenable to some modification.

3.1.23 hard core: A predesigned block of functional logic such as a macro, megacell, or memory that has a physical implementation that cannot be modified.

3.1.24 hybrid instruction: A wrapper instruction that has mixed use of wrapper serial port (WSP) and wrapper parallel port (WPP) terminals.

3.1.25 input cell: A wrapper boundary register (WBR) cell that is provided on a core input.

3.1.26 internal safe state: A configuration of safe state whereby a core is protected from the impact of a test outside the core. *See also* **external safe state**; **safe state**.

3.1.27 interoperability: *See* **plug-and-play (PnP)**.

3.1.28 inward facing (IF) mode: The test mode where core inputs are controlled by the wrapper boundary register (WBR) and core outputs are observed by the WBR.

3.1.29 mergeable core: With respect to testability, a core that can be integrated with other cores and user-defined logic (UDL) into a system on chip (SoC) so that a uniform design-for-test (DFT) methodology can be applied across the entire system. A typical mergeable core is provided using a register transfer level (RTL) or gate-level description.

3.1.30 merged core: With respect to testability, a core that is integrated with other cores and user-defined logic (UDL) into a system on chip (SoC) so that a uniform design-for-test (DFT) methodology could be applied across the entire system.

3.1.31 nonmergeable core: With respect to testability, a core that cannot be integrated to apply a uniform design-for-test (DFT) methodology to the entire system on chip (SoC). A typical nonmergeable core comes with a physical design implementation that does not accommodate modification of the test methodology. A nonmegeable core may be represented as a block-box design, making standard automatic test pattern generation (ATPG) impossible on such a core.

3.1.32 nonmerged core: With respect to testability, a core that has not been integrated with other cores and user-defined logic (UDL) into a system on chip (SoC) so that a uniform design-for-test (DFT) methodology could be applied across the entire system.

3.1.33 normal mode: The mode in which the wrapper boundary register (WBR) does not interfere with the functional operation of a wrapped core.

3.1.34 observation: The process of monitoring pattern response.

3.1.35 output cell: A wrapper boundary register (WBR) cell that is provided for a core output.

3.1.36 outward facing (OF) mode: The test mode where wrapper functional outputs (WFOs) are controlled by the wrapper boundary register (WBR) and wrapper functional inputs (WFIs) are observed by the WBR.

3.1.37 parallel instruction: A wrapper instruction that uses wrapper parallel port (WPP) terminals and also configures the wrapper bypass register (WBR) between wrapper serial input (WSI) and wrapper serial output (WSO).

3.1.38 pattern set: A collection of test vectors intended for manufacturing test. In the context of core test language (CTL), a pattern set is a collection of pattern constructs and their associated macros and procedures brought together with PatternBurst and PatternExecs.

3.1.39 plug-and-play (PnP): A minimum level of interoperability between various core wrappers in a system on chip (SoC).

3.1.40 safe data: Data that satisfy safe state configuration requirements. These data are user-defined.

3.1.41 safe state: A property whereby a test of one block of logic is prevented from interfering with or damaging another block of logic. *See also* **external safe state**; **internal safe state**.

3.1.42 selectWIR: The IEEE 1500 wrapper terminal that determines the selection of a wrapper register (WR). A value of 1 represents selection of the wrapper instruction register (WIR), and a value of 0 represents selection of a wrapper data register (WDR).

3.1.43 serial instruction: A wrapper instruction that exclusively uses wrapper serial port (WSP) terminals.

3.1.44 serial scan chain: The scan chain configuration inside a wrapped core where an internal scan chain is concatenated with the wrapper boundary register (WBR) chain for the purpose of running the WS_INTEST_SCAN instruction.

3.1.45 shared wrapper (cell): The wrapper style that shares logic between the test and functional modes of operation. Shared cells typically include registered core inputs and outputs that can be used in test mode to control and observe core test data. This style prevents simultaneous functional and test operation uses of the shared register.

3.1.46 shiftWR: The wrapper terminal used to enable and control a Shift operation in the selected IEEE 1500 wrapper register (WR).

3.1.47 silent shift path: A wrapper boundary register (WBR) shift path comprising a dedicated shift path and implemented to support a Shift operation that keeps wrapper functional output (WFO) terminals static.

3.1.48 soft core: A predesigned block of functional logic such as a macro, megacell, or memory with a register transfer level (RTL) representation. Soft cores are inherently process technology independent.

3.1.49 standard test interface language (STIL): The language in IEEE Std 1450™ for representing digital test vector data. The core test language (CTL) is an extension of STIL to support a standard way of representing test data for a core.

3.1.50 system on chip (SoC): An entire system integrated on a single chip. It may include one or more cores with user-defined logic (UDL) integrated by the core user or system integrator.

3.1.51 test access mechanism (TAM): A feature of a system-on-chip (SoC) design that enables the delivery of test data to and from cores or core wrappers.

3.1.52 test access mechanism (TAM) harness: Wrapper boundary register (WBR) logic that enables the coupling of a TAM to cell test inputs (CTIs) and cell test outputs (CTOs).

3.1.53 test input (TI): The serial test data input of a wrapper boundary register (WBR).

NOTE—See TI in Figure 16.

3.1.54 test mode: A configuration whereby a block of logic is made ready for test.

3.1.55 test output (TO): The serial test data output of a wrapper boundary register (WBR).

NOTE—See TO in Figure 16.

3.1.56 test protocol: A sequence of control operations required for a test. A test protocol comprises functions and/or sequences. Functions may consist of other functions and/or sequences, while sequences comprise a series of logic 0 and 1 values applied to specified terminals. At the lowest level, a test protocol is just a series of logic 0 and 1 applied to specified test control terminals. The protocol will typically also contain symbolic references to the test data that are to be applied to or observed at a specified test data or system data port. For example, a scan protocol might involve the repetition of the following operations:

a) Apply a logic level to assert an internal scan chain SCAN_ENABLE control signal.
b) Apply a sequence of *n* clock pulses to a clock port while applying data to the SCAN_DATA_IN of the scan chain(s). Observe data at the SCAN_DATA_OUT of the scan chain(s) as the data are clocked through the scan chains.
c) De-assert SCAN_ENABLE and clock the scan chains one or more times while controlling primary inputs and observing primary outputs.
d) Repeat steps (a), (b), and (c) until the application of the scan chain patterns is complete.

3.1.57 test reuse: The ability to apply a predetermined test pattern associated with a core after this core has been integrated into a system on chip (SoC). Test reuse is a consequence of design reuse and often requires the adaptation of a test protocol to reflect the core's new environment within an SoC.

3.1.58 transferDR: The IEEE 1500 wrapper terminal provided to enable and control the Transfer operation for the wrapper boundary register (WBR).

3.1.59 update register: The register used to prevent the outputs of a shift register from propagating to other circuitry during Shift operation. After shifting is complete, the content of the associated shift register is parallel-loaded (updated) into the update register.

3.1.60 updateWR: The wrapper terminal used to enable and control an Update operation in the selected IEEE 1500 wrapper register (WR).

3.1.61 user-defined logic (UDL): Logic added by the system chip integrator (i.e., not a reused design), as interface circuitry or part of the feature set that differentiates the system-on-chip (SoC) product.

3.1.62 wrapper: Circuitry added around an embedded core to facilitate test reuse and to interface between a test access mechanism (TAM) and the embedded core.

3.1.63 wrapper boundary register (WBR): The portion of a wrapper comprising wrapper cells. *See* **wrapper cell**.

3.1.64 wrapper bypass register (WBY): The IEEE 1500 register providing an abbreviated data register between wrapper serial input (WSI) and wrapper serial output (WSO).

3.1.65 wrapper cell: The wrapper element associated with a single core terminal.

3.1.66 wrapper clock (WRCK): The IEEE 1500 clock.

3.1.67 wrapper configuration: An arrangement of interconnected wrappers in a system on chip (SoC).

3.1.68 wrapper data register (WDR): The IEEE 1500 register [e.g, wrapper boundary register (WBR), wrapper bypass register (WBY)] used to perform IEEE 1500 operations.

3.1.69 wrapper functional input (WFI): The wrapper input terminal corresponding to the functional core input of a wrapped core.

3.1.70 wrapper functional output (WFO): The wrapper output terminal corresponding to the functional core output of a wrapped core.

3.1.71 wrapper input: An input terminal to a wrapped core. In the case where there is a shared wrapper cell, the term *wrapper input* takes precedence over the term *core input*.

3.1.72 wrapper instruction register (WIR): The IEEE 1500 register used to serially load and store IEEE 1500 instructions.

3.1.73 wrapper output: An output terminal to a wrapped core. In the case where there is a shared wrapper cell, the term *wrapper output* takes precedence over the term *core output*.

3.1.74 wrapper parallel port (WPP): A user-defined set of wrapper terminals corresponding to a test access mechanism (TAM) and providing parallel access to the wrapper boundary register (WBR) and/or an embedded core.

3.1.75 wrapper register (WR): A wrapper instruction register (WIR) or wrapper data register (WDR).

3.1.76 wrapper reset (WRSTN): The IEEE 1500 active low reset.

3.1.77 wrapper serial input (WSI): The wrapper serial data input.

3.1.78 wrapper serial output (WSO): The wrapper serial data output.

3.1.79 wrapper serial port (WSP): Standard wrapper terminals providing serial access to the IEEE 1500 wrapper. The following signals comprise the WSP: wrapper reset (WRSTN), wrapper clock (WRCK), SelectWIR, TransferDR, ShiftWR, CaptureWR, UpdateWR, wrapper serial input (WSI), and wrapper serial output (WSO). Additionally, user-defined auxiliary clock(s) (AUXCKs) may be part of the WSP.

3.2 Acronyms and abbreviations

ASIC	application-specific integrated circuit
ATPG	automatic test pattern generation
AUXCK	auxiliary clock
CDR	core data register
CFI	cell functional input
CFO	cell functional output
CTI	cell test input
CTL	core test language
CTO	cell test output
DFT	design-for-test
FPGA	field-programmable gate array
IC	integrated circuit
IF	inward facing
I/O	input/output
IP	intellectual property
OF	outward facing
PnP	plug-and-play
RF	radio frequency
RTL	register transfer level
SoC	system on chip
STIL	standard test interface language
TAM	test access mechanism
TAP	test access port
TCK	IEEE 1149.1 test clock
TI	test input
TO	test output
UDL	user-defined logic
WBR	wrapper boundary register

WBY	wrapper bypass register
WDR	wrapper data register
WFI	wrapper functional input
WFO	wrapper functional output
WGL	waveform generation language
WIR	wrapper instruction register
WPC	wrapper parallel control
WPI	wrapper parallel input
WPO	wrapper parallel output
WPP	wrapper parallel port
WR	wrapper register
WRCK	wrapper clock
WRSTN	wrapper reset
w.r.t	with respect to
WSC	wrapper serial control
WSI	wrapper serial input
WSO	wrapper serial output
WSP	wrapper serial port

4. Structure of this standard

The three mandatory hardware blocks along with the standard instructions and the mandatory serial port are described in separate clauses of this standard:

— Standard instructions: Clause 7
— Wrapper serial port (WSP): Clause 8 and Clause 14
— Wrapper instruction register (WIR): Clause 10
— Wrapper bypass register (WBY): Clause 11
— Wrapper boundary register (WBR): Clause 12

The above hardware clauses are complemented with information model and compliance discussion split across the following clauses:

— Compliance definitions common to wrapped and unwrapped cores: Clause 17
— Compliance definitions specific to unwrapped cores: Clause 18
— Compliance definitions specific to wrapped cores: Clause 19

4.1 Specifications

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

a) *Rules* specify the mandatory aspects of this standard. Subclauses that are rules contain the word *shall*.

b) *Recommendations* indicate preferred practice for designs that seek to conform to this standard. Subclauses that are recommendations contain the word *should*.

c) *Permissions* show how optional features may be introduced into a design that seeks to conform to this standard. These features will extend the application of the test circuitry defined by this standard. Subclauses that are permissions contain the word *may*.

It is assumed throughout this standard that rules take precedence over recommendations and recommendations take precedence over permissions.

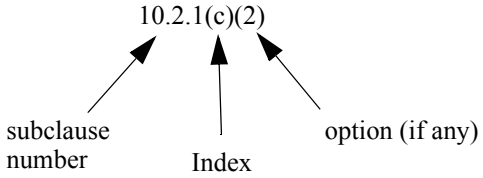
4.2 Descriptions

Material not contained in subclauses entitled “Specification” is descriptive material that illustrates the need for the features being specified or their application. This material includes schematics that illustrate a possible implementation of the specifications in this standard. Annex clauses to this standard contain alternative implementation examples. The descriptive material also discusses rationale for inclusion of certain features.

The descriptive material contained in this standard is for illustrative purposes only and does not define a preferred implementation. Most examples provided are intentionally somewhat ambiguous to not display bias towards a particular implementation style. Where discrepancies between examples and specifications may occur, the specifications always take precedence. Readers should exercise caution when using these examples to ensure full compliance in their specific applications. In particular, it is emphasized that the examples are designed to effectively communicate the meaning of this standard. As always, a particular implementation may not operate properly with respect to timing and other parametric characteristics.

The following structural conventions are used in this standard:

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



— When the reference and the referring text belong to the same subclause, only the index is indicated (i.e., the subclause number is omitted).

5. Introduction and motivations of two compliance levels

IEEE Std 1500 recognizes the heterogeneous aspect of the embedded core market, i.e., a diversity driven by the need to cover a variety of design functions implemented in digital logic, analog logic, memory, radio frequency (RF), field-programmable gate arrays (FPGAs), or combinations of the above. Cores come in many different “flavors” (e.g., hard, firm, soft) and are being used and/or sold within companies as well as between companies. Within its focus on nonmerged digital logic and memory cores, IEEE Std 1500 intends to support the above diversity and associated business models. This requirement poses a need for flexibility that was translated into the definition of the following two levels of compliance to this standard:

— **IEEE 1500 unwrapped compliance:** This compliance level refers to a core that does not have a (complete) IEEE 1500 wrapper, but does have a IEEE 1500 CTL description on the basis of which the core could be made IEEE 1500 wrapped compliant, either manually or by using dedicated tools. The CTL program describes the core test knowledge at the bare core terminals.

— **IEEE 1500 wrapped compliance:** This compliance level refers to a core that incorporates an IEEE 1500 wrapper function and comes with an IEEE 1500 CTL program. The CTL program describes the core test knowledge, including how to operate the wrapper, at the wrapper's external terminals.

The motivation behind the two different levels of compliance is a need to match the core-provider/core-user business model and provide the flexibility that is required in testing core-based system chips. The ultimate compliance goal should be to make a core compliant to the IEEE 1500 wrapped level. Nevertheless, the two levels of compliance provide the option to become IEEE 1500 wrapped directly or via the intermediate step of being IEEE 1500 unwrapped.

The first alternative, i.e., direct generation of a IEEE 1500 wrapped core, provides the possibility to integrate the wrapper functionality with the core itself and hence minimize the performance and area impact of the wrapper. The second alternative is to first create an IEEE 1500 unwrapped core, which is then, in a separate step, turned into an IEEE 1500 wrapped core. This allows the SoC integrator to take advantage of the scalability of the standardized wrapper and instantiate the wrapper with particular parameter values, which take into account certain aspects of the system chip environment in which this particular core version is used.

6. Overview of the IEEE 1500 scalable hardware architecture

The IEEE 1500 core wrapper comprises the following:

— Serial interface terminals forming the WSP

— A user-defined set of wrapper terminals forming the wrapper parallel port (WPP) and providing parallel access to the wrapper

— A WIR

— A WBY

— A WBR

Figure 1 illustrates the standard components of the IEEE 1500 wrapper.

6.1 Wrapper serial port (WSP)

The WSP terminals serve as the primary interface to the IEEE 1500 wrapper. This set of serial terminals could be sourced from chip-level pins or from an embedded controller such as an IEEE 1149.1-based controller, as described in Annex C. The WSP is used to load and unload instructions and data into and out of the IEEE 1500 registers. In addition to the wrapper serial input (WSI) and wrapper serial output (WSO) terminals shown in Figure 1, the WSP contains wrapper serial control (WSC) terminals used to control the operation of all IEEE 1500 registers. The WSP is described with further details in Clause 8.

6.2 Wrapper parallel port (WPP)

The WPP is a user-defined set of wrapper terminals providing a parallel interface to the IEEE 1500 wrapper. These terminals are used when the wrapper is configured into parallel mode. The WPP terminals consist of the wrapper parallel input (WPI) terminal(s), wrapper parallel output (WPO) terminal(s), and wrapper parallel control (WPC) terminals. The set of WPC terminals may include elements of the set of WSC terminals. The WPP is described with further details in Clause 9.

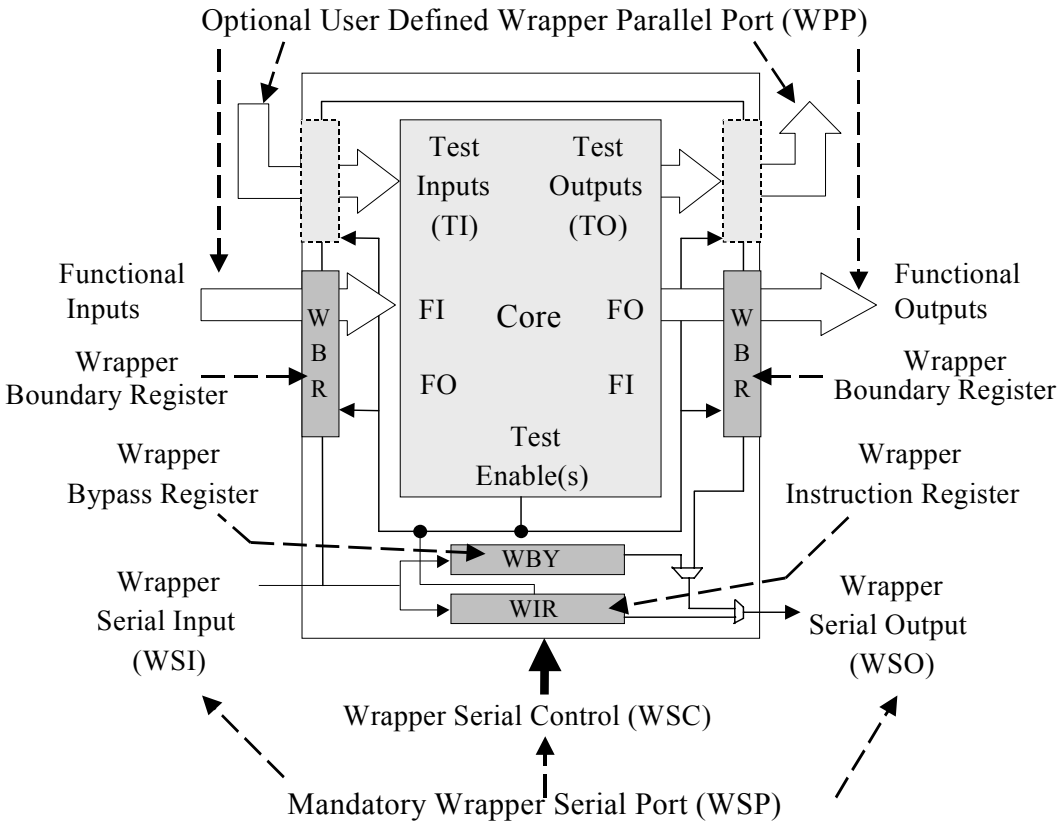


Figure 1—Standard IEEE 1500 wrapper components

6.3 Wrapper instruction register (WIR)

The WIR enables all IEEE 1500 wrapper operations. This register is loaded via the WSP with instructions that select an IEEE 1500 data register. The WIR can optionally be interfaced to the core for establishing test mode or functional operation. The WIR is described with further details in Clause 10.

6.4 Wrapper bypass register (WBY)

The WBY provides a bypass path for the WSI-WSO terminals of the WSP. The WBY is the default data register between WSI and WSO and should be selected by the current wrapper instruction when no other data register is selected. The WBY is intended to provide a minimum length scan path through the wrapper, so that when several IEEE 1500 wrappers are serially chained together in a system on chip (SoC), the wrappers that do not require a data register to be accessed can be bypassed with a short scan path through their WSI-WSO terminals. The WBY is described with further details in Clause 11.

6.5 Wrapper boundary register (WBR)

The WBR is the data register through which test data stimuli are applied and pattern responses are captured. This register allows internal testing of the core, as well as testing of external connectivity to other cores and SoC integration circuitry, in response to an instruction loaded into the WIR. The WBR is described with further details in Clause 12.

7. WIR instructions

The WIR allows instructions to be serially entered into the wrapper circuitry via the WSI and the WSC. This clause defines the minimum set of instructions that shall be supplied and the operations that occur in response to those instructions. Optional instructions and the resulting operation of the wrapper circuitry are also defined, together with the requirements for extensions to the instruction set defined in this standard. The tests applied using IEEE 1500 instructions are to be completely specified with CTL. The CTL identifies attributes of the wrapped core such as the following:

— Location and number of core wrapper input and output terminals interfacing with external circuitry,
— Protocol used to input and output the wrapper scan data,
— Condition or state of the core input and output terminals during the external circuitry test, excluding the test input (TI) and test output (TO) terminals and the terminals interfacing with the external circuitry, and
— Configuration of the WBR.

7.1 Introduction

The IEEE 1500 wrapper has various modes of operation. There are modes for functional (nontest) operation, inward facing (IF) test operation, and outward facing (OF) test operation. Different test modes also determine whether the serial test data mechanism (WSI–WSO) or the parallel test data mechanism (WPI–WPO), if present, is being utilized.

Instructions loaded into the WIR, together with the IEEE 1500 wrapper signals, determine the mode of operation of the wrapper and possibly the core itself. There is a minimum set of instructions and corresponding operations that shall be supplied. Optional instructions and their corresponding behavior are also defined, together with the requirements for extension of the instruction set. All instructions that establish test modes that utilize the parallel port WPI and WPO are optional, as the presence of this port is optional. Furthermore, IEEE Std 1500 also allows for user-defined instructions.

IEEE Std 1500 has a set of instructions that are defined to use only the serial interface (WSP) and a corresponding set of instructions that are defined for the parallel interface. IEEE Std 1500 must allow accessibility to test the core. There is one main core test instruction—Wx_INTEST (user-specified core-test instruction)—that is flexible enough to allow any core test to execute. IEEE Std 1500 does not define the test for the core, and this very flexible instruction was specified so that it could be defined differently for each core test. There are two other instructions that are mandatory: an instruction for functional mode (WS_BYPASS) and an instruction for external test mode (WS_EXTEST). WS_BYPASS puts the wrapper into the bypass configuration and allows access to all functional terminals of the core. WS_EXTEST is the serial EXTEST configuration of the wrapper. Even if there is a WP_EXTEST mode (for parallel access), there must still be a WS_EXTEST instruction capability.

7.2 Response of the wrapper circuitry to instructions

7.2.1 Specifications

Rules

a) Each instruction shall select the wrapper register(s) (WRs) that operate while the instruction is active.
b) WRs that are not selected by the active instruction shall be controlled so that they do not interfere with the operation of the core circuitry or the selected WRs.
c) Each instruction shall cause a WR path to be selected to shift data between WSI and WSO with the proper WSC signals.

d) The data register path selected between WSI and WSO shall be of fixed length for each instruction, and the length shall remain constant while the instruction is active.

e) The WBY shall be selected to shift data between WSI and WSO while parallel instructions are active.

f) Data that are loaded per an instruction into the WBR and to be used by a subsequent instruction must be held until the subsequent instruction becomes active.

Permissions

g) The WR path [of rule 7.2.1(c)] between WSI and WSO may comprise one or more serially connected wrapper data registers (WDRs) or core data registers (CDRs).

h) A user-defined wrapper instruction may enable an alternate mechanism for access to WDRs and/or CDRs.

7.2.2 Description

The instructions loaded into the WIR are interpreted in order to achieve two key functions. The active instruction identifies the wrapper or CDRs that may operate while the instruction is active. Several wrapper or CDRs may be set into test modes simultaneously. Also, an instruction identifies the WR path that is used to shift data between WSI and WSO during scan operation. A particular instruction may result in one or more wrapper or CDRs being serially connected between WSI and WSO or WPI and WPO. Further, the active instruction may select one or more other registers, separate from the register(s) between WSI and WSO or WPI and WPO, to perform other test functions. Nonselected wrapper or CDRs should be controlled so that they will not interfere with the operation of the core circuitry or with the operation of a WR path. Rule 7.2.1(f) allows for interdependent instructions such as WS_PRELOAD and WS_CLAMP to properly establish and use setup data.

The standard instructions are defined in Table 1. Only the WSP is utilized during the serial instructions.

Table 1—Instruction list

Instruction	Mandate	Description
WS_BYPASS	Required	Allows normal (functional) mode and puts the wrapper into bypass mode.
WS_EXTEST	Required	Allows external test using a single chain configuration in the WBR.
WP_EXTEST	Optional	Allows external test using a multiple scan chain configuration in the WBR.
Wx_EXTEST	Optional	A user-specified external test instruction.
WS_SAFE	Optional	Puts the core into a quiet mode and outputs a predefined static (safe) state from all output ports. It also puts the WBR into bypass mode.
WS_CLAMP	Optional	Outputs a programmable static (safe) state from all output ports. It also puts the wrapper into bypass mode. Preceded by a Wx_PRELOAD instruction.
WS_PRELOAD	Conditionally required[a]	Loads data into the single silent shift path of the WBR.
WP_PRELOAD	Optional	Loads data into the multiple silent shift paths of the WBR.
WS_INTEST_RING	Optional[b]	Allows internal testing using a single chain configuration in the WBR.

Table 1—Instruction list *(continued)*

Instruction	Mandate	Description
WS_INTEST_SCAN	Optional[b]	Allows internal testing by concatenation of the wrapper chain with a single internal chain.
Wx_INTEST	Required	A user-specified core test instruction.

[a]WS_PRELOAD is required only if there is a silent shift path in the WBR.
[b]This instruction is also considered a Wx_INTEST instruction, just more completely (specifically) defined. This instruction can be used as the required Wx_INTEST instruction.

7.3 Wrapper instruction rules and naming convention

There is a naming convention for the instructions.

W<Parallel/Serial/Hybrid> _<Mode>_<Configuration> (e.g. WS_INTEST_SCAN)

— **W:** Prefaces all standard IEEE 1500 instructions.

— **Parallel/Serial/Hybrid:** An S denotes a serial mode instruction. A P denotes a parallel mode instruction. An H denotes a hybrid instruction in which both the serial and parallel data ports are utilized. Note that a standard parallel instruction (e.g., WP_EXTEST) must have the WBY between WSI and WSO but remain distinct from a hybrid instruction. An instruction by which any register other than WBY is configured between WSI and WSO and that also uses the parallel port is classified as a hybrid instruction.

— **Mode:** A shortened description of the instruction mode, such as BYPASS, PRELOAD, etc.

— **Configuration:** A shortened description of the configuration selected by a particular instruction. For instance, during the serial instructions WS_INTEST_SCAN and WS_INTEST_RING, SCAN denotes that internal scan chains are included in the single scan chain between WSI and WSO. RING indicates that only the wrapper chain is between the WSI and WSO.

7.3.1 Specifications

Rules

a) All serial instructions (e.g., WS_EXTEST) shall use the WSP solely.

b) All parallel instructions (e.g., WP_EXTEST) shall use the WPP solely.

c) At least one INTEST instruction is required.

d) Each instruction name in Table 1 shall be used for the standard instruction associated with it.

e) No user instruction shall utilize the standard instruction names.

f) When Wx is stated in this standard, the x shall be replaced with P, S, or H.

g) WS_EXTEST instruction is required.

h) WS_BYPASS instruction is required and shall be used during any operation where the wrapper is disabled.

i) All standard instructions shall have a unique opcode.

j) Standard instruction names, such as WP_EXTEST, defined in this standard shall not be reused as names for instructions with functionality other than the functionality described in this standard. The standard instruction names are reserved.

Recommendations

k) User and private instructions should follow the same naming convention as the standard instructions.

Permissions

l) User-defined instructions may share opcodes with standard instructions.

m) Standard instruction names may be used as a prefix for other instructions defined by the user, such as WP_EXTEST_**BIST**.

7.3.2 Description

Rules 7.3.1(a) and 7.3.1(b) are meant to specify usage of the WSP and WPP for serial and parallel instructions. Additional instruction-specific description is provided in 7.4 through 7.14.

Where permission 7.3.1(l) is exercised, the corresponding user-defined instruction is expected to be identical or a superset of the standard instruction that has the same opcode as this user-defined instruction.

7.4 WS_BYPASS Instruction

The mandatory WS_BYPASS instruction enables the functional configuration of the wrapper. WS_BYPASS is selected when no test operation of that core is required and allows only the WBY to be selected. The WBY provides a minimum-length serial path between the wrapper's WSI and the WSO. This allows more rapid movement of test data to and from other core wrappers, provided the wrappers are connected serially.

7.4.1 Specifications

Rules

a) Each wrapper shall provide a WS_BYPASS instruction.

b) The WS_BYPASS instruction shall select the WBY to be connected for serial access between the WSI and WSO of the wrapper.

c) While the WS_BYPASS instruction is selected, the operation of the wrapper circuitry shall have no effect on the operation of the core circuitry.

d) While the WS_BYPASS instruction is selected, all wrapper boundary cells that can operate in either system or test modes shall perform their system function.

e) While the WS_BYPASS instruction is selected, the WSC shall be utilized for test control.

Permissions

f) The binary code for the WS_BYPASS instruction may be selected by the wrapper designer.

7.4.2 Description

During testing of a particular core or cluster of cores on an SoC, it may be inconvenient to drive data through the entire length of a WBR. Instead the WBR is replaced by the WBY. This allows data to transfer through the wrappers of the cores that are not being tested, more directly to and from the wrappers of the cores that are being tested, provided these wrappers are serially connected. The bypassed cores will have the ability to continue to run in system mode. The WS_BYPASS instruction could be used for this purpose. This instruction is the active instruction after wrapper reset (WRSTN) is asserted. Figure 2 shows an example of the configuration of the wrapper during the WS_BYPASS instruction.

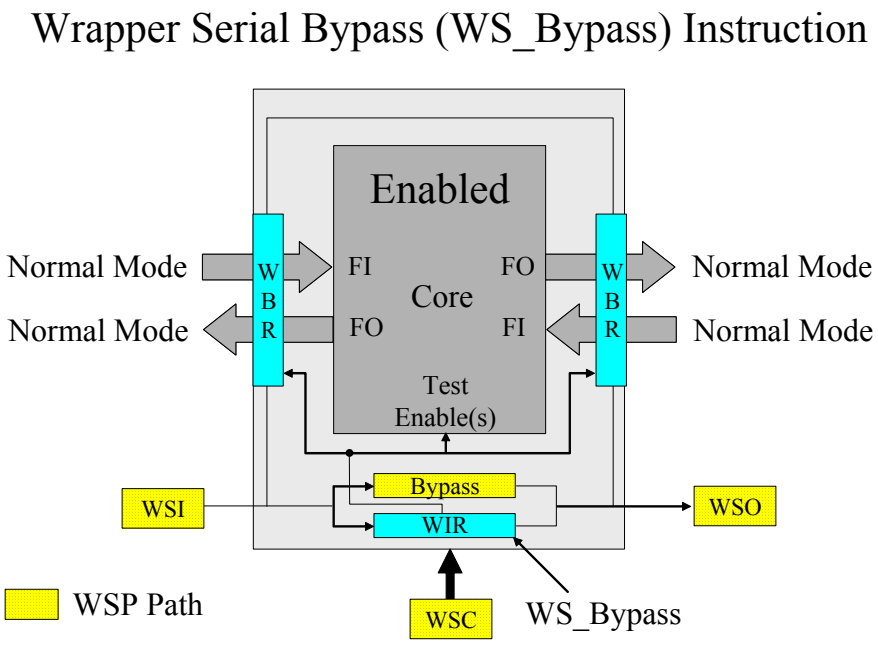


Figure 2—WS_BYPASS example

7.5 WS_EXTEST instruction

The mandatory WS_EXTEST instruction allows testing of off-core circuitry and core-to-core interconnections.

NOTE 1—Wrapper boundary cells that support the Wx_PRELOAD instruction (see 7.9 and 7.10) are presumed to have data loaded into the WBR using the Wx_PRELOAD instruction prior to loading the Wx_EXTEST instruction.[3]

NOTE 2—Following use of the WS_EXTEST instruction, the core circuitry may be in an indeterminate state that will persist until a system reset is applied. Therefore, the core circuitry may need to be reset on return to normal (i.e., nontest) operation.

7.5.1 Specifications

Rules

a) Each wrapper shall provide a WS_EXTEST instruction.

b) While the WS_EXTEST instruction is selected, only the WBR shall be connected for serial access between WSI and WSO during the Shift operation (i.e., no other test data register may be connected in series with the WBR).

c) While the WS_EXTEST instruction is selected, the core circuitry shall be controlled so that it cannot be damaged as a result of signals received at core input or core clock input terminals.

   NOTE—This might be achieved by placing the core circuitry in a reset or hold state while the WS_EXTEST instruction is selected.

d) While the WS_EXTEST instruction is selected, the state of all signals driven from the WBR wrapper functional outputs (WFOs) shall be completely defined by the data held in the WBR cell associated with that terminal. The data at the output shall be valid with the Apply event and remain stable until the next WBR event.

[3]Notes in text, tables, and figures are given for information only and do not contain requirements needed to implement this standard.

NOTE 1—Where the WBR provides a silent shift path, the data held in the WBR are presumed to have been loaded by the previous use of an instruction such as WS_PRELOAD.

NOTE 2—Where the WBR comprises shared wrapper boundary cells (i.e., the wrapper boundary cells are shared between functional and test operation) that do not have a silent shift path, the data held in the WBR exist as a result of the previous instruction or of entering the WS_EXTEST instruction.

e) While the WS_EXTEST instruction is selected, the state of all signals received at the core input terminals required to be provisioned with wrapper boundary cells per rule 12.1.1(a) shall be loaded into the WBR during the Capture event.

f) While the WS_EXTEST instruction is selected, the state of WFO terminals shall not change in response to the Capture event.

g) While the WS_EXTEST instruction is selected, the WBR shall be in OF mode.

h) While the WS_EXTEST instruction is selected, the WSC shall be utilized for test control.

Recommendations

i) While the WS_EXTEST instruction is selected, the core should be put into a quiet mode (e.g., reset or clock off).

j) While the WS_EXTEST instruction is selected and where output cells that are in the WBR shift path are used, the WFOs should be controlled to safe values during shift.

Permissions

k) A binary code for the WS_EXTEST instruction may be selected by the wrapper designer.

7.5.2 Description

The WS_EXTEST instruction allows circuitry external to the core wrapper [typically the interconnects and user-defined logic (UDL)] to be tested. The wrapper boundary cells at WFOs are used to apply test stimuli, while the cells at wrapper input terminals capture test results. This instruction also allows testing of blocks of UDL between cores that do not themselves incorporate wrappers.

While the WS_EXTEST instruction is selected, the core circuitry may receive input signals that differ significantly from those expected during normal operation. If the core circuitry can tolerate any permutation of input signals that is received, then no specific design changes are required. However, for some cores, there may be input sequences that could place the core circuitry in a state where damage may result or excessive current may be drawn. In these cases, it is the responsibility of the wrapper designer to prevent the core circuitry from processing the “illegal” inputs while the WS_EXTEST instruction is selected. This may be achieved by placing the core circuitry into a reset or hold state during WS_EXTEST or by controlling the WFOs to the core.

The wrapper input terminals provisioned with wrapper boundary cells per rule 12.1.1(a) may optionally be designed to allow signals to be driven into the core circuitry when the WS_EXTEST instruction is selected. This allows user-defined values to be established at the core input terminals, preventing undesired operation of the core in response to unknown signals arriving at core inputs during operation of the WS_EXTEST instruction. The values driven may either be constant for the duration that WS_EXTEST is selected (i.e., by using signal blocking gates at the core input terminals) or be loaded serially through the WBR. Figure 3 shows an example of the configuration of the wrapper during the WS_EXTEST instruction.

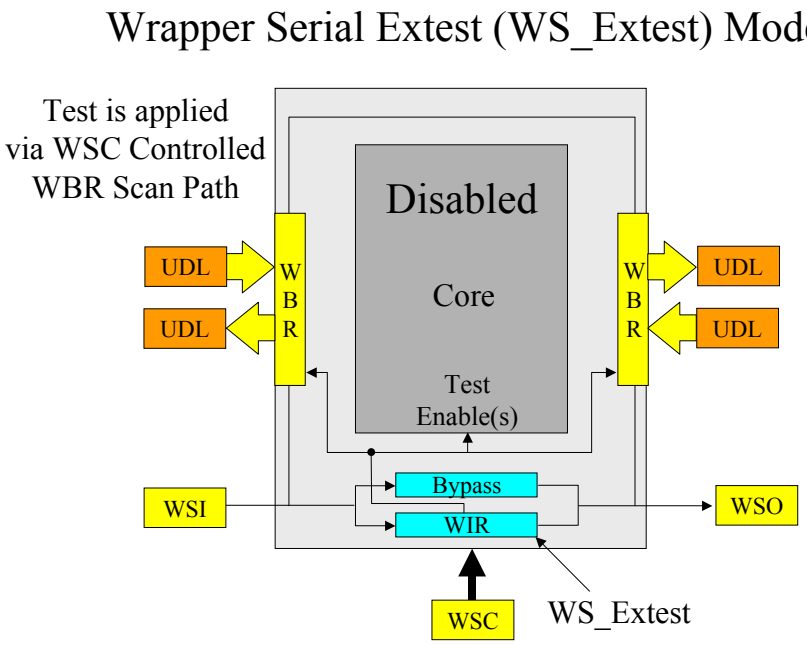


Figure 3—WS_EXTEST example

7.6 WP_EXTEST instruction

The WP_EXTEST instruction allows testing of off-core circuitry and core-to-core interconnections. The WP_EXTEST instruction allows the WBR to be divided into segments (multiple scan chains). It also allows wrapper inputs and wrapper outputs, other than the WSP, to control WBR inputs and observe WBR outputs. In all other behaviors, WP_EXTEST mimics the WS_EXTEST instruction.

NOTE 1—Wrapper boundary cells that support "preload" are presumed to have data loaded into the WBR using the WS_PRELOAD or WP_PRELOAD instruction prior to loading the WP_EXTEST instruction.

NOTE 2—Following use of the WP_EXTEST instruction, the core circuitry may be in an indeterminate state that will persist until a system reset is applied. Therefore, the core circuitry may need to be reset on return to normal (i.e., nontest) operation.

7.6.1 Specifications

Rules

a) While the WP_EXTEST instruction is selected, only the WBR shall be selected between WPI and WPO during the Shift operation (i.e., no other test data register may be connected in series with the WBR).

b) While the WP_EXTEST instruction is selected, the core circuitry shall be controlled so that it cannot be damaged as a result of signals received at core input or core clock input terminals.

   NOTE—This might be achieved by placing the core circuitry in a reset or hold state while the WP_EXTEST instruction is selected.

c) While the WP_EXTEST instruction is selected, the state of all signals received at the wrapper input terminals required to be provisioned with wrapper boundary cells per rule 12.1.1(a) shall be loaded into the WBR during the Capture event.

d) While the WP_EXTEST instruction is selected, the state of all signals driven from the WBR WFOs shall be completely defined by the data held in the WBR cell associated with that terminal. The data at the output shall be valid with the Apply event and remain stable until the next WBR event.

NOTE 1—Where the WBR provides a silent shift path, the data held in the WBR are presumed to have been loaded by the previous use of an instruction such as WS_PRELOAD.

NOTE 2—Where the WBR comprises shared wrapper boundary cells (i.e., the wrapper boundary cells are shared between functional and test operation) that do not have a silent shift path, the data held in the WBR must exist as a result of either the previous instruction or of entering the WP_EXTEST instruction.

e) While the WP_EXTEST instruction is selected, the WBR shall be in OF mode.

f) While the WP_EXTEST instruction is selected, the state of WFO terminals shall not change in response to the Capture event.

Recommendations

g) While the WP_EXTEST instruction is selected, the core should be put into a quiet mode (e.g., reset or clock off).

h) While the WP_EXTEST instruction is selected, where output cells that are in the WBR shift path are used, the WFOs should be controlled to safe values during shift.

Permissions

i) The WBR may be segmented into one or more scan chains with the input and output port of the scan chains interfaced to the TAM.

j) While the WP_EXTEST instruction is selected, the ShiftWR, CaptureWR, UpdateWR, and TransferDR terminals may be used.

k) The binary code for the WP_EXTEST instruction may be selected by the wrapper designer.

7.6.2 Description

The optional WP_EXTEST instruction allows circuitry external to the core wrapper (typically the interconnects and glue logic) to be tested. Wrapper boundary cells at WFOs are used to apply test stimuli, while the cells at wrapper input terminals capture test results. This instruction also allows testing of blocks of UDL between cores that do not themselves incorporate wrappers.

While the WP_EXTEST instruction is selected, the core circuitry may receive input signals that differ significantly from those expected during normal operation. If the core circuitry can tolerate any permutation of input signals that is received, then no specific design changes are required. However, for some cores, there may be input sequences that could place the core circuitry in a state where damage may result or excessive current may be drawn. In these cases, it is the responsibility of the designer to prevent the core circuitry from processing the “illegal” inputs while the WP_EXTEST instruction is selected. This may be achieved by placing the core circuitry into a reset or hold state during WP_EXTEST or by controlling the cell functional outputs (CFOs) connected to the core.

ShiftWR, CaptureWR, UpdateWR, and TransferDR terminals are defined in 8.1.2.

The wrapper input terminals provisioned with wrapper boundary cells per rule 12.1.1(a) may optionally be designed to allow signals to be driven into the core circuitry when the WP_EXTEST instruction is selected. This allows user-defined values to be established at the core input terminals, preventing undesired operation of the core in response to unknown signals arriving at core inputs during operation of the WP_EXTEST instruction. The values driven may either be constant for the duration that WP_EXTEST is selected (e.g., by using signal blocking gates at the core input terminals) or be loaded serially through the WBR. Figure 4 shows an example of the configuration of the wrapper during the WP_EXTEST instruction. Note that both the WPC and the WSC control pins are shown in the figure; however, either the WPC or the WSC can be used during the WP_EXTEST instruction, not a combination of both. This is the only parallel standard instruction that can utilize the WSC.

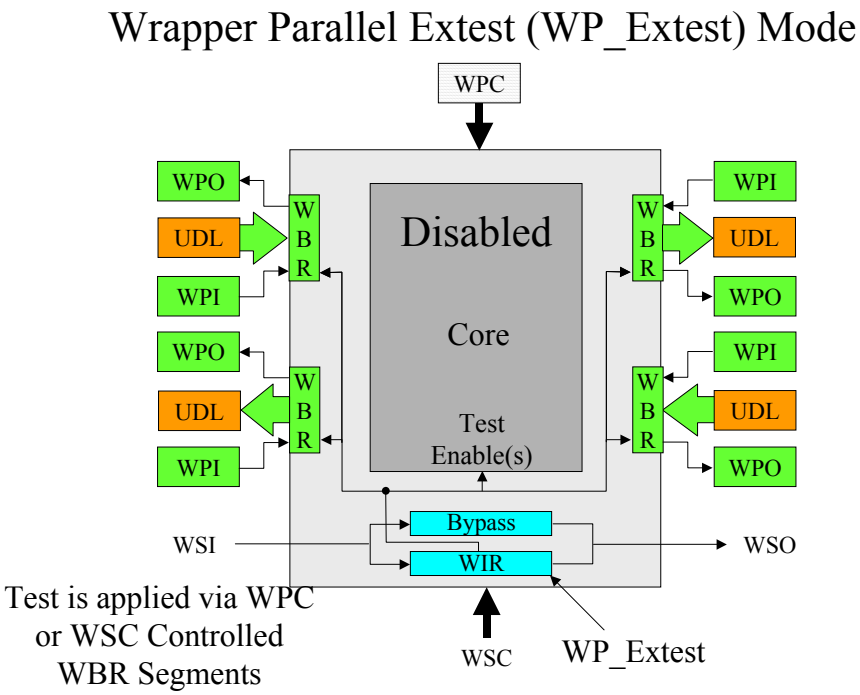


Figure 4—WP_EXTEST example

7.7 Wx_EXTEST instruction

Wx_EXTEST is an external test instruction that allows the circuitry external to the core to be tested according to the system integrator's requirements. The x in Wx is a place holder for an S, P, or H to indicate whether the instruction is serial, parallel, or hybrid, respectively.

The Wx_EXTEST instruction encompasses the WS_EXTEST and the WP_EXTEST instructions. This instruction provides the same flexibility as the Wx_INTEST instruction. The difference between the two instructions is that Wx_INTEST is an IF test and Wx_EXTEST is an OF test.

NOTE—Following use of the Wx_EXTEST instruction, the core circuitry may be in an indeterminate state that will persist until a system reset is applied. Therefore, the core circuitry may need to be reset on return to normal (i.e., nontest) operation.

The rules in 7.7.1 apply where a Wx_EXTEST instruction is provided.

7.7.1 Specifications

Rules

a) While the Wx_EXTEST instruction is selected, the WBR shall be in OF mode.

Recommendations

b) While the Wx_EXTEST instruction is selected, the core should be put into a quiet mode (e.g., reset or clock off).

c) While the Wx_EXTEST instruction is selected, the testing of the circuitry external to the core should not disturb circuitry internal to the core.

d) While the Wx_EXTEST instruction is selected, for core output terminals provisioned with wrapper boundary cells per rule 12.1.1(a), the data loaded during the Capture event should be independent of the operation of core circuitry.

e) WS_EXTEST or WP_EXTEST instructions should be utilized unless the more flexible capabilities of the Wx_EXTEST instruction are needed.

Permissions

f) The binary code(s) for the Wx_EXTEST instruction may be selected by the wrapper designer.

g) Any number of Wx_EXTEST instructions may be included by the wrapper designer.

7.7.2 Description

The Wx_EXTEST instruction allows testing of circuitry external to the core. While the Wx_EXTEST instruction is selected, the WBR assumes the role of virtual test points. Cells at core output terminals are used to apply the test stimulus, while those at core input terminals capture the test response. Stimuli and responses are moved into and out of the circuit by shifting through the WBR.

7.8 WS_SAFE instruction

The optional WS_SAFE instruction allows the state of the signals driven from WFOs to be determined from the WBR while the WBY is selected as the serial path between the serial input (WSI) and serial output (WSO) of the wrapper. The signals driven from the WFOs will not change while the WS_SAFE instruction is selected. The WS_CLAMP instruction can also be used to put the wrapper into a safe state by utilizing a Wx_PRELOAD instruction to shift in the proper states and then clamping them with the WS_CLAMP instruction. However, it is important to allow a straightforward way for the core integrator to put wrappers into a safe state. This is why there is a separate instruction for WS_SAFE. This instruction allows the core integrator to more easily put various wrappers into safe states while other portions of an SoC are being tested.

The rules in 7.8.1 apply where the WS_SAFE instruction is provided.

NOTE—Following use of the WS_SAFE instruction, the core circuitry may be in an indeterminate state that will persist until a system reset is applied. Therefore, the core circuitry may need to be reset on return to normal (i.e., nontest) operation.

7.8.1 Specifications

Rules

a) While the WS_SAFE instruction is selected, the WBY shall be connected for serial access between WSI and WSO during wrapper data scan operations.

b) While the WS_SAFE instruction is selected, the state of all signals driven from WFOs shall be hard-wired to fixed values that have been predetermined at wrapper instantiation.

NOTE—The data output from the WBR shall be safe and established as a result of entering the WS_SAFE instruction.

c) When the WS_SAFE instruction is selected, the core circuitry shall be controlled so that it cannot be damaged as a result of signals received at the core data input or core clock input terminals.

d) While the WS_SAFE instruction is selected, the WSC shall be utilized for test control.

Recommendations

e) While the WS_SAFE instruction is selected, the core circuitry should be put into a quiet mode (e.g., reset or clock off).

Permissions

f) The binary code for the WS_SAFE instruction may be selected by the wrapper designer.

7.8.2 Description

During testing of a particular core or cluster of cores on a system chip, it may be necessary to place static values on signals that control operation of circuitry not involved in the test, e.g., to place the core in a state where it will not respond to signals received from the circuitry under test.

The optional WS_SAFE instruction is similar to the WS_CLAMP instruction in that it allows safe static values to be applied using the WBRs of the appropriate core wrapper, but does not retain these registers in the serial path during test application. In a case in which the WS_SAFE instruction is used to create safe static signals, the following process would be used.

It is presumed in the following example that every wrapper implements the optional WS_SAFE instruction.

— Shift the WS_SAFE instruction into all core wrappers that will provide safe static signals during the upcoming test. Call this group of wrappers *G*. If test setup data are required in wrappers not in *G* (i.e., in the wrappers that will participate actively in the upcoming test), another instruction (e.g., WS_EXTEST) may be loaded into these core wrappers simultaneously with the loading of WS_SAFE into core group *G*.

— From this point on, until the test is concluded, every time instructions are to be scanned into core wrappers in the SoC, enter the WS_SAFE instruction into the wrappers in *G*. As long as the WS_SAFE instruction is maintained as the active instruction in the wrappers of *G*, the output signal values of these wrappers will be determined by the safe data at their WBR outputs. Also, as a consequence of the use of the WS_SAFE instruction, the wrappers in *G* all have their WBYs selected throughout the test; thus, they contribute very little to the overall test time. Figure 5 shows an example of the configuration of the wrapper during the WS_SAFE instruction.

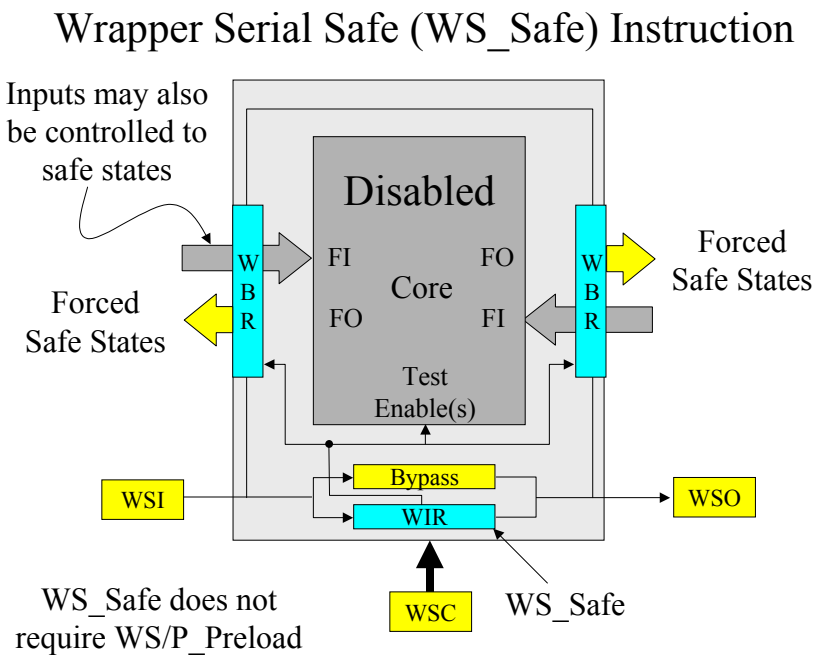


Figure 5—WS_SAFE example

7.9 WS_PRELOAD instruction

The WS_PRELOAD instruction enables the wrapper to be functionally configured. Wrappers that have a WBR composed entirely of cells with a silent shift path must include the WS_PRELOAD instruction. The WS_PRELOAD instruction allows data values to be loaded into the latched parallel outputs of the registers in the silent shift path of the WBR. The data are loaded into the silent shift path of wrapper cells, and then the data are loaded into the update stage, if present, during the Update operation. This instruction would typically be utilized before other instructions are selected (e.g., WS_EXTEST, WS_CLAMP).

7.9.1 Specifications

Rules

a) If the WBR contains a silent shift path, there shall be a WS_PRELOAD instruction.

b) While the WS_PRELOAD instruction is selected, only the WBR shall be connected for serial access between WSI and WSO during the Shift operation (i.e., no other test data register may be connected in series with the WBR).

c) While the WS_PRELOAD instruction is selected, the Shift operation of the WBR shall have no effect on the operation of the core or UDL.

d) While the WS_PRELOAD instruction is selected, the WSC shall be utilized for test control.

Recommendations

e) While the WS_PRELOAD instruction is selected, if the core is not being used in its normal operation, the core circuitry should be put into a quiet mode (e.g., reset or clock off).

Permissions

f) The binary code for the WS_PRELOAD instruction may be selected by the wrapper designer.

7.9.2 Description

The WS_PRELOAD instruction is used to allow shifting of the WBR, via WSI to WSO, without causing interference to the operation of the core or UDL attached to the WBR. WS_PRELOAD allows an initial data pattern to be placed at the latched parallel outputs of the silent shift path register cells through a scan load operation. For example, prior to selection of the WS_EXTEST instruction, data can be loaded onto the latched parallel outputs using WS_PRELOAD. As soon as the WS_EXTEST instruction has been transferred to the update register of the instruction register, the preloaded data are driven from the WFOs. This ensures that known data are driven immediately when the WS_EXTEST instruction is entered. Without WS_PRELOAD, indeterminate data may be driven until the first scan load is complete. Figure 6 shows an example of the configuration of the wrapper during the WS_PRELOAD instruction.

7.10 WP_PRELOAD instruction

The WP_PRELOAD instruction enables the wrapper to be functionally configured. Wrappers that have a WBR composed entirely of cells with a silent shift path and have parallel instruction capability, may include the WP_PRELOAD instruction. The WP_PRELOAD instruction allows data values to be loaded into the latched parallel outputs of the registers in the silent shift path(s) of the WBR. The data are loaded into the silent shift paths of wrapper cells, and then the data are loaded into the update stage, if present, during the Update operation. This instruction would typically be utilized before other defined instructions are selected (e.g., WP_EXTEST). The WP_PRELOAD instruction allows the WBR to be divided into segments (multiple scan chains), and may be preferred over the WS_PRELOAD instruction before entering the WP_EXTEST instruction. It also allows wrapper inputs and outputs, other than the WSP, to control WBR inputs and observe WBR outputs. In all other behaviors, WP_PRELOAD mimics the WS_PRELOAD instruction.

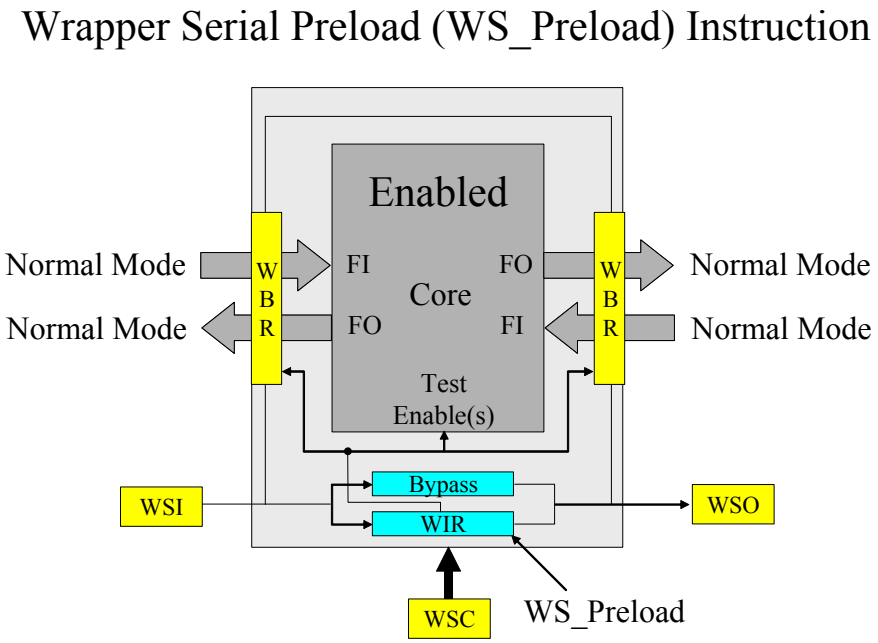


Figure 6—WS_PRELOAD example

7.10.1 Specifications

Rules

a) While the WP_PRELOAD instruction is selected, only the WBR shall be connected during the Shift operation (i.e., no other test data register may be connected in series with the WBR).

b) While the WP_PRELOAD instruction is selected, the data in the wrapper boundary cell shall be available for the Update operation, if applicable.

c) While the WS_PRELOAD instruction is selected, the Shift operation of the WBR shall have no effect on the operation of the core or UDL.

d) While the WP_PRELOAD instruction is selected, the WPC shall be utilized for test control.

Recommendations

e) While the WP_PRELOAD instruction is selected, if the core is not being used in its normal operation, the core circuitry should be put into a quiet mode (e.g., reset or clock off).

Permissions

f) The WBR may be segmented into one or more scan chains with the input and output port of the scan chains interfaced to the TAM.

g) The binary code for the WP_PRELOAD instruction may be selected by the wrapper designer.

7.10.2 Description

The WP_PRELOAD instruction is used to allow shifting of the WBR without causing interference to the operation of the core or UDL attached to the WBR. WP_PRELOAD allows an initial data pattern to be placed at the latched parallel outputs of the silent shift path register cells through a scan load operation. For example, prior to selection of the WP_EXTEST instruction, data can be loaded onto the latched parallel outputs using WP_PRELOAD. As soon as the WP_EXTEST instruction has been transferred to the parallel output of the instruction register, the preloaded data are driven from the output wrapper terminals. This

ensures that known data are driven immediately when the WP_EXTEST instruction is entered. Without WP_PRELOAD, indeterminate data may be driven until the first scan load is complete. Figure 7 shows an example of the configuration of the wrapper during the WP_PRELOAD instruction.

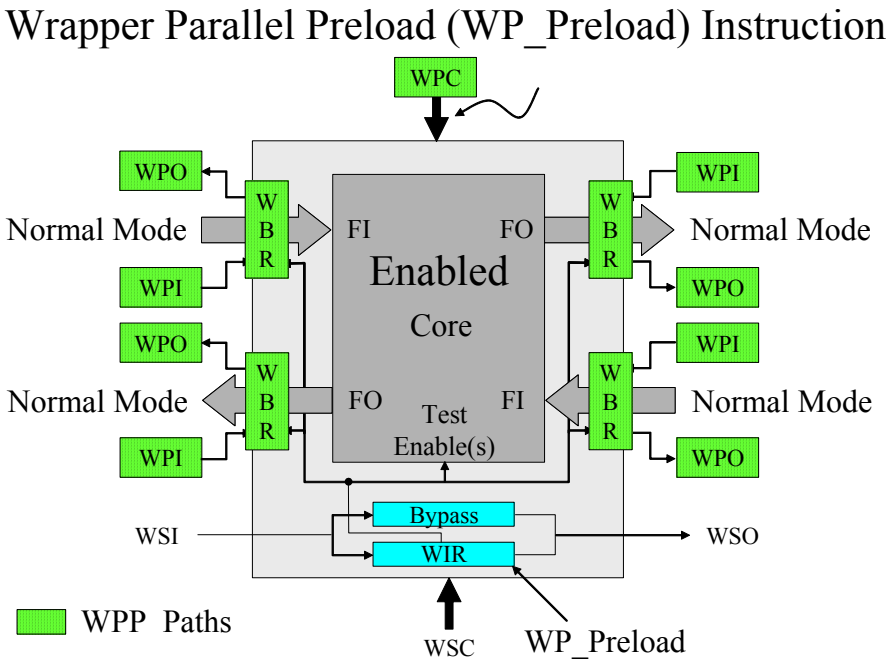


Figure 7—WP_PRELOAD example

7.11 WS_CLAMP instruction

The optional WS_CLAMP instruction allows the state of the signals driven from WFOs to be determined from the WBR while the WBY is selected as the serial path between WSI and WSO. The signals driven from the WFOs will not change while the WS_CLAMP instruction is selected.

The rules in 7.11.1 apply where the WS_CLAMP instruction is provided.

NOTE—Following use of the WS_CLAMP instruction, the core circuitry may be in an indeterminate state that will persist until a system reset is applied. Therefore, the core circuitry may need to be reset on return to normal (i.e., nontest) operation.

7.11.1 Specifications

Rules

a) While the WS_CLAMP instruction is selected, the WBY shall be connected for serial access between WSI and WSO.

b) While the WS_CLAMP instruction is selected, the state of all signals driven from WFOs shall be completely defined by the data held in the WBR.

NOTE—Data held in the WBR might be achieved by the previous use of an instruction (e.g., WS_PRELOAD).

c) While the WS_CLAMP instruction is selected, the WSC shall be utilized for test control.

Recommendations

d) While the WS_CLAMP instruction is selected, the core circuitry should be put into a quiet mode (e.g., reset or clock off).

Permissions

e) The binary code for the WS_CLAMP instruction may be selected by the wrapper designer.

7.11.2 Description

During testing of a particular core or cluster of cores on a system chip, it may be necessary to place static values on signals that control operation of circuitry not involved in the test, e.g., to place the aforementioned circuitry in a state where it cannot respond to signals received from the circuitry under test.

The optional WS_CLAMP instruction allows static values to be applied using the WBRs of the appropriate core wrapper, but does not retain these registers in the serial path during test application. In a case in which the WS_CLAMP instruction is used to create static values, the following processes would be used.

It is presumed in the following example that every wrapper implements the optional WS_CLAMP instruction.

Example: WBRs having dedicated wrapper boundary cells

— Prior to the test, a Wx_EXTEST or Wx_PRELOAD (the x denotes S, P, or H) instruction would be loaded into all core wrappers that will provide specific values during the upcoming test. Call this group of wrappers *G*. If test setup data are required in wrappers not in *G* (i.e., in the wrappers that will participate actively in the upcoming test), a Wx_EXTEST or Wx_PRELOAD instruction may be loaded into these core wrappers simultaneously with the loading of WS_CLAMP into core group *G*.

— Shift the desired pattern into all relevant wrapper boundary cells of the wrappers in *G*. Any test setup data required for the SoC to be tested are also loaded.

— Load WS_CLAMP instruction into the wrappers in *G*.

— From this point on, until the test is concluded, every time instructions are to be scanned into core wrappers in the SoC, enter the WS_CLAMP instruction into the wrappers in *G*. As long as the WS_CLAMP instruction is maintained as the active instruction in the wrappers of *G*, the output signal values of these wrappers will be determined by the data in their WBRs. Also, as a consequence of the use of the WS_CLAMP instruction, the wrappers in *G* all have their WBYs selected throughout the test; thus, they contribute very little to the overall test time.

Figure 8 shows an example of the configuration of the wrapper during the WS_CLAMP instruction. This instruction would be loaded serially into the core wrappers that drive the signals on which static values are required. The required signal values could be loaded as a part of the complete serial data stream shifted into the chip-level WBR path, both at the start of the test and each time a new test pattern is entered. However, a limitation of this approach is that the length of the data pattern to be shifted for each test is increased by inclusion of the WBR of each core wrapper involved in the process. As a result, the test application rate is reduced. WS_CLAMP reduces overall shift length when the target wrapper does not take an active part in the test.

Note that this is just one example of how to clamp a signal. Instructions such as Wx_PRELOAD or Wx_EXTEST can be used to load specific values into the wrapper before WS_CLAMP is enabled.

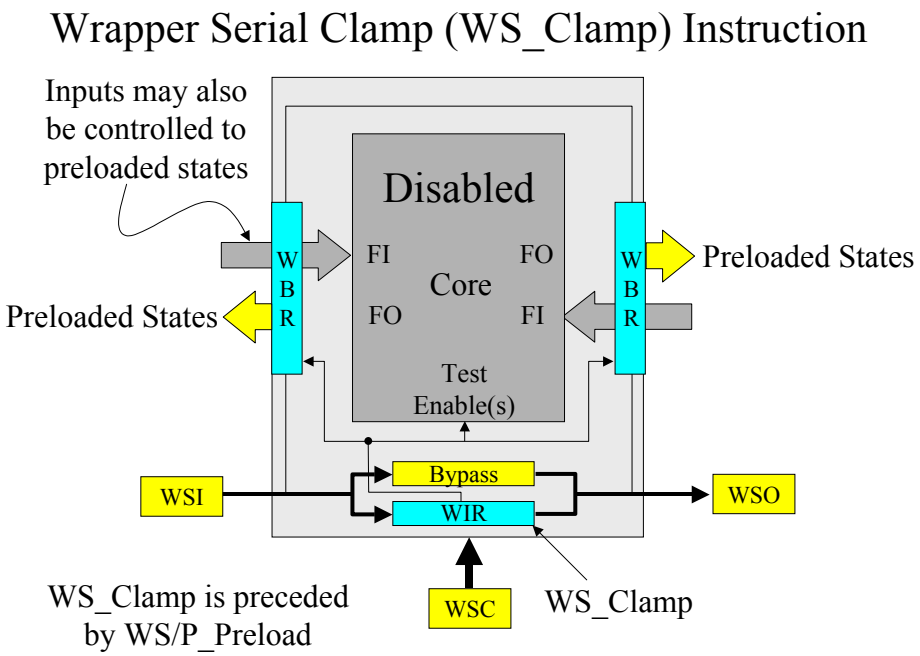


Figure 8—WS_CLAMP example

7.12 WS_INTEST_RING instruction

The optional serial wrapper core test WS_INTEST_RING instruction allows testing of the core circuitry. During the WS_INTEST_RING instruction, test stimuli are shifted in 1 bit at a time and applied to the core functional terminals via the WBR only. The test results are captured into the WBR and are shifted out 1 bit at a time for examination. The rules in 7.12.1 apply where the WS_INTEST_RING instruction is provided.

NOTE—Following the use of the WS_INTEST_RING instruction, the core circuitry may be in an indeterminate state that will persist until a core reset is applied. Therefore, the core circuitry may need to be reset on return to normal operation.

7.12.1 Specifications

Rules

a) The WS_INTEST_RING instruction shall select only the WBR to be connected for serial access between WSI and WSO (i.e., no other test data register may be connected in series with the WBR).

b) While the WS_INTEST_RING instruction is selected, the WBR shall be in IF mode.

c) While the WS_INTEST_RING instruction is selected, the WSC shall be utilized for test control.

Recommendations

d) While the WS_INTEST_RING instruction is selected, all WFOs should be placed in a safe or static drive state.

e) While the WS_INTEST_RING instruction is selected, for core input terminals provisioned with wrapper boundary cells per rule 12.1.1(a), the data loaded during the Capture event should be independent of the operation of off-core circuitry or chip-level interconnections.

Permissions

f) The binary code for the WS_INTEST_RING instruction may be selected by the wrapper designer.

7.12.2 Description

The WS_INTEST_RING instruction allows single-step testing of core circuitry with each test pattern and response being shifted through the WBR. The WS_INTEST_RING instruction requires that the core circuitry can be operated in a single-step mode, where the circuitry moves one step forward in its operation each time shifting of the WBR is completed.

While the WS_INTEST_RING instruction is selected, the WBR assumes the role of virtual test points. Cells at nonclock core input terminals are used to apply the test stimulus, while those at core output terminals capture the response. Stimuli and responses are moved into and out of the circuit by shifting through the WBR. Note that this requires that the core input terminals provisioned with wrapper boundary cells per rule 12.1.1(a) are able to drive signals into the core circuitry.

Typically, the core circuitry will receive a sequence of clock events between application of the stimulus and capture of the response so that single-step operation is achieved. The specification of wrapper boundary cells for core clock input terminals allows the clocks for the core circuitry to be obtained in several ways while the WS_INTEST_RING instruction is selected. The following are offered as examples:

a) The signals received at core clock terminals can be fed directly to the core circuitry as during normal operation of the core. Where this option is selected, the core design shall guarantee that precisely one single step of operation of the core circuitry occurs between each shifting (filling) of the WBR.

b) Circuitry may be built into the core that allows the core circuitry to complete one step of operation on completion of shifting in data into the WBR. If the core were a microprocessor, it would be permitted to complete a single processing cycle by, for example, internal generation of a pulse on the hold signal. In this case, the clock(s) applied at the core clock terminal(s) during the test could be free-running.

While the WS_INTEST_RING instruction is selected, the state of all WFOs is determined by the wrapper circuitry. Every WFO should be forced to a static or inactive drive state. This ensures that surrounding circuitry on an SoC is supplied known, safe signal levels while the core circuitry test is in progress.

Recommendation 7.12.1(e), where followed, ensures that data shifted out of the core in response to the WS_INTEST_RING instruction are not altered by the presence of defects in off-core circuitry, chip-level interconnections, etc. This simplifies diagnosis, since any errors in the output bit stream can be caused only by defects in the core circuitry or in the WBR. Figure 9 shows an example of the configuration of the wrapper during the WS_INTEST_RING instruction.

7.13 WS_INTEST_SCAN instruction

The optional serial core test WS_INTEST_SCAN instruction is one of the instructions defined by this standard that allows testing of the core circuitry. During the WS_INTEST_SCAN instruction, test stimuli are shifted in 1 bit at a time and applied to the core functional terminals via the WBR only. This instruction is identical to the WS_INTEST_RING instruction, with the exception of internal scan chain concatenation to the WBR scan chain, referred to in this standard as the serial scan chain. The test results are captured into the serial scan chain and are examined by subsequent shifting.

The rules in 7.13.1 apply where the WS_INTEST_SCAN instruction is provided.

NOTE—Following the use of the WS_INTEST_SCAN instruction, the core circuitry may be in an indeterminate state that will persist until a core reset is applied. Therefore, the core circuitry may need to be reset on return to normal operation.

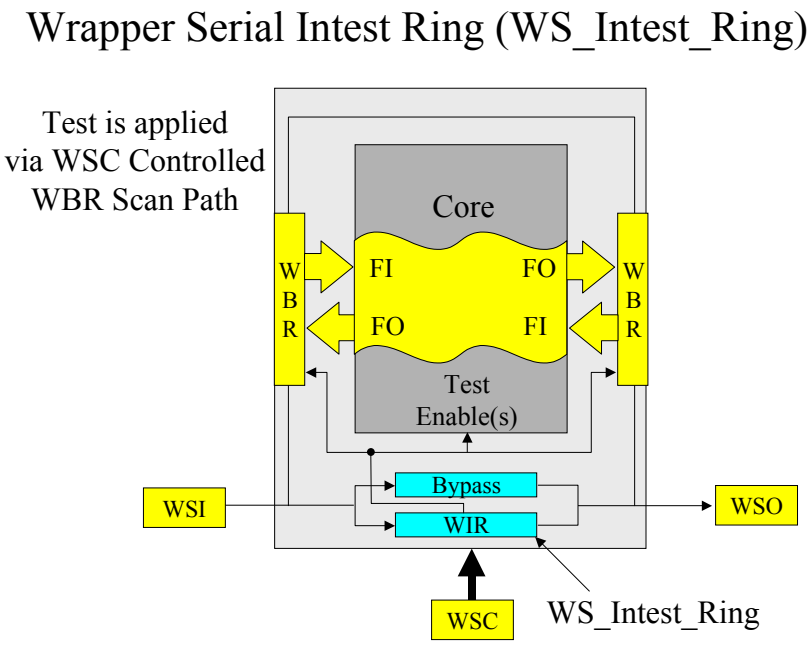


Figure 9—WS_INTEST_RING example

7.13.1 Specifications

Rules

a) While the WS_INTEST_SCAN instruction is selected, the WBR concatenated with an internal scan chain, which includes all internal scan cells, shall be connected for serial access between WSI and WSO.

b) While the WS_INTEST_SCAN instruction is selected, the WBR shall be in IF mode.

c) While the WS_INTEST_SCAN instruction is selected, the WSC shall be utilized for test control.

Recommendations

d) While the WS_INTEST_SCAN instruction is selected, all WFOs should be placed in a safe or static drive state.

e) While the WS_INTEST_SCAN instruction is selected, for core input terminals provisioned with wrapper boundary cells per rule 12.1.1(a), the data loaded during the Capture event should be independent of the operation of off-core circuitry or chip-level interconnections.

f) The serial scan chain should comprise the WBR and every scanned element in the core.

Permissions

g) The binary code for the WS_INTEST_SCAN instruction may be selected by the wrapper designer.

7.13.2 Description

The WS_INTEST_SCAN instruction allows single-step testing of core circuitry with each test pattern and response being shifted through the serial scan chain.

While the WS_INTEST_SCAN instruction is selected, the serial scan chain assumes the role of virtual test points. Cells at nonclock core input terminals are used to apply the test stimulus, while those at core output terminals capture the response. Stimuli and responses are moved into and out of the circuit by shifting the

serial scan chain. This operation requires that the core input terminals provisioned with wrapper boundary cells per rule 12.1.1(a) are able to drive signals into the core circuitry.

Typically, the core circuitry will receive a sequence of clock events between application of the stimulus and capture of the response so that single-step operation is achieved. The specification of wrapper boundary cells for core clock input terminals allows the clocks for the core circuitry to be obtained in several ways while the WS_INTEST_SCAN instruction is selected. The following are offered as examples:

a) The signals received at core clock terminals can be fed directly to the core circuitry as during normal operation of the core. Where this option is selected, the core design shall guarantee that precisely one single step of operation of the core circuitry occurs between each shifting (filling) of the serial scan chain.

b) Circuitry may be built into the core that allows the core circuitry to complete one step of operation on completion of shifting in data into the serial scan chain. If the core were a microprocessor, it would be permitted to complete a single processing cycle by, for example, internal generation of a pulse on a hold signal. In this case, the clock(s) applied at the core clock terminal(s) during the test could be free-running.

While the WS_INTEST_SCAN instruction is selected, the state of all WFOs is determined by the wrapper circuitry. Every WFO should be forced to a static or inactive drive state. This ensures that surrounding circuitry on an SoC is supplied known, safe signal levels while the core circuitry test is in progress.

Recommendation 7.13.1(e), where followed, ensures that data shifted out of the core in response to the WS_INTEST_SCAN instruction are not altered by the presence of defects in off-core circuitry, chip-level interconnections, etc. This simplifies diagnosis, since any errors in the output bit stream can be caused only by defects in the core circuitry or in the WBR. Figure 10 shows an example of the configuration of the wrapper during the WS_INTEST_SCAN instruction.

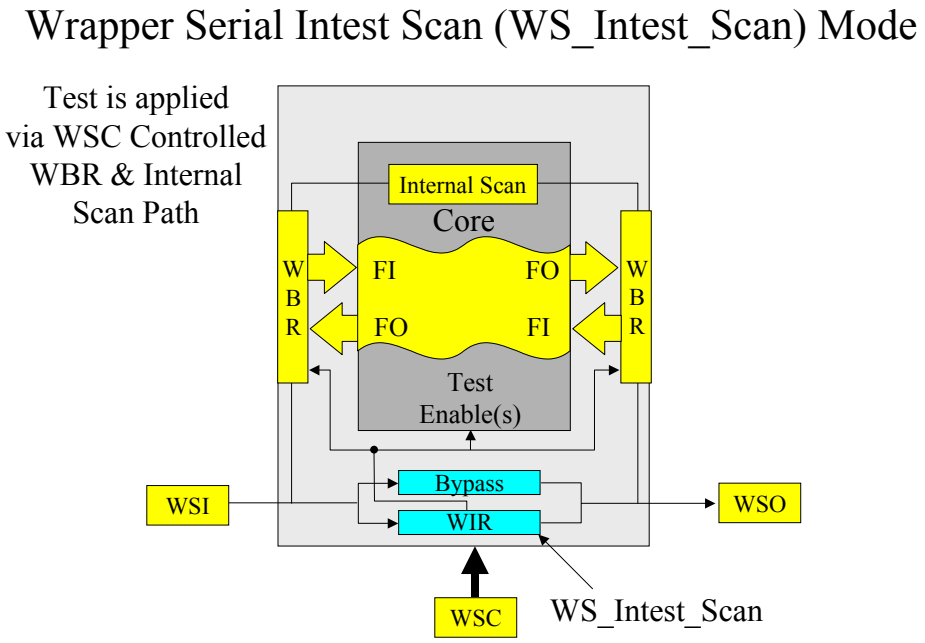


Figure 10—WS_INTEST_SCAN example

7.14 Wx_INTEST instruction

One core test instruction that allows the core to be tested according to a test procedure specified by the core provider or core user is required. IEEE Std 1500 does not describe how to test individual cores; this is the responsibility of the core provider. The core test invoked by the Wx_INTEST instruction (the x in Wx is a place holder for an S, P, or H to indicate whether the instruction is serial, parallel, or hybrid) is completely specified with the CTL provided for the core. The Wx_INTEST CTL describes the attributes of the core test such as the following:

a) Number of cell test input (CTI) and cell test output (CTO) terminals,

b) Test pattern set, including control and/or data, that is applied to the core via associated TI and TO terminals,

c) Protocol used to input and output the test patterns,

d) Condition or state of core input and output terminals during the core test, excluding the TI and TO terminals, and

e) Whether the WBR is used during the core test, and if used, in what configuration it is placed.

The Wx_INTEST instruction encompasses the WS_INTEST_RING and the WS_INTEST_SCAN instruction.

The rules in 7.14.1 apply for the Wx_INTEST instruction.

7.14.1 Specifications

Rules

a) Each wrapper shall provide at least one Wx_INTEST instruction.

b) While the Wx_INTEST instruction is selected, the WBR shall be in IF mode.

Recommendations

c) While the Wx_INTEST instruction is selected, the operation of the core should not disturb circuitry external to the core.

d) While the Wx_INTEST instruction is selected, all WFOs should be placed in a safe or static drive state.

e) While the Wx_INTEST instruction is selected, for core input terminals provisioned with wrapper boundary cells per rule 12.1.1(a), the data loaded during the Capture event should be independent of the operation of off-core circuitry or chip-level interconnections.

Permissions

f) The WBR may be segmented into one or more scan chains with the input and output port of the scan chains interfaced to the TAM.

g) The binary code for the Wx_INTEST instruction may be selected by the wrapper designer.

h) Any number of Wx_INTEST instructions (Wx_INTEST_user_defined)) may be included by the wrapper designer.

7.14.2 Description

The Wx_INTEST instruction allows testing of core circuitry.

While the Wx_INTEST instruction is selected, the WBR may participate and assume the role of virtual test points. Cells at nonclock core input terminals are used to apply the test stimulus, while those at core output terminals capture the response. Stimuli and responses are moved into and out of the circuit by shifting the

WBR. Note that this requires that the core input terminals provisioned with wrapper boundary cells per rule 12.1.1(a) are able to drive signals into the core circuitry.

Typically, the core circuitry will receive a sequence of one or more clock events between application of the stimulus and capture of the response. The specification of wrapper boundary cells for core clock input terminals allows the clocks for the core circuitry to be received in several ways while the Wx_INTEST instruction is selected. The following are offered as examples:

a) The signals received at core clock terminals can be fed directly to the core circuitry as during normal operation of the core.

b) Circuitry may be built into the core that allows the core circuitry to complete one step of operation on completion of shifting in data into the WBR. If the core were a microprocessor, it would be permitted to complete a single processing cycle by, for example, internal generation of a pulse on the hold signal. In this case, the clock(s) applied at the core clock terminal(s) during the test could be free-running.

While the Wx_INTEST instruction is selected, the state of all WFOs is determined by the wrapper circuitry. Every WFO should be forced to a static or inactive drive state. This ensures that surrounding circuitry on an SoC are supplied known, safe signal levels while the core circuitry test is in progress.

Recommendation 7.14.1(e), where followed, ensures that data shifted out of the core in response to the Wx_INTEST instruction are not altered by the presence of defects in off-core circuitry, chip-level interconnections, etc. This simplifies diagnosis, since any errors in the output bit stream can be caused only by defects in the core circuitry or in the WBR.

8. Wrapper serial port (WSP)

The WSP is a set of wrapper terminals that facilitate standard plug-and-play (PnP) operation of a core that implements the IEEE 1500 architecture. IEEE Std 1500 defines mandatory WSP terminals to control serial access to the WIR, the WBY, and the WBR.

The WSP includes the following mandatory terminals: WSI, WSO, and a set of WSC terminals. The WSC comprises the wrapper clock (WRCK), wrapper reset (WRSTN), SelectWIR, CaptureWR, ShiftWR, and UpdateWR terminals. If required for operation of the WBR, the optional TransferDR terminal is also included in the WSC. Some WBR implementations may also operate using one or more optional auxiliary clock (AUXCK) inputs, which are depicted as the AUXCKn terminal(s) in Figure 11. The AUXCKn terminal(s) are included in the WSC. The above standard WSP terminals are described in further detail in 8.1.

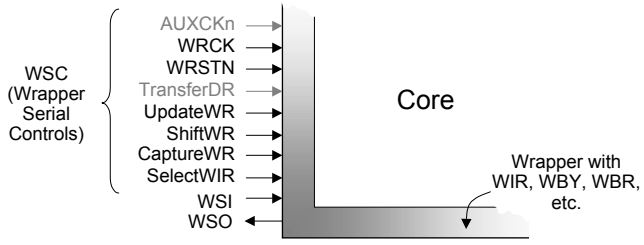


Figure 11—WSP

8.1 WSP terminals

This subclause defines the standard WSP terminals of the IEEE 1500 wrapper. Note that only the WSP's terminals are specified in this subclause, whereas the WSP timing and protocol, as used to operate the standard features of the IEEE 1500 architecture, are specified and described in subsequent clauses of this standard.

8.1.1 Specifications

Rules

a) The WSP is mandatory and shall include the following mandatory terminals: WSI, WSO, WRCK, WRSTN, SelectWIR, CaptureWR, ShiftWR, and UpdateWR.

b) All mandatory WSP terminals, and the optional TransferDR terminal, shall be dedicated wrapper terminals (i.e., they shall not be used for any other system function).

c) If user-defined AUXCK(s) are used, per permission (e), the timing relationship between WRCK and the AUXCK(s) shall be sufficiently specified and documented for use by the SoC integrator.

Permissions

d) The WSP may include the TransferDR terminal.

e) The WSP may include AUXCKn terminal(s) in addition to the dedicated WRCK for operation of registers other than the WBY and WIR. Such clocks may be shared with other system clocks and may be used to operate other core features.

f) The user may select a user-defined name for the AUXCK terminal(s).

8.1.2 Description

The mandatory WSP terminals are WSI, WSO, WRCK, WRSTN, SelectWIR, CaptureWR, ShiftWR, and UpdateWR. The signals connected to these WSP terminals are used to select whether the WIR or a WDR is connected between WSI and WSO and to select and operate either the WIR or a selected WDR between WSI and WSO. The WSP may also include the optional TransferDR and AUXCKn terminals.

The following definitions apply to the rules specified in 8.1.1.

WSI and **WSO:** These WSP terminals are used to scan in and scan out wrapper instructions and data.

WRCK: The signal connected to the WRCK terminal is a dedicated clock used to operate IEEE 1500 functions.

AUXCKn: AUXCKn are optional auxiliary clocks (AUXCK) used [per permission 8.1.1(e)] with some implementations of the WBR. These may also be shared with system clocks and must have a user-specified timing relationship to WRCK for operation of the WBR. It is anticipated that more than one AUXCK terminal could exist in an IEEE 1500 wrapper. In such a case, the integer n is used to differentiate these terminals (e.g., AUXCK1, AUXCK2). Subclause 16.1 provides further discussion on AUXCKs.

WRSTN: When asserted, WRSTN puts the wrapper into its normal system mode. The signal may be used to reset other WRs or wrapper circuitry as needed.

SelectWIR: SelectWIR determines what type of WR operation, i.e., instruction or data, is to be performed. The signal value selects between the WIR and the data registers, e.g., WBY or WBR. While SelectWIR is asserted to logic 1, the WIR is selected and connected between WSI and WSO. SelectWIR must be deasserted to logic 0 in order for any WDR or CDR to be selected and connected between WSI and WSO.

CaptureWR, ShiftWR, UpdateWR: These terminals control and enable WR operations. When one of these signals is asserted to logic 1 and the other two signals are deasserted to logic 0, a corresponding Capture, Shift, or Update operation will be enabled for the selected WR. While the WIR or WBY is selected, the enabled operation occurs synchronously to WRCK.

TransferDR: TransferDR is required when the WBR includes cells with a transfer capability.

The operation of the WIR, WBY, and WBR using the WSP is described in further detail in Clause 10, Clause 11, and Clause 12, respectively.

9. Wrapper parallel port (WPP)

In addition to the mandatory WSP, IEEE Std 1500 supports an optional WPP as depicted in Figure 1.

9.1 WPP terminals

WPP terminals are user-defined with the specifications in 9.1.1.

9.1.1 Specifications

Rules

a) If a user-defined WPP is used per permission (c), the timing relationships between its constituent signals shall be sufficiently specified and documented for use by the SoC integrator.

b) A WPP shall not include the ShiftWR, CaptureWR, UpdateWR, TransferDR, SelectWIR, WSI, or WSO terminals.

Permissions

c) The wrapper designer may define one or more dedicated WPP(s).

d) A WPP may include the WRCK and AUXCK terminals.

9.1.2 Description

It is permitted to have a parallel access mechanism for increased data bandwidth to the wrapped core. Figure 1, Figure 17, and Figure 20 show the WPP and its usage. A WPP port is distinct from the WSP, but may share the WRCK and AUXCKs with the WSP.

10. Wrapper instruction register (WIR)

This clause specifies and describes the WIR. The WIR is an instruction register in which IEEE 1500 wrapper instructions are serially loaded through the standard WSP. The WIR also comprises interface circuitry to other IEEE 1500 components, such as the WBR and WBY, and may also interface to the circuitry of the core.

10.1 WIR configuration and DR selection

Figure 12 illustrates the WIR within the IEEE 1500 architecture. This figure shows how the standard data registers (i.e., WBR and WBY) and other optional data registers (i.e., WDRs and CDRs) are configured for access via the WSP and how they are selected by the WIR. The WIR contains a shift stage, instruction

decode, and update stage. The WIR interfaces to the WSP, the IEEE 1500 wrapper, and conditionally the core per permission 10.2.1(h).

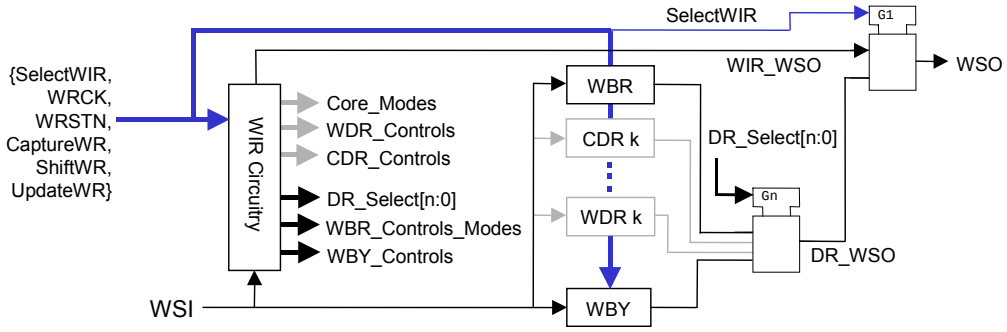


Figure 12—Example of WIR interface to WBY, WBR, and core

10.1.1 Specifications

Rules

a) The IEEE 1500 wrapper shall contain a single WIR.

b) The WIR shift register shall be selected and connected between WSI and WSO when SelectWIR is asserted to logic 1, regardless of the current wrapper instruction.

c) Selection of registers other than the WIR (i.e., WDRs or CDRs) between WSI and WSO shall be determined by the current wrapper instruction, and these registers are selected when SelectWIR is asserted to logic 0.

Permissions

d) Alternate mechanisms for access to WDRs or CDRs may be enabled by a user-defined wrapper instruction.

10.1.2 Description

The WIR is unconditionally accessible by the WSP by asserting SelectWIR. SelectWIR always takes precedence in selecting the WIR, so that when it is logic 1, the WIR is connected between WSI and WSO and is enabled to shift, update, or capture using the WSP. Thus, access to the WIR for loading wrapper instructions can always be realized simply by asserting SelectWIR to logic 1, regardless of the current wrapper instruction and the selected WDR or CDR.

The access of any WDR or CDR through the WSP requires that SelectWIR be logic 0. The register accessed through WSI and WSO is then determined based on the wrapper instruction previously updated in the WIR.

User-defined wrapper instructions may enable an optional user-defined parallel port. Such a user-defined parallel interface may encompass an alternative selection mechanism for access to WDRs and/or CDRs. Permission 10.1.1(d) establishes that control of such alternate access mechanisms originates and terminates exclusively from the WIR.

10.2 WIR design

A block diagram of a WIR design is depicted in Figure 13. The WIR is controlled and clocked by the standard WSC terminals.

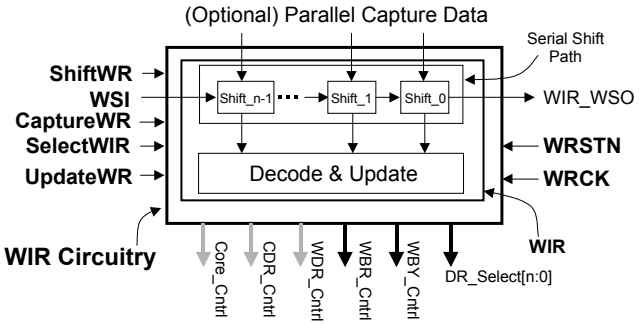


Figure 13—WIR circuitry design

The design of the WIR and its associated interface circuitry are shown in Figure 12 and Figure 13.

Figure 12 shows an example of the interface signals generated by the WIR circuitry. This interface circuitry must be designed to provide the necessary control signals for standard data register selection and operation and to control the standard wrapper modes. The WIR circuitry may be optionally extended to generate controls for the selection and operation of other data registers, for the enabling of a WPP, or for any other core modes that may be needed.

Control signals generated by the WIR circuitry are derived from the current wrapper instruction and the states of the signals connected to the WSC terminals. For the standard IEEE 1500 data registers (i.e., the WBR and the WBY), signals generated by the WIR circuitry will include controls for connecting the WBR or WBY between the WSI and WSO terminals. For the WBY, the WIR circuitry must generate controls for its Shift operation, e.g., a ShiftWBY shift enable signal. For standard serial instructions, the WIR circuitry must also generate control signals for operation of the WBR and its wrapper modes. For example, the WIR would generate wrapper modes for multiplexing the WBR cells between the test and normal modes of the wrapper.

Figure 13 shows a block diagram of the WIR circuitry. The WIR is controlled and clocked by the standard WSC terminals, as shown in the figure. Circuitry in the WIR must generate the necessary controls for its operation from the signals connected to the standard WSC terminals. For example, a ShiftWIR enable signal may be derived from the signals connected to the SelectWIR and ShiftWR terminal. The specific circuitry required for WIR operation is dependent on the design implementation of the WIR and is not shown in Figure 13.

As illustrated in Figure 13, the WIR circuitry includes an *n*-bit serial shift stage, logic for (optionally) decoding the shift stage contents, and circuitry for updating the wrapper instruction that was loaded into the WIR. In order to accommodate the mandatory wrapper instructions, the WIR serial shift stage must be at least 2 bits long. The wrapper instruction opcode is serially shifted into the *n*-bit shift stage and may be encoded, partially encoded, or unencoded. Thus, logic for decoding the wrapper instruction is optional, depending on implementation. The update stage latches the wrapper instruction that was loaded into the WIR to ensure that the current (i.e., previously updated) instruction does not change during subsequent WIR Shift operations.

The WIR may optionally include parallel capture inputs, as shown in Figure 13. This permits the WIR to capture test control information or to capture data that can be used for testing the WIR, or other IEEE 1500 circuitry. Further specification and description of the WIR design are given in 10.2.1 and 10.2.2.

10.2.1 Specifications

Rules

a) The WIR shall provide signals for selecting the WBR or WBY between WSI and WSO as required by the wrapper instructions defined in this standard.

b) The WIR shall provide signals to enable and control the operational modes of the WBR, as defined by this standard.

c) The WIR shall be dedicated IEEE 1500 logic.

d) The shift path through the WIR shall include at least 2 bits.

e) There shall be no inversion of logic values from WSI to WSO for WIR Shift operations.

f) The WIR shall be designed so that data shifted into the WIR shift register do not affect the currently active wrapper and core modes until a WIR Update operation occurs.

g) Where optional parallel inputs are provided to the WIR, the data at the parallel inputs shall be captured into the WIR shift register upon execution of a WIR Capture operation.

h) Where cores are provided with test mode enable input terminals, these terminals shall be directly controlled by the WIR circuitry.

Recommendations

i) When parallel capture inputs are provided to the WIR, any shift register stages that are unused for capture should be designed to always capture a fixed binary logic value (i.e., either a 0 or a 1) when a WIR Capture operation occurs. The fixed binary code captured may be determined by the designer.

j) Operation of the WRSTN to reset the wrapper should load fixed logic values into the WIR shift path.

Permissions

k) Modes of user-defined WDRs may be enabled and controlled by the current instruction in the WIR.

l) Core system operation modes may be enabled and controlled by the current instruction in the WIR.

m) Signals applied to the terminals of the WSC may be used to directly control and/or clock WDRs, CDRs, or core test functions without going through the WIR.

n) The WIR may capture data values during WIR Capture operations.

o) Control signals provided by the WIR may be output to IEEE 1500 wrapper terminals in order to facilitate external circuitry that may be required for PnP operation of the WBR or other data registers.

10.2.2 Description

The WIR circuitry as shown in Figure 12 and Figure 13 is loaded and updated with a wrapper instruction via the WSP terminals. Based on the protocol applied to the signals connected to the WSC terminals, the WIR circuitry generates the necessary control signals for shifting and updating wrapper instructions or the data register specified by the current wrapper instruction. The WIR circuitry also generates the necessary signals for controlling any wrapper and core modes associated with the instruction.

Interface signals generated by the WIR circuitry for the standard IEEE 1500 instructions must include controls for the following:

— Selection of which data register is to be connected between WSI and WSO. This includes controls to select between the WBY and the WBR.

— Control signals for the WBY Shift operation and the optional WBY Capture operation.

— Control signals for the wrapper modes of the WBR cells.

The WIR circuitry may optionally generate other control signals, such as signals to configure core test modes, controls to select and enable CDRs and/or WDRs, or controls to select and enable the optional WPP and its associated data registers.

Control signals generated by the WIR are static control signals, i.e., signals that change relatively infrequently. Depending on the implementation of the WBR or other data registers, WSC signals may interface directly to the data registers or the core. For example, the WRCK may directly clock the WBR cells. The state of the control signals output from the WIR circuitry, as shown in Figure 12, is generated based on the state of the WSC terminals and the currently active wrapper instruction in the WIR.

As an example, consider the standard WS_EXTEST instruction. When this instruction is loaded into the WIR, the WBR must be accessed between the WSI and WSO terminals of the wrapper, and the WBR Cells are put into their OF wrapper mode. Similarly, when the WS_BYPASS instruction is loaded in the WIR, the WBY is to be accessed between WSI and WSO, and the WBR must be put into a state so that it allows normal operation of the core. Figure 12 shows a set of control signals called DR_Select[n:0], as being generated by the WIR circuitry. These controls are then connected to the DR_WSO multiplexer. In this example of WIR implementation, the DR_Select[n:0] controls are used for selecting the data register to be connected between WSI and WSO. The DR_Select[n:0] signals select among the WBY, the WBR, and any number of other optional registers, per the currently active wrapper instruction. The scan output of the selected data register is then output onto the DR_WSO output of the multiplexer, and the DR_WSO multiplexer output would then be selected at the wrapper's WSO multiplexer by the SelectWIR signal. The WIR circuitry must also generate signals to control WBY operations, e.g., a ShiftWBY signal to enable shifting the WBY.

Figure 13 shows optional parallel inputs to the WIR. These inputs may be provided to permit data to be captured into the WIR. These capture data may be used for test control, testing of the WIR circuitry, or testing of other IEEE 1500 circuitry.

When optional parallel capture inputs are provided to the WIR, it is recommended that any unused capture inputs be designed to capture fixed binary logic values. This is intended to prevent the capture of unknown data into any unused bits, which may then need to be masked when examining the captured data that were scanned out of the WIR. In addition, the designer may consider choosing fixed logic values that will aid in testing and diagnosis of the WIR logic and SoC scan path integrity. However, care should be taken in the values that are captured into the WIR, and the operation of the WIR, so that unknown or harmful instructions are not inadvertently updated in the WIR. This can be avoided after a WIR Capture operation, by first shifting in a known instruction, prior to performing a WIR Update operation.

10.3 WIR operation

Operation of the WIR is controlled by the signals connected to the WSC terminals of the IEEE 1500 wrapper. Figure 14 shows a timing diagram of WIR Shift operation followed by a WIR Update operation. Standard WIR operations are specified and described in 10.3.1 and 10.3.2.

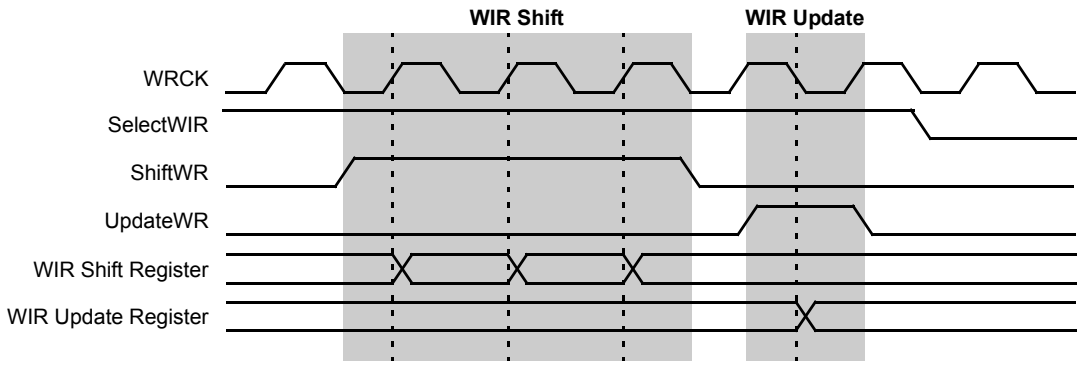


Figure 14—Timing for WIR Shift operation followed by WIR Update operation

10.3.1 Specifications

Rules

a) The behavior and operation of the WIR shall be defined by WSP protocol applied to the signals connected to the WSC terminals, as follows:

   1) When a WRSTN occurs, the WIR update register shall be set to WS_BYPASS, and the WBR shall be set to normal mode.

   2) When a WIR Shift operation occurs, the WIR shift register shall shift instruction data from WSI to WSO.

   3) When a WIR Update operation occurs, the wrapper instruction shall be updated from the WIR shift register into the WIR update register, and updated wrapper and core modes shall be set.

   4) When the optional WIR Capture operation occurs, the WIR shift register shall be loaded with parallel capture data.

   5) During all other WSP operations, the WIR shift and update registers shall retain their current state or instruction.

b) The WIR and its associated circuitry shall be clocked and controlled only with the dedicated WSC signals, as defined by this standard.

c) A single WSP protocol (i.e., WRSTN, WIR shift, WIR update, or WIR capture) shall be applied, as defined by this standard, to operate the WIR so that only one WIR operation is enabled for the given protocol.

d) The current wrapper modes and core modes shall remain active until either a WRSTN or WIR Update operation occurs, which shall cause the specified wrapper instruction and modes to take effect and become the currently active instruction and modes.

   NOTE—The previous instruction and modes shall become inactive subsequent to the new Instruction and Modes becoming active.

e) The WIR circuitry shall retain its current state (i.e., shift stage values and currently active modes) indefinitely while the WRCK signal is stopped (i.e., WRCK held at a fixed logic value of 1 or 0) and the signal connected to the WRSTN terminal is logic 1.

f) The WIR circuitry shall retain its current state while there are no active WIR operations in progress.

g) A WRSTN shall occur when the signal connected to the WRSTN terminal transitions to logic 0 and shall cause the wrapper modes and core modes to be set to normal system mode.

h) When the signals connected to the WSC's SelectWIR and ShiftWR terminals are logic 1, a WIR Shift operation shall occur on the next rising edge of WRCK and shall cause instruction data to shift through the WIR, from WSI to WSO. The shift stage data shall shift 1 bit toward WSO for each

rising edge of WRCK of the WIR Shift operation. SelectWIR and ShiftWR shall remain logic 1 during the WIR Shift operation.

i) During a WIR Shift operation, WSI shall be sampled on the rising edge of WRCK and WSO shall change only on the falling edge of WRCK.

j) When the signals connected to the WSC's SelectWIR and UpdateWR terminal are logic 1, a WIR Update operation shall occur on the next falling edge of WRCK and shall cause the wrapper modes and core modes, as determined by the wrapper instruction in the WIR serial shift stage, to take effect.

k) When the signals connected to the WSC's SelectWIR and CaptureWR terminals are logic 1, a WIR Capture operation, if implemented, shall occur on the next rising edge of WRCK and shall cause the data at the parallel inputs of the WIR to be latched into the serial shift stage of the WIR.

NOTE—For all WIR operations the appropriate WSC terminals must be at valid logic levels prior to the corresponding clock edge of WRCK specified for the operation, and they must meet the setup and hold timing specifications of the design. Per rule (c), the signals connected to the WSC terminals that are not specified in the protocol shall be de-asserted in the given protocol.

Recommendations

l) WRSTN should not be used to initialize any other system logic within the core.

m) When a WIR Capture operation has been performed, a WIR Shift operation should be performed prior to a WIR Update operation.

NOTE—This will prevent an unintended wrapper instruction (i.e., based on the WIR's capture data) from being updated in the WIR. Either a WIR Shift operation should always precede a WIR Update operation, or the designer should implement the WIR so that unintended wrapper instructions are not updated in the WIR.

n) When a WRSTN occurs, the WIR shift register should be set to WS_BYPASS.

Permissions

o) The sequences and number of WIR operations may occur in any order.

10.3.2 Description

Protocol applied to the WSP terminals controls three mandatory operations of the WIR (i.e., WRSTN, WIR Shift, and WIR Update) and an optional operation (i.e., WIR Capture). For all other data register operations, the WIR is required to retain the wrapper instruction last updated in the WIR. This guarantees that any wrapper modes or core modes controlled by the instruction will not change until a new wrapper instruction is updated in the WIR. Further, when the WRCK is stopped, the WIR must retain its current state; thus the WRCK is not required to be free-running.

The WIR circuitry outputs WS_BYPASS when the WSC's WRSTN terminal transitions to logic 0. This causes the mode of the core to be set to normal system operation.

A wrapper instruction is loaded into the WIR by asserting the signals connected to the SelectWIR and ShiftWR terminals to logic 1 and then shifting in the instruction opcode on the WSI terminal with WRCK. The WSI data will be sampled on the rising edge of WRCK, and the WIR will shift 1 bit of data, towards WSO, for each rising edge of WRCK. WSO will change on the falling edge of WRCK. Once the proper number of wrapper instruction bits has been shifted into the serial shift stage of the WIR, the ShiftWR signal must be deasserted before the next rising edge of WRCK. Following this, the signal connected to the UpdateWR terminal is asserted to logic 1, and the WIR will then update on the next falling edge of WRCK. A timing diagram with a WIR Shift operation, followed by a WIR Update operation, is shown in Figure 14. This falling-edge behavior was adopted in order to prevent race conditions between shift and update and as an alternative to additional protocol restrictions. This falling-edge behavior also improves PnP.

When a wrapper instruction is updated in the WIR, the corresponding data register is enabled to be selected, and the appropriate wrapper modes and core modes are set. Once the new instructions and modes are active, deasserting the SelectWIR signal selects operation of the data register specified by the wrapper instruction.

11. Wrapper bypass register (WBY)

This clause specifies and describes the WBY.

11.1 WBY register configuration and selection

Figure 12 illustrates how the WBY fits into the IEEE 1500 WR configuration and is connected between the WSI and WSO of the WSP. The WBY is the WDR selected when the WIR contains the WS_BYPASS instruction. Once selected, the WBY is connected between the WSI and WSO terminals of the WSP and can be shifted using protocol applied to the signals of the WSC terminals.

11.1.1 Specifications

Rules

a) The IEEE 1500 wrapper shall contain a single WBY.

b) The WBY shall be selected for access between WSI and WSO when the currently active wrapper instruction is WS_BYPASS and the signal connected to the WSC's SelectWIR terminal is logic 0.

c) The WBY shall be selected for access between WSI and WSO when the currently active wrapper instruction is WS_CLAMP or WS_SAFE, if support for those instructions is provided in the wrapper.

Recommendations

d) The WBY should be selected for all unused wrapper instruction opcodes or for when a wrapper instruction does not require a WDR for its operation.

Permissions

e) The WBY may be selected by wrapper instructions other than the WS_BYPASS instruction.

11.1.2 Description

The WBY is a mandatory WDR and must always be selected by the WS_BYPASS instruction. The WBY may also be selected by other wrapper instructions and should be considered as the default data register (i.e., for unused wrapper instruction opcodes or for when the instruction does not require a specific data register) selected between the WSI and WSO terminals of the WSP. This will ensure that there is always a default WDR connected between the WSI and WSO terminals so that the scan path for the WSP is never undefined. Once selected, the WBY can be utilized when the signal connected to the WSC's SelectWIR terminal is deasserted to logic 0.

11.2 WBY design

A block diagram for the design of the WBY is shown in Figure 15. The WBY is an n-bit serial shift register. It can optionally capture parallel data into its shift register stage and does not have an update stage. Further specification and description of the WBY design are given in 11.2.1 and 11.2.2.

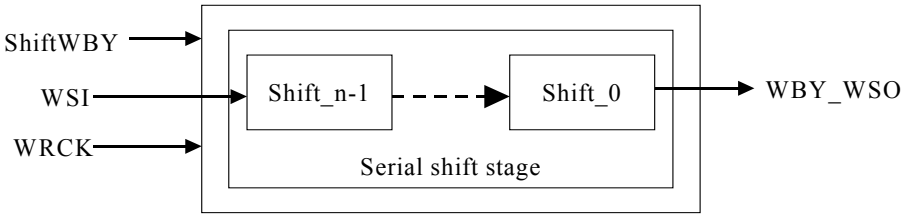


Figure 15—WBY

11.2.1 Specifications

Rules

a) The circuitry that forms the WBY shall be dedicated IEEE 1500 logic (i.e., the circuitry shall not be shared with, or used to perform, system functions).

b) The WBY shall include at least 1 bit of serial shift data.

c) There shall be no inversion of logic values between WSI and WSO for WBY Shift operations.

Recommendations

d) The serial shift stage of the WBY should be kept to a minimal length in order to provide the shortest possible bypass path through the WSI-WSO terminals of the WSP. A 1 bit WBY is the preferred length.

Permissions

e) Parallel input(s) to the WBY that permit capture of data value(s) during WBY Capture operations may be provided.

11.2.2 Description

The WBY provides an *n*-bit bypass of the WSP. In most IEEE 1500 wrapper implementations, the WBY will be a single bit in length; however, it is permitted to be multiple bits in length in order to facilitate connection of the WSP at the system chip level (see Clause 15). In the case where more than a single bit WBY is implemented, it is recommended that the WBY length be kept as short as possible.

The WBY interfaces to the WSC terminals and to the controls generated by the WIR circuitry. Particular controls signals for the WBY circuitry will depend on the actual implementation of the WBY.

11.3 WBY operation

Operation of the WBY register involves first loading a wrapper instruction to select the WBY and then applying the proper protocol to the WSC terminals in order to shift the WBY or optionally capture parallel data into its shift stage. Detailed specifications and description on operation of the WBY are given in 11.3.1 and 11.3.2.

11.3.1 Specifications

Rules

a) The operation of the WBY shall be defined by protocol applied to the dedicated WSP signals (e.g., WRCK, ShiftWR).

b) The WBY shall retain its current shift stage value indefinitely while the WRCK signal is stopped (i.e., WRCK held at a fixed logic value of 1 or 0).

c) The WBY shall retain its current shift stage value while there are no active WBY operations in progress.

d) While the WBY is selected and when the signals connected to the WSC's SelectWIR and ShiftWR terminals are logic 0 and 1, respectively, a WBY Shift operation shall occur on the next rising edge of WRCK.

e) A WBY Shift operation shall cause data to shift through the WBY, from WSI to WSO, and the shift stage data shall shift 1 bit toward WSO for each rising edge of WRCK.

f) During a WBY Shift operation, WSI shall be sampled on the rising edge of WRCK and WSO shall change only on the falling edge of WRCK.

NOTE—For all WBY operations, the appropriate WSP terminals shall be at valid logic levels prior to the corresponding clock edge of WRCK specified for the operation, and they shall meet the setup and hold timing specifications of the design.

11.3.2 Description

The operation of the WBY is controlled and clocked with protocols applied to the WSP signals. The WBY interfaces to the WIR circuitry, where clock and control inputs to the WBY from the WIR will depend on the actual WBY implementation. For example, in the WBY register illustrated in Figure 15, a ShiftWBY control input is generated by the WIR circuitry and the WBY receives the WRCK and WSI signals directly from the WSP. The WBY-WSO output of the WBY is then connected to the DR_WSO multiplexer in Figure 12 and will be output on the DR_WSO signal when the WBY is selected as the data register.

12. Wrapper boundary register (WBR)

IEEE Std 1500 mandates a serial interface and supports a parallel interface for accessing a wrapped core. The serial interface is characterized by a single chain composed of all WBR cells, whereas the parallel interface is characterized by one or multiple IEEE 1500 wrapper cell chains surrounding a core. Figure 16 shows the WBR in an IEEE 1500 serial interface architecture. The WRSTN shown in this figure may optionally be used to reset the WBR cells to a known state.

The WBR is constructed of WBR cells, each of which (except for those permitted by permission 12.2.1(g)) has four data terminals: cell functional input (CFI), cell functional output (CFO), cell test input (CTI), and cell test output (CTO).

Figure 16 shows the following:

a) Wrapper functional inputs (WFIs) are connected to the CFI terminals of WBR cells provided for core input terminals.

b) CFO terminals of WBR cells provided for core input terminals are connected to the corresponding core input terminals.

c) Core output terminals are connected to the CFI terminals of corresponding WBR cells provided for those core output terminals.

d) The CFO terminals of cells provided for core output terminals are connected to corresponding WFOs.

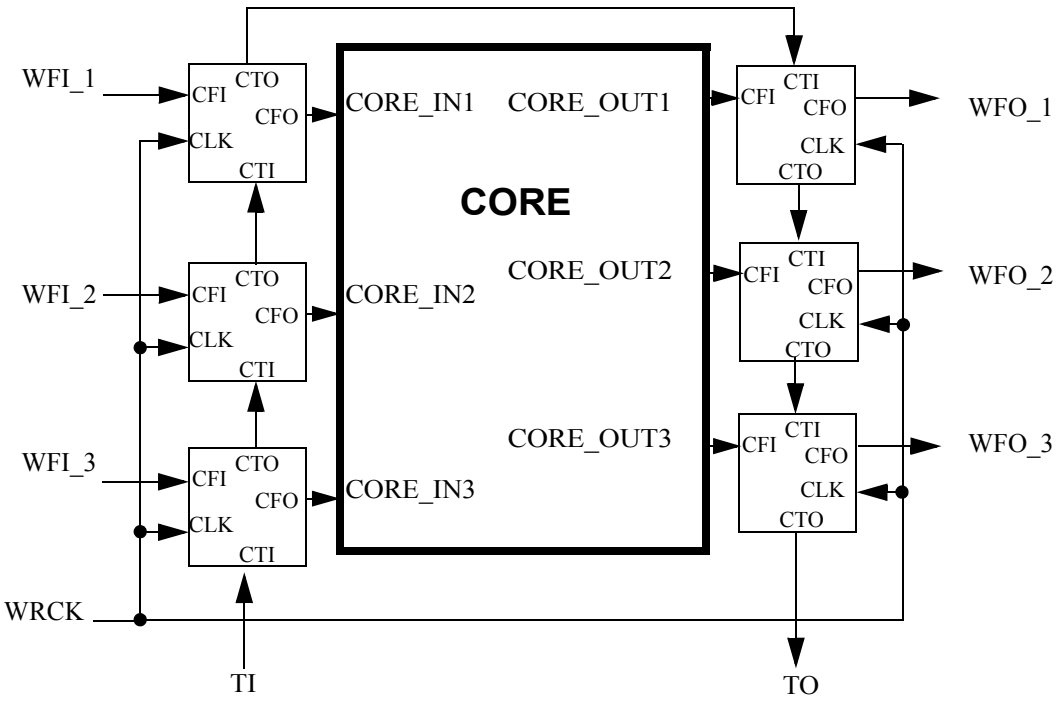


Figure 16—Example of WBR in serial interface configuration to be accessed from WSP

WFI and WFO are generic names for wrapped core terminals that are intended to be connected within an SoC to integrate the wrapped core within the SoC. It is often the case that identical WBR cells are used for wrapping both core input terminals and core output terminals. These WBR cells are connected differently and may receive different control signals (not shown). The WBR shift path is formed by connecting WBR cells CTO to CTI. TI and TO are defined to be the scan input and scan output terminals of the WBR as a whole.

In Figure 16 and Figure 17, the WFI_k terminals are WFI terminals, while the WFO_k terminals are WFO terminals. In the case where an AUXCK is used to operate the WBR, Figure 40 shows an example timing relationship between WRCK and the AUXCK.

In Figure 17, a parallel interface implementation using two scan chains is depicted. Although this figure shows all inputs placed on one scan chain and all outputs on a second scan chain, the IEEE 1500 parallel interface is not limited to this configuration. In addition, although WRCK is shown in both Figure 16 and Figure 17, IEEE Std 1500 does not mandate the use of WRCK for clocking the wrapper cells. A different clock, i.e., AUXCK, may be used although the WRCK is required at the wrapper level.

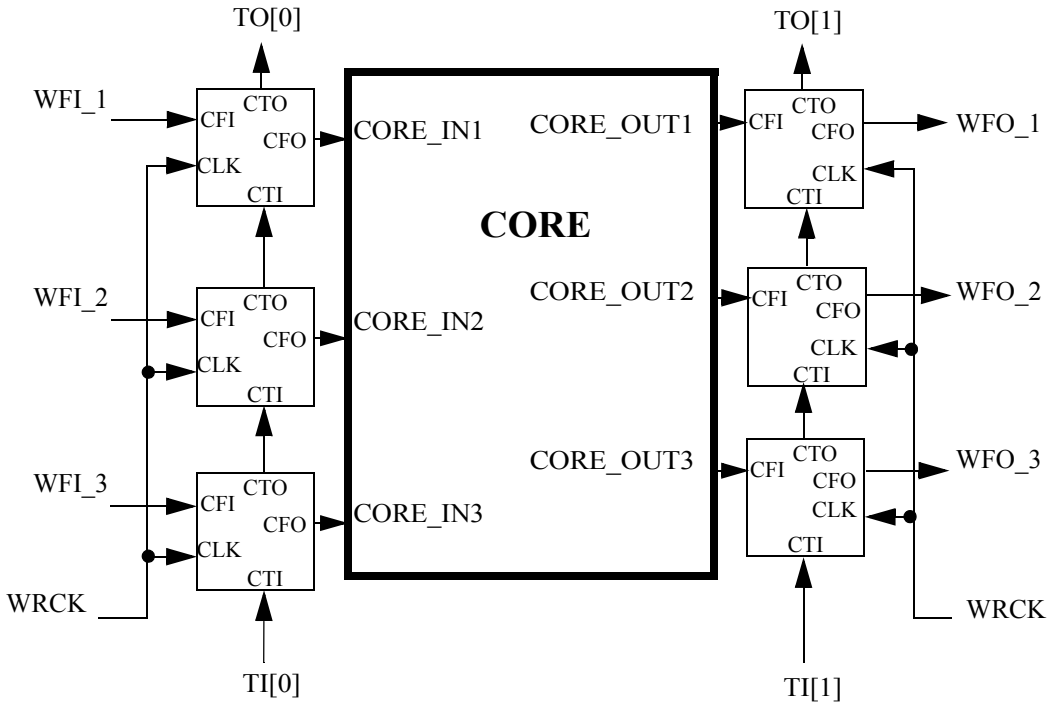


Figure 17—Example of WBR in parallel interface configuration to be accessed from WPP

12.1 WBR structure and operation

12.1.1 Specifications

Rules

a) Every core signal terminal that is digital, except for signals that cause data to be loaded into a sequential element (e.g., the clock of a flip-flop, the gate of a latch, or the asynchronous set or reset of either a flip-flop or a latch) or dedicated test signal, shall be provisioned with a WBR cell.

b) The WBR shall have at least one configuration in response to the state of the WIR, allowing serial access to and from all WBR cells between TI and TO.

c) Every wrapped terminal shall be uniquely associated with at least one WBR cell, except as exempted under permission 12.1.1(h).

d) The WBR must respond to shift, capture, and apply as defined in 12.3.1, except as exempted under permission 12.2.1(g).

e) Core terminals that have been provided for the purpose of statically enabling one or more core test modes shall be provisioned with a WIR control.

Recommendations

f) Core terminals exempted from wrapper cell insertion per rule 12.1.1(a) should be provided with direct access from the SoC to their corresponding wrapped core terminals, except where doing so would conflict with rule 12.1.1(e).

Permissions

g) Any core terminal may be provisioned with a WBR cell, provided doing so does not alter the behavior of standard instructions.

h) In a case in which a core input terminal is used solely as the source of data for an output terminal, a single WBR cell may be shared between the input and the output terminals.

i) A core input terminal may be provisioned with more than one WBR cell.

12.1.2 Description

For the IEEE 1500 serial interface, the WBR should have a single, uniform shift path between its TI and TO. WBR structure rules 12.2.1(a) and 12.2.1(b) support this. A shift path in a WBR cell is composed of one or more storage elements serially connected between CTI and CTO. Likewise, for the serial mode of the IEEE 1500 wrapper, a shift path is the same storage elements of all the cells in the WBR concatenated into a single serial path.

In the case where core terminals are exempted from wrapper cell insertion, the wrapped core patterns will likely presume that a direct access is provided from the SoC to these terminals except where prohibited by rule 12.1.1(e) since those terminals are controlled by the WIR. The SoC integrator has to take this into account for proper operation at the SoC level.

12.2 WBR cell structure and operation

The WBR cell operation relies on the Shift, Capture, Update, Transfer, and Apply events more fully defined in 12.3.1.

12.2.1 Specifications

Rules

a) Every WBR cell shall have at least one storage element connected between its CTI and CTO terminals.

b) Every WBR cell required per rule 12.1.1(a) shall have a storage element provisioned for the purpose of servicing the Capture event and this element shall be the shift path storage element closest to CTI, the shift path storage element closest to CTO, or the optional update storage element, if it exists.

c) Provided that the update storage element is provisioned for servicing the Capture event, the WBR cell shall support the Transfer event.

d) Provided that the WBR cell's shift path is composed of more than one storage element, the WBR cell shall support the Transfer event.

e) Storage elements in the shift path of a WBR cell shall not respond to the Update event.

Permissions

f) Any WBR cell may have a storage element provisioned for servicing the Update event.

g) For core terminals exempted from being wrapped per rule 12.1.1(a), reduced-functionality WBR cells may be provisioned. When these cells are provided, they must support the Shift event and at least one of the following:

   1) The Capture event to observe the CFI terminal or the CFO terminal

   2) The Apply event to control the CFO terminal

12.2.2 Description

WBR cell structure rules 12.2.1(a) and 12.2.1(b) provide for a minimal implementation consisting of only a means to select test data or functional data as input to a shared storage element.

WBR cell structure rule 12.2.1(a) allows for multiple storage elements in the shift path in order to support test methodologies requiring the application of sequential patterns (e.g., path-delay, transition delay, piece-wise functional).

Captured data may enter the shift path of a cell at the storage element closest to the CTI or CTO of this cell. However, in order to prevent captured data from overwriting other test data during sequential tests, captured data should ultimately move into the shift path of a cell via either the Capture event or Transfer event. WBR cell structure rules 12.2.1(c) and 12.2.1(d) support this.

This standard anticipates test methodologies requiring the application of functional timing in test mode. For example, permission 12.2.1(f) and rule 12.2.1(b) allow the Update element to be shared with normal operation and also service the Capture event.

Unidirectional IEEE 1149.1 boundary-scan cells are usable as IEEE 1500 WBR cells. WBR cell structure permission 12.2.1(f) supports this.

The WBR cell is fully self-sufficient for the processing of a test so that, after shifting in test data and before shifting out test data, all data for the test are sourced from and/or sampled into one or more storage elements in the single cell, i.e., the terminals' test needs are met solely by their respective WBR cells.

Cells provisioned on core terminals that need not be wrapped per rule 12.1.1(a) may have reduced-functionality cells as described in permission 12.2.1(g). For instance, one may wish to have an observe-only cell on clocks as defined in rule 12.1.1(a) or other types of observe cells supporting the Shift, the Capture, and perhaps the Transfer event in OF mode. If the core terminal provided with a reduced-functionality cell is a clock terminal and gets connected to the clock input of the reduced-functionality cell, then this clock would be considered an AUXCK and would have to follow AUXCK requirements and allow for proper operation of the WBR.

12.3 WBR operation events

An event is an uninterrupted, predefined sequence of one or more steps. Within the interval of the predefined sequence there may be a particular instant that characterizes the event, when the nominal action of the event occurs. This is referred to as the characteristic instant. Predefined events are Shift, Update, Transfer, Capture, and Apply. Some events may overlap with others as indicated in permission 12.3.1(e). Events apply to the WBR as a whole.

12.3.1 Specifications

12.3.1.1 Definitions

The following definitions apply to the rules specified in 12.2.1.

Shift event: An event whereby the data stored in the WBR shift path are advanced one storage position closer to the WBR's TO. The data present at the WBR's TI are loaded into the shift path storage element closest to the WBR's TI. If multiple WBR segments are implemented per Figure 17, this definition applies individually to each WBR segment.

Capture event: An event whereby the value present on the CFI, or on the CFO, at a characteristic instant is stored in a sequential element within a WBR cell. The data shall be stored in the shift path storage element closest to CTI, or in the shift path storage element closest to CTO, or in the off-shift-path update storage element.

Update event: An optional event whereby data stored in a WBR cell's shift path storage element closest to CTO at a characteristic instant are loaded into an off-shift-path storage element.

Transfer event: An optional event that moves data to or within the shift path of a WBR cell dependent on both or either

a) The case where data stored by the Capture event are stored in any storage element other than the shift path storage element closest to CTI (see rule 12.2.1(b)): The Transfer event will cause the value present in the storage element used for provisioning the Capture event to be stored into the shift path storage element closest to CTI;

b) The case where more than one storage element exists in the shift path: The Transfer event will cause data stored in the shift path to move one element closer to CTO.

Figure 18 illustrates Case a, and Figure 19 illustrates Case b. The legend of these figures is in Annex A.

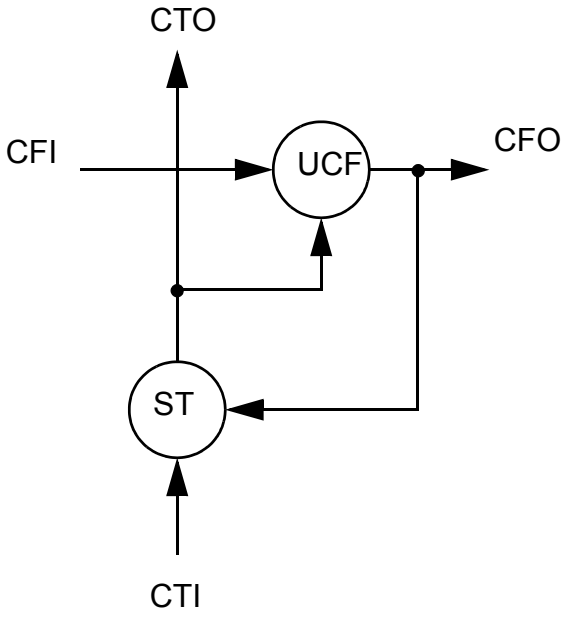


Figure 18—WBR cell supporting Transfer event with off-shift-path capture storage element

The purposes of the Transfer event are to properly position captured data in the shift path for observation and to provide sequential stimuli data. In order to preserve as many capture values as there are storage elements in the shift path, captured data should enter a WBR cell's shift path via the storage element closest to CTI (see rule 12.2.1(b)). To support this, the Transfer event causes the captured data to be transferred from the storage element where the Capture event occurred into the shift path storage element closest to CTI. Certain tests require application of sequential data. Transfer moves data through the shift path to make them available for the Apply event at the cell's CFO. In the case where the update storage element is present, transfer will move data through the shift path so that each bit may be sequentially loaded into the update storage element.

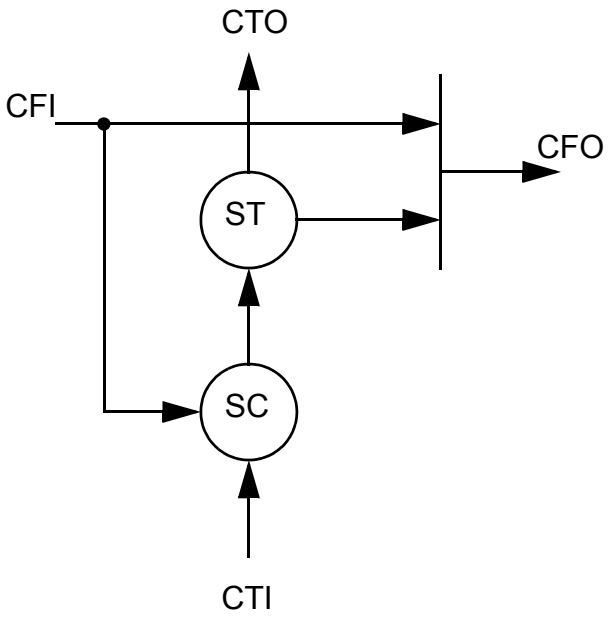


Figure 19—WBR cell supporting the Transfer event with multiple shift path storage elements

Apply: A derivative event (in the sense that it is inferred from the operation of the other four events: Shift, Update, Transfer, and Capture), whereby test data become active and effective as test stimuli. While the wrapper is in IF mode, the Apply event causes test data to be applied from input cells onto core inputs. While the wrapper is in OF mode, the Apply event causes test data to be applied from output cells onto WBR functional outputs. The test data are the data stored in the shift path storage element closest to CTO unless the Update event is supported, in which case the test data shall be the data stored in the off-shift-path storage element described in the Update event. The Apply event is only meaningful in the context of the WBR as a whole and not on a cell-by-cell basis.

Rules

a) The Shift event shall not occur simultaneously with either Transfer, Capture, or Update events.

b) While the SelectWIR signal is low and in the absence of an active value on other WSC or any WPC terminal (as defined in the CTL), WRCK shall not cause a WBR storage element to change state.

c) For serial instructions, in the absence of an active edge on WRCK or pulse on WRSTN, WBR storage elements shall not change state.

Recommendations

d) While the SelectWIR signal is high and in the presence of active values on WSC terminals, with the exception of UpdateWR, WRCK should not cause a WBR storage element to change state.

Permissions

e) Except where excluded by rule (a), events may be discrete, simultaneous, or overlapping.

f) WBR cells may respond to user-defined events, provided that these events are documented and distinguished in their behavior from standard IEEE 1500 events.

12.3.2 Description

WBR events are enabled by the active edge of the clock (rising edge of WRCK for Capture, Shift, and Transfer; falling edge of WRCK for Update) in conjunction with the assertion of a WSC signal (CaptureWR, ShiftWR, TransferDR, or UpdateWR). WRCK pulses or edges applied in the absence of active WSC signals do not result in any event. WRCK may be held in a high or low state for arbitrarily long intervals without loss of WBR state. PnP requirements may restrict the degree to which events may be overlapped. Subclause 16.3 discusses such restriction. Recommendation 12.3.2(d) allows the WBR to maintain its state for subsequent instructions that may require the current WBR data.

12.4 WBR operation modes

A mode is a static condition or configuration of the WBR that exists in response to the state of the WIR. This subclause describes the normal mode, the IF mode, the OF mode, and the nonhazardous mode.

General permissions

a) IF mode and OF mode may be operative at the same time.

b) Whereas the four modes defined in this subclause (i.e., normal, IF, OF, and nonhazardous) are applied homogeneously across the entire WBR, other modes may be defined in which the cells respond on an individual basis.

12.4.1 Normal mode

The normal mode is the mode in which the WBR does not interfere with the functional operation of the core.

12.4.1.1 Specifications

Rules

a) While in normal mode, the WBR shall not interfere with the operation of the core or with the flow of signals to and from the core.

Permissions

b) While in normal mode, the WBR may respond to the Capture, Shift, or Transfer events, provided that such response does not conflict with rule (a).

12.4.2 Inward facing (IF) mode

The IF mode is a test mode where core inputs are controlled by the WBR and core outputs are observed by the WBR.

12.4.2.1 Specifications

Rules

a) While in IF mode, cells provided for core inputs shall respond to Shift, Apply, and, if provisioned for them (under rule 12.2.1(d) or permission 12.2.1(f)), the Transfer or Update events, respectively.

b) While in IF mode, cells provided for core outputs shall respond to Capture, Shift, and, if provisioned for them (under rule 12.2.1(c) or rule 12.2.1(d)), Transfer events.

Recommendations

c) While in IF mode, cells provided for core outputs should present safe data at their CFO terminals (i.e., external safe state).

12.4.3 Outward facing (OF) mode

The OF mode is a test mode where WFOs are controlled by the WBR and WFIs are observed by the WBR.

12.4.3.1 Specifications

Rules

a) While in OF mode, cells provided for core inputs shall respond to Capture, Shift, and, if provisioned for them (under rule 12.2.1(c) or rule 12.2.1(d)), Transfer events.

b) While in OF mode, cells provided for core outputs shall respond to Shift, Apply, and, if provisioned for them (under rule 12.2.1(d) or permission12.2.1(f)), the Transfer or Update events, respectively.

Recommendations

c) While in OF mode, cells provided for core inputs should present safe data at their CFO terminals (i.e., internal safe state).

12.4.4 Nonhazardous mode

In nonhazardous mode, core inputs are controlled to safe data by the WBR, and WFOs are controlled to safe data by the WBR.

12.4.4.1 Specifications

Recommendations

a) While in nonhazardous mode, all cells should present safe data at their CFO terminals.

Permissions

b) While in nonhazardous mode, the WBR may respond to any events.

12.4.4.2 Description

Safe data are data that will not harm circuitry internal or external to the core. These data can be static or dynamic.

12.5 Parallel access to the WBR

In addition to its mandatory serial wrapper access mechanism, IEEE Std 1500 provides for an optional parallel wrapper access mechanism. There are two possible ways to implement parallel access of the IEEE 1500 WBR, distinguished by the presence or absence of an IEEE 1500 wrapper cell on every core terminal:

— In the case where every core terminal contains an IEEE 1500 wrapper cell, as shown in Figure 17, parallel access of the WBR is achieved via segmentation of the WBR chain.

— In the case where not every core terminal has a WBR cell, per rule 12.1.1(a), parallel harnessing is implemented, as shown in Figure 20, in order to allow enabling of the parallel test being applied to the core as a response to a parallel instruction in the WIR.

NOTE—Harnessing logic enables coupling of a TAM to CTIs and CTOs. This includes any logic that is needed between the core and a TAM provided that such logic obey wrapper states rules defined in Clause 13.

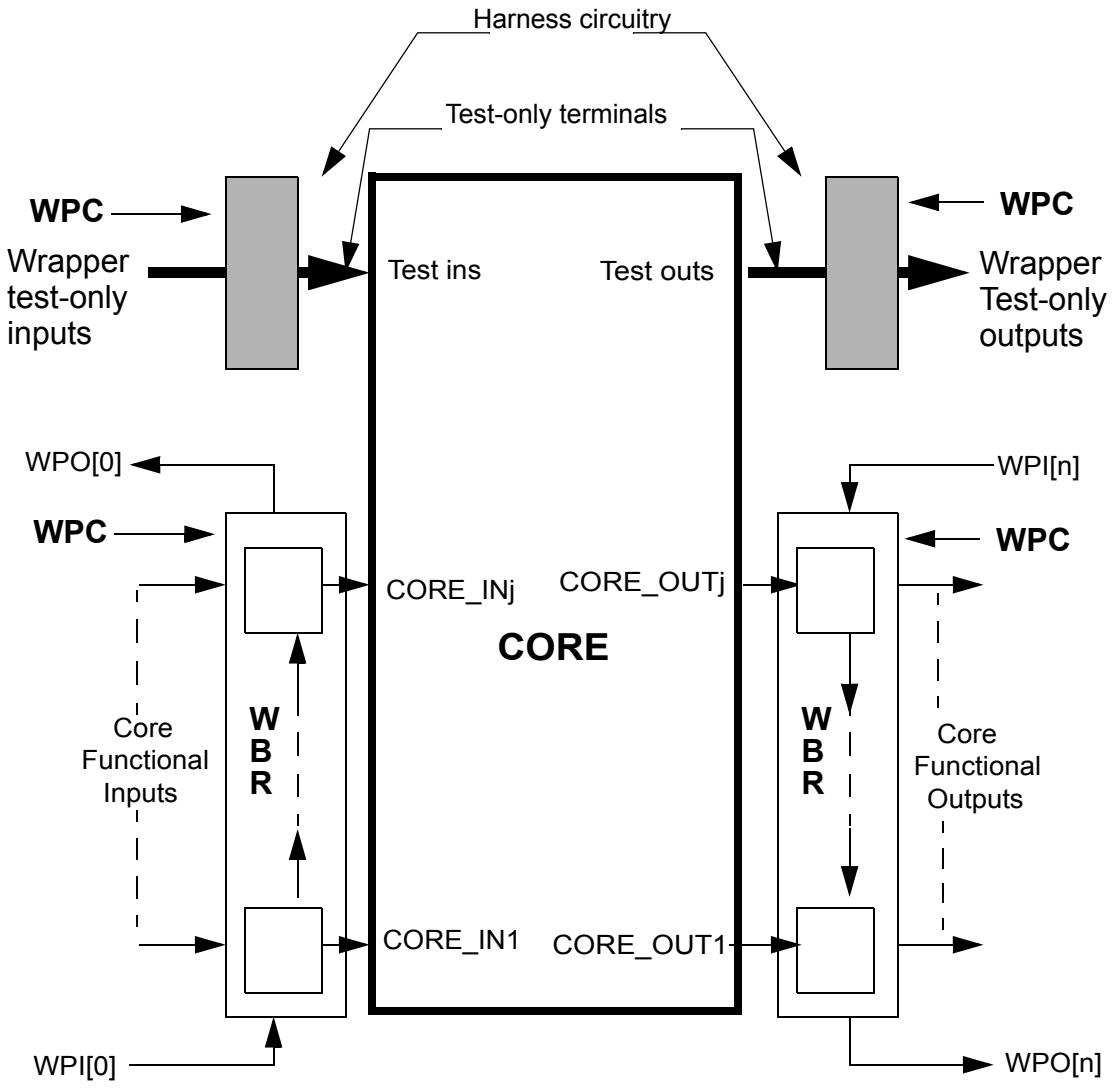


Figure 20—WBR harnessing via test-only terminals

12.5.1 Parallel configuration of the WBR

This subclause applies to both parallel access types described in 12.5.

12.5.1.1 Specifications

Rules

a) Parallel configuration of the WBR shall be enabled by the WIR, as a response to a parallel or hybrid instruction, e.g. WP_PRELOAD, WH_EXTEST, WP_INTEST_SCAN.

b) WPI and WPO terminals shall be distinct from the mandatory WSP terminals, provided that the parallel interface exists.

c) For every segmentation of the WBR in every parallel configuration, every WBR cell shall be a member of a unique WBR segment and be provided a direct or indirect path from a WPI to a WPO.

d) In parallel mode, all WBR segments shall be under the control of the WPC.

Recommendations

e) In cases where a WBR segment drives an internal scan chain, circuitry should be inserted to prevent certain wrapper events, such as Transfer, from causing data corruption in the internal scan chain.

Permissions

f) The WBR may be segmented into subsets of WBR scan chains accessible via WPI and WPO.

g) When the WBR is configured into parallel mode, the serial input (TI) and serial output (TO) of each segment may be connected to one of the following:

1) Another WBR segment from the same wrapper (in order to create a larger segment),

2) a TAM via WPI or WPO, respectively,

3) a core internal scan chain output or input, respectively.

h) There may be more than one parallel configuration of the WBR, allowing different parallel instructions to access different segmentations of the WBR.

12.5.1.2 Description

IEEE Std 1500 allows for a configuration of the WBR as a parallel interface to a core, in order to provide an increase in data bandwidth to the core. This parallel configuration of the WBR allows flexibility for the connectivity of the WBR segments and for how the parallel test applied to the core is controlled. The flexibility in connectivity allows WBR segments to be connected via their WPI and WPO to a TAM interface or to internal scan chains. For standard parallel instructions, nonclock WPC signals that are distinct from WSC signals, are used to control the WBR segments during the application of a parallel test. User-defined parallel instructions are not required to use WPC signals that are distinct from WSC signals for controlling the parallel test.

12.5.2 Harnessing of the WBR

This subclause discusses additional rules and permissions specific to the second type of parallel access described in 12.5, which is the parallel harnessing of the WBR.

12.5.2.1 Specifications

Rules

a) Harnessing logic inserted on test-only terminals shall maintain compliance with Wrapper Disabled state rules described in 13.1.

b) Parallel harnessing shall be configured only in response to an instruction loaded in the WIR.

Permissions

c) Test-only terminals [which are exempted from the requirement for wrapper cell insertion per rule 12.1.1(a)] may be provided with wrapper harnessing logic that, in response to an instruction present in the WIR, couples these terminals to a TAM for the application of parallel tests.

d) In response to nonstandard or hybrid/parallel instructions in the WIR, the configuration of the harnessing logic may be actuated via a TAM signal, a WBR cell, or a WPC signal.

e) Parallel harnessing may be reduced to a direct connection from a TAM to a core input, provided that compliance to Wrapper Disabled state rules (described in 13.1) is maintained via control from the WIR.

f) Parallel harnessing may be configured by a combination of WBR storage elements provided uniquely for this purpose, provided that the configuration is enabled in response to the WIR. Such WBR storage elements need not correspond uniquely (one to one) to the core terminals so harnessed.

12.5.2.2 Description

Rules defined in this subclause complement those defined in 12.5.1 in that harnessing requires compliance to both 12.5.1 and 12.5.2.

Although flexibility is provided to the user in terms of defining the nature of the harnessing logic, the WIR remains the enabler of the parallel mode. In particular, core terminals with direct harnessing (direct wire) to a TAM should not interfere with the Wrapper Disabled state mode of the IEEE 1500 wrapper. Test modes specific to the parallel mode of the wrapper should be enabled only as a response to the WIR. Therefore, appropriate gating logic is expected to be inserted whereby the connection between the TAM and the core terminal is enabled by the WIR as part of a parallel test instruction. WBR storage elements may be used to provide decoding logic for the configuration of parallel harnessing, assuming that this configuration is requested from the WIR.

12.6 WBR cell naming

IEEE Std 1500 mandates a naming convention for WBR cells.

12.6.1 Specifications

Rules

a) All IEEE 1500 WBR cell names shall match the following regular expression: /((WC)|(WH))((_S[DF]\d+)|(_CI?))(_C([IOB][IOU])|N)?(_U[DF])?(_O)?(_G[01]?)?/.

12.6.2 Description

WBR cell structures are defined in Perl regular expression format to ease software tool automation as well as integration with the CTL. The regular expression defined in 12.6.1 allows for the identification of any IEEE 1500 WBR cell by parsing CTL code.

The naming of the various types of WBR cells shall be done in a descriptive, parsable method. Each cell type name shall begin with WC (WBR cell) or WH (WBR harness cell). This prefix is followed by a sequence of characters that describe the capabilities and structure of the cell. The information will indicate whether a particular storage element is shared or dedicated to wrapper operation, how many shift path storage elements exist, the existence and type of the optional update cell, and the existence of safe data support. In addition, for harness cells, the presence of an inversion will be noted and also if the cell is combinational or sequential. To support this, from one to five fields will exist in the name in a specified order. Each field begins with an underscore.

— The first field is mandatory and describes the nature and number of shift path storage elements or the combinational and/or inverted nature of a harness cell. The first two characters of this field are _S (shift) or _C (combinational). If _S is selected, the third character is either D (dedicated) or F [(shared) functional], followed by an integer indicating the number of shift path storage elements. If _C is selected, a third character, I, may be used to indicate an inversion in the data path. This first field has the following regular expression format: /(_S[DF]\d+)|(_CI?)/.

— The second field indicates the site where data are captured. The first two characters of this field are _C (capture). The third character of this field specifies the origin of captured data: I (CFI), O (CFO), or B (selectable by design, i.e., can be CFI or CFO). The last letter of this field indicates the capture site: I (the first element of the shift path), O (the last element of the shift path), or U (the update storage element). In cases where the wrapper cell being described does not perform the capture function, both third and fourth characters in this field should be replaced by N (none) to indicate that there is no origin for captured data and that the capture site does not exist. When _C or _CI is selected for

the first field, _CN is mandatory for the second field. The regular expression matching this second field is /(_C([IOB][IOU])|N)?/.

— The third field describes the nature of the update element. It is composed of three characters, the first two of which are _U (update). The third character is either D (dedicated) or F [(shared) functional]. Its regular expression format is /(_U[DF])?/. The absence of this field indicates the absence of an update storage element.

— The fourth field indicates, by its presence or absence, the presence or absence of an observe-only characteristic. It is composed of an underscore followed by an O. Its regular expression format is /_O/.

— The final field indicates, by its presence or absence, the presence or absence of safe data support in nonhazardous mode. It is composed of an underscore followed by a G (guarded data) and optionally by a 0 or 1 indicating the static value. Its regular expression format is /_G[01]?/.

The following is a generic CTL code describing IEEE 1500 WBR cells:

```
Environment (Env_name) {
   CTL (CTL_name){
    External {
     (sigref_expr {
       (connectTo {
         Wrapper IEEE1500
         (CellID cell_enum | PinID user_defined);
       })*
      })+
     }
   }
}
cell_enum = /((WC)|(WH))((_S[DF]\d+)|(_CI?))(_C([IOB][IOU])|N)?(_U[DF])?(_O)?(_G[01]?)?/
```

IEEE P1450.6 (CTL) documentation contains an in-depth description of the above CTL syntax.

12.7 WBR cell examples

Table 2 describes IEEE 1500 WBR cell example names, and a gallery of bubble diagrams depicting each example follows the table. The bubble diagrams used in these examples are defined in Annex A. It is understood that the means of data path selection shown in these bubble diagram figures is to be configured by the content of the WIR.

Table 2—WBR cell example list

Cell description	Name	Figure number
One storage element shared with functional operation.	WC_SF1_CII	Figure 21
One storage element in shift path dedicated to wrapper function, capturing from CFO.	WC_SD1_COI	Figure 22
Two dedicated shift path storage elements, capturing from CFI into the shift storage element closest to CTO. This cell captures data into the storage element closest to the scan output.	WC_SD2_CIO	Figure 23

Table 2—WBR cell example list *(continued)*

Cell description	Name	Figure number
One dedicated storage element in the shift path and a shared update storage element. This cell captures into the update storage element.	WC_SD1_CIU_UF	Figure 24
A reduced-functionality cell with one dedicated storage element in the shift path, performing observe-only function.	WC_SD1_CII_O	Figure 25

The WC_SF1_CII cell types have the least circuitry. As this cell services both normal operation and test operation, the two operations are likely mutually exclusive. Also, the CFO terminal toggles as data are shifted from CTI to CTO and when a capture occurs. WC_SF1_CII_G may drive a safe value in addition to supporting the functionality described for WC_SF1_CII. See Figure 21.

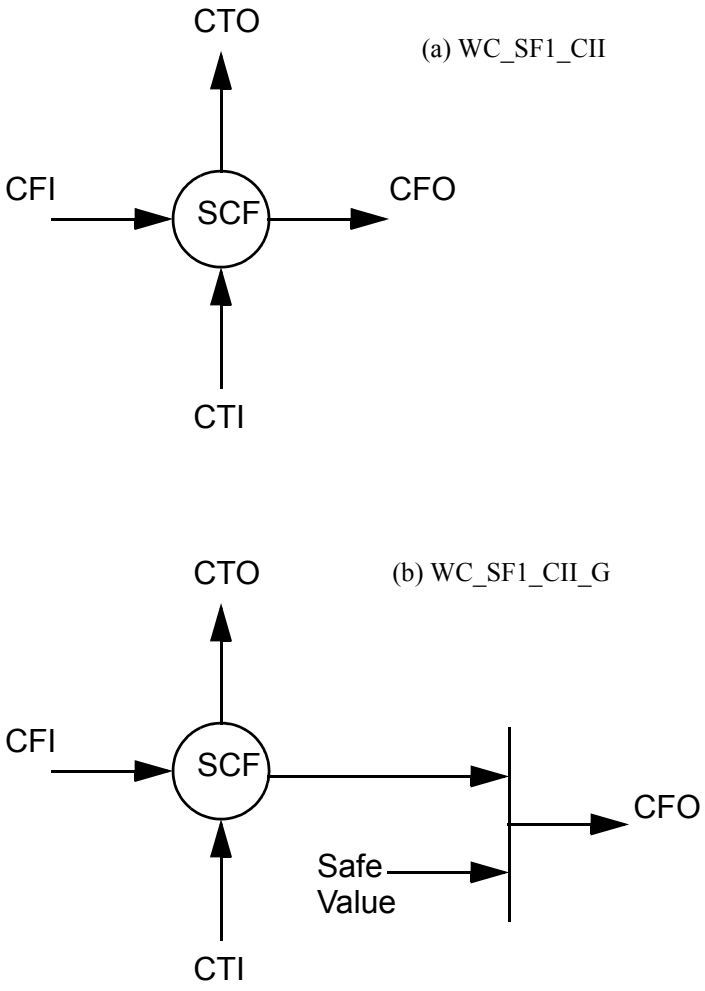


Figure 21—WC_SF1_CII WBR cell

WC_SD1_COI has a dedicated shift path, and its CFO may toggle during Shift or Capture operations. However, compared to the WC_SD1_CII cell, the CFI to CFO connection has superior testability. See Figure 22.

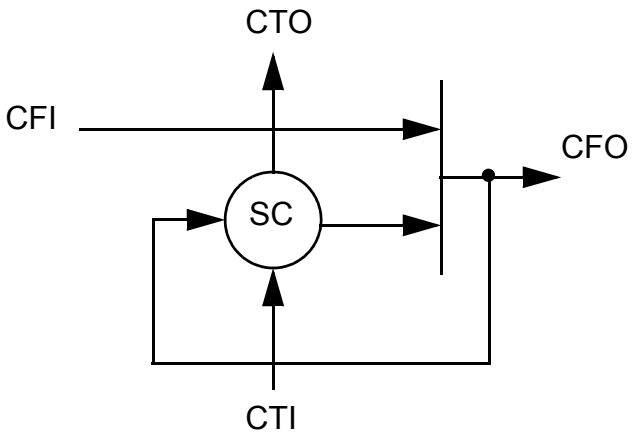


Figure 22—WC_SD1_COI WBR cell

The WC_SD2_CIO cell has the capability to circulate a sequential pattern for delay testing applications. Note that this cell has a dedicated shift path and responds to the Transfer event. A detailed example illustrating the use of this cell is discussed in 12.8. See Figure 23.

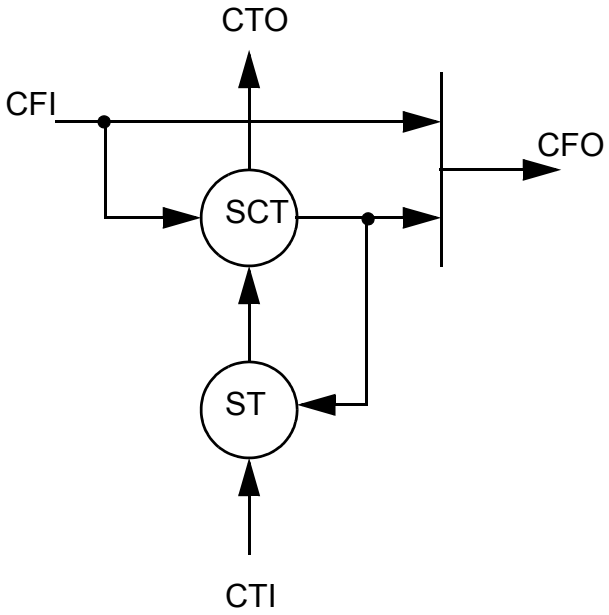
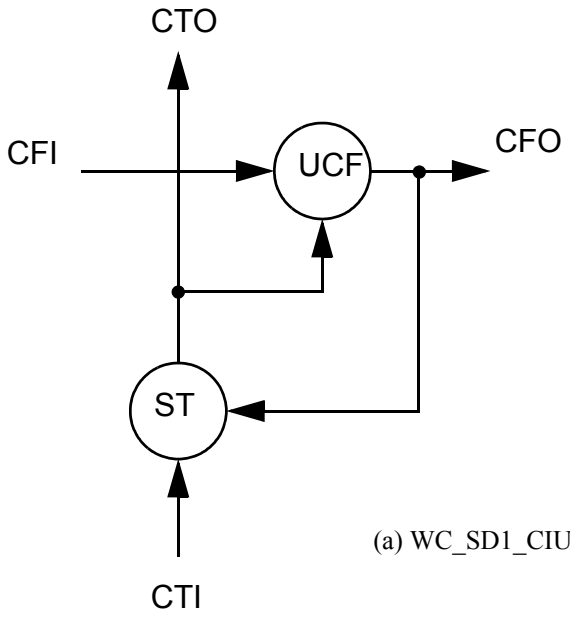
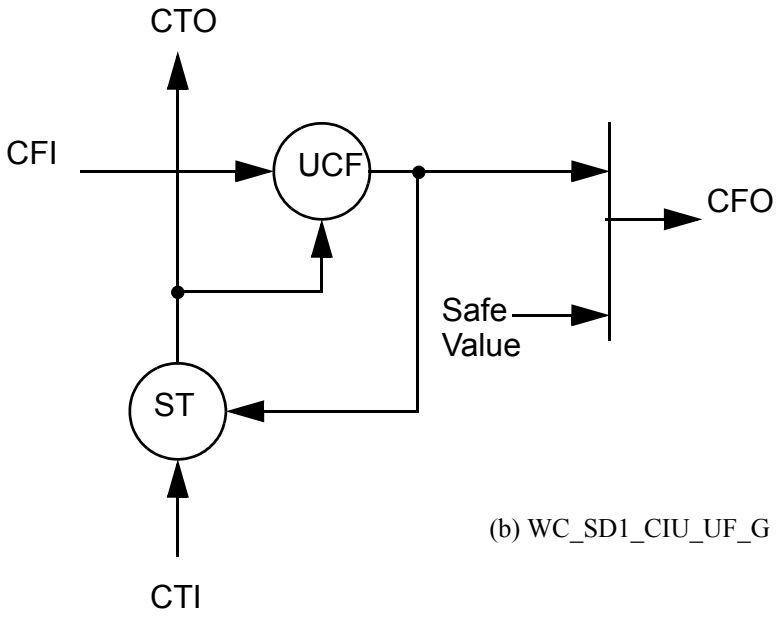


Figure 23—WC_SD2_CIO WBR cell

WC_SD1_CIU_UF has a dedicated shift path. An update storage element that is shared with functional operation serves as capture site for this cell. Accordingly, the path from CFO to the shift path storage element is provided for the Transfer operation to move captured data into the shift path. WC_SD1_CIU_UF_G may be forced to a safe value in addition to supporting the functionality described for WC_SD1_CIU_UF. See Figure 24.



(a) WC_SD1_CIU_UF



(b) WC_SD1_CIU_UF_G

Figure 24—WC_SD1_CIU_UF WBR cell

WC_SD1_CII_O is a observe-only harnessing cell. This cell is to be used only in accordance with permission 12.2.1(g). See Figure 25.

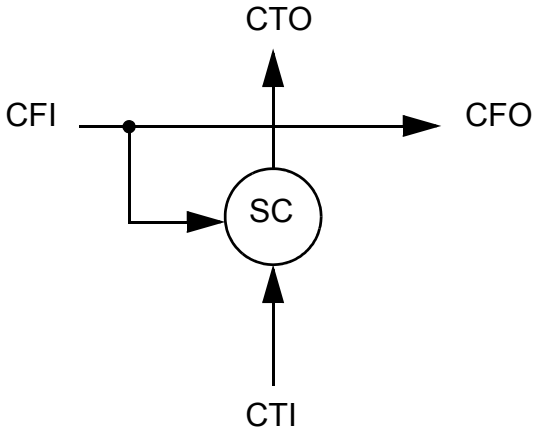


Figure 25—WC_SD1_CII_O WBR cell

12.8 IEEE 1500 WBR example

This subclause describes an example of how the WBR and WBR events may be used to perform a delay test. Figure 26 depicts an example logic schematic of a WC_SD2_CIO WBR cell. The bubble diagram corresponding to this WBR cell is shown in Figure 23. The WBR chosen for this example supports the Transfer event.

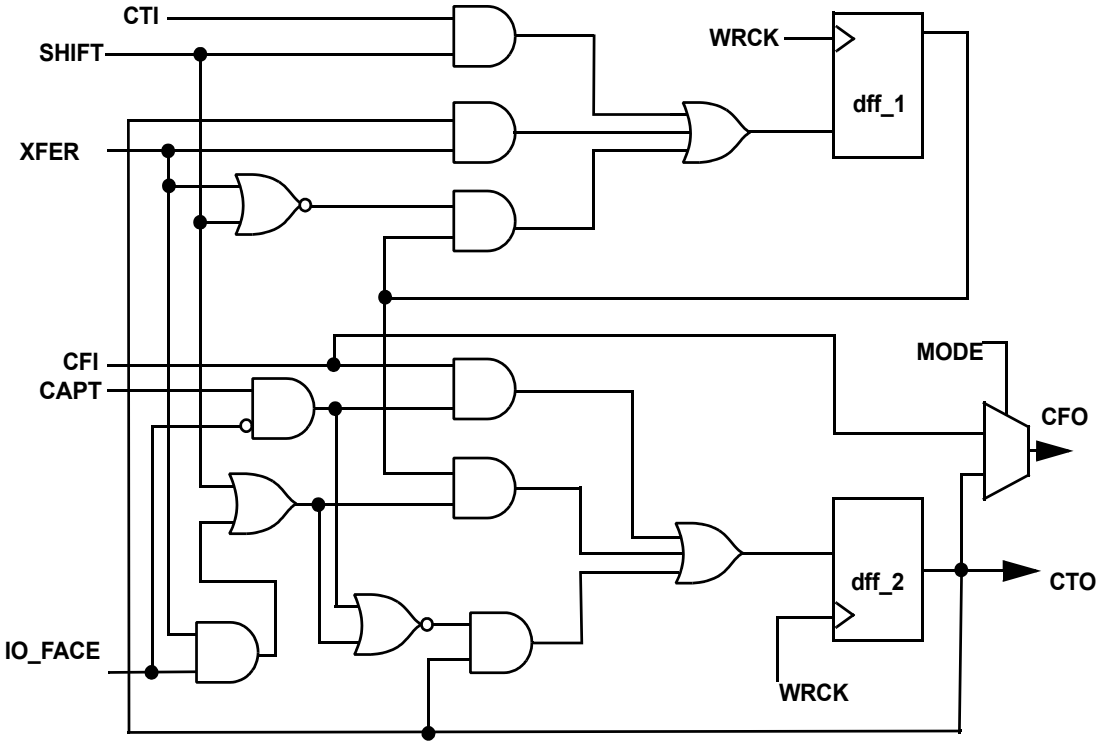


Figure 26—Example logic schematic of a "WC_SD2_CIO" WBR cell

This cell has a dedicated shift path of two independent data positions; therefore, it responds to the Transfer event (XFER). It captures data from its CFI port into the shift path storage element closest to CTO (dff_2). This cell may be used for either core input terminals or core output terminals but it is to be connected differently for these two cases per Table 3. SHIFT and CAPT are the control signals for the Shift and Capture events, respectively.

Table 3—Value applied to IO_FACE signal of WC_SD2_CIO example for different cases

		Instruction type	
		IF	OF
Terminal type	Core input	1	0
	Core output	0	1

IO_FACE is derived from the WIR. Dff_1 participates in the Shift and Transfer events unconditionally and holds state in the absence of these events. Dff_2 participates in the Shift event unconditionally and responds to transfer if IO_FACE = 1 and responds to capture if IO_FACE = 0; otherwise, DFF_2 holds state. The combinational logic shown decodes the SHIFT, TRANSFER, and CAPTURE signals along with the WIR-derived IO_FACE signal to determine the source of the data clocked into dff_1 and dff_2 in response to the clock.

Figure 27 depicts a simplified example of an arrangement of an AND gate as a core and three of these WC_SD2_CIO cells as a WBR. Mode indicates whether the cell is in test or normal mode. The transfer, shift, and capture signals are derived from TransferDR, ShiftWR, and CaptureWR.

Consider the scan order through 2 bits in each of the cells for terminals IN0, IN1, and OUT. The cell provided for terminal OUT is configured to receive the opposite polarity value of the IO_FACE signal from that applied to the other two cells. Also depicted are the logic values of a delay test of three vectors applied to the AND gate inside the simplified core.

Figure 28 depicts a timing diagram of the delay test sequence. For ease of explanation, the rising edges of WRCK are numbered sequentially.

The sequence begins with 6 data bits being shifted into the WBR. Following the Shift operation, data in dff_1 and dff_2 of the wc_in0 wrapper cell are both 1; dff_1 and dff_2 of the wc_in1 cell are loaded with 1 and 0, respectively; data in dff_1 and dff_2 of the wc_out wrapper cell are both 0.

With this data pattern, wc_in1 has an initial value of 0 applied, but there is a 1 value ready in that cell's dff_1 storage element.

Clock cycle 7 is a Transfer event where wc_in0 and wc_in1 internally exchange data between their respective dff_1 and dff_2 bits. This results in a rising-edge delay test stimulus being applied to the IN1 input of the AND gate.

Clock cycle 8 performs both the Transfer and Capture events. If the time interval between clock events 7 and 8 is according to the delay time specification of the path from IN1 to OUT, then a first delay test response is captured in dff_2 of wc_out. Simultaneously, a falling-edge delay test stimulus is applied to the IN1 input of the AND gate. After a settling time, the WBR Test Output TO reflects the result of the capture.

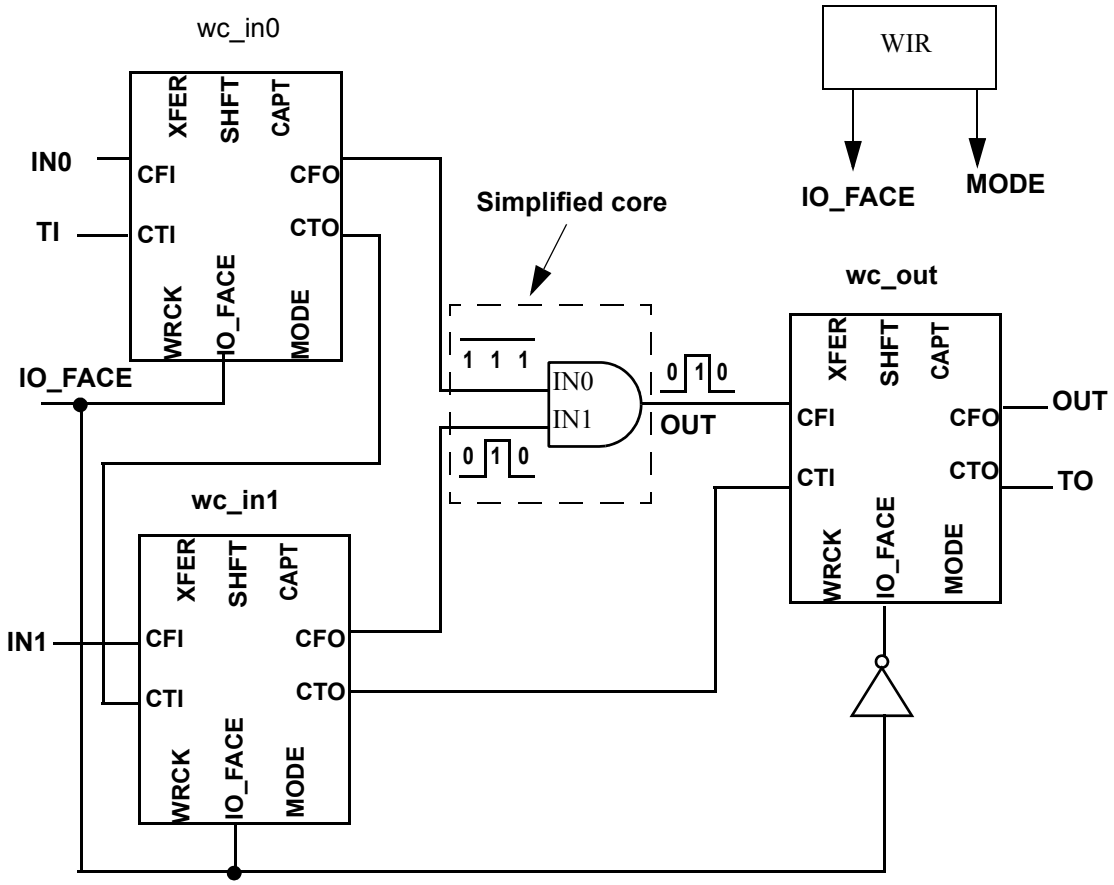


Figure 27—WBR cell connectivity around a core

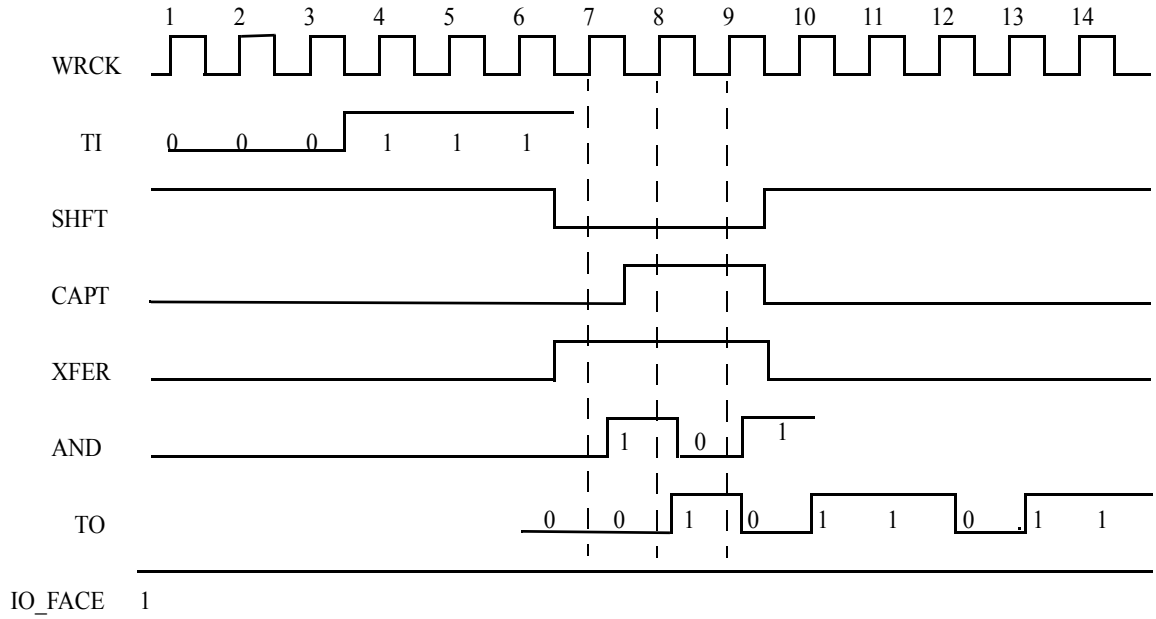


Figure 28—Delay test sequence timing diagram

Clock cycle 9 again performs both the Transfer and Capture events. If the time interval between clocks 8 and 9 is according to the delay specification of the AND gate a second delay test response is captured in dff_2 of wc_out. Simultaneously, the data value that was captured during clock 8 is transferred to dff_1 of wc_out. After a settling time, the WBR TO reflects the results of the second capture.

Clocks 10 through 14 are used to shift out the last 5 bits of the WBR where the results of the first capture come out on clock 10.

This simplified example has been presented in the context of a combinational path from a core input terminal, through the core, to a core output terminal while the WBR is in IF mode. It should be understood that while this simplified example has used the AND gate as a core, the AND gate could have been a UDL, tested with WBR cells in OF mode.

13. Wrapper states

13.1 Wrapper Disabled and Wrapper Enabled states

The WRSTN signal can be used to force the wrapper logic into a state that enables functional operation of the core. Figure 29 and Figure 30 identify two main wrapper states: Wrapper Disabled and Wrapper Enabled, respectively. Both have substates that are entered in a sequence controlled by the WSP as defined by this standard. The wrapper is forced unconditionally to a Wrapper Disabled state when the WRSTN signal of the WSP is low. In Wrapper Disabled state, the wrapper is inactive and functional operation of the core logic can continue unhindered. Test operations are performed while the wrapper is kept in Wrapper Enabled state by a high value on the WRSTN signal.

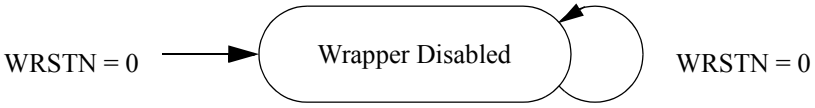


Figure 29—Wrapper Disabled state conditions

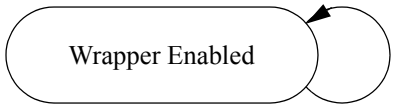


Figure 30—Wrapper Enabled state conditions

13.1.1 Specifications

Rules

a) Asserting logic 0 on WRSTN (WRSTN = 0) shall unconditionally force the wrapper into Wrapper Disabled state; no actions of the wrapper logic (e.g., WIR, WBR) shall occur in response to WSP signals and/or WPP signals other than WRSTN and/or WRCK.

b) Asserting logic 0 on WRSTN (WRSTN = 0) shall unconditionally force the WS_BYPASS instruction to become effective.

c) Asserting logic 1 on WRSTN (WRSTN = 1) shall unconditionally enable the wrapper to respond to the WSP signals.

d) When the wrapper logic becomes enabled after being disabled, the effective instruction present in the WIR shall be the WS_BYPASS instruction.

Recommendations

e) The protocol used to control the WRSTN signal should ensure that at least one complete WRCK pulse is applied when WRSTN is set to logic 0 (WRSTN = 0) before WRSTN can be set to logic 1 (WRSTN = 1).

Permissions

f) The wrapper may require an active WRCK during Wrapper Disabled state (WRSTN = 0), prior to becoming enabled by logic 1 on WRSTN (WRSTN = 1).

13.1.2 Description

Independent of the current state of a wrapper, this wrapper will enter Wrapper Disabled state when the active low WRSTN signal is asserted. The wrapper will remain in this state while the WRSTN signal remains low. In the Wrapper Disabled state, the wrapper is forced to its inactive state. This is achieved either by asynchronously resetting the WIR contents to WS_BYPASS or by gating the WIR outputs (with WRSTN = 0) to create the same effect. In the latter case, a synchronous reset operation is still needed to load the WS_BYPASS instruction into the WIR before the wrapper is enabled by de-asserting the WRSTN signal. For this reason, it is allowed to require an active WRCK during Wrapper Disabled state, prior to using the wrapper in Wrapper Enabled state.

If asynchronous reset of the WIR is used to implement the Wrapper Disabled state, the sequence prior to test operations consists only of deasserting WRSTN. When gating of WIR outputs is used to implement the Wrapper Disabled state and a synchronous reset of the WIR is needed to load WS_BYPASS, the sequence prior to test operations needs an active WRCK during the Wrapper Disabled state. Because this sequence can also be used for wrappers with an asynchronous reset of the WIR, it is recommended for all wrappers. In order to have a reliable operation of the wrapper, WRSTN should be deasserted only on the falling edge of WRCK so that at least 1/2 WRCK period is left for signals to propagate within the wrapper. Figure 34 describes a synchronous reset timing.

All rules described in this standard apply to the Wrapper Enabled state, except rules specific to the Wrapper Disabled state of the IEEE 1500 wrapper.

14. WSP timing diagram

This clause defines timing relationships between the mandatory WSP terminals. The actual values, denoted TBD, are to be supplied in the CTL by the wrapper provider. These timing specification parameters apply to any wrapper implementation. Values with a hyphen (-) indicate they are not necessary or not pertinent to the usage. Timing diagrams shown in this clause are not meant to show protocol, but rather timing relationships between wrapper signals.

14.1 Specifications

Rules

a) SelectWIR, ShiftWR, CaptureWR, and TransferDR shall be sampled on the rising edge of WRCK.

b) The UpdateWR signal shall be sampled on the falling edge of WRCK.

c) Actions of IEEE 1500 standard components (e.g., WIR, WDRs) shall occur on either the rising or falling edge of WRCK in response to WSC signals as depicted in Figure 31, Figure 32 and Figure 33.

d) Changes on the SelectWIR signal shall not occur coincident with Shift, Capture, Transfer, or Update operations.

e) Provided that permission 13.1.1(f) is exercised, the timing relationships depicted in Figure 34 shall be applicable.

14.2 Description

Even though ShiftWR may change while WRCK is high, it is anticipated that data driven from WSO at the falling edge of WRCK will result from the data or instruction register that was selected at the immediately preceding rising edge of WRCK.

Table 4 and Figure 31 describe timing parameters for WSP operation. See Clause 8 for details on these signals.

Table 4—WSC and WBR timing parameters for WSP operation

Timing parameter	Symbol	Min	Max
Clock pulse high	t_{ckwh}	TBD	—
Clock pulse low	t_{ckwl}	TBD	—
WRSTN Pulse Width Low	t_{rstl}	TBD	—
WRSTN negation setup time w.r.t. WRCK rising edge	t_{rstsu}	TBD	—
SelectWIR setup time w.r.t. WRCK rising edge	t_{swsu}	TBD	—
SelectWIR hold time w.r.t. WRCK rising edge	t_{swhd}	TBD	—
ShiftWR, CaptureWR, or TransferDR setup time w.r.t. WRCK rising edge	t_{ctlsu}	TBD	—
ShiftWR, CaptureWR, or TransferDR hold time w.r.t. WRCK rising edge	t_{ctlhd}	TBD	—
UpdateWR setup time w.r.t. WRCK falling edge	t_{updsu}	TBD	—
UpdateWR hold time w.r.t. WRCK falling edge	t_{updhd}	TBD	—
WSI setup time w.r.t. WRCK rising edge	t_{sisu}	TBD	—
WSI hold time w.r.t. WRCK rising edge	t_{sihd}	TBD	—
WSO output valid time w.r.t. WRCK falling edge	t_{sov}	—	TBD

Timing relationship of the AUXCK with respect to WRCK is discussed in Clause 16.

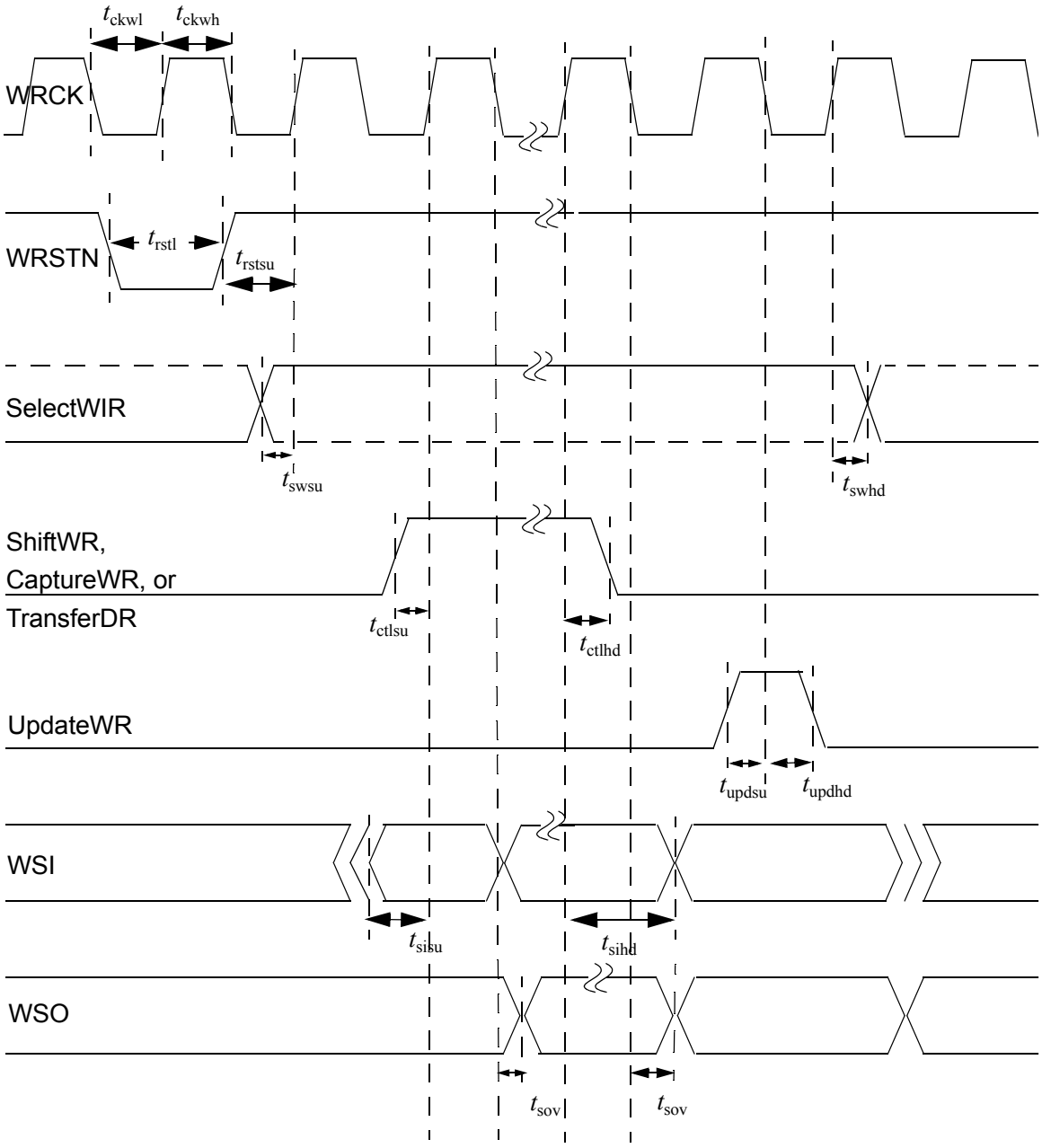


Figure 31— IEEE 1500 WSP timing

14.2.1 Timing parameters for event and functional input/output (I/O)

Table 5, Figure 32, and Figure 33 describe timing parameters regarding event and WFI/WFO. Figure 32 describes timing relationships for wrapper cells with minimally required terminals, CaptureWR and ShiftWR. Figure 33 describes timing relationships for wrapper cells that support Update and Transfer events. Note that multiple TransferDR operations are possible depending on WBR cell capability.

Table 5—Event and functional timing

Timing parameter	Symbol	Min	Max
CaptureWR or TransferDR setup time w.r.t. WRCK rising edge	t_{ctlsu}	TBD	—
CaptureWR or TransferDR hold time w.r.t. WRCK rising edge	t_{ctlhd}	TBD	—
UpdateWR setup time w.r.t. WRCK falling edge	t_{updsu}	TBD	—
UpdateWR hold time w.r.t. WRCK falling edge	t_{updhd}	TBD	—
WFO output valid time w.r.t. WRCK rising edge without update	t_{rov}	—	TBD
WFO output valid time w.r.t. WRCK falling edge with update	t_{fov}	—	TBD
WFI setup time w.r.t. WRCK rising edge	t_{fisu}	TBD	—
WFI hold time w.r.t. WRCK rising edge	t_{fihd}	TBD	—

The minimal configuration timing specification shown in Figure 32 can apply, as an example, to wrapper cell WC_SF1_CII from Figure 21, where CFI is to be connected to WFI and CFO to be connected to WFO as shown in Figure 16. An example of a wrapper cell corresponding to the timing specification of Figure 33 would be wrapper cell WC_SD2_CIU_UF shown in Figure B.5.

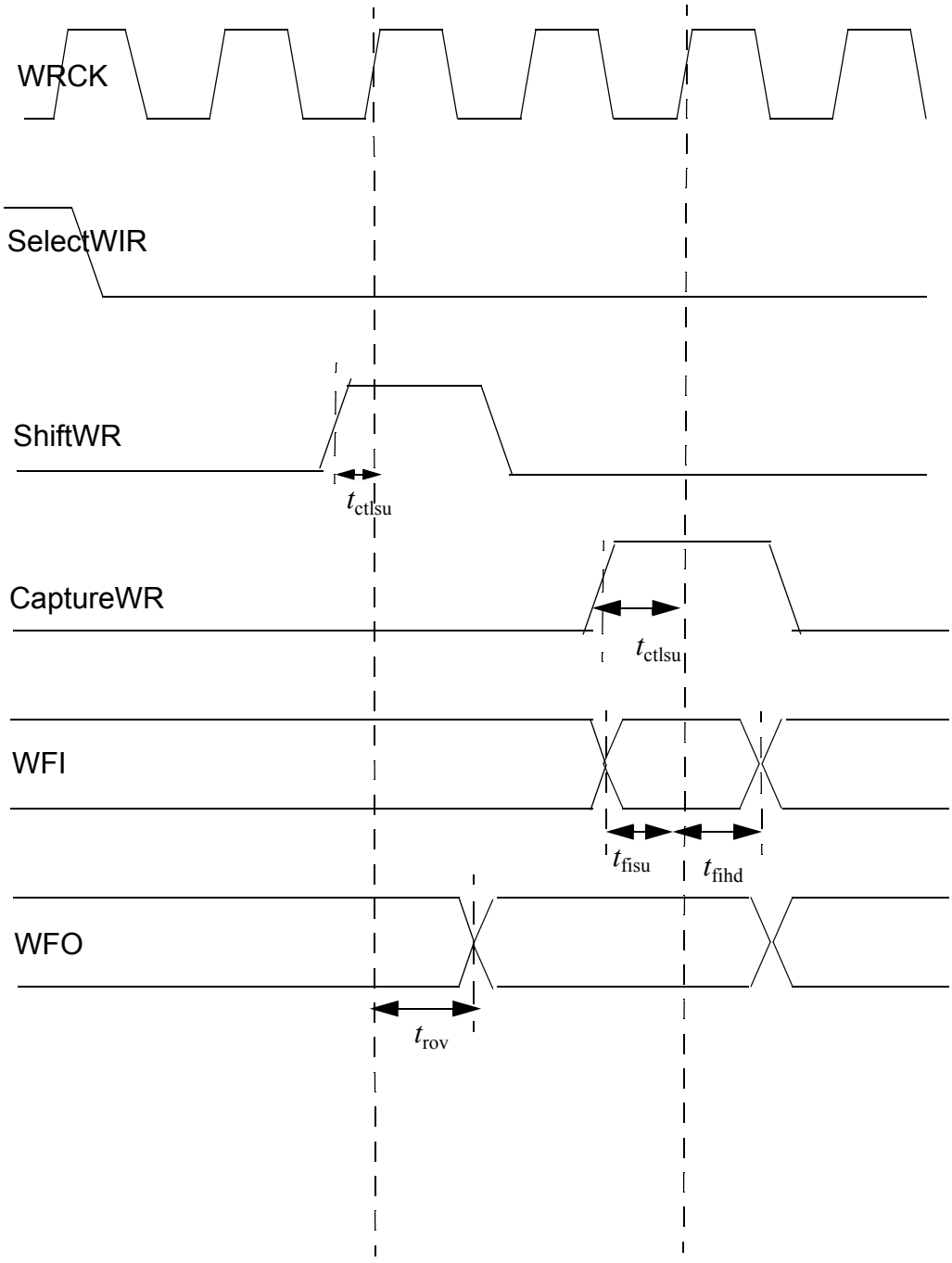


Figure 32—IEEE 1500 minimal event and functional I/O timing

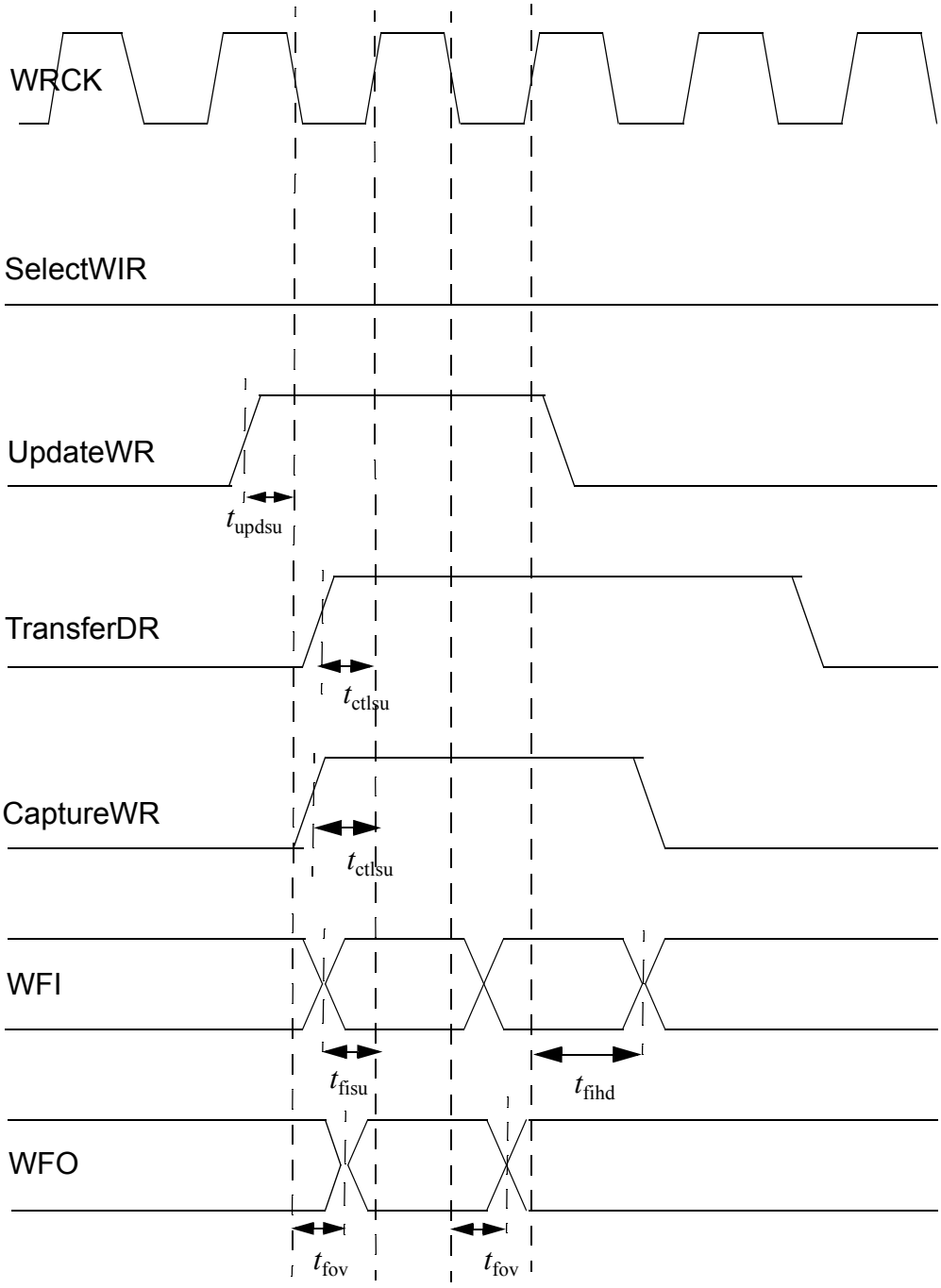


Figure 33—IEEE 1500 event and functional I/O timing

14.3 Synchronous reset timing

Table 6 and Figure 34 show the timing parameters for an optional synchronous reset.

Table 6—Synchronous reset timing

Timing parameter	Symbol	Min	Max
Clock pulse high	t_{ckwh}	TBD	—
Clock pulse low	t_{ckwl}	TBD	—
Synchronous WRSTN assertion setup time w.r.t. WRCK rising edge	t_{srstsu}	TBD	—
Synchronous WRSTN assertion hold time w.r.t. WRCK falling edge	t_{srsthd}	TBD	—

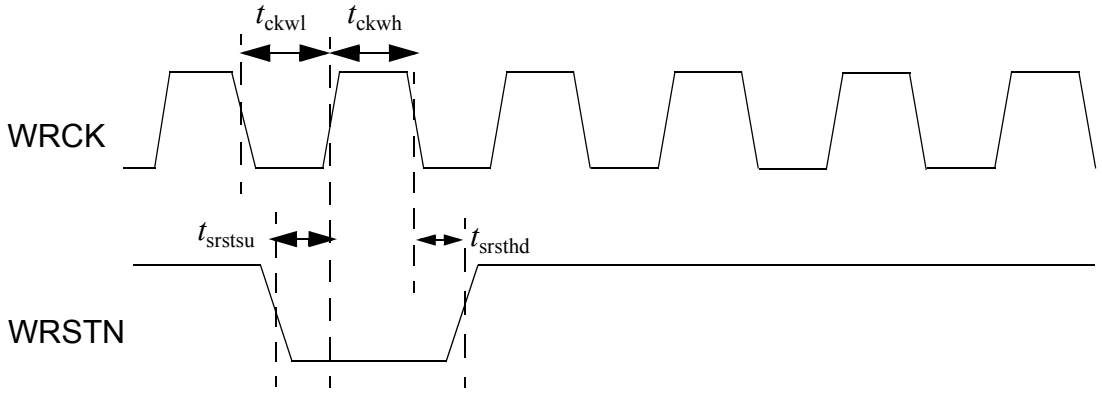


Figure 34—IEEE 1500 synchronous reset timing

15. WSP configurations for IEEE 1500 system chips

This clause provides specifications and description for configuring multiple IEEE 1500 WSPs at the system chip level.

15.1 Connecting multiple WSPs

This subclause exemplifies the system chip configurations for the WIR and WBY registers when connecting the WSP of IEEE 1500 wrappers. Figure 35 through Figure 39 show various configurations of IEEE 1500 wrapper connections for the WSP. In the figures, the system chip configuration is defined by the way the WSI-WSO scan path is connected at the system chip or core levels and how the WSPs are controlled. IEEE Std 1500 allows single or multiple WSI-WSO paths and WSPs to be configured, depending on the application needs of the system chip and cores.

In Figure 35, the WSP scan path order is Core1-Core2-CoreN, and the signals connected to the WSC terminals are globally bussed. Therefore, all wrappers are controlled together. This allows different instructions to be loaded into different WIRs, and the WSP protocol of the cores is controlled in a lock-step fashion. Therefore, for example, the SelectWIR signal would be bussed and is common to all N cores. Thus, all of the N WIRs will always be selected simultaneously. To control the WIRs and WDRs separately and, for example, load the WBRs of some cores with test data concurrent with a WIR Shift operations in other cores would require the signals connected to the SelectWIR terminals of the cores to be controlled separately for each core. IEEE Std 1500 recommends against this in the serial WSP configuration.

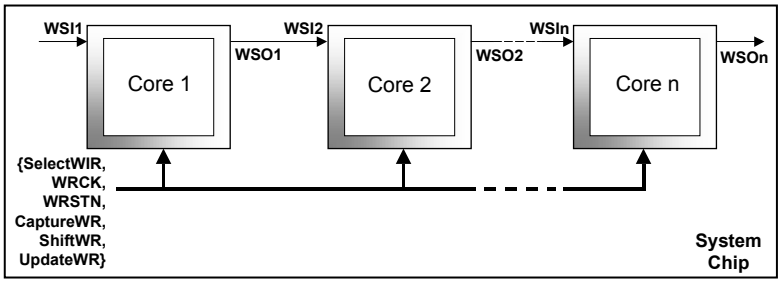


Figure 35—Serial connection of WSPs

The multiple parallel configuration shown in Figure 36 shows how the WSC signals can be bussed, as in the serial WSP configuration of Figure 35, yet the WSI-WSO scan paths of the three cores are separate parallel scan paths, one scan path for each core. This configuration supports parallel access to the three core WSPs.

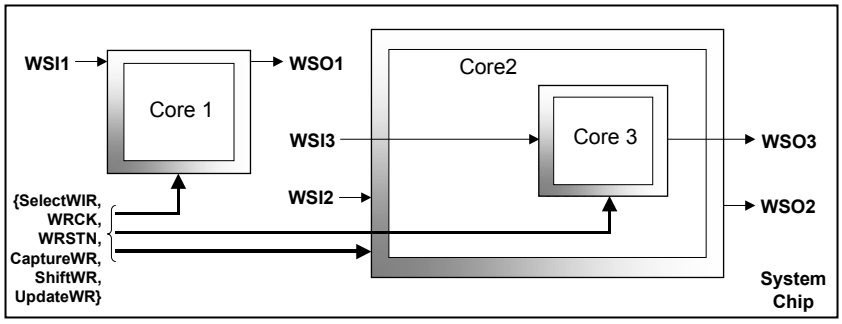


Figure 36—Multiple parallel WSP configuration

Figure 37 shows a WSP configuration with a core, Core 2 (which has two subcores), Core 3, and Core 4. This configuration uses a single WSC bus, as in Figure 35. This is similar to the single serial WSP configuration of Figure 35 in that there is a single WSP scan path through the system chip. The WSP scan path order for this configuration is Core1-Core2-Core-3-Core4.

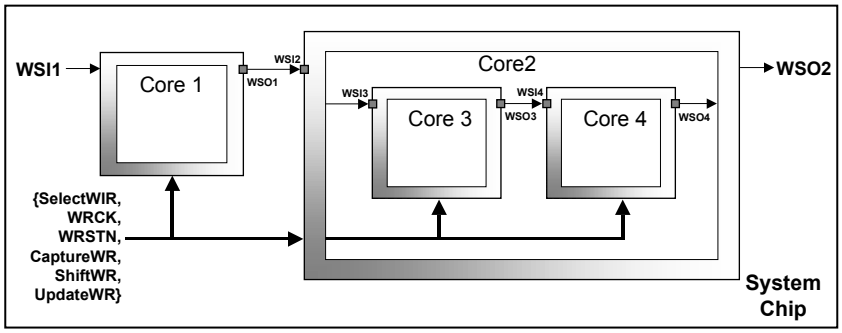


Figure 37—Serial WSPs with subcores

The WSI-WSO connection from Core2 to Core3 requires that the Core2 WSO be brought out of the wrapper and into the top level of the core. This is shown as the To_WSI3 output of WSP2 in the detailed figures, Figure 38 and Figure 39. Further, the WSO of Core4 must be routed to the Core2 wrapper for the final WSO output. Figure 38 and Figure 39 show two examples of how the WSO4 input to WSP2 can be selected at WSO2. In Figure 38, the subcore WSPs are always chained with that of Core2, so that WSO4 passes directly through the wrapper of Core2 and is output on WSO2. In Figure 39, there is a multiplexer for WSO2 that enables the subcore WSPs to be bypassed.

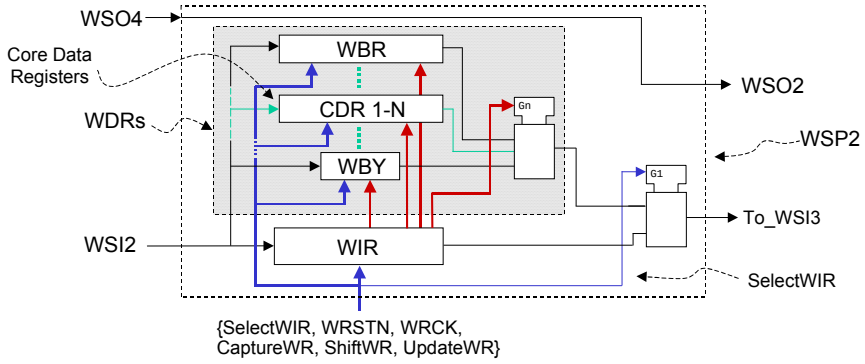


Figure 38—WSP2 with serial subcore paths

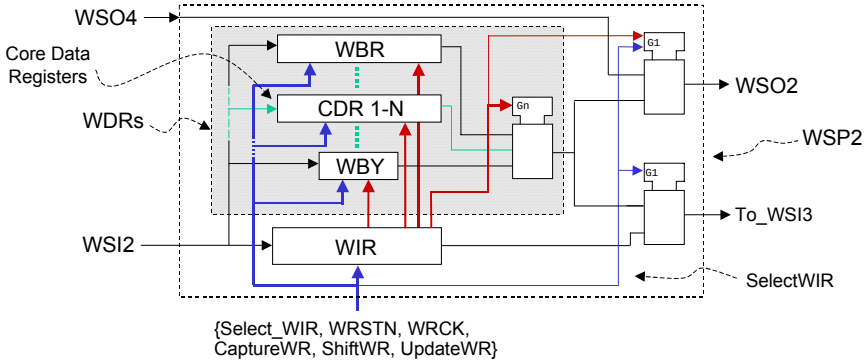


Figure 39—WSP2 with subcore WDR bypass

15.1.1 Specifications

Recommendations

a) When connecting the WSPs of two or more wrappers in series (i.e., daisy-chaining their WSI-WSO terminals) at the system chip level, the SelectWIR, WRSTN, WRCK, TransferDR, CaptureWR, ShiftWR, and UpdateWR terminals from each of the wrappers should be bussed, so that there will be a common source to all wrappers for each of the above WSC terminals.

Permissions

b) The WSP terminal interfaces may be configured at the system chip level in a manner appropriate to the system chip's requirements.

15.1.2 Description

When connecting the WSP interfaces of multiple IEEE 1500 wrappers within a system chip, the core integrator may connect and configure the interfaces as is required by the system chip. Three examples of WSP configurations are shown in Figure 35, Figure 36, and Figure 37. Multiple WSPs connected in series, multiple parallel WSPs, and a serial WSP with subcores are shown in the respective figures.

When chaining the WSPs of subcores in series with their parent core, the subcore WSP may be fixed, as in Figure 37. In this example, the WRs of the cores are always chained. Therefore, for example, when the WS_BYPASS instruction is loaded into Core2, Core3, and Core4, there will be a multiple-bit bypass path from WSI2 to WSO2. Figure 39 shows an example of where the subcore scan paths are multiplexed with the parent cores WDRs. This allows the subcore paths to be bypassed during certain instructions. Other WSP configurations are possible.

When connecting the WSPs of two or more wrappers in series, it is recommended that the WSC control signals be bussed and controlled by a common source. This will facilitate PnP of multiple serially connected WSPs.

16. Plug-and-play (PnP)

This clause provides specifications and descriptions for enabling interoperation between multiple IEEE 1500 wrappers at the system chip level.

16.1 Background and definition

PnP is the term used to describe a level of interoperability between different cores' wrappers on the same SoC design. This level of interoperability was settled upon as a compromise between these somewhat differing objectives:

— The architecture should provide for all cores having a uniform test capability utilizing the WSC and standard instructions;

— The architecture should be amenable to a variety of best practices of design and test of cores and SoCs. Core designers and SoC integrators should have flexibility to adopt a variety of implementation styles to meet their system and test design objectives;

— It should be permissible that the resources of the WBR be shared with normal system functionality (shared use).

As IEEE 1500 entirely owns the WIR and the WBY, it is a simple matter for this standard to specify those registers so that they may easily interoperate between multiple wrappers. However, specification of the interoperability of the WBRs is complicated by the notion that WBR may be shared with system functionality. This shared use of WBR resources is the primary constraint on achieving PnP. In the case that a core designer exercises the liberty to co-mingle WBR functionality with system functionality, it is recognized that core system clocks may operate the sequential elements in which WBR features reside. In the case that system functionality is separated from wrapper functionality, PnP is assured by design. At the same time, it is recognized that the WBR must be responsive to WRCK and the other WSC signals to support a common test capability. In order to resolve such WBR clocking issues and still mandate a uniform level of interoperability, the compromise accepts a limited decoupling of the cause-and-effect relationship between WRCK and events in the WBR.

The governing principle of IEEE 1500 PnP is as follows:

> A user of this standard in applying tests on an SoC comprising multiple IEEE 1500 cores via the WSC is assured that the multiple cores' wrappers will interoperate robustly at or below some maximum WRCK frequency.

In other words, robust PnP means that timing margins will improve with decreasing frequency, so below some maximum frequency, interoperability is assured. Note that this allows SoC testing to be accomplished without requiring careful timing of any IEEE 1500 test signals and without requiring knowledge of wrapper timing characteristics.

Permission 8.1.1(e) is repeated here for convenience.

The WSP may include AUXCKn terminal(s) in addition to the dedicated WRCK for operation of registers other than the WBY and WIR. Such clocks may be shared with other system clocks and may be used to operate other core features.

Core designers may use AUXCKs of their choice to actuate WSP-directed WBR events. Such use does not change the relationship between any of the other WSC terminals and WRCK, nor does it alter any of the rules governing those relationships. However, it may impose additional operational limitations. For example, one design implementation style that is accommodated by these rules is where WBR sequential elements do not receive WRCK exactly as a clock, but rather as an enable on the data port of the sequential elements. With such an implementation style, WBR operation would be dependent upon adherence to setup and hold time specifications of WRCK with respect to the underlying AUXCK as depicted in Figure 40. The timing parameters shown in this figure are sample parameters representing setup and hold timing of WRCK with respect to the AUXCK for some hypothetical wrapper implementation employing an AUXCK.

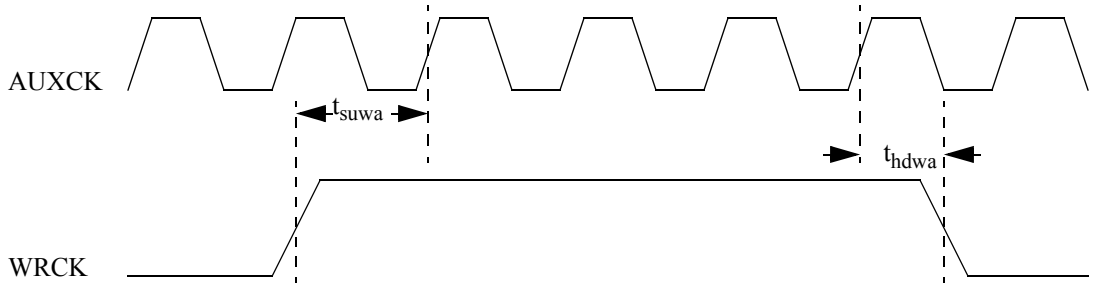


Figure 40—Example of timing relationship specification of WRCK w.r.t. an AUXCK

16.2 PnP aspects of standard instructions

16.2.1 Specifications

Rules

a) All standard serial instructions (WS_BYPASS, WS_SAFE, WS_CLAMP, WS_INTEST, WS_INTEST_SCAN, WS_INTEST_RING, WS_PRELOAD, and WS_EXTEST) must be robustly PnP with respect to the Shift operation of their selected registers. The respective registers of these instructions must be responsive to ShiftWR, SelectWIR, and WRCK.

b) All wrappers must be PnP with respect to Capture, Update, and optional Transfer operations of the WSC while executing the WS_EXTEST instruction.

c) Provided that permission 8.1.1(e) is exercised, the timing relationships required between WRCK and the AUXCK(s) to ensure PnP operation shall be specified.

16.2.2 Description

All wrappers must shift interoperatively when arranged on an SoC with their WSI and WSO terminals serially concatenated. All wrappers interact during Shift operation.

WS_EXTEST is the only standard instruction provided for testing of SoC circuitry that connects to core terminals. It is assumed that the wrappers of multiple cores are simultaneously involved in testing this circuitry. Because of this extensive interaction, WS_EXTEST uniquely has the requirement to be fully PnP for all WSC operations.

WRCK and auxiliary timing relationships required to ensure PnP operation are expected to be provided in CTL per rule 17.3.1(b). A sample representation of such timing relationship is shown in Figure 40.

16.3 PnP limitations on protocols

16.3.1 Specifications

Recommendation

a) PnP WS_EXTEST operation using the WBR should use protocols in which the intended Apply event and the corresponding Capture events do not occur simultaneously.

16.3.2 Description

In order for a WBR cell to reliably capture a new test response, the PnP definition requires that there be sufficient timing margin. If the Apply event providing a test stimulus and the Capture event for observing a response are triggered by the same edge of the same pulse of WRCK, there may exist a timing race. This may be ameliorated by applying the various events sequentially rather than simultaneously.

16.4 Non-PnP in IEEE Std 1500

Although this standard provides a basic PnP testing capability, there is no intention to inhibit non-PnP uses of this standard. For example, timing-related testing is inherently timing critical, and such testing is facilitated by such aspects of this standard as the WBR Transfer event and the allowance for AUXCKs. The user-defined parallel port and user-defined instructions and events are all opportunities for innovation within IEEE Std 1500 that are nonetheless non-PnP.

17. Compliance definitions common to wrapped and unwrapped cores

This clause lists and describes rules that must be followed to claim compliance with this standard. When used without any qualification, the word *core* applies to both wrapped and unwrapped cores in these rules. Words in **bold** represent CTL (IEEE P1450.6) keywords.

17.1 General rules

The rules in this subclause are general rules, not specific to individual terminals, core wrappers, or test patterns.

17.1.1 Specifications

Rules

a) All test information pertinent to core integration shall be provided in CTL, without relying on user-specific standard test interface language (STIL) extensions or functions.

b) The IEEE 1500 compliance level (wrapped or unwrapped) and version to which the core and associated information apply shall be specified using the **Compliancy** statement:
Compliancy IEEE1500 2005 <**Wrapped** | **Unwrapped**>.

c) The information related to the length of an existing WBR shall be supplied in CTL using **ScanStructures** {**Length** integer;}

d) The information related to the order of the wrapper boundary cells within the register shall be supplied in CTL using **ScanStructures** {**Cells** cell-list;}

e) The scan terminals of the WBR shall be specified using the **ScanDataType** statement:
signame {
DataType ScanDataIn {**ScanDataType** Boundary;}
}

f) All state elements of the WBR shall be described in CTL as part of a scan chain using the **ScanStructures** construct of CTL.

17.1.2 Description

Here is the reasoning behind the general rules:

a) For automation needs, the information delivered by the core provider to the system integrator is provided in CTL (IEEE P1450.6). Any use of user-defined mechanisms in the language that are provided for extensibility will limit interoperability. Furthermore, user-defined mechanisms should not be used when keywords already exist to convey the same information.

As an example, a core provider wants to communicate to the system integrator that there are three test modes in the design: two for testing the internal logic of the design and one for providing access to the wrapper scan chain for testing the external logic of the design. Also, the core provider wants to communicate the fact that the two internal test modes are different configurations that perform the same task. In other words, only one of the patterns in each of the internal test modes needs to be executed to complete the testing of the core. This text would have been sufficient to describe the information; however, it is not easy to automate processing of such text. The following CTL describes this information in an automatable form:

```
Environment {
    CTL config1 {
        TestMode InternalTest {AlternateTestMode config2;}

    }
    CTL config2 {
        TestMode InternalTest {AlternateTestMode config1;}

    }
    CTL config3 {
         TestMode ExternalTest;

    }
}
```


While the semantics are relevant to the understanding of the example, CTL's syntax is easily readable because of the verboseness of its keywords. Configurations of the design are test modes that are defined in CTL blocks of statements (CTL {}). Each test mode is assigned a type or purpose through the **TestMode** statement. Alternate test modes are identifiable through the **AlternateTestMode** statement. The reader should observe that the information defined in the syntax shown reflects the text that was intended to be described.

b) For ease of reference, the CTL shall specify which type of compliance is being provided (IEEE 1500 wrapped or IEEE 1500 unwrapped) and also an identification of which version of the IEEE Std 1500 was used to define this compliance. This is needed to handle any future, enhanced versions of this standard.

c) The intention is for the high-level information model provided by CTL to allow such pattern generation to be possible without need for access to the core's or even the wrapper's netlist definition. For an unwrapped core, providing full information about an existing WBR (if included) enables integration with other wrapper components. Any core terminals that have IEEE 1500 WBR cells provided with the core will need to be correctly identified so that the final integrated WBR can include those cells as well as any additional WBR cells. Also, knowing the correct association between WBR cells within the built-in WBR's bit positions and their corresponding core interface terminals is necessary so that the completed boundary chains can be correctly described for the fully wrapped core.

d) See (c).

e) The WBR chains are differentiated from other scan chains of the core using this statement.

f) This information aids in automatically recognizing and operating the existing WBR.

17.2 Per-terminal rules

The rules in this subclause are specific to the core's external interface terminals.

17.2.1 Specifications

Rules

a) All interface terminals of the core shall be identified using the **Signals** block of statements in CTL.

b) All nondigital interface terminals identified for the core shall be classified according to their electrical characteristics using the following statement:
signame {
ElectricalProperty property_type;
}

c) All digital interface terminals of the core shall be categorized according to their test function for all test modes using the following CTL statement:
signame {
DataType (data_type)+;
}

d) All digital terminals of IEEE 1500 wrapped or unwrapped cores with corresponding wrapper cells shall be identified in CTL using the following statement:
Internal {
signame {
IsConnected {
Wrapper IEEE1500 CellID cell_type;
}
}
}

e) All digital terminals of IEEE 1500 wrapped or unwrapped cores with no WBR cells or intended to have no WBR cells shall be identified in CTL using the following statement:

signame {
Wrapper None;
}

f) Active states of test terminals needed for the validity of test information of the core shall be specified using the **ActiveState** statement associated with the **DataType** statement as follows:
signame {
DataType data_type {**ActiveState** active_state;}
}

g) Certain terminals such as clock, test mode, and set/reset terminals are assumed to be at a certain state at the beginning of every test protocol for sequences to be valid. That assumption shall be specified using the **AssumedInitialState** statement associated with the **DataType** statement as follows:
signame {
DataType data_type {**AssumedInitialState** assumed_state;}
}

17.2.2 Description

Here is the reasoning behind the per-terminal rules:

a) There are many requirements to accurately refer to the core interface terminals. The test patterns will refer to them, and the core integrator will need to understand which terminals of the core have certain chip-level test interface connection requirements. Labelling the terminals is the first step required to pass along all of the important information from the core provider through to the core integrator.

b) CTL allows for the description of property types that include digital, analog, power, and ground terminals. Unless specified, the property type of a terminal is assumed to be digital since IEEE Std 1500 is targeted for digital terminals. Use of this property exempts some terminals of the core from other compliance rules.

c) A core comes with many types of test terminals that are used in special ways. For example, scan-related terminals serve a special function in the operation of a scan chain. These are to be identified using the **DataType** statement in CTL for all the test modes required for the associated compliance level.

d) The correspondence between the WBR cells and the core terminals is identified within the CTL that accompanies the wrapped core.

e) Digital terminals that do not have wrapper cells will be explicitly stated.

f) While more data types may have active states associated with them, the ones listed in this rule are required to be specified.

g) It is important to know if certain terminals need to always be at a specific value at the start of every migrated test, and this is the mechanism used to provide this information.

17.3 Test pattern information rules

The rules in this subclause pertain to requirements placed on the test patterns for the core.

17.3.1 Specifications

Rules

a) All test patterns shall be supplied using the IEEE P1450.6 (CTL) format, without making use of the **Foreign** pattern construct.

b) All WSC-related critical timing relationships and other design-specific critical timing relationships shall be specified in the CTL **DriveRequirements** and **StrobeRequirements** blocks.

c) Test patterns shall contain test protocols (**Macro** or **Procedure** blocks) in CTL so that the protocol does not assume that consecutive test patterns are (scan) overlapped.

d) Patterns used to initialize a test mode shall be separated from the test data patterns and identified as such using the **EstablishMode** keyword.

e) Any terminal that is not included in the supplied test pattern shall not affect the test result.

f) Values on terminals that are exempted from having WBR cells by IEEE Std 1500, with the exception of clocks and test signals, shall not affect the test results.

g) There shall be a sufficient set of patterns supplied with the core to validate every test protocol (**Macro** and **Procedure** blocks) provided in the CTL.

h) Provided that the full internal test pattern set is not supplied with the core (to the core integrator), the complete pattern set shall be made available to the device/chip manufacturer or testing company to use during production test.

Recommendations

i) Fault grading results for all internal patterns and/or pattern-bursts should be provided in CTL including the total number of faults, the number of faults detected, the number of redundant faults, the number of automatic test pattern generation (ATPG) untestable faults, and the number of possibly tested faults.

j) Timing information in the CTL **DriveRequirements** and **StrobeRequirements** blocks should be specified with acceptable margin to allow maximum flexibility for event placement.

17.3.2 Description

Here is the reasoning behind the test pattern information rules:

a) The only acceptable format for test patterns will be the format defined by IEEE P1450.6 (CTL). While other test pattern formats may be acceptable for certain tools, only IEEE P1450.6 patterns will be allowed for an IEEE 1500 core.

However, just saying “use CTL” is not sufficient for ensuring interoperability. CTL allows for the handling of legacy cores and their associated test patterns through the **Foreign** construct. While the **Foreign** construct can utilize the CTL infrastructure for scenarios that are not associated with the reuse needs for core test-pattern integration in an embedded environment, the **Foreign** construct is restricted by IEEE Std 1500 because of the test pattern reuse requirement. Patterns described in CTL without the **Foreign** construct have properties such as data-protocol separation that are critical for pattern mapping tasks. The **Foreign** construct allows for patterns to be supplied in private formats, waveform generation language (WGL), or STIL syntax, which do not separate data and protocol information. Thus, it is disallowed for IEEE 1500 cores.

b) **DriveRequirements** and **StrobeRequirements** are CTL constructs for specifying timing relationships between the pins of wrapped or unwrapped cores. These must be provided to ensure proper timing operation by the core integrator. Subclause 20.2 shows usage examples of the **DriveRequirements** and **StrobeRequirements** constructs.

c) CTL requires that the data and protocol be separated. Any test pattern will use a protocol that is defined in a **Macro** or **Procedure** blocks and identified with a purpose of **DoTest** or **DoTestOverlapped**. CTL allows for test protocols to come in both “flavors.” However, when unwrapped cores are provided with **DoTestOverlapped** patterns, the test pattern data that use the protocol need to have redundant parameters to allow for serialization of the parallel values in a **DoTestOverlapped** environment. Although this solution works in CTL, this is not a very streamlined approach.

The problem of overlapped tests may not be fully apparent, so the following example is provided to illustrate why overlapping of test is undesirable:

Suppose an unwrapped core has three patterns that are overlapped. The core contains one scan chain and has three parallel inputs (a, b, and c) and three parallel outputs (x, y, and z). The CTL pattern data might look as follows when the patterns are overlapped:

```
//Overlapped patterns for unwrapped core
//Note: the data values encode the pattern number for convenience in this example.

Pattern overlapped_3 {
P { si=001; so=xxx; abc=001; xyz=001; }
P { si=010; so=001; abc=010; xyz=010; }
P { si=011; so=010; abc=011; xyz=011; }
P { si=000; so=011; abc=000; xyz=xxx; }
}
```

In the above example, parallel stimulus and response data appear together—within the **Macro** application that is associated with that test. Note that the stimulus data are not significant on the final overlapped test, but the STIL syntax does not allow for stimulus of x.

When the patterns are modified to account for the wrapping of the core, the parallel data will have to be converted to scan data since these data are now applied through the wrapper chain. With this, data that were expected on xyz during the load of pattern number n in the unwrapped core patterns will appear during the load of pattern number n + 1. This requires the unwrapped core patterns to be modified after wrapping and to look as follows:

```
//Overlapped patterns for wrapped core

Pattern overlapped_3 {
P { si=001; so=xxx; abc=001; xyz=xxx; }
P { si=010; so=001; abc=010; xyz=001; }
P { si=011; so=010; abc=011; xyz=010; }
P { si=000; so=011; abc=000; xyz=011; }
}
```

In order to avoid this manipulation of the patterns and provide more flexibility during translation, patterns are required without overlapping.

d) Separating out the test setup patterns allows the core integrator to merge the setup patterns for multiple cores to ensure that all such cores being tested in parallel will enter their respective test modes prior to the application of any tests to any of the cores.

e) If the test pattern associated to a wrapped or unwrapped core does not include some core terminals, the values on these terminals shall not change the test result.

f) If there is one or more core terminals that do not have wrapper cells (perhaps because these terminals are analog inputs) and do not serve a test function as do clocks or scan enable terminals, the test patterns shall not presume any specific values on these pins because it is not possible to ensure that such values can be applied on these pins. Without the control provided by the IEEE 1500 wrapper mechanism, an unwrapped I/O must be assumed to be at an unknown state.

g) It is important for verification purposes that every migrated pattern work. Since test migration is effectively done by migrating/modifying the protocols without modifying the pattern data, the main tasks needing verification are the updated/migrated protocols.

h) Full pattern sets must eventually be accessible for production test.

i) Core users want to be able to determine the overall chip test coverage. To be able to compute this they will need to combine the fault counts for all of the cores and the surrounding

application-specific integrated circuit (ASIC) logic. Also, if they cannot apply all of the test patterns for whatever reason, it is important that they be able to know the effect on test coverage if certain tests are dropped.

j) Because of the generally unknown amount of delay in paths between the chip I/O pins and the internal core pins, it may be difficult to meet very strict timing requirements on test patterns for the core when they are migrated out to the chip pins. Cores can be provided with information that selects a certain working timing for the operations possible on it. An integrator can only try to achieve all the timing specification as depicted by the instance of the timing that is selected for the events. In reality, the core may have some rigid requirements and some flexibility in the timing. This should be shown in the information for the system integrator to successfully incorporate the core for the purposes of test.

18. Compliance definitions specific to unwrapped cores

18.1 General rules

The rules in this subclause pertain to any test logic within the unwrapped cores that will need to be understood prior to attempting to add or complete the wrapping of the core. Words in **bold** represent CTL (IEEE P1450.6) keywords.

18.1.1 Specifications

Rules

a) All core terminals assigned to share functional registers with the IEEE 1500 wrapper shall be provided with a complete WBR cell, which includes the functional registers.

b) All core terminals that cannot be wrapped without logical modification to the unwrapped core shall have a WBR cell built into the core.

c) Full control of any embedded WBR(s) shall be provided from core terminals.

d) If the state of the core relies on the stability of certain core input terminals during the scan Shift operation of the embedded environment, this information shall be specified using the **InputProperty** statement as follows:
signame {
InputProperty ScanStable;
}

e) If the core provider determines the state of the core environment relies on stability of certain core output terminals during the internal test operation of the core, this information shall be specified in CTL.

f) Every core shall come with at least one definition of an internal test mode for the core in CTL.

18.1.2 Description

Here is the reasoning behind the test logic information rules:

a) A functional I/O register can be reused as part of a WBR cell provided that the full WBR cell is implemented within the core.

b) Since it is not feasible to change the logic already built into a hard core or a black-box core, its unwrapped version shall have WBR cells built in to be compliant with this standard. For example, a core output driven by a tristate driver should have wrapper cells on both the data and enable inputs to the tristate driver. A wrapper cell cannot be added to the output of a tristate driver.

c) The existing WBR shall be fully controllable from the core's terminals so that it can be included as part of the core WBR when the core is wrapped with a IEEE 1500 wrapper.

d) If the test patterns for the core depend on certain terminals remaining stable during scan operations for the wrapper and in any logic surrounding the core, the core integrator needs to know this because it has implications on how the embedded core is connected to the TAM. The input property **ScanStable** specifies this requirement.

e) Any signal made stable to prevent damaging effect on other signals must be described using this statement. This would be identified by the following:
signame {
OutputProperty ScanStable;
DisableState ExpectOff;
IsDisabledBy Macro scan_macro_name
}

f) At least one core internal test mode must be fully defined in CTL so that test patterns for testing the core itself can be migrated to the SoC for that test mode.

18.2 Per-terminal rules

The rules in this subclause pertain to core interface terminals that require wrapping or that are already wrapped.

18.2.1 Specifications

Rules

a) The preferred wrapper cell type for every core terminal that is to be wrapped shall be described in CTL using the **ConnectTo** wrapper statement.

18.2.2 Description

Here is the reasoning behind the per-terminal rules:

a) For a wrapper to be automatically created for an unwrapped core, information must be provided to help the wrapper generator understand which type of wrapper cell should be used for each pin of the core. The **External** block in the CTL construct used to specify this information as follows:
signal_name {
ConnectTo{
Wrapper IEEE 1500 **CellID** cell_enum**;**
}}

The term *cell_enum* represents a WBR cell name as defined in 12.6.

18.3 Additional test information rules

18.3.1 Specifications

Rules

a) Any existing quiet or powered-down mode shall be defined in CTL using the **TestMode** keyword.

Recommendations

b) If a safe mode exists, its associated safe values shall be described in CTL using the following construct: **DataType** data_type {**ActiveState** active_state;}.

18.3.2 Description

Here is the reasoning behind the additional test information rules:

a) In order to perform I_{DDQ} testing for the SoC, it is important to ensure there are no high-current conditions within the design. Each embedded core should be placed into a quiet mode to minimize current drain during such testing. By specifying a quiet test mode, this simplifies the I_{DDQ} test generation process for the core integrator. Not providing a quiet mode definition may result in the core being placed into a state that consumes more current than would otherwise be necessary.

b) It is important that no damage be done to a core while testing of other cores or surrounding logic is being performed. To ensure this damage cannot happen, a core safe state should be defined (even for cores that have no unsafe states).

19. Compliance definitions specific to wrapped cores

19.1 General rules

The rules in this subclause apply to all IEEE 1500 wrapped cores and are not specific to individual terminals. Words in **bold** represent CTL (IEEE P1450.6) keywords.

19.1.1 Specifications

Rules

a) IEEE 1500 wrapper architecture and operation shall be compliant to all IEEE 1500 hardware rules described in this standard.

b) The information related to the length of the WIR, implemented public instructions, and their corresponding opcodes shall be provided in CTL.
   1) The WIR is to be defined using the **ScanStructure** construct.
   2) The scan cells composing the WIR are to be identified using the **ScanDataType** construct.
   3) The protocol operating the WIR is to be defined as **Instruction** using the **Purpose** construct.
   4) Instructions and their opcodes are to be defined using the **TestModeForWrapper** construct.

c) All IEEE 1500 wrapper terminals shall be described in CTL.

d) Available information about the safe state of input and output terminals shall be provided for all possible test operating modes.

Recommendations

e) The number of (tester) cycles needed to execute the test patterns should be specified for all the patterns of the core.

19.1.2 Description

Here is the reasoning behind the general rules for wrapped cores:

a) This rule reiterates that all of the wrapper logic must conform to the requirements and behaviors of the IEEE 1500 wrapper architecture as described in other clauses of this standard.

b) All wrapper-supported public instructions must be defined and described using CTL in order to ensure they can be utilized by standard software. Because instructions are considered test modes in CTL, the **TestModeForWrapper** construct is to be used to define instructions.

c) All wrapper pins must be described accurately in CTL in order for standard software to understand how to integrate the core into surrounding logic. Since the netlist for the wrapper may not be

provided, accurate descriptions of all wrapped pins are necessary in order for software to understand how to test the logic connected to each wrapped pin.

d) In order for the core integrator to know how to prevent unwanted conflicts from occurring during test, it is important to note the safe state values required on core/wrapper terminals.

e) The ability to schedule pattern application tasks on the SoC relies on this information being provided. Without this information, optimized scheduling of the tests would be difficult.

19.2 Per-terminal rules

The rules in this subclause pertain to specific external interface terminals of the wrapped core.

19.2.1 Specifications

Rules

a) All IEEE 1500 terminals that are used to operate the wrapper shall be identified with the following statement:
signame {
Wrapper IEEE1500 PinID pin_type;
}

19.2.2 Description

Here is the reasoning behind the per-terminal rules:

a) By specifying explicitly the function of special IEEE 1500 wrapper control terminals, there is no requirement that these terminals need to have specific pin names at the wrapped core boundary. The pin_type field is expected to be a member of the WSP terminal list described in 8.1.2.

19.3 Wrapper protocol information rules

19.3.1 Specifications

Rules

a) The protocol that operates the WIR through a scan operation to establish the test mode is required to shift values into all state elements of the WIR.

Recommendations

b) The Capture, Shift, and Update events of the protocol that operates the WIR should be identified in CTL for the wrapped core using the **Identifiers** syntax or as a purpose of the protocol.

19.3.2 Description

Here is the reasoning behind the wrapper protocol information rules:

a) Partial loads of the WIR are disallowed as they cause problems when multiple cores are integrated. A typical integration step would daisy-chain all the WIRs into a single scan chain. If the protocol that operates the WIRs does not shift the complete daisy-chained set of WIRs, the synchronized scan operation to establish the modes of the top-level scan chain of the SoC may not work.

b) A typical integration of wrapped cores would daisy-chain the WIRs of all the wrapped cores. To allow for this form of connection, the activities of the WIRs need to be synchronized. Since there is only one possible protocol at this time, this is a recommended practice.

20. IEEE 1500 application

This clause is informative only. Words in **bold** represent CTL (IEEE P1450.6) keywords.

20.1 CTL (IEEE P1450.6) overview

IEEE P1450.6 defines the CTL. A CTL description of an IEEE 1500 wrapper is required, so a brief overview of CTL is included in IEEE Std 1500 to explain the pertinent connection between the two standards.

CTL is a language for capturing and expressing test-related information for reusable cores, which is meant to co-exist with and complement information expressed as a netlist. It is utilized to describe IEEE 1500 wrappers. Test information about a core can be captured in CTL so that the SoC integrator or automation tools can successfully create a complete test for the SoC. Using the CTL description of a core, a wrapper can be constructed, and the appropriate TAM can be determined based on the test constraints in the CTL description. Once all the structures are in place, the test patterns that are also a part of the CTL description can be translated from the core boundary to the SoC boundary. CTL is the language to support all the information that the core provider needs to give the system integrator so that the latter can successfully test the embedded core and any UDL around the core. This language is broad enough to describe cores on which IEEE 1500 wrappers are to be implemented, IEEE 1500 wrapped cores, their different test methodologies, and the different ways in which they are integrated in the SoC.

One of the most important requirements for CTL is that the patterns, which contain the bulk of the data, are reusable without any modification whatsoever. This is accomplished by creating patterns using protocol statements (P statements) as opposed to vector statements (V statements) as used in traditional STIL (IEEE Std 1450). This allows the vector application protocol to be modified by the core integrator in an expeditious manner. Each pattern is identified by its intended purpose and associated test mode, so that a test synthesis tool can select and reorder as desired, again, without actually having to adjust the bulk pattern data.

CTL is both human and computer readable, as is STIL. Hence, it can be utilized for documentation purposes, as well as for driving chip test integration tools.

Every mode of the core can have a mechanism to initialize the mode. The initialization sequences are described in CTL on a per-mode basis. Some modes of the core contain test patterns with their associated timing information, constraints, and statistics. Some modes contain structural information so that the structure can be used to create patterns at another level of integration of the design. CTL is designed to describe all these needs of different modes of the core. For any given test mode of a core, typically only a subset of the full CTL syntax will be required to adequately describe the attributes of the mode.

At the basic level, **Signals** and **SignalGroups** are defined with their attributes. The following text explains CTL keywords. For complete definitions, refer to IEEE P1450.6.

Signals: This block defines each of the signal names of the core. Attributes can be attached to signals for additional information, e.g., **ScanIn** and **ScanOut** length, indicating the length of the scan chain connected to the signal.

SignalGroups: Signals can be clustered into signal groups. Also here attributes can be attached. A signal can occur in multiple signal groups, each time having different attributes.

Timing: Each parallel or scan vector in a pattern macro has an associated timing block to define the waveform and corresponding timing on each signal.

Pattern: This block contains the parallel and scan data for testing the core. Since pattern data are typically the bulk of the CTL file, they are often split off into a separate file.

MacroDefs: This block contains the protocol for applying test pattern data to the core or chip. Protocols for patterns provided with the core are written at the core boundary. Once integrated, this protocol is expanded to the chip level. The actual patterns are not changed in this process so they still reference the same macro name, but in fact a new, expanded macro definition has replaced the core-level definition.

PatternBurst: This collection of patterns represents a schedule and they are to be executed on the core. Patterns can be run serially or in parallel.

ScanStructures: This block can be used to describe in detail the internal scan chains of a core. In addition to the scan-in and scan-out port, a list of scan cells and clocking information is specified using the **ScanStructures** block.

Environment: This block is the top-level construct in CTL and is used to define the test modes for the core. Information in each test mode is categorized in subblocks such as the **Internal**, **External**, and **PatternInformation** blocks.

Internal: This block is used to describe the internal characteristics of the core signals. This information is provided to allow the core integrator to know the pertinent test information for each terminal of the core without needing full access to the design information. Examples of this information include wrapper type, timing accuracy required, and electrical characteristics such as analog or digital.

External: This block is used to describe the external characteristics that are expected from the perspective of the core boundary. Examples include connect to chip pins (input, output, bidirectional), connect to another named core, and connect to TAM, and connect to UDL.

PatternInformation: This block defines the purpose of each of the test patterns provided and the test mode necessary for the execution of each pattern. Other information like the fault model used and achieved coverage can also be given.

20.2 IEEE 1500 examples

A core and wrapper are depicted in Figure 41. Note that, whereas this figure shows some implementation details, IEEE Std 1500 defines only the behavior of the wrapper and leaves the implementation open. In this particular example, the wrapper can connect Core A both to a serial TAM, via WSI and WSO, as well as to a 3 bit parallel TAM, via WPI[0:2] and WPO[0:2]. There are six instructions that are described by the figures in this subclause, supporting normal mode and test modes with connections to either serial or parallel TAMs. Mode setting through the WIR actually means setting the controls of the wrapper's multiplexers. In Figure 41, the multiplexers that do occur in the wrapper are named m1, m2, ... ,m12. The dark circles in the multiplexers show which path is enabled if the multiplexer control signal is set to 1.

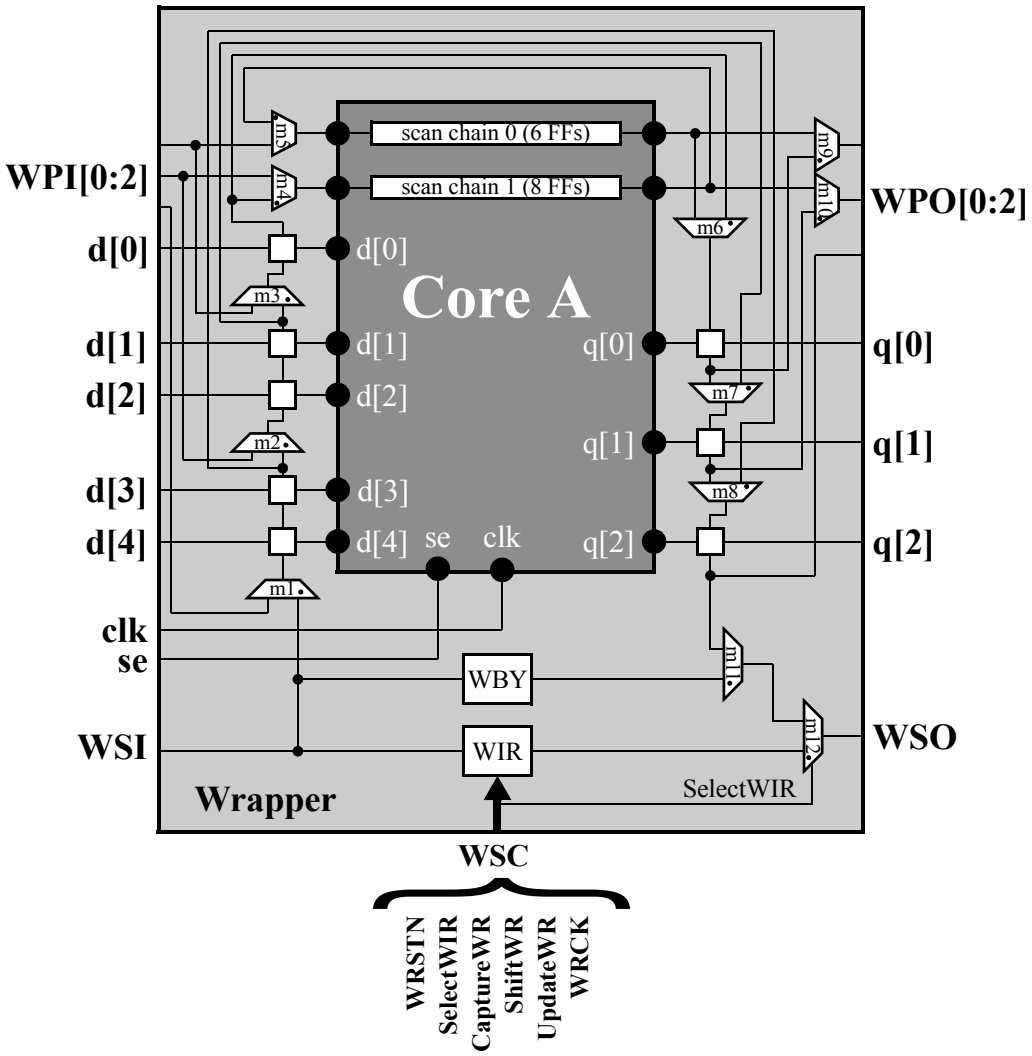


Figure 41—Example Core A and its IEEE 1500 wrapper

Table 7 lists the six instructions and the corresponding multiplexer settings they cause.

Table 7—Instruction decoding for multiplexer for the wrapper of Core A

Mode	Instruction	Opcode	Multiplexer settings											
			m1	m2	m3	m4	m5	m6	m7	m8	m9	m10	m11	m12
Normal	WS_BYPASS	0000	X	X	X	X	X	X	X	X	X	X	1	0
Serial bypass	WS_BYPASS	0000	X	X	X	X	X	X	X	X	X	X	1	0
Serial intest	WS_INTEST_ SCAN	1010	1	1	1	1	1	0	0	0	X	X	0	0
Serial extest	WS_EXTEST	1001	1	1	1	X	X	1	0	0	X	X	0	0

Table 7—Instruction decoding for multiplexer for the wrapper of Core A *(continued)*

Mode	Instruction	Opcode	Multiplexer settings											
			m1	m2	m3	m4	m5	m6	m7	m8	m9	m10	m11	m12
Parallel intest	WP_INTEST	1110	0	1	1	0	0	1	0	0	0	0	X	X
Parallel extest	WP_EXTEST	1101	0	0	0	X	X	1	1	1	1	1	X	X

In Figure 41 through Figure 44, the six modes are shown by bold highlighting of the active wires in a particular mode. The CTL description is included for three of the instructions: WS_BYPASS, WS_INTEST_SCAN, and WS_EXTEST.

20.2.1 WS_INTEST_SCAN

Figure 42 depicts the wrapper in its serial internal test mode in which the wrapper and internal scan chains are concatenated into one long scan chain. Following Figure 42 is the CTL code that describes the wrapper configuration and test for this. It is anticipated that the user will refer to IEEE P1450.6 for details on the CTL constructs shown in the code.

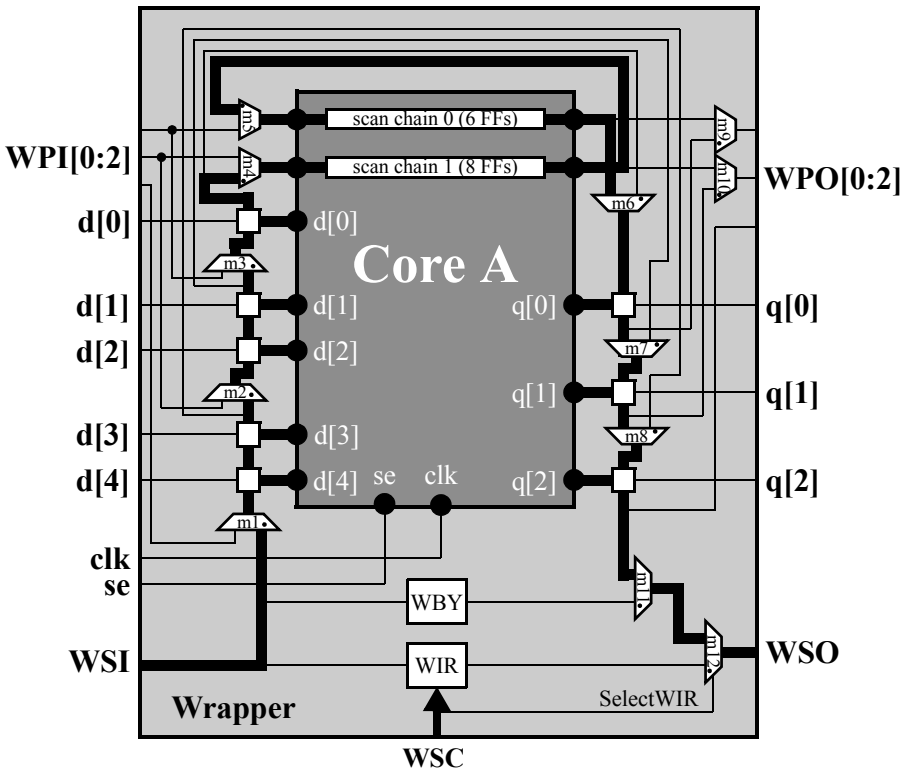


Figure 42—Serial intest mode using WS_INTEST_SCAN

```
STIL 1.0 {
      Design A-26-2003;
      CTL 2004;
}
Header {
      Title "Serial Intest";
      Date "2004";
}
Signals {
      WSI In {DefaultState N;}
      se In {DefaultState D;}
      clk In {DefaultState D;}
      d[0..4] In {DefaultState N;}
      WPI[0..2] In {DefaultState N;}
      WPO[0..2] Out;
      q[0..2] Out;
      WSO Out;
      WRSTN In {DefaultState U;}
      SelectWIR In {DefaultState D;}
      CaptureWR In {DefaultState D;}
      ShiftWR In {DefaultState D;}
      UpdateWR In {DefaultState D;}
      WRCK In {DefaultState D;}
}
SignalGroups {
      si_m123 = 'WSI' {ScanIn 22;}
      si_wir = 'WSI' {ScanIn 4;}
      so_wir = 'WSO' {ScanOut 4;}
      so_m123 = 'WSO'{ScanOut 22;}
      wsp[0..5] = 'WRSTN+SelectWIR+CaptureWR+ShiftWR+UpdateWR+WRCK';
}
Variables {
      SignalVariable si[0..1];
      SignalVariable so[0..1];
      SignalVariable instruction [0..3];
}

ScanStructures {
      ScanChain wir_chain {
            ScanLength 4;
            ScanIn WSI;
            ScanOut WSO;
            ScanCells wcell[0..3];
            ScanMasterClock WRCK;
      }
}


Timing {
      WaveformTable default_WFT {
            Period '100ns';
            Waveforms {
                  'wsp[0..5]+WPI[0..2]+d[0..4]+clk+se+WSI' {
                        01X {'5ns' D/U/N;}
                  }
                  'clk+WRCK' {
                        P {'0ns' D; '45ns' U; '55ns' D;}
                  }
                  'WPO[0..2]+q[0..2]+WSO' {
                        01X {'0ns' X; '95ns' L/H/X;}
                  }
            }
      }
}
```

```
MacroDefs {
      setup_intest {
            W default_WFT;
            V {WRSTN=0;}
            V {wsp[0..5] = 110100;}
            C {instruction=#;}
            Shift {V {si_wir= \W instruction[0..3]; WRCK=P;}}
            V {si_wir=X; WRCK=0; ShiftWR=0; UpdateWR=1;}
            V {WRCK=P;}
            V {WRCK = 0; SelectWIR=0; UpdateWR = 0;}
      }
      do_intest {
            W default_WFT;
            C {se=1; ShiftWR=1; si=#; so=#;}
            Shift {V {si_m123= \W si[1] \W si[0] \W d[0..4];
                  clk=P; WRCK=P;}}
            V {WRCK=P; ShiftWR=0; UpdateWR=1;}

            V {clk=0; WRCK=0; se=0; CaptureWR=1; ShiftWR=0;}
            V {clk=P; WRCK=P;}
            V {CaptureWR=0; clk=0; WRCK=0;}
            C {se=1; ShiftWR=1;}
            Shift {V {so_m123= \W q[2..0] \W so[1] \W so[0]; clk=P;
                  WRCK=P;}}

      }
}
PatternBurst all_pats {
      PatList {
            initSerialInternalTestMode {Protocol Macro setup_intest;}
            pat1 {Protocol Macro do_intest};
      }
}
PatternExec run_all {
      PatternBurst all_pats;
}
Environment wrapped_core {
      CTL serial_internal_test_mode {
            TestMode InternalTest;
            Compliancy IEEE1500 2004 Wrapped;
            TestModeForWrapper WS_INTEST_SCAN 1010;
            Internal {
                  'd[0..4]+q[0..2]+WPI[0..2]+WPO[0..2]' {
                                    DataType Unused;}

                  clk {
                        DataType TestControl
                        CaptureClock ScanMasterClock
                              {AssumedInitialState D;}
                        DriveRequirements {
                              TimingSensitive {
                                    Period Min '50ns';
                                    Pulse High Min '10ns';
                                                }
                        }
                  }

                  se {DataType TestControl ScanEnable {ActiveState U;}
                        DriveRequirements {
                              TimingSensitive {
                                    Reference clk {
                                          ReferenceEdge Leading;
                                          Setup '5ns'; Hold '5ns';
                                    }
```

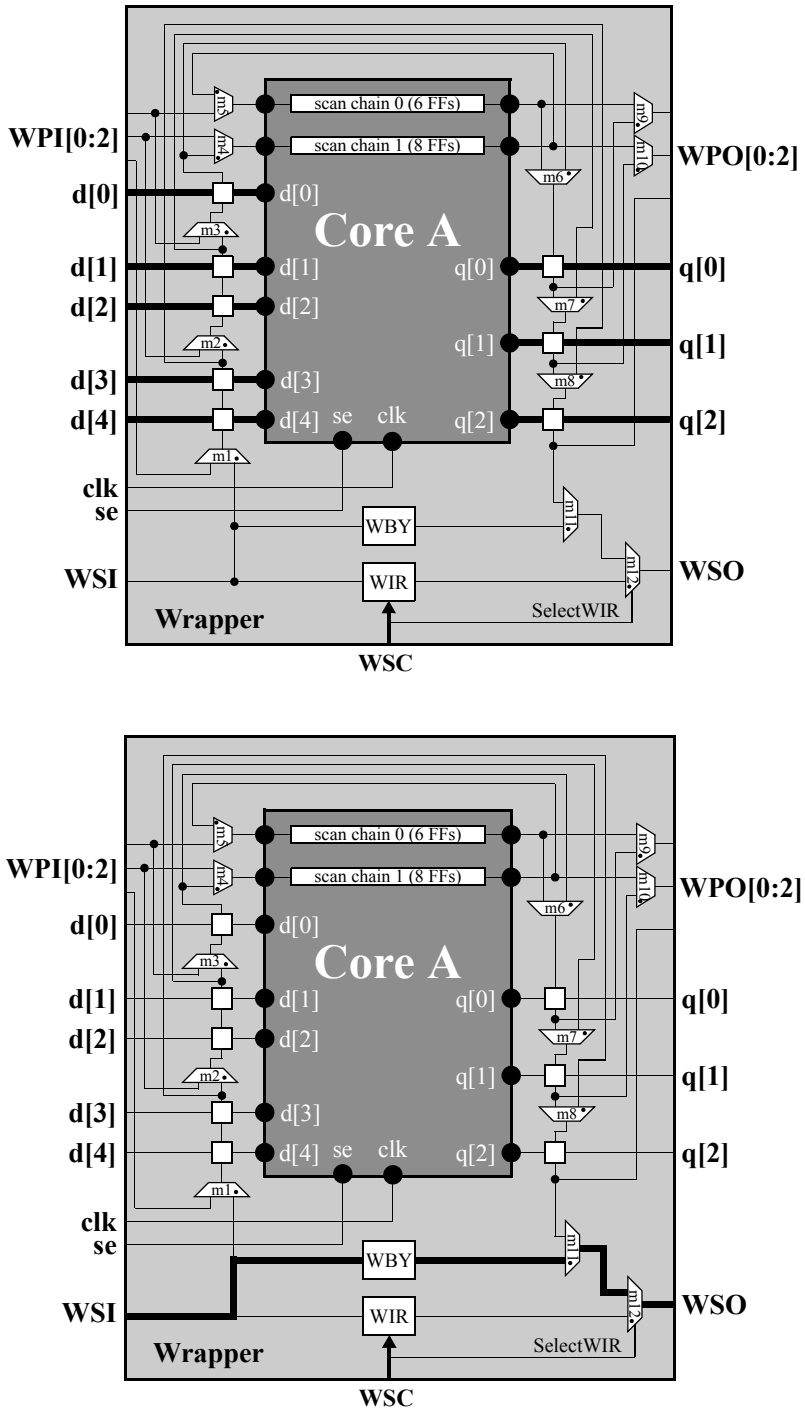


```
                  }
            }
      }

      si_m123 {DataType TestData ScanDataIn
                  {ScanDataType Boundary Internal;}
            DriveRequirements {
                  TimingSensitive {
                        Reference WRCK {
                              ReferenceEdge Leading;
                              Setup '5ns'; Hold '5ns';
                        }
                  }
            }

      }

      so_m123 {DataType TestData ScanDataOut
                  {ScanDataType Boundary Internal;}
            StrobeRequirements {
                  TimingSensitive {
                        Reference WRCK {
                              ReferenceEdge Trailing;
                              EarliestTimeValid '5ns';
                        }
                  }
            }
            LaunchClock WRCK {TrailingEdge;}

      }

      WRSTN {
            Wrapper IEEE1500 PinID WRSTN;
            DataType Asynchronous TestControl Reset {
                  ActiveState D;
                  AssumedInitialState U;
            }
      }

      SelectWIR {
            Wrapper IEEE1500 PinID SelectWIR;
            DataType TestControl TestWrapperControl;
      }

      CaptureWR {
            Wrapper IEEE1500 PinID CaptureWR;
            DataType TestControl {
                  ActiveState U;
            }
      }

      ShiftWR {
            Wrapper IEEE1500 PinID ShiftWR;
            DataType TestControl ScanEnable {
                  ActiveState U;
            }
      }
      UpdateWR {
            Wrapper IEEE1500 PinID UpdateWR;
            DataType TestControl {
                  ActiveState U;
            }
      }
      WRCK {
```

```
                        Wrapper IEEE1500 PinID WRCK;
                        DataType TestClock ScanMasterClock {
                              AssumedInitialState D;
                        }
                        DriveRequirements {
                              TimingSensitive {
                                    Period Min '50ns';
                                    Pulse High Min '10ns';
                                                }
                        }

                  }
            }
            External {
                  'clk+se+wsp[0..5]' {ConnectTo {DataType In;}}
                  WSI {ConnectTo {DataType In;}
                        ConnectTo {Wrapper IEEE1500 PinID WSO;}}
                  WSO {ConnectTo {DataType Out;}
                        ConnectTo {Wrapper IEEE1500 PinID WSI;}}
            }
            PatternInformation {
                  PatternExec run_all {
                        Purpose Production;
                        PatternBurst all_pats;
                        Fault {
                              Type StuckAt;
                              FaultCount 1092;
                              FaultsDetected 999;
                        }
                  }
                  Pattern initSerialInternalTestMode {
                        Purpose EstablishMode;
                  }
                  Macro setup_intest {
                        Purpose ModeControl;
                        UseByPattern EstablishMode;
                  }
                  Macro do_intest {
                        Purpose DoTest;
                  }
            }
      }
}

Pattern initSerialInternalTestMode {
      P {instruction[0..3]=1010;}
}

Pattern pat1 {
      P {d[0..4]=00000; si[0]=111000; si[1]=11110000;
            so[0]=001X11; so[1]=111100X1; q[0..2]=001;}
      P {d[0..4]=01101; si[0]=011010; si[1]=01011101;
            so[0]=1100X1; so[1]=10110000; q[0..2]=110;}
      P {d[0..4]=11001; si[0]=110010; si[1]=00011100;
            so[0]=010001; so[1]=1X110100; q[0..2]=00X;}
      P {d[0..4]=01010; si[0]=001101; si[1]=10011101;
            so[0]=11X000; so[1]=101110XX; q[0..2]=011;}
      P {d[0..4]=01111; si[0]=010001; si[1]=10101101;
            so[0]=0X1111; so[1]=00001X10; q[0..2]=000;}
}
```


20.2.2 WE_BYPASS

Note that both the multiplexer settings in Table 7 AND the highlighted active wires in Figure 43 show that the functional and serial bypass modes are nonconflicting and hence can be established by one combined instruction, WS_BYPASS. Following Figure 43 is the CTL code that describes the wrapper configuration and test.



a) Normal mode: WS_BYPASS

b) Bypass mode: WS_BYPASS

Figure 43—WS_BYPASS configuration

```
STIL 1.0 {
      Design A-26-2003;
      CTL 2004;
}
Header {
      Title "Bypass Mode - WS_BYPASS";
      Date "2004";
}
Signals {
      WSI In {DefaultState N;}
      se In {DefaultState D;}
      clk In {DefaultState D;}
      d[0..4] In {DefaultState N;}
      WPI[0..2] In {DefaultState N;}
      WPO[0..2] Out;
      q[0..2] Out;
      WSO Out;
      WRSTN In {DefaultState U;}
      SelectWIR In {DefaultState D;}
      CaptureWR In {DefaultState D;}
      ShiftWR In {DefaultState D;}
      UpdateWR In {DefaultState D;}
      WRCK In {DefaultState D;}
}
SignalGroups {
      si_byp = 'WSI' {ScanIn 1;}
      si_wir = 'WSI' {ScanIn 4;}
      so_byp = 'WSO'{ScanOut 1}
      wsp[0..5] = 'WRSTN+SelectWIR+CaptureWR+ShiftWR+UpdateWR+WRCK';
}
Variables {
            SignalVariable instruction[0..3];
}
MacroDefs bypass_mode_macros{
      operateScanChain {
            W default_WFT;
            C {ShiftWR=1; WRCK=0;}
            Shift {V {WSI=#; WSO=#; WRCK=P;}}
      }
}
ScanStuctures bypass_mode_chains {
      ScanChain bypassChain {
            ScanIn WSI;
            ScanOut WSO;
            ScanLength 1;
            ScanCells c0;
            ScanMasterClock WRCK;
      }
}
ScanStructures {
      ScanChain wir_chain {
            ScanLength 4;
            ScanIn WSI;
            ScanOut WSO;
            ScanCells wcell[0..3];
            ScanMasterClock WRCK;
      }
}


Timing {
      WaveformTable default_WFT {
            Period '100ns';
```



```
            Waveforms {
                  'wsp[0..5]+WPI[0..2]+d[0..4]+clk+se+WSI' {
                        01X {'5ns' D/U/N;}
                  }
                  'clk+WRCK' {
                        P {'0ns' D; '45ns' U; '55ns' D;}
                  }
                  'WPO[0..2]+q[0..2]+WSO' {
                        01X {'0ns' X; '95ns' L/H/X;}
                  }
            }
      }
}
MacroDefs {
      setup_testmode{
            W default_WFT;
            V {WRSTN=0;}
            V {wsp[0..5] = 110100;}
            Shift {V {si_wir= \W instruction [0..3]; WRCK=P;}}
            V {si_wir=X; WRCK=0; ShiftWR=0; UpdateWR=1;}
            V {WRCK=P;}
            V {WRCK = 0; SelectWIR=0; UpdateWR = 0;}
            }
}
Environment wrapped_core {
      CTL {
                  WRSTN {
                        Wrapper IEEE1500 PinID WRSTN;
                        DataType Asynchronous TestControl Reset {
                              ActiveState D;
                              AssumedInitialState U;
                        }
                  }
                  SelectWIR {
                        Wrapper IEEE1500 PinID SelectWIR;
                        DataType TestControl TestWrapperControl;
                  }
                  CaptureWR {
                        Wrapper IEEE1500 PinID CaptureWR;
                        DataType TestControl {
                              ActiveState U;
                        }
                  }
                  ShiftWR {
                        Wrapper IEEE1500 PinID ShiftWR;
                        DataType TestControl ScanEnable {
                              ActiveState U;
                        }
                  }
                  UpdateWR {
                        Wrapper IEEE1500 PinID UpdateWR;
                        DataType TestControl {
                              ActiveState U;
                        }
                  }
                  WRCK {
                        Wrapper IEEE1500 PinID WRCK;
                        DataType TestControl TestClock ScanMasterClock {
                              AssumedInitialState D;
                        }
                  }
            }
            External {
            'clk+se+wsp[0..5]' {ConnectTo {DataType In;}}
```

```
            WSI {ConnectTo {DataType In;}
                  ConnectTo {Wrapper IEEE1500 PinID WSO;}}
            WSO {ConnectTo {DataType Out;}
                  ConnectTo {Wrapper IEEE1500 PinID WSI;}}
            }
      }
      CTL bypass_test_mode {
            TestMode Bypass Normal;
            Compliancy IEEE1500 2004 Wrapped;
            TestModeForWrapper WS_BYPASS 0000;
            DomainReferences {
                  ScanStructures bypass_mode_chains;
                  MacroDefs bypass_mode_macros;
            }
            Internal {
                  'd[0..4]+q[0..2]+se' {DataType Functional;}
                  'WPI[0..2]+WPO[0..2]' {DataType TestData Unused;}
                  clk {
                        DataType TestControl MasterClock
                              {AssumedInitialState D;}
                        DriveRequirements { TimingSensitive {
                              Period Min '50ns';
                              Period Max '100ns';
                        }}
                  }
                  WSI {DataType TestData ScanDataIn
                              {ScanDataType ByPass;}}
                  WSO {DataType TestData ScanDataOut
                              {ScanDataType ByPass;}}
            }
            PatternInformation {
                  Pattern initByPass TestMode{
                        Purpose EstablishMode;
                        Protocol Macro setup_testmode;
                  }
                  Macro setup_testmode{
                        Purpose ModeControl;
                        UseByPattern EstablishMode;
                  }
                  Macro operateScanChain{
                        Purpose ControlObserve;
                        ScanChain bypassChain;
                  }
            }
      }
}
Pattern initByPass_TestMode {
      P {instruction[0..3]=0000;}
}
```


20.2.3 WS_EXTEST

Figure 44 depicts the wrapper in its serial external test mode. Following Figure 44 is the CTL code that describes the wrapper configuration and test for this.

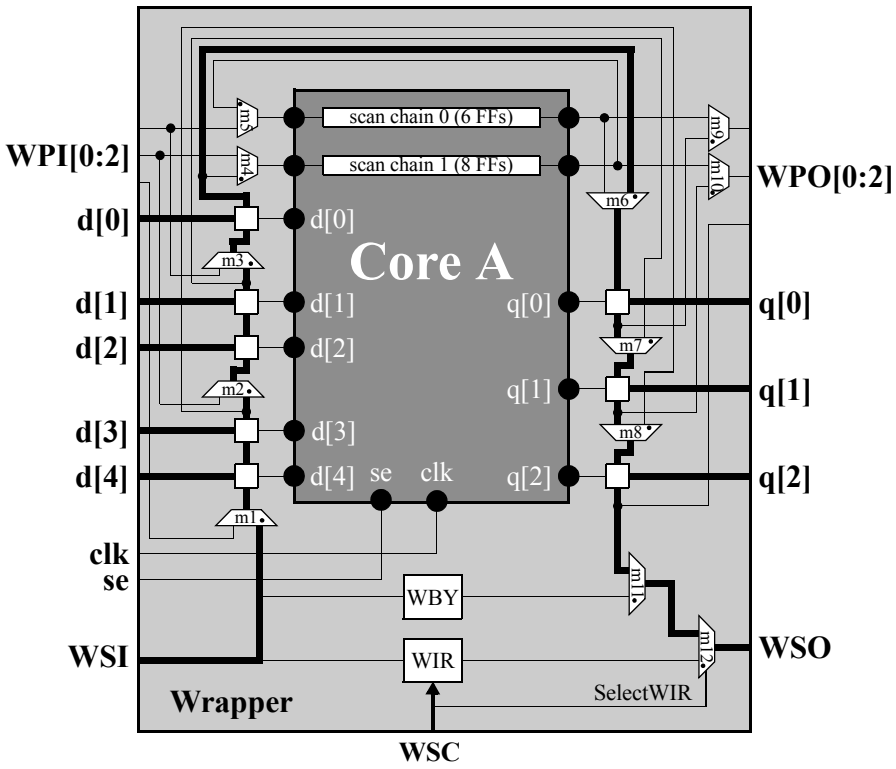


Figure 44—WS_EXTEST configuration

```
STIL 1.0 {
      Design A-26-2003;
      CTL 2004;
}
Header {
      Title "Extest Mode - WS_EXTEST";
      Date "2004";
}
Signals {
      WSI In {DefaultState N;}
      se In {DefaultState D;}
      clk In {DefaultState D;}
      d[0..4] In {DefaultState N;}
      WPI[0..2] In {DefaultState N;}
      WPO[0..2] Out;
      q[0..2] Out;
      WSO Out;
      WRSTN In {DefaultState U;}
      SelectWIR In {DefaultState D;}
      CaptureWR In {DefaultState D;}
      ShiftWR In {DefaultState D;}
      UpdateWR In {DefaultState D;}
```

```
      WRCK In {DefaultState D;}
}
SignalGroups {
      si_ext = 'WSI';
      si_wir = 'WSI';
      so_ext = 'WSO';
      wsp[0..5] = 'WRSTN+SelectWIR+CaptureWR+ShiftWR+UpdateWR+WRCK';
}
Variables {
            SignalVariable instruction[0..3];
}
MacroDefs extest_mode_macros{
      operateScanChain {
            W default_WFT;
            C {ShiftWR=1; WRCK=0;}
            Shift {V {WSI=#; WSO=#; WRCK=P;}}
      }
}
ScanStuctures extest_mode_chains {
      ScanChain wrapperChain {
            ScanIn WSI;
            ScanOut WSO;
            ScanLength 8;
            ScanCells c[0..7];
            ScanMasterClock WRCK;
      }
}
Timing {
      WaveformTable default_WFT {
            Period '100ns';
            Waveforms {
                  'wsp[0..5]+WPI[0..2]+d[0..4]+clk+se+WSI' {
                        01X {'5ns' D/U/N;}
                  }
                  'clk+WRCK' {
                        P {'0ns' D; '45ns' U; '55ns' D;}
                  }
                  'WPO[0..2]+q[0..2]+WSO' {
                        01X {'0ns' X; '95ns' L/H/X;}
                  }
            }
      }
}
MacroDefs {
      setup_testmode{
            W default_WFT;
            V {WRSTN=0;}
            V {wsp[0..5] = 110100;}
            Shift {V {si_wir= \W instruction[0..3]; WRCK=P;}}
            V {si_wir=X; WRCK=0; ShiftWR=0; UpdateWR=1;}
            V {WRCK=P;}
            V {WRCK = 0; SelectWIR=0; UpdateWR = 0;}
            }
}
Environment wrapped_core {
      CTL {
             Internal {
                  WRSTN {
                        Wrapper IEEE1500 PinID WRSTN;
                        DataType Asynchronous TestControl Reset {
                              ActiveState D;
                              AssumedInitialState U;
                        }
                  }
```

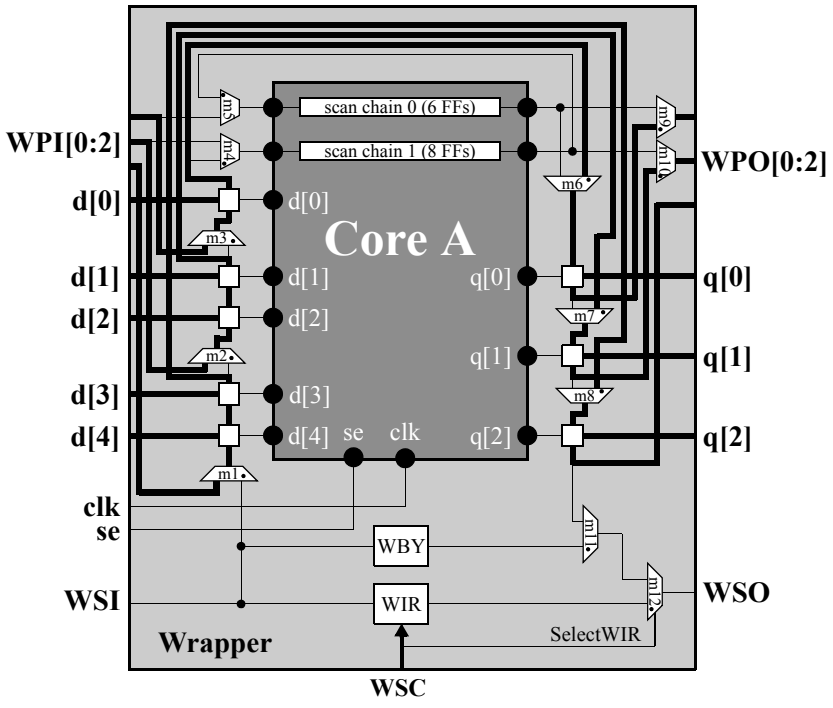


```
                SelectWIR {
                      Wrapper IEEE1500 PinID SelectWIR;
                      DataType TestControl TestWrapperControl;
                }
                CaptureWR {
                      Wrapper IEEE1500 PinID CaptureWR;
                      DataType TestControl {
                            ActiveState U;
                      }
                }
                ShiftWR {
                      Wrapper IEEE1500 PinID ShiftWR;
                      DataType TestControl ScanEnable {
                            ActiveState U;
                      }
                }
                UpdateWR {
                      Wrapper IEEE1500 PinID UpdateWR;
                      DataType TestControl {
                            ActiveState U;
                      }
                }
                WRCK {
                      Wrapper IEEE1500 PinID WRCK;
                      DataType TestControl TestClock ScanMasterClock {
                            AssumedInitialState D;
                      }
                }
          }
          External {
                'clk+se+wsp[0..5]' {ConnectTo {DataType In;}}
                WSI {ConnectTo {DataType In;}
                      ConnectTo {Wrapper IEEE1500 PinID WSO;}}
                WSO {ConnectTo {DataType Out;}
                      ConnectTo {Wrapper IEEE1500 PinID WSI;}}
          }
    }
    CTL extest_test_mode {
          TestMode ExternalTest;
          Compliancy IEEE1500 2004 Wrapped;
          TestModeForWrapper WS_EXTEST 1001;
          DomainReferences {
                ScanStructures extest_mode_chains;
                MacroDefs extest_mode_macros;
          }
          Internal {
                'd[0..4]' { DataType Functional TestData;
                      IsConnected In {
                            StateElement Scan 'c[4..0]';
                            CaptureClock WRCK {
                                  LeadingEdge;
                                  StateAfterClock Connection;
                            }
                      }
                }
                'q[0..2]' {DataType Functional TestData;
                      IsConnected Out {
                            StateElement Scan 'c[5..7]';
                            LaunchClock WRCK {
                                  LeadingEdge;
                                  StateAfterClock ExpectUnknown;
                            }
                      }
                }
```

```
                    se {DataType Functional Unused;}
                    'WPI[0..2]+WPO[0..2]' {DataType TestData Unused;}
                    clk {
                          DataType TestControl MasterClock
                          UseByPattern EstablishMode;
                    }
                    Macro operateScanChain{
                          Purpose ControlObserve;
                          ScanChain wrapperChain;
                    }
              }
        }
}
Pattern initExtest_TestMode {
        P {instruction[0..3]=1001;}
}
```

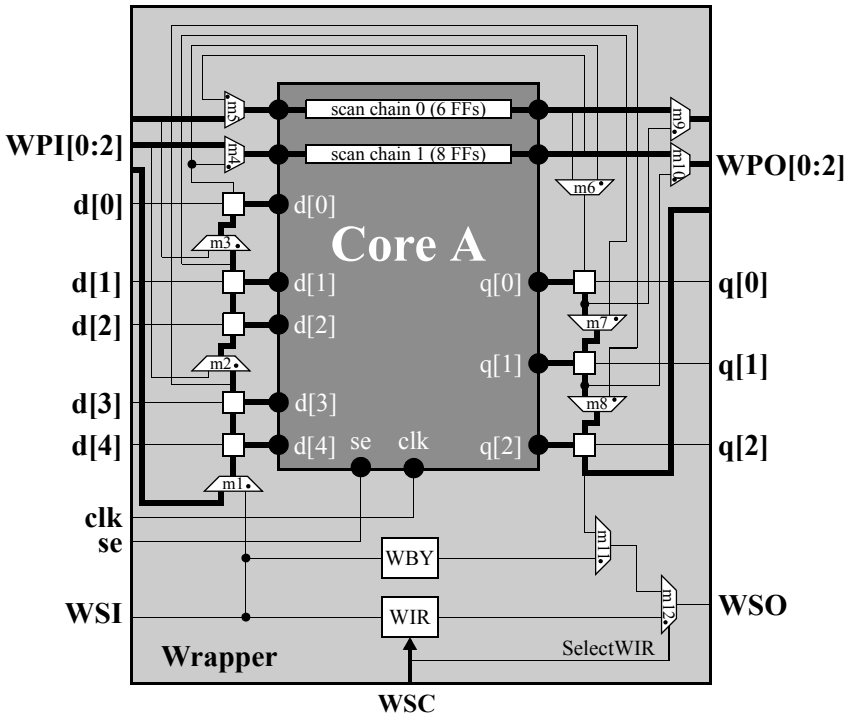
20.2.4 WP_EXTEST and WP_INTEST

Figure 45 illustrates the configuration of the wrapper for the parallel extest and parallel intest modes.



a) Parallel extest mode: WP_EXTEST

Figure 45—Parallel extest and parallel intest modes



b) Parallel intest mode: WP_INTEST

Figure 45—Parallel extest and parallel intest modes *(continued)*

Annex A

(normative)

Bubble diagram definition

In IEEE Std 1500, an attempt has been made to focus on architecture and remain neutral to implementation styles. In this spirit, a new graphical vocabulary has been implemented to describe hardware behavior visually without describing the physical implementation of the hardware. Informally, diagrams drawn using these symbols have been called *bubble diagrams* because the much-used symbol for a storage element is a circle.

The symbols to describe behavior are few. In fact, there are only four as shown in Figure A.1. The first is the symbol for a storage element, which is a circle. The second is a data path, which is represented as a line with an arrowhead indicating data flow direction. The third is a dataflow decision point, which is represented as a vertical line with no arrowhead. The fourth symbol is a connection point, represented as a small filled circle located at the junction of two datapaths.

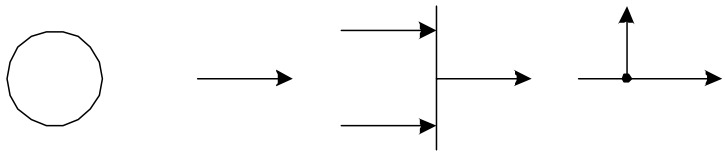


Figure A.1—The four symbols used in bubble diagrams

The storage element has a characteristic associated with it that indicates the events to which it responds. This characteristic is displayed as the presence of one or more characters inside the circle from the set S, C, T, U, and F indicating the Shift, Capture, Transfer, Update, and any Functional event, respectively. Figure A.2 shows various storage elements.

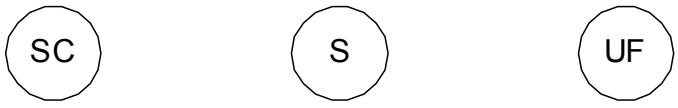


Figure A.2—Various storage elements

Shift data flows from CTI to CTO and can go through multiple storage elements represented by the bubbles. Capture data come into a storage element from the CFI terminal or the CFO terminal. Transfer data come from the input sourced from another storage element. Update data come from the shift path storage element closest to CTO. Functional data come from CFI.

To show how these elements are used together, a IEEE 1149.1 BC_1 cell using the IEEE 1500 cell notation is shown in Figure A.3. The BC_1 cell supports the Shift, Capture, Update, and Apply events. It should be noted that the Apply event is a virtual event, composed of other events or modes, and, therefore, is not specifically represented in these diagrams.

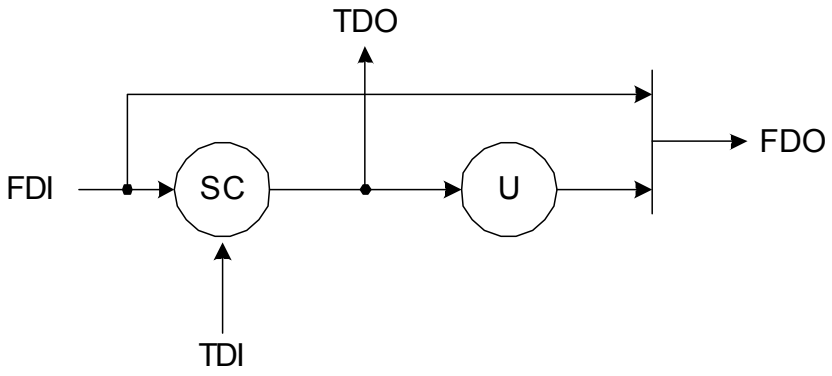


Figure A.3—Example of IEEE 1149.1 BC_1 cell

For further examples of usage, refer to Annex B where various bubble diagrams are reduced to implementation examples.

Annex B

(informative)

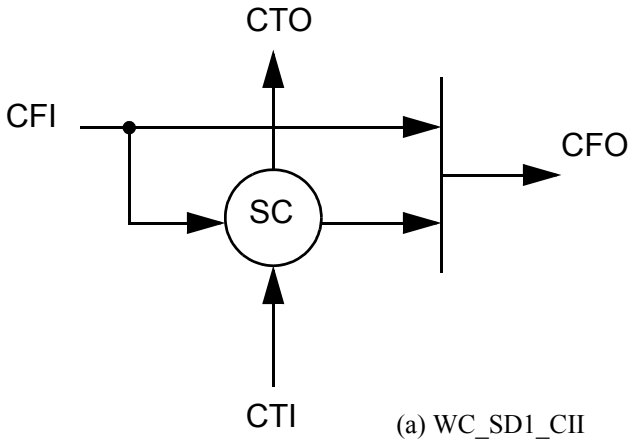
WBR cell examples

Table B.1 describes IEEE 1500 WBR cell example names, and a gallery of bubble diagrams depicting each example follows the table. The bubble diagrams used in these examples are defined in Annex A. It is understood that the means of data path selection shown in these bubble diagram figures is to be configured by the content of the WIR.

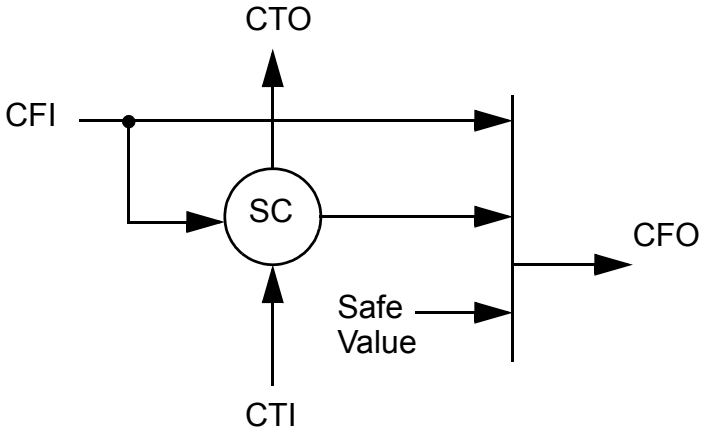
Table B.1—WBR cell examples

Cell description	Name	Figure number
One storage element in shift path dedicated to wrapper function.	WC_SD1_CII	Figure B.1
Two storage elements in the shift path with one shared with functional operation.	WC_SF2_CIO	Figure B.2
Two dedicated storage elements in the shift path.	WC_SD2_CII	Figure B.3
One dedicated storage element in the shift path and a dedicated update storage element.	WC_SD1_CII_UD	Figure B.4
Two dedicated storage elements in the shift path and a shared update storage element.	WC_SD2_CIU_UF	Figure B.5
n dedicated storage elements in the shift path and a dedicated update storage element	WC_SDn_CII_UD	Figure B.6
A reduced-functionality cell performing control-only function with one dedicated storage element in the shift path.	WC_SD1_CN	Figure B.7
One dedicated storage element in the shift path, selectively capturing from CFI or CFO, and a dedicated update storage element.	WC_SD1_CBI_UD	Figure B.8

WC_SD1_CII has the advantage of a dedicated shift path, however, depending on the instruction, the CFO terminal may toggle during Shift or Capture operations. The connection between CFI and CFO may be difficult to test. WC_SD1_CII_G may be forced to a safe value in addition the supporting the functionality described for WC_SD1_CII. See Figure B.1.



(a) WC_SD1_CII



(b) WC_SD1_CII_G

Figure B.1—WC_SD1_CII WBR cell

WC_SF2_CIO is similar to WC_SD2_CIO except that functional operation and test operation share the sequential element closest to CTO. This cell may be used to meet a test objective in which the capture and apply paths are identical. WC_SF2_CIO_G may be forced to a safe value in addition the supporting the functionality described for WC_SF2_CIO. See Figure B.2.

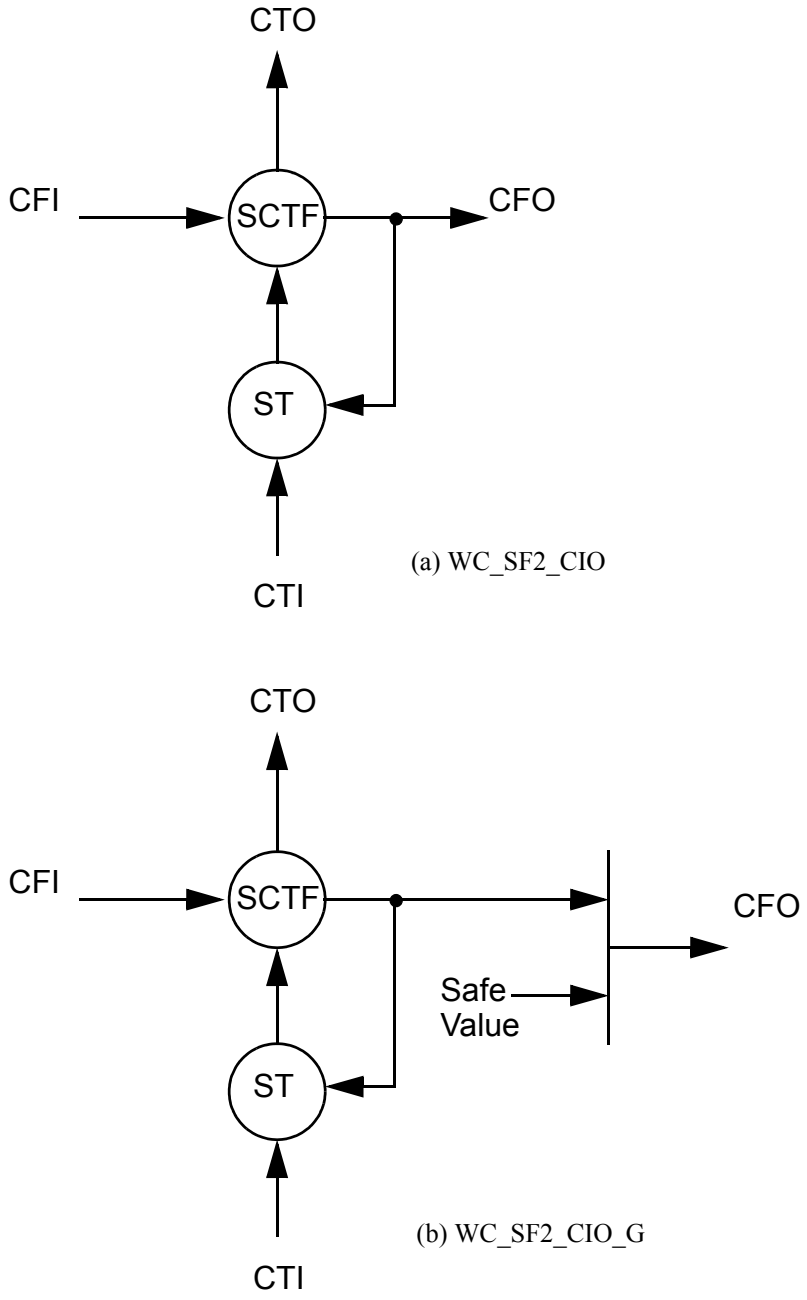


Figure B.2—WC_SF2_CIO WBR cell

WC_SD2_CII and WC_SD2_CII_G support the same functionality as the WC_SD1_CII and WC_SD1_CII_G, respectively, and additionally support a transition test application. See Figure B.3.

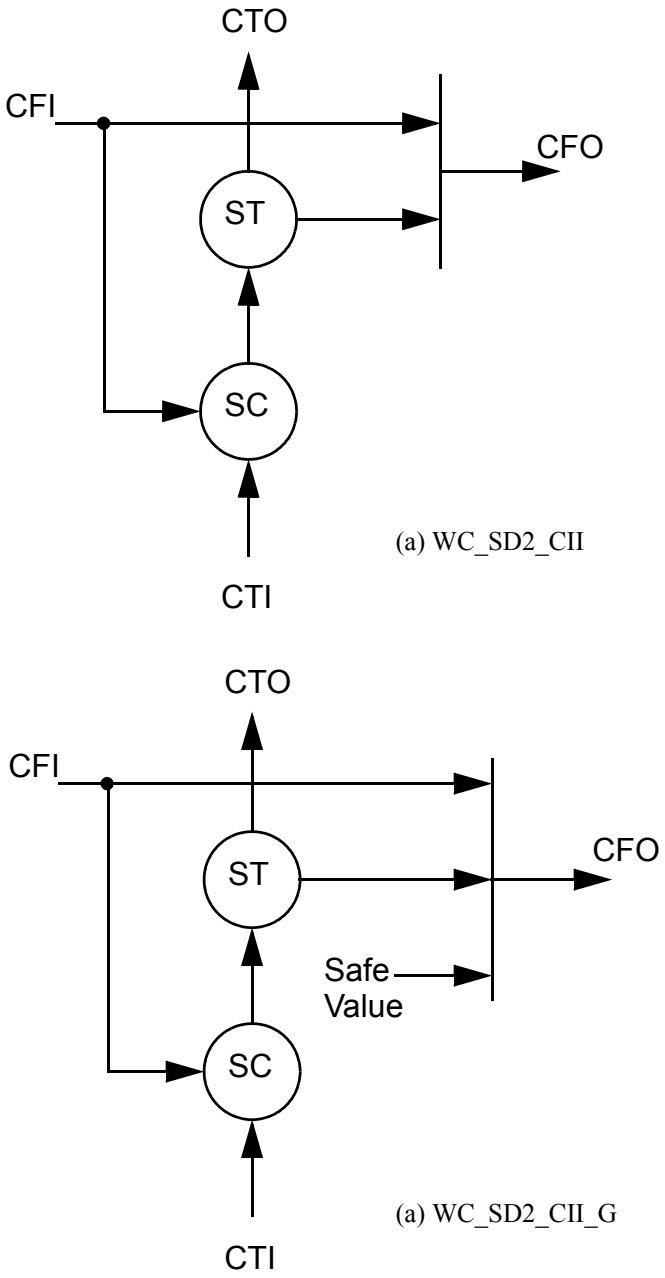


Figure B.3—WC_SD2_CII WBR cell

WC_SD1_CII_UD corresponds to an IEEE 1149.1 BC_1 cell. It has dedicated shift path and a dedicated update storage element. WC_SD1_CII_UD_G may be forced to a safe value in addition the supporting the functionality described for WC_SD1_CII_UD. See Figure B.4.

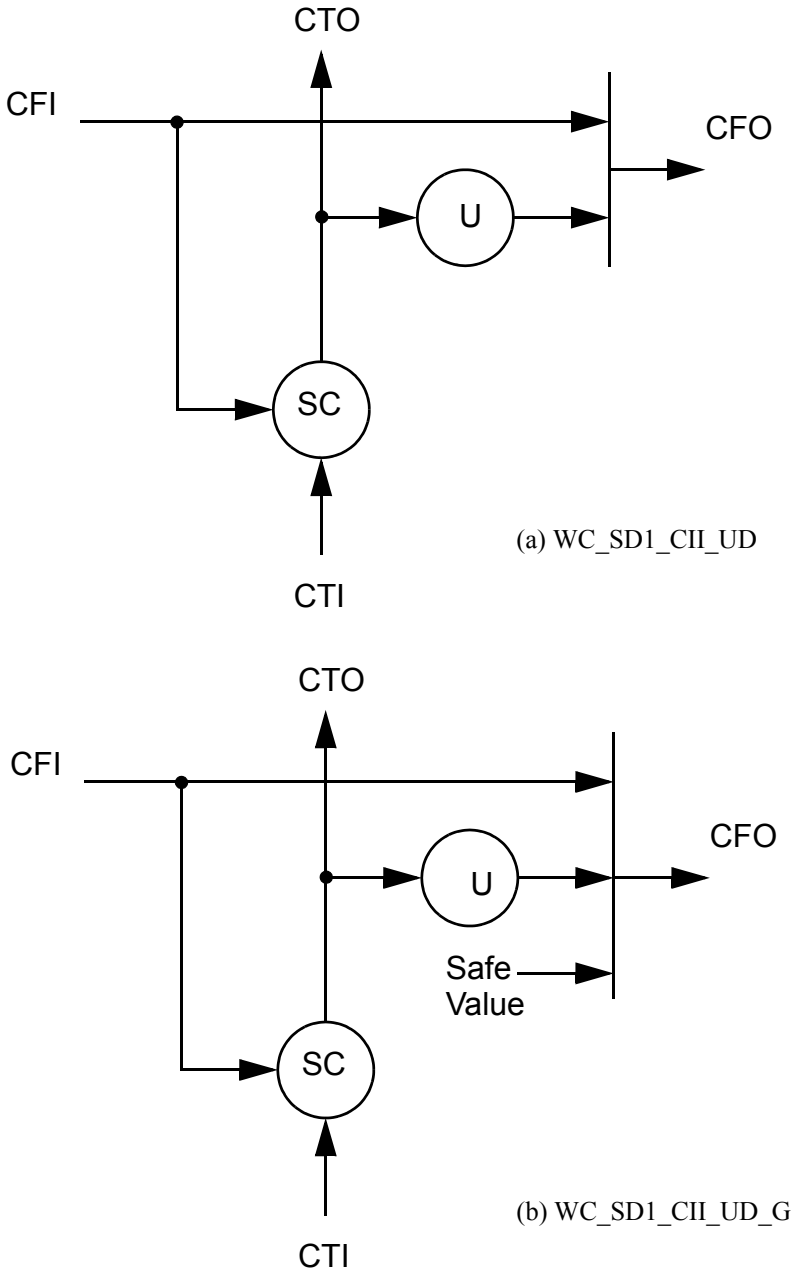


Figure B.4—WC_SD1_CII_UD WBR cell

WC_SD2_CIU_UF and WC_SD2_CIU_UF_G support the same functionality as the WC_SD1_CIU_UF and WC_SD1_CIU_UF_G, respectively (see Figure 24) except that additionally they have two shift path storage elements and, therefore, may be used for transition test applications. See Figure B.5.

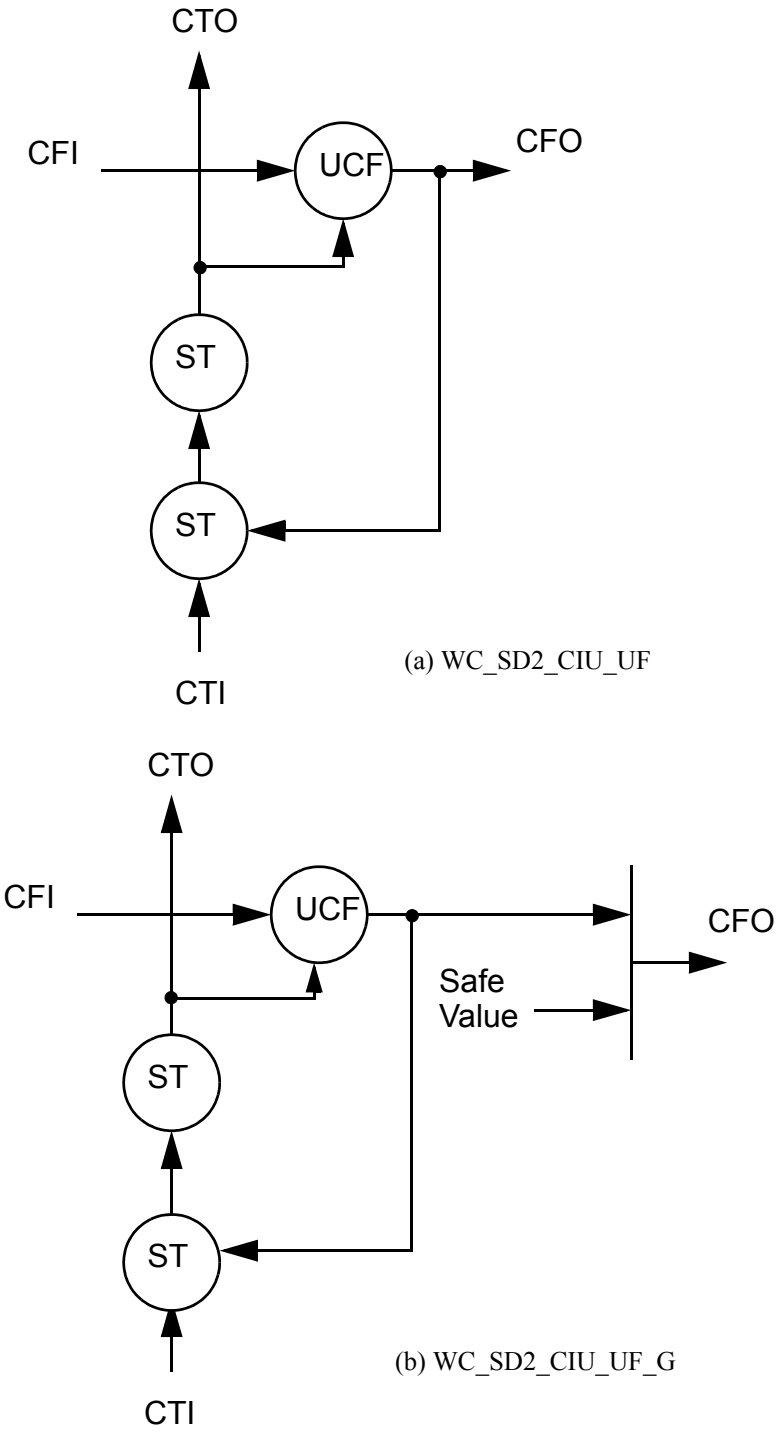


Figure B.5—WC_SD2_CIU_UF WBR cell

WC_SDn_CII_UD and WC_SDn_CII_UD_G support the same functionality as the WC_SD1_CII_UD and WC_SD1_CII_UD_G, respectively, except that additionally they have two or more shift path storage elements and, therefore, may be used for transition test applications. Similar to the BC_1, the capture sites are the storage elements closest to CTI. See Figure B.6.

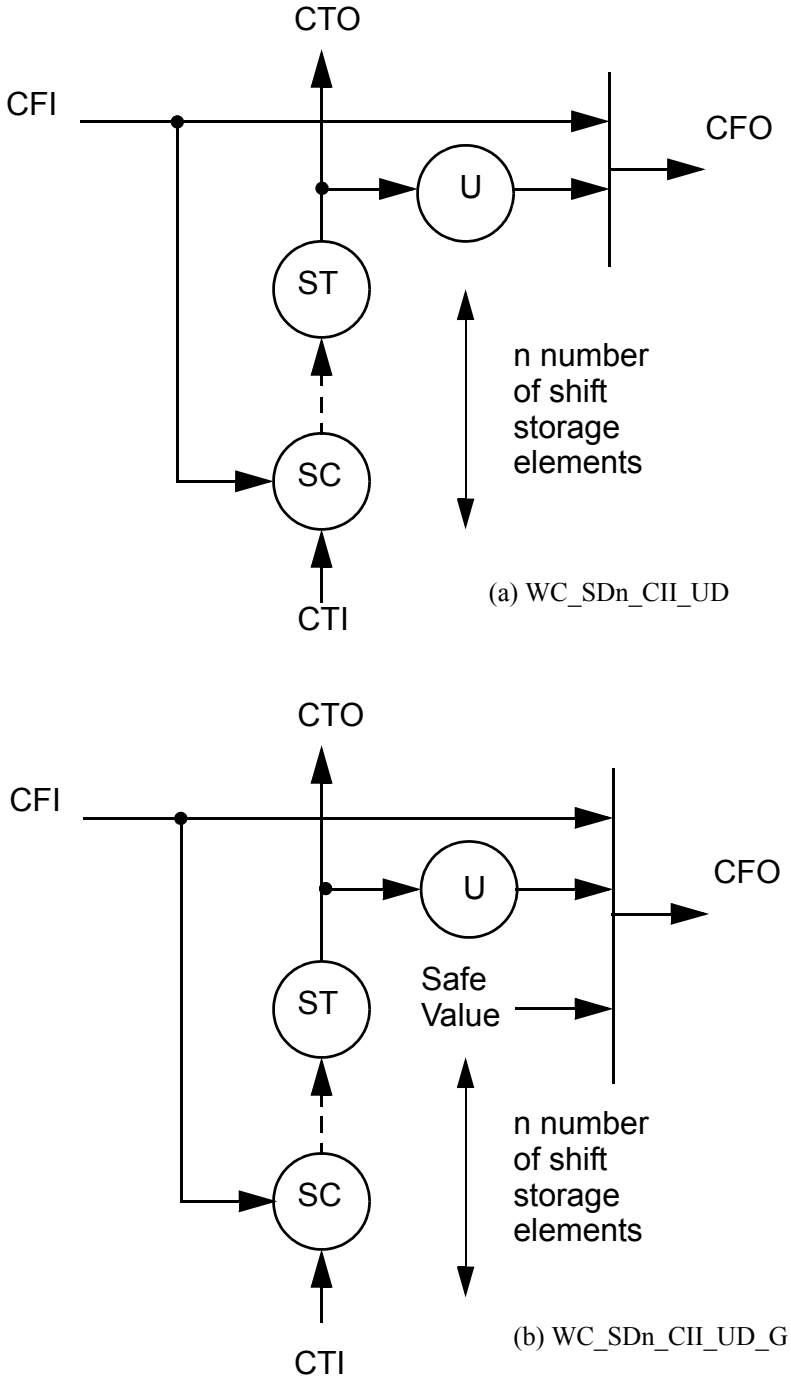


Figure B.6—WC_SDn_CII_UD WBR cell

WC_SD1_CN is a control-only harnessing cell. This cell is to be used only in accordance with permission 12.2.1(g). See Figure B.7.

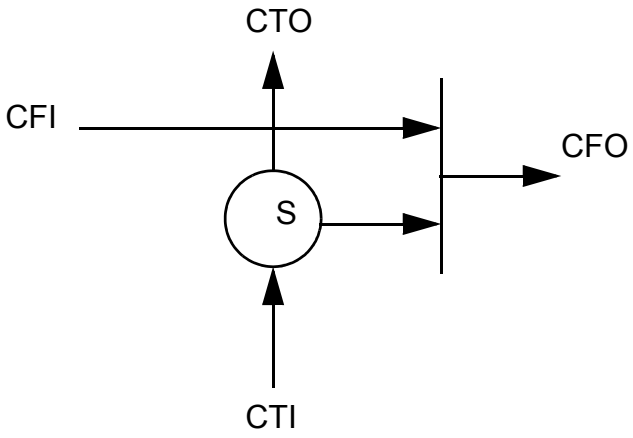
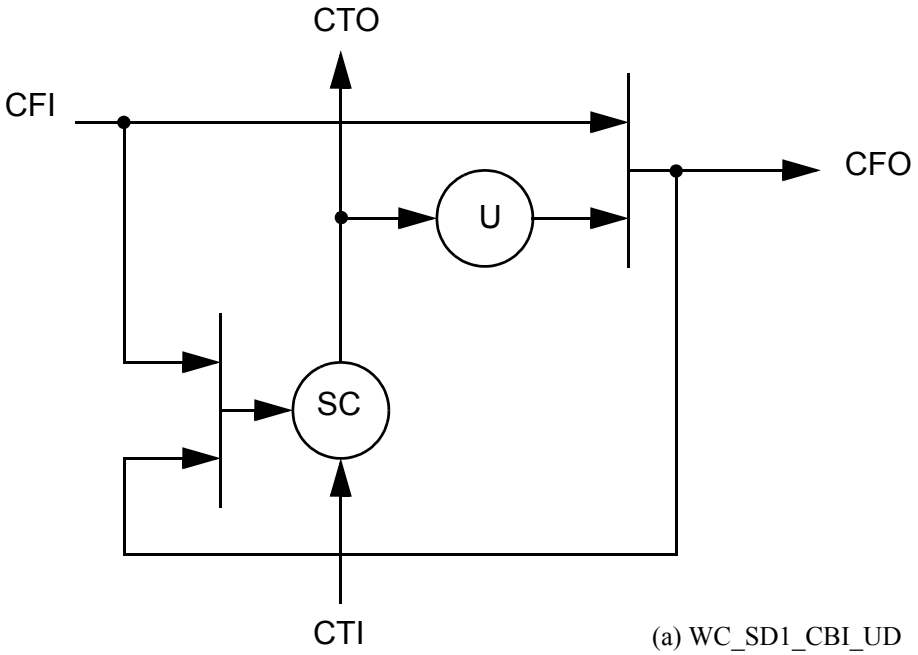
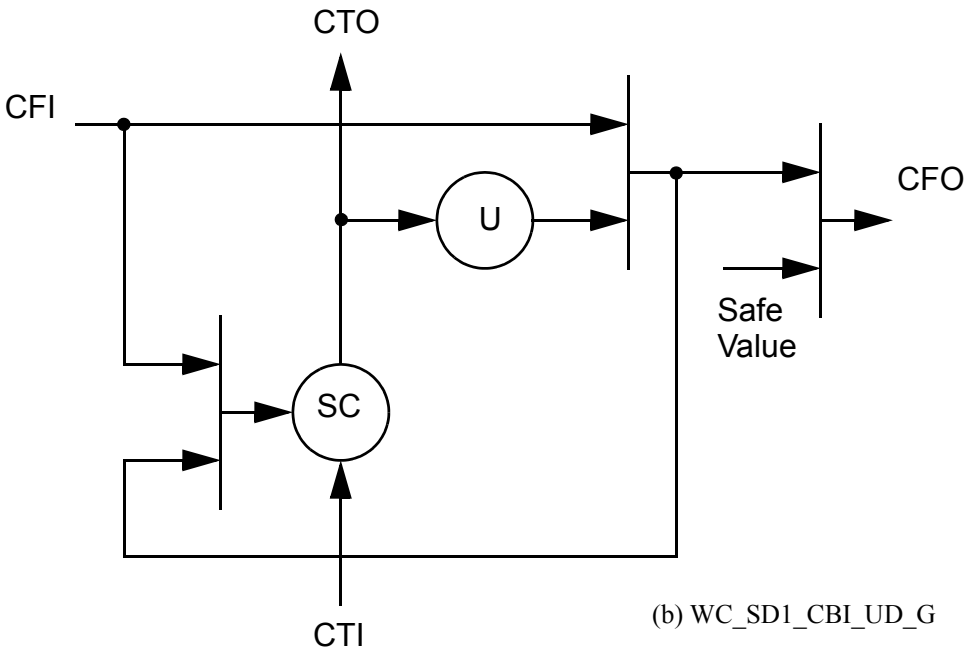


Figure B.7—WC_SD1_CN WBR cell

WC_SD1_CBI_UD and WC_SD1_CBI_UD_G support the same functionality as the WC_SD1_CII_UD and the WC_SD1_CII_UD_G cells, respectively, except additionally the origin of the capture data is selectable between CFI and CFO. See Figure B.8.



(a) WC_SD1_CBI_UD



(b) WC_SD1_CBI_UD_G

Figure B.8—WC_SD1_CBI_UD WBR cell

Annex C

(informative)

Relationship of IEEE Std 1500 to IEEE Std 1149.1

The reader will likely note that this standard bears much resemblance to IEEE Std 1149.1, and that is not accidental. IEEE Std 1149.1 has proven to be very valuable to the electronics industry and has been widely adopted. While the objectives of both IEEE Std 1149.1 and IEEE Std 1500 are to provide improvements in testing, their focus differs significantly. It is natural, therefore, that some divergence exists between the two standards. IEEE Std 1149.1 is primarily concerned with board testing and secondarily concerned with internal IC testing. IEEE Std 1500 is primarily concerned with testing embedded cores and UDL within an IC.

The IEEE 1500 architecture was designed to allow interface compatibility with the IEEE 1149.1 test access port (TAP) controller. Indeed, the wrapper's WSC interface (with the exception of the optional TransferDR signal) can correspond to the control outputs from the IEEE 1149.1 TAP controller. For example, the WSP timing diagram in Figure 31 (see 14.2) is compatible with the protocol control issued from a TAP controller. Thus, while IEEE Std 1500 does not require or suggest it, the WSC interface may be operated by an IEEE 1149.1 TAP controller if the SoC designer wishes to do so to allow access to IEEE 1500 wrappers via the dedicated TAP pins on the SoC. This would advantageously allow IEEE 1500 to be accessible at any point in the SoC life cycle.

A consideration of IEEE 1500 has been to facilitate delay testing of cores and UDL within SoCs. The two and a half cycles of IEEE 1149.1 test clock (TCK) latency between the TAP's UpdateDR and CaptureDR states limits the TAP protocol from being able to apply some delay tests. Using the IEEE 1500 WSP interface signals directly (i.e., not via a TAP) allows for delay testing since there are no protocol timing restrictions between any desired IEEE 1500 WSC event. For example, a Capture event may immediately follow either an Update or a Transfer event to facilitate delay testing. In summary, it is possible to design an interface between an IEEE 1149.1 TAP and IEEE 1500 WSC so that the WSC may be operated by the TAP to achieve all wrapper test operations with the exception of delay testing and use of the Transfer event as shown in C.1.

C.1 Sample IEEE 1149.1 TAP controller interface

To reduce the congestion of test control interface wiring of IEEE 1500 wrappers in SoCs and/or to consolidate the operation of IEEE 1500 wrappers with IEEE 1149.1-based standards (1149.1, 1149.4, 1149.5, 1149.6, 1532, and 5001) in SoCs, the use of an IEEE 1149.1 TAP controller may be required. The following description provides insight into one way of interfacing the IEEE 1500 wrapper's WSP to an IEEE 1149.1 TAP controller.

Figure C.1 illustrates the WSP of an IEEE 1500 wrapper being interfaced to an IEEE 1149.1 TAP controller. As shown, the standard reset and select outputs of the IEEE 1149.1 TAP controller can be directly connected to the WRSTN and SelectWIR inputs of the IEEE 1500 WSP, respectively. However, the WRCK, ShiftWR, CaptureWR, and UpdateWR signals of the WSP must be interfaced to the IEEE 1149.1 TAP controller via glue logic. This implementation does not support the Transfer event. An example of glue logic implementation is shown in Figure C.2.

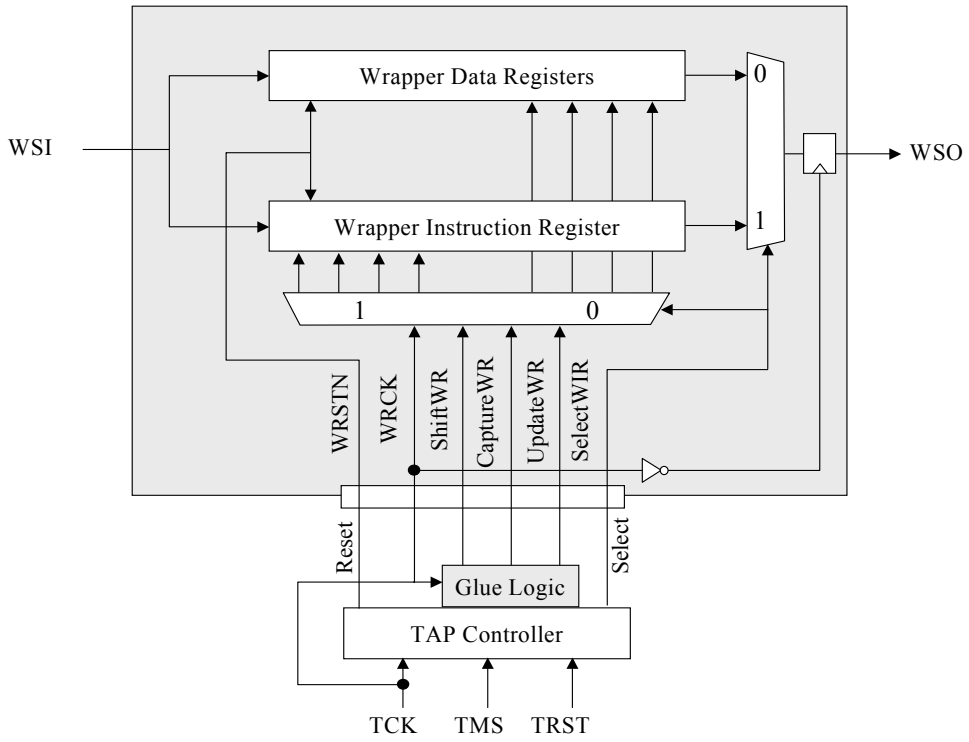


Figure C.1—IEEE 1500 WSP and IEEE 1149.1 TAP controller interface

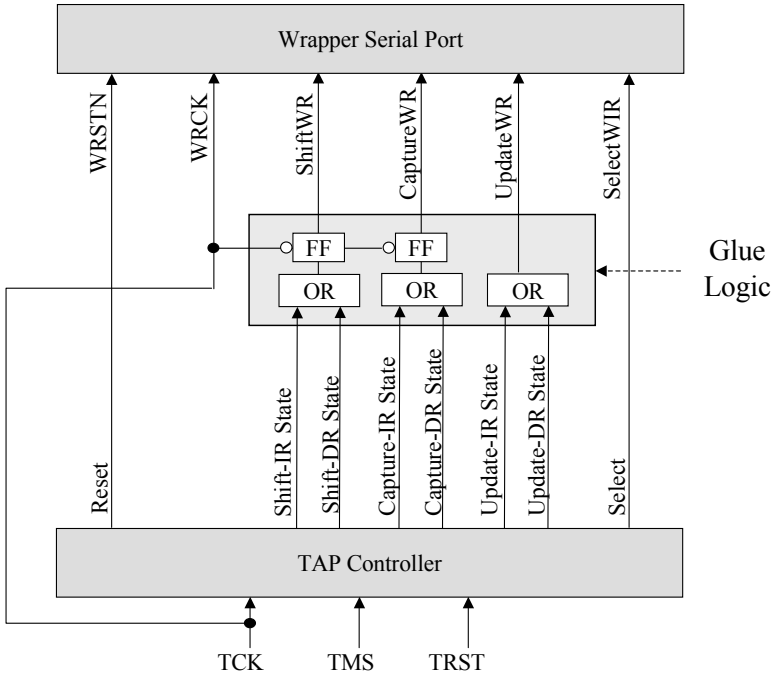


Figure C.2—IEEE 1149.1 TAP controller interface logic

Figure C.2 illustrates a simple implementation of the glue logic of Figure C.1, which interfaces a standard IEEE 1149.1 TAP controller to an IEEE 1500 WSP interface via logical OR functions. This figure shows a conceptual model and appropriate timing should be verified for proper operation. As seen in Figure C.2,

— An IEEE 1500 WRCK event occurs in response to an IEEE 1149.1 ClockIR OR ClockDR event.

— An IEEE 1500 ShiftWR event occurs in response to the IEEE 1149.1 TAP controller being in either the Shift-IR State OR the Shift-DR state.

— An IEEE 1500 CaptureWR event occurs in response to the IEEE 1149.1 TAP controller being in either the Capture-IR state OR the Capture-DR state.

— An IEEE 1500 UpdateWR event occurs in response to an IEEE 1149.1 TAP controller UpdateIR OR UpdateDR event.

Figure C.3 illustrates IEEE 1500 Capture, Shift, and Update events generated using the IEEE 1149.1 TAP controller shown in Figure C.1.

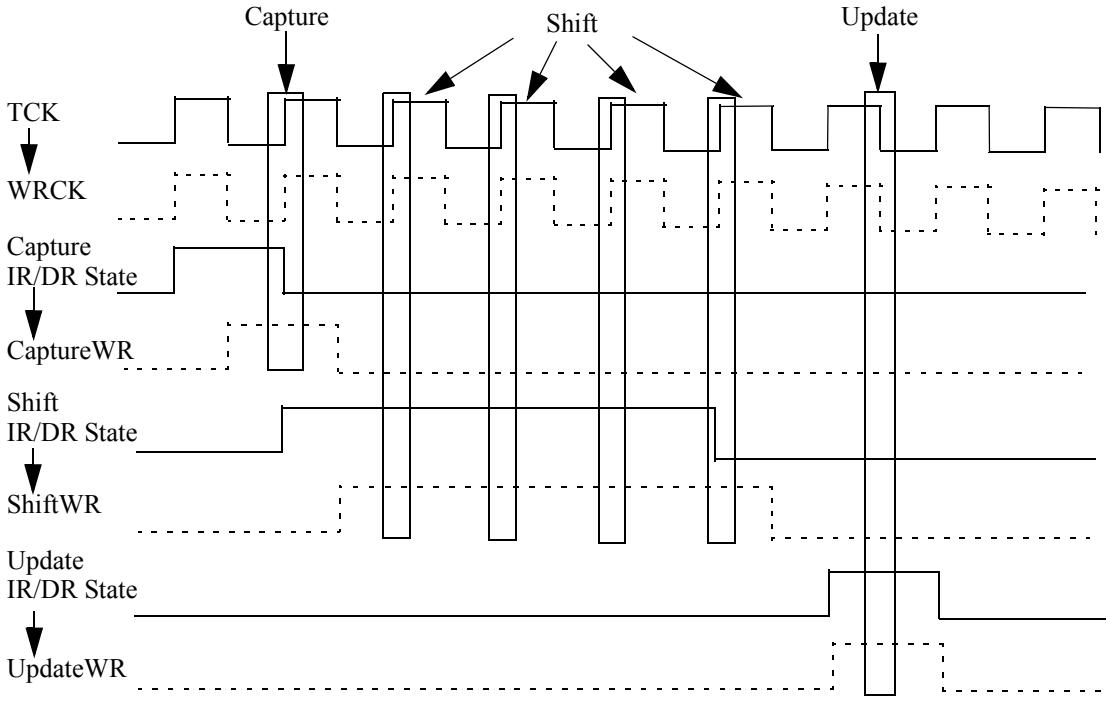


Figure C.3—IEEE 1500 Capture/Shift/Update events generated using TAP controller

It is clear via the glue logic circuit description given in this subclause with respect to Figure C.1 and Figure C.2 that IEEE 1500 wrappers are easily and efficiently controllable via legacy IEEE Standard 1149.1 TAP based tools, processes, and technologies.