

Standard Testability Method for Embedded Core-based Integrated Circuits

ICS 31.200

Confirmed November 2010

National foreword

This British Standard is the UK implementation of IEC 62528:2007.

The UK participation in its preparation was entrusted to Technical Committee GEL/93, Design automation.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 December 2007

© BSI 2007

ISBN 978 0 580 59318 5

Amendments issued since publication

Amd. No.	Date	Comments



IEC 62528

Edition 1.0 2007-11

INTERNATIONAL STANDARD

IEEE 1500™

Standard Testability Method for Embedded Core-based Integrated Circuits

CONTENTS

IEE Introduction	4
1. Overview	6
1.1 Scope	7
1.2 Purpose	7
2. Normative references	7
3. Definitions, acronyms, and abbreviations	8
3.1 Definitions	8
3.2 Acronyms and abbreviations	13
4. Structure of this standard	14
4.1 Specifications	14
4.2 Descriptions	15
5. Introduction and motivations of two compliance levels	15
6. Overview of the IEEE 1500 scalable hardware architecture	16
6.1 Wrapper serial port (WSP)	16
6.2 Wrapper parallel port (WPP)	16
6.3 Wrapper instruction register (WIR)	17
6.4 Wrapper bypass register (WBY)	17
6.5 Wrapper boundary register (WBR)	17
7. WIR instructions	18
7.1 Introduction	18
7.2 Response of the wrapper circuitry to instructions	18
7.3 Wrapper instruction rules and naming convention	20
7.4 WS_BYPASS Instruction	21
7.5 WS_EXTEST instruction	22
7.6 WP_EXTEST instruction	24
7.7 Wx_EXTEST instruction	26
7.8 WS_SAFE instruction	27
7.9 WS_PRELOAD instruction	29
7.10 WP_PRELOAD instruction	29
7.11 WS_CLAMP instruction	31
7.12 WS_INTEST_RING instruction	33
7.13 WS_INTEST_SCAN instruction	34
7.14 Wx_INTEST instruction	37
8. Wrapper serial port (WSP)	38
8.1 WSP terminals	39
9. Wrapper parallel port (WPP)	40
9.1 WPP terminals	40
10. Wrapper instruction register (WIR)	40
10.1 WIR configuration and DR selection	40
10.2 WIR design	41
10.3 WIR operation	44
11. Wrapper bypass register (WBY)	47
11.1 WBY register configuration and selection	47

11.2 WBY design.....	47
11.3 WBY operation	48
12. Wrapper boundary register (WBR).....	49
12.1 WBR structure and operation	51
12.2 WBR cell structure and operation.....	52
12.3 WBR operation events	53
12.4 WBR operation modes	56
12.5 Parallel access to the WBR	57
12.6 WBR cell naming.....	60
12.7 WBR cell examples	61
12.8 IEEE 1500 WBR example	65
13. Wrapper states.....	68
13.1 Wrapper Disabled and Wrapper Enabled states	68
14. WSP timing diagram.....	69
14.1 Specifications	69
14.2 Description.....	70
14.3 Synchronous reset timing.....	74
15. WSP configurations for IEEE 1500 system chips	75
15.1 Connecting multiple WSPs	75
16. Plug-and-play (PnP).....	78
16.1 Background and definition.....	78
16.2 PnP aspects of standard instructions	79
16.3 PnP limitations on protocols	80
16.4 Non-PnP in IEEE Std 1500.....	80
17. Compliance definitions common to wrapped and unwrapped cores	80
17.1 General rules	80
17.2 Per-terminal rules.....	82
17.3 Test pattern information rules.....	83
18. Compliance definitions specific to unwrapped cores	86
18.1 General rules.....	86
18.2 Per-terminal rules.....	87
18.3 Additional test information rules	87
19. Compliance definitions specific to wrapped cores	88
19.1 General rules	88
19.2 Per-terminal rules.....	89
19.3 Wrapper protocol information rules	89
20. IEEE 1500 application	90
20.1 CTL (IEEE P1450.6) overview	90
20.2 IEEE 1500 examples.....	91
Annex A (normative) Bubble diagram definition.....	107
Annex B (informative) WBR cell examples	109
Annex C (informative) Relationship of IEEE Std 1500 to IEEE Std 1149.1	118
Annex D (informative) List of participants.....	121

IEEE Introduction

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 TM^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 1149.1TM 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

^aInformation 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.

STANDARD TESTABILITY METHOD FOR EMBEDDED CORE-BASED INTEGRATED CIRCUITS

1. Overview

IEEE Std 1500TM 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 pre-designed 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 P1450.6TM¹ 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 1149.1TM-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

¹Information 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/>.

²IEEE 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 nonmergeable 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.

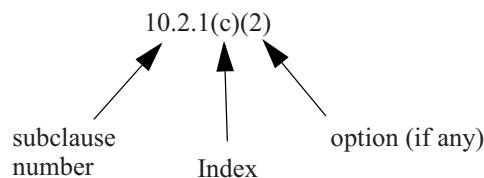
4.2 Descriptions

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

This standard 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, this standard 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.

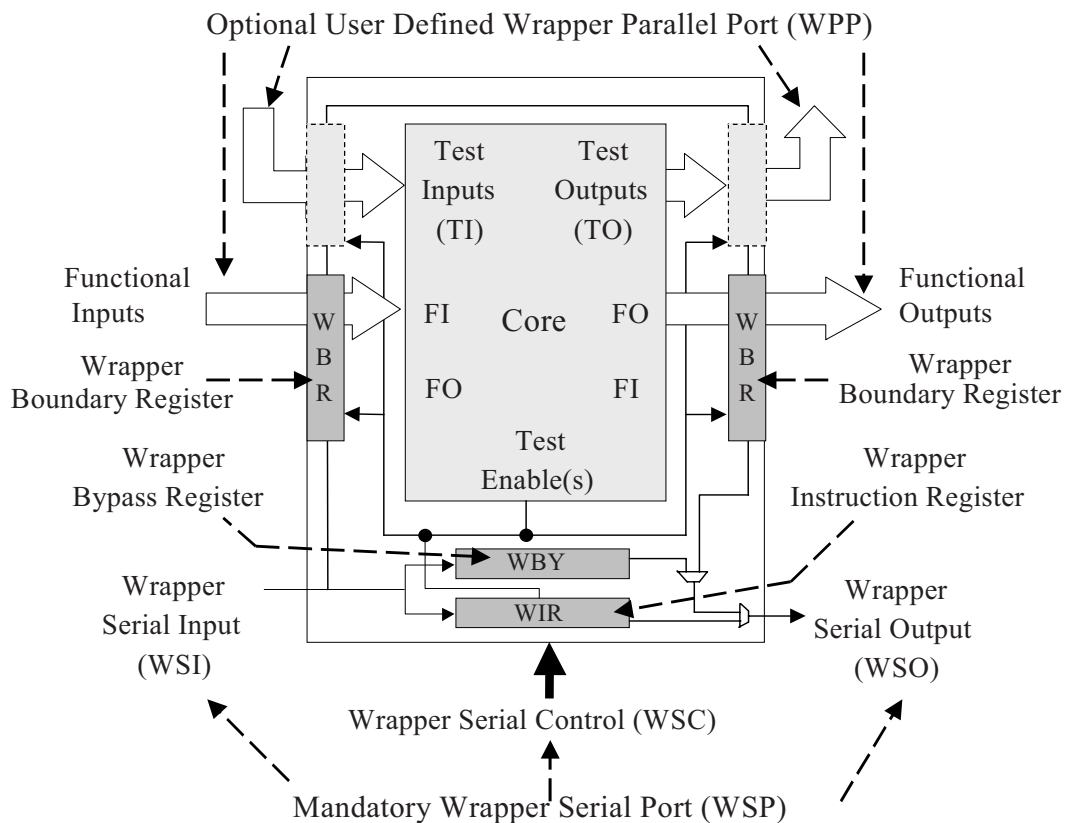


Figure 1—Standard IEEE 1500 wrapper components

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 (WB)

The WB provides a bypass path for the WSI-WSO terminals of the WSP. The WB 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 WB 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 WB 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.

^aWS_PRELOAD is required only if there is a silent shift path in the WBR.

^bThis 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.

Wrapper Serial Bypass (WS_Bypass) Instruction

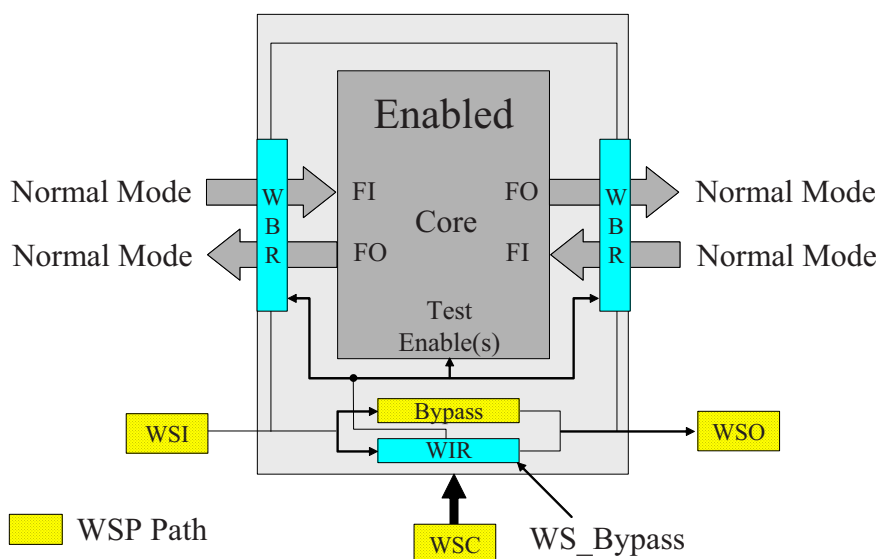


Figure 2—WS_BYPASS example

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

- Each wrapper shall provide a WS_EXTEST instruction.
- 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).
- 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.
- 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.

³Notes 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.

Wrapper Serial Extest (WS_Extest) Mode

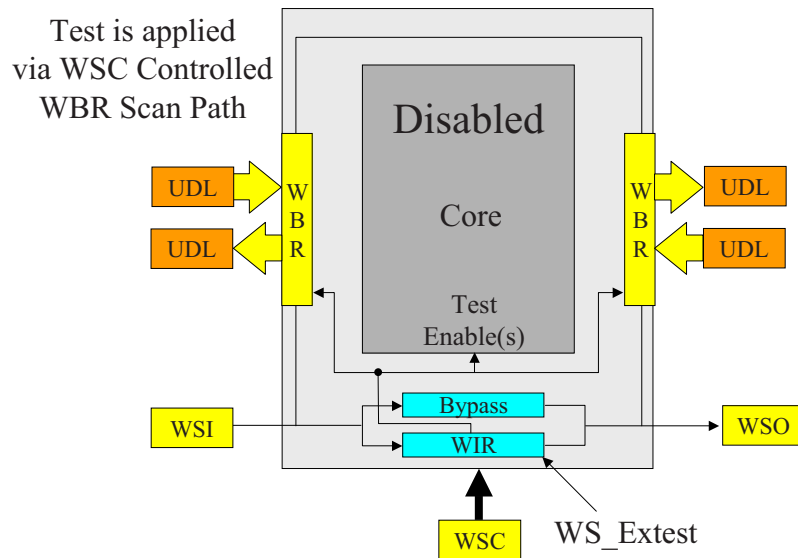


Figure 3—WS_EXTTEST example

7.6 WP_EXTTEST instruction

The WP_EXTTEST instruction allows testing of off-core circuitry and core-to-core interconnections. The WP_EXTTEST 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_EXTTEST mimics the WS_EXTTEST 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_EXTTEST instruction.

NOTE 2—Following use of the WP_EXTTEST 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

- While the WP_EXTTEST 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).
- While the WP_EXTTEST 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_EXTTEST instruction is selected.
- While the WP_EXTTEST 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.
- While the WP_EXTTEST 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.

Wrapper Parallel Extest (WP_Extest) Mode

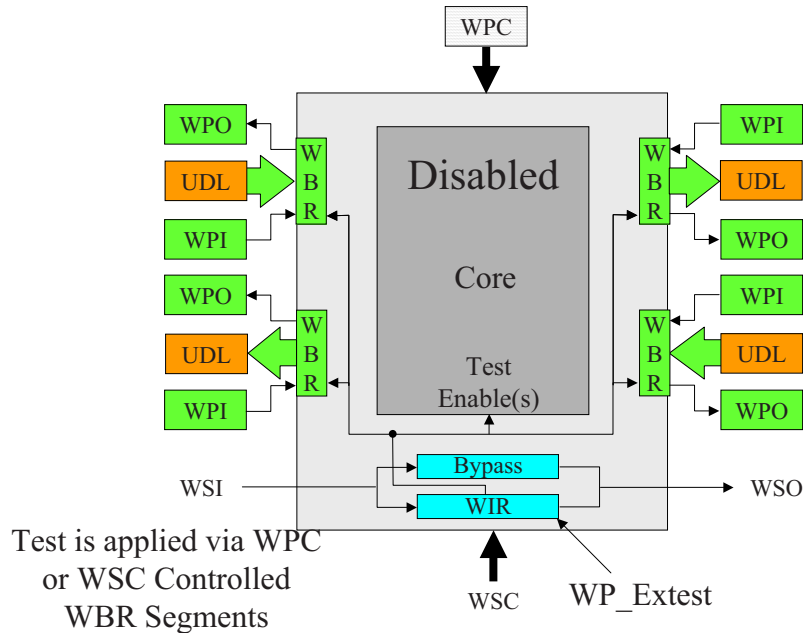


Figure 4—WP_EXTTEST example

7.7 Wx_EXTTEST instruction

Wx_EXTTEST 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_EXTTEST instruction encompasses the WS_EXTTEST and the WP_EXTTEST 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_EXTTEST is an OF test.

NOTE—Following use of the Wx_EXTTEST 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_EXTTEST instruction is provided.

7.7.1 Specifications

Rules

- While the Wx_EXTTEST instruction is selected, the WBR shall be in OF mode.

Recommendations

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

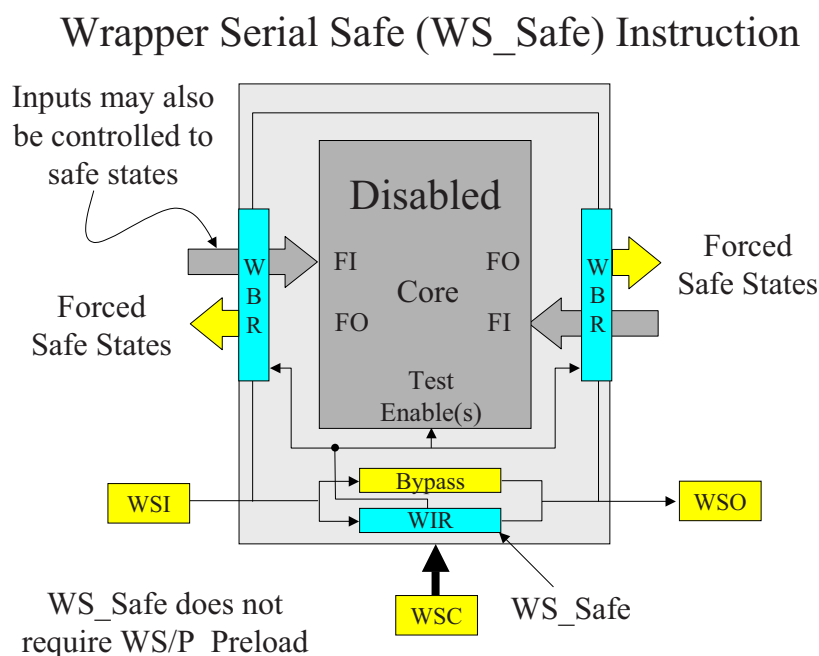
7.8.2 Description

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.



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.

Wrapper Serial Preload (WS_Preload) Instruction

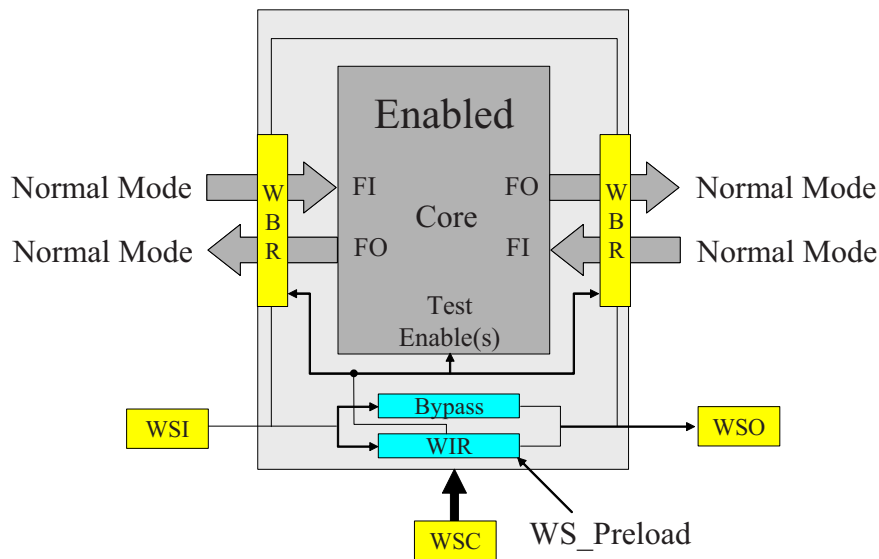


Figure 6—WS_PRELOAD example

7.10.1 Specifications

Rules

- 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).
- While the WP_PRELOAD instruction is selected, the data in the wrapper boundary cell shall be available for the Update operation, if applicable.
- 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.
- While the WP_PRELOAD instruction is selected, the WPC shall be utilized for test control.

Recommendations

- 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

- 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.
- 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.

Wrapper Parallel Preload (WP_Preload) Instruction

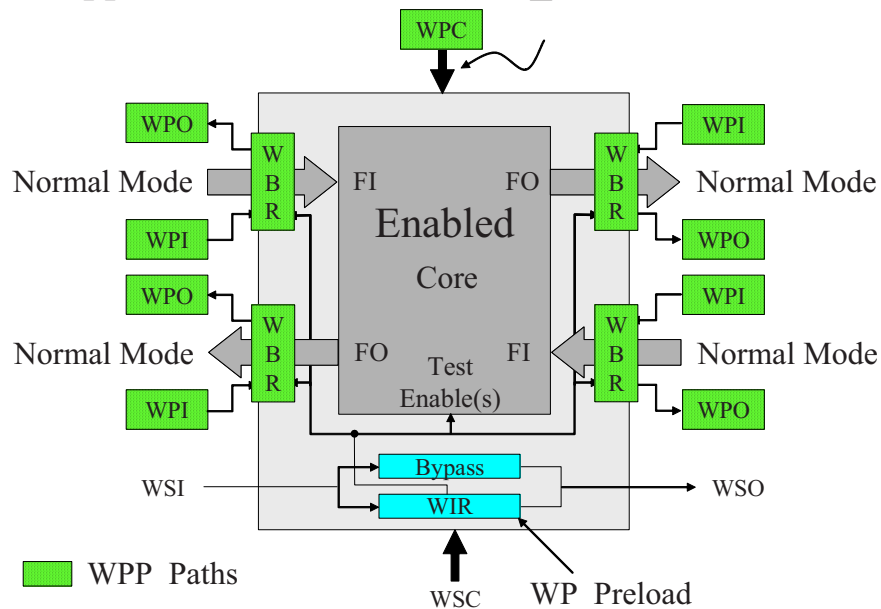


Figure 7—WP_PRELOAD example

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

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

- 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_EXTTEST 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_EXTTEST 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_EXTTEST can be used to load specific values into the wrapper before WS_CLAMP is enabled.

Wrapper Serial Clamp (WS_Clamp) Instruction

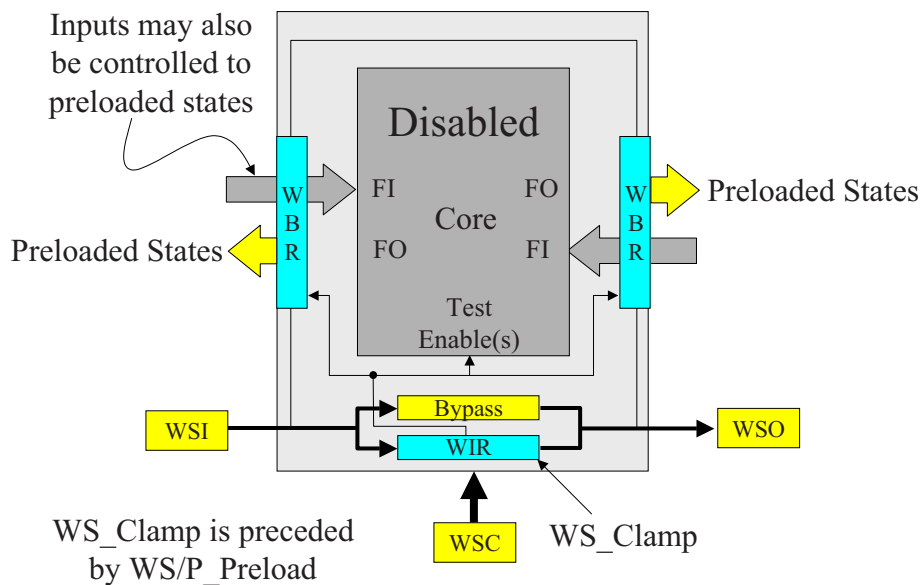


Figure 8—WS_CLAMP example

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

- 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).
- While the WS_INTEST_RING instruction is selected, the WBR shall be in IF mode.
- While the WS_INTEST_RING instruction is selected, the WSC shall be utilized for test control.

Recommendations

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

- 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.

Test is applied via WSC Controlled WBR Scan Path

Core

FI FO

FO FI

Test Enable(s)

WBR

WBR

WSI

Bypass

WIR

WSO

WSC

WS_Intest_Ring

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.

Wrapper Serial Intest Scan (WS Intest Scan) Mode

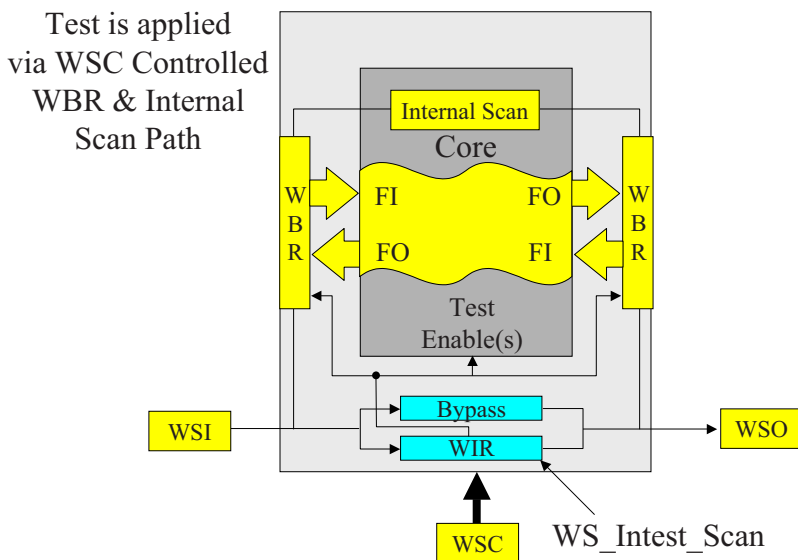


Figure 10—WS_INTEST_SCAN example