

OCP



International Partnership

OCP 2.0 / 2.1 Compliance Checks

Version 0.17
OCP-IP Functional Verification Working Group

OCP 2.0 / 2.1

Compliance Checks

The Overriding Document On OCP Certification Is the Current OCP Specification

Copyright © 2005 OCP-IP

This document contains material that is confidential to OCP-IP and its members and licensors. The user should assume that all materials contained and/or references d in this document are confidential and proprietary unless otherwise indicated or apparent from the nature of such materials (for example, references to publicly available forms or documents). Disclosure or use of this document or any material contained herein, other than as expressly permitted, is prohibited without the prior written consent of OCP-IP or such other party that may grant permission to use its proprietary material.

The trademarks, logos, and service marks displayed in this document are the registered and unregistered trademarks of OCP-IP, its members and its licensors.

The copyright and trademarks owned by OCP-IP, whether registered or unregistered, may not be used in connection with any product or service that is not owned, approved or distributed by OCP-IP, and may not be used in any manner that is likely to cause customer confusion or that disparages OCP-IP. Nothing contained in this document should be construed as granting by implication, estoppel, or otherwise, any license or right to use any copyright without the express written consent of OCP-IP, its licensors or a third party owner of any such trademark.

DISCLAIMER

This OCP-IP document is provided "as is" with no warranties whatsoever, including any warranty of merchantability, noninfringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification or sample. OCP-IP disclaims all liability for infringement of proprietary rights, relating to use of information in this document. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

OCP International Partnership (OCP-IP) disclaims all warranties and liability for the use of this document and the information contained herein and assumes no responsibility for any errors that may appear in this document, nor does OCP-IP make a commitment to update the information contained herein.

Contact the OCP-IP office to obtain the latest revision of this document.

Questions regarding this document or membership in OCP-IP may be forwarded to:

OCP-IP
www.ocpip.org
E-mail: admin@ocpip.org
Phone: +1 503-619-0560
Fax: +1 503-644-6708
OCP-IP Technical Support
techsupport@ocpip.org

Revision History

Version	Date	Author	Comments
0.1	08/05/2004	Samuel Dellacherie (TransEDA)	Original text-based proposal
0.2	09/11/2004	Samuel Dellacherie (TransEDA)	takes into account : Sonics list of checks Yogitech list of checks new TransEDA list v1.5.4 TI feedback (Jeroen Vliegen) Yogitech feedback (Natale Barsotti)
0.3			
0.4	19/11/2004	Samuel Dellacherie (TransEDA)	Reorganization of all properties for a better clarity
0.5		Samuel Dellacherie (TransEDA)	Feedback from OCP-IP support and xls file from TI Elimination of: cross-constraint section performance properties sampling properties
0.6	14/02/2005	Samuel Dellacherie (TransEDA)	Includes corrections based on the final golden xls sheet from the FVWG. Follows the order of the golden xls list. Adds references to the <i>OCP Specification</i>
0.7	08/04/2005	Samuel Dellacherie (TransEDA)	Minor corrections
08	30/04/2005	John Robiola (Sonics Inc.)	Added summary table and formatted document to conform
0.9	06/06/2005	Samuel Dellacherie (TransEDA)	Filled-in all blanks, added intro, changed section names
0.10	05/07/2005	Samuel Dellacherie	Added Jeroen's Intro and formatting Corrections based on Jeroen's, Anshul's and Ravi's feedbacks. Plus minor changes.
0.11	04/08/2005	Jeroen Vliegen (Texas Instruments)	Incorporated feedback of Drew Wingard (Sonics/SWG) Rewrote Introduction chapter for more clarity Added acknowledgements
0.12	29/08/2005	Jeroen Vliegen (Texas Instruments)	1) Added Steven McMaster's (Synopsys) feedback for the formal verification paragraph (new text) 2) Added Anssi Haverinen's (Nokia) feedback for property 3.4.11 3) Added Christophe Sucur's (TI) feedback 4) Updated figure 1 and former core_rtl.conf paragraph based on FVWG discussion of 16/08/2005 5) Added Yogitech's feedback on document v0.11 6) Added the 2.1 OCP properties 7) Added 2.1 specification references 8) did some overall formatting

Version	Date	Author	Comments
0.13	05/09/2005	Samuel Dellacherie	<p>1) modifications to the « dynamic verification » and « static verification » sub-sections</p> <p>2) added section 2.3 on which parameters activate which checks</p> <p>3) checked Anshul's feedback</p> <p>4) erased the "transaction" properties of 2.1 that were duplicates of the 2.0 versions</p>
0.14	22/09/2005	Jeroen Vliegen	<p>Many updates. Mainly around 2.1 properties. Please refer to "changes_v14.xls" for the entire list!</p> <p>Add burst_hold_MTagInOrder</p> <p>Change name of ... see email Samuel</p>
0.15	03/10/2005	Jeroen Vliegen	<p>Final review of FVWG required 2 changes:</p> <p>1. Add burst_hold_MTagInOrder property</p> <p>2. Change burst_order_MTagID_when_MTagInOrder_zero to burst_hold_MTagID_when_MTagInOrder_zero</p> <p>Tables are updated / TOC is updated / document is ready for Tec Writer review!!</p>
0.16	16/02/2006	Jeroen Vliegen	<p>1. Added functional coverage Chapter (4)</p> <p>2. Added activation table fixes reported by JEDA Technologies on v0.15 (activation parameters)</p>
0.17	03/05/2006	Jeroen Vliegen	<p>Incorporated feedback of Drew Wingard (Sonics Inc)</p> <p>Incorporated FVWG feedback (on Drew's feedback)</p>

Table of Contents

1	Introduction	1
1.1	OCP Compliance Definition.....	2
1.2	OCP Configurability.....	3
1.3	Verification Techniques.....	6
1.4	This Document Contains.....	9
1.5	Acknowledgements.....	10
2	Compliance Checks Tables	11
2.1	Introduction.....	12
2.2	Table of Checks.....	13
2.3	Activation Table.....	21
3	Compliance Checks Details	29
3.1	Introduction.....	30
3.2	Dataflow Signals Checks.....	31
3.2.1	signal_valid_<signal>_when_reset_inactive.....	31
3.2.2	request_valid_<signal>.....	32
3.2.3	datahs_valid_<signal>.....	33
3.2.4	response_valid_<signal>.....	34
3.2.5	request_valid_MTagInOrder.....	35
3.2.6	response_valid_STagInOrder.....	36
3.2.7	request_valid_MTagID_when_MTagInOrder_zero.....	37
3.2.8	datahs_valid_MDataTagID_when_MTagInOrder_zero.....	38
3.2.9	response_valid_STagID_when_STagInOrder_zero.....	39
3.3	DataFlow Phase Checks.....	40
3.3.1	request_exact_SThreadBusy.....	40
3.3.2	request_hold_<signal>.....	41
3.3.3	request_value_MCmd_<command>.....	43
3.3.4	request_value_MAddr_word_aligned.....	44
3.3.5	request_value_MAtomicLength_0x0.....	45
3.3.6	request_value_MBurstSeq_<sequence>.....	46
3.3.7	request_value_MBurstSeq_0x7.....	47
3.3.8	request_value_MBurstLength_0x0.....	48
3.3.9	request_value_MByteEn_force_aligned.....	49
3.3.10	request_value_MThreadID.....	51
3.3.11	datahs_exact_SDataThreadBusy.....	52
3.3.12	datahs_hold_<signal>.....	53
3.3.13	datahs_value_MDataByteEn_force_aligned.....	54
3.3.14	datahs_value_MDataThreadID.....	56
3.3.15	response_exact_MThreadBusy.....	57
3.3.16	response_hold_<signal>.....	58
3.3.17	response_value_SResp_FAIL_without_WRC.....	59

3.3.18	response_value_SThreadID	60
3.3.19	request_hold_MTagInOrder	61
3.3.20	response_hold_STagInOrder	62
3.3.21	request_hold_MTagID_when_MTagInOrder_zero	63
3.3.22	datahs_hold_MDataTagID_when_MTagInOrder_zero	64
3.3.23	response_hold_STagID_when_STagInOrder_zero	65
3.3.24	request_value_MTagID_when_MTagInOrder_zero	66
3.3.25	datahs_value_MDataTagID_when_MTagInOrder_zero	67
3.3.26	response_value_STagID_when_STagInOrder_zero	68
3.3.27	datahs_order_MDataTagID_when_MTagInOrder_zero	69
3.3.28	response_reorder_STagID_tag_interleave_size	70
3.3.29	response_reorder_STagID_overlapping_addresses	71
3.4	Dataflow Burst Checks.....	72
3.4.1	burst_hold_MBurstLength_precise	72
3.4.2	burst_hold_<signal>	73
3.4.3	burst_hold_<signal>_STRM	74
3.4.4	burst_phase_order_reqdata_together	75
3.4.5	burst_sequence_MAddr_INCR	76
3.4.6	burst_sequence_MAddr_STRM	77
3.4.7	burst_sequence_MAddr_WRAP	78
3.4.8	burst_sequence_MAddr_XOR	79
3.4.9	burst_value_<signal>_<sequence>	80
3.4.10	burst_value_<signal>_SINGLE	81
3.4.11	burst_value_MAddr_INCR_burst_aligned	82
3.4.12	burst_value_MAddr_INCR_no_wrap	83
3.4.13	burst_value_MBurstLength_<sequence>	84
3.4.14	burst_value_MBurstLength_INCR_burst_aligned	85
3.4.15	burst_value_MBurstPrecise_<sequence>	86
3.4.16	burst_value_MBurstPrecise_INCR_burst_aligned	87
3.4.17	burst_value_MBurstPrecise_SRMD	88
3.4.18	burst_value_MBurstSeq_UNKN_SRMD	89
3.4.19	burst_value_MCmd_<command>	90
3.4.20	burst_value_MDataLast_<mode>	91
3.4.21	burst_value_MReqLast_MRMD	92
3.4.22	burst_value_MReqLast_SRMD	93
3.4.23	burst_value_SRespLast_<mode>	94
3.4.24	burst_hold_MTagID_when_MTagInOrder_zero	95
3.4.25	burst_hold_MTagInOrder	96
3.5	DataFlow Transfer Checks.....	97
3.5.1	transfer_phase_order_datahs_before_request_begin	97
3.5.2	transfer_phase_order_datahs_before_request_end	98
3.5.3	transfer_phase_order_response_before_request_begin ..	99
3.5.4	transfer_phase_order_response_before_request_end ...	100
3.5.5	transfer_phase_order_response_before_datahs_begin ..	101

3.5.6	transfer_phase_order_response_before_datahs_end	102
3.5.7	transfer_phase_order_response_before_last_datahs_begin_ SRMD_wr103	
3.5.8	transfer_phase_order_response_before_last_datahs_end_ SRMD_wr	104
3.6	DataFlow ReadEx Checks	105
3.6.1	rdex_hold_ <signal>	105
3.6.2	rdex_hold_ <signal>	106
3.6.3	rdex_lock_release_no_WR/WRNP	107
3.6.4	rdex_lock_release_no_burst_allowed	108
3.7	Sideband Checks	109
3.7.1	signal_valid_ <signal>	109
3.7.2	signal_valid_ <signal> when_reset_inactive	110
3.7.3	signal_hold_ <signal>_16_cycles	111
3.7.4	signal_hold_Control_after_reset	112
3.7.5	signal_hold_Control_2_cycles	113
3.7.6	signal_hold_Control_ControlBusy_active	114
3.7.7	signal_hold_ControlWr_after_reset	115
3.7.8	signal_value_ControlWr_Control_transitioned	116
3.7.9	signal_value_ControlWr_ControlBusy_active	117
3.7.10	signal_hold_ControlWr_2_cycles	118
3.7.11	signal_value_ControlBusy	119
3.7.12	signal_hold_StatusRd_2_cycles	120
3.7.13	signal_value_StatusRd_StatusBusy_active	121
4	Functional Coverage	123
4.1	Introduction	124
4.2	Signal Level	125
4.3	Transfer Level	127
4.4	Transaction Level	127
4.5	Sideband Signals Coverage	131
4.6	Naming Conventions	132

1 Introduction

This document contains the OCP 2.0 / 2.1 compliance checks. It was developed by leading industry experts participating in the Open Core Protocol International Partnership Functional Verification Workgroup (OCP-IP™ FVWG).

The purpose of this document is to empower Verification IP/VIP developers to create OCP compliancy checking solutions in the language and tool of their choice.

The technical materials listed in this document are based on the OCP 2.0 / 2.1 specifications. They are guidelines to verify an IP/VIP for OCP 2.0 / 2.1 compliance. In all cases, the “Part I Specification” of the OCP 2.0 / 2.1 specifications is the definitive reference. Any reference made to “Part II Guidelines” of the OCP 2.0 / 2.1 specifications in this document is not definitive. The specification supersedes the guidelines.

This document is maintained by the OCP-IP™, a trade organization solely dedicated to OCP, supporting products and services. For all technical support inquiries, please contact techsupport@ocpip.org. For any other information or comments, please contact admin@ocpip.org.

1.1 OCP Compliance Definition

The OCP compliance definition is stated on page three (3) of the OCP 2.0 / 2.1 specifications.

For a core to be considered OCP compliant, it must satisfy the following conditions:

1. The core must include at least one OCP interface.
2. The core and OCP interfaces must be described using an RTL configuration file with the syntax specified in Chapter 6 of the OCP 2.0 / 2.1 specifications.
3. Each OCP interface on the core must:
 - a. Comply with all aspects of the OCP interface specification.
 - b. Have its timing described using a synthesis configuration file following the syntax specified in Chapter 7 of the OCP 2.0 / 2.1 specifications.

1.2 OCP Configurability

The main challenge in developing an OCP VIP lies in adequately contemplating the high degree of configurability of OCP. Figure 1 shows the different inputs that can affect OCP configurability.

OCP Interface Configuration

To properly define the OCP interface(s) of an IP/VIP, the following contexts must be considered:

a) Open System Context

In the context of an open system, upon delivery of an IP/VIP all the OCP interface(s) must be described using the `<core>_rtl.conf` syntax. The existence of the `<core>_rtl.conf` file is a hard requirement for OCP compliance. The metadata properties described in the `<core>_rtl.conf` file are clearly specified in Chapter 6 of the OCP 2.0 / 2.1 specifications.

For a configurable IP/VIP supporting multiple OCP configurations, several `<core>_rtl.conf` files are to be provided. Possibly they could be produced with a generator.

b) Closed System Context

In the context of a closed system, the verification of an IP/VIP with one (1) or more OCP interfaces may, or may not, be driven from a `<core>_rtl.conf` file. A vendor is free to implement any other solution. For example, a VERA verification environment could have a VERA object to control the OCP stimuli generators in its VIP instead of a `<core>_rtl.conf` file.

However, if the IP/VIP is being developed in a closed system context for delivery in an open system context, then the verification must include the `<core>_rtl.conf` file(s) and, if applicable, the `<core>_rtl.conf` generator which the IP/VIP is to be delivered with.

OCP Configuration Parameters Extraction

Depending on the system context, the VIP must extract the OCP configuration parameters from the `<core>_rtl.conf` file (open) or from any other solution (closed). Parameters for which the value is not explicitly specified must be retrieved using the configuration parameter defaults summarized in Table 22 of the OCP 2.0 specification or Table 25 of the OCP 2.1 specification. Note that certain parameters are always needed in certain configurations, and for those, no default is specified. For example: `addr_wdth` must always be specified if `addr == 1`.

OCP Configuration Parameters Cross-constraints Checking

Once all the OCP configuration parameters are known, illegal OCP interface parameter configurations must be flagged. Chapter 4 of the OCP 2.0 / 2.1 specifications contains most of the cross-constraints. For example: if `readex_enable` is set to 1, `write_enable` or `writenonpost_enable` must be set to 1.

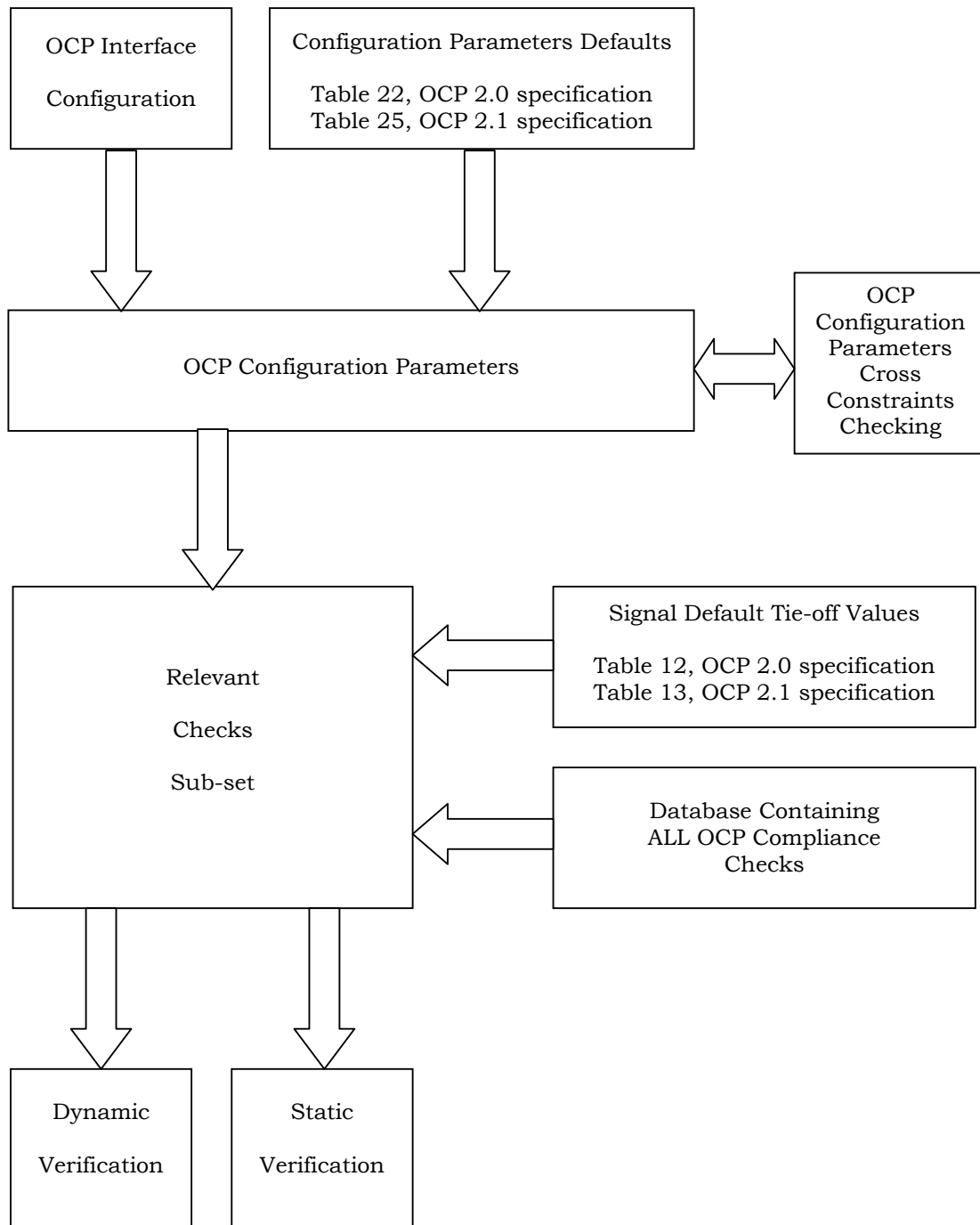
Select the Relevant Checks

Based on the OCP configuration parameters, select a subset of the checks in the VIP OCP library. This subset will be used for the actual verification. Note that if a signal used by a check is not configured in the OCP interface and if no other tie-off value is specified, Table 12 of the OCP 2.0

specification or Table 13 of the OCP 2.1 specification specifies the inferred default tie-off values. For example: MBurstPrecise default tie-off value is '1' or precise.

Compliance Checking

Check the compliance of the DUT OCP interface(s) using static or dynamic verification techniques. For more details , see 1.3, Verification Techniques.

Figure 1: OCP Configurability

1.3 Verification Techniques

Dynamic Verification

This verification technique relies on the following key elements:

- A 'good' set of stimuli
- Protocol checks
- Functional + code coverage

In summary, the methodology consists of driving a set of stimuli through the OCP interface into the DUT. The VIP contains a monitor which traces the OCP interface activity. The traces must be checked by a protocol checker to make sure that the protocol is never violated. The quality of the set of stimuli must be assessed using functional and code coverage.

Note that the OCP configuration parameters determine:

- Which protocol checks must be active.
- How the OCP functional coverage is defined.

Stimuli

The 'good' set of stimuli is not described in this document. This decision was made because it is not trivial for traditional directed-test methodologies to define a 'golden' set of stimuli for any OCP interface configuration. It is up to the Verification Engineer to implement a smart and efficient set of stimuli. For example, this may be done using constraint-driven random stimuli generation. As indicated before, the quality of the stimuli must be assessed using both functional and code coverage (see below).

Protocol Checks

The protocol checker is a passive component which monitors whether the OCP protocol is violated for a specific set of OCP configuration parameters. The protocol checker can be written in several languages; for example: HDL, PSL, SVA, E, VERA. The protocol checker must be instantiated on each OCP interface of the DUT.

Note that the names of the protocol checks must be matched with the names given to the compliance checks described in this document such that this document can easily be referenced by a Verification Engineer.

Functional Coverage

The quality of the set of applied stimuli must be measured with OCP functional coverage. One hundred percent OCP functional coverage must be targeted. Additionally, code coverage must be run on the RTL to determine whether there are any verification holes such as uncovered FSM states or missed branches. Based on the code coverage analysis, additional coverage metrics may be required.

The guidelines for the OCP functional coverage are given in the last chapter of this document. Additional coverage metrics needed, based on code-coverage analysis, are design dependent and therefore are out of the scope of this document.

Static Verification

This verification approach is also referred to as 'formal verification'. It relies on the following key elements:

- OCP protocol assertions (or protocol checkers).
- OCP protocol constraints (optional).
- OCP functional coverage.

In this verification approach, the Verification Engineer will use a formal tool to prove that, given the stimulus limits defined by the OCP protocol constraints, the OCP interface of the DUT never violates the OCP protocol assertions. The formal proof may either be exhaustive (the assertions are never violated) or bounded (up to a certain depth of the state space the assertions are never violated). No stimuli are needed; instead the tool relies on the 'all acceptable stimuli' definitions provided via the OCP protocol constraints.

Note that the OCP configuration parameters determine:

1. Which protocol assertions must be active.
2. How the functional coverage must be defined.
3. Which protocol constraints must be active.

Protocol Assertions

Formal verification revolves around taking the protocol assertions and attempting to prove that they are never violated. If a violation is found, the formal tool provides a test sequence that illustrates the violation on the design. The assertions can be written in different languages such as HDL, PSL or SVA.

Note that the names of the assertions must be matched with the names given to the compliance checks described in this document such that this document can easily be referenced by a Verification Engineer.

Protocol Constraints

In order to place bounds on the stimuli the formal engine must consider, the design must be connected to some protocol constraints or some form of generator description. The constraints can be specified using the same language as for protocol assertions, typically in HDL, PSL, SVA, or OVA.

A good set of protocol constraints for the OCP protocol is not described in this document and must be provided with the formal tool or created by the Verification Engineer. The set of protocol constraints should not be under-constrained as this will cause some "false negative" test sequences and the protocol assertions will be violated. This same set of protocol constraints should not be over-constrained or the protocol properties can be wrongly proved correct.

Functional Coverage

The Verification Engineer must insure that all of the protocol assertions are verified to a reasonable extent, and that the protocol constraints are sound. To accomplish this, some functional coverage must be added as a function of the OCP configuration parameters.

The functional coverage definition should cover:

1. Which assertions warrant exhaustive proofs?
2. Which assertions are ok with just bounded proofs, at what depth?
3. Detect and correct an over-constrained environment.

The guidelines for the OCP functional coverage are given in the last chapter

1.4 This Document Contains

As indicated before, the compliance checks listed in this document are extracted from the OCP 2.0 / 2.1 specifications. These compliance checks are guidelines to verify an IP for OCP 2.0 / 2.1 compliance. In all cases the “Part I Specification” of the OCP 2.0 / 2.1 specifications is the definitive reference. Any reference to “Part II Guidelines” of the OCP 2.0 / 2.1 specifications in this document is not definitive. The specification supersedes the guidelines.

This document contains:

Chapter 2: Compliance Checks Tables

- Table of Checks

This table defines the full set of OCP 2.0 / 2.1 compliance checks. Each table entry consists of a name and a concise description of the check in 'natural' English. English was chosen to be implementation independent.

- Activation Table

This table defines for each compliance check the set of OCP 2.0 / 2.1 parameters required for the check to be activated.

Chapter 3: Compliance Checks Details

This chapter provides more information for each compliance check: the nomenclature, a profound English description and OCP 2.0 / 2.1 specification references.

Chapter 4: Functional Coverage

This chapter outlines the functional coverage approach for the OCP protocol.

1.5 Acknowledgements

The following companies were instrumental in the development of the OCP 2.0 / 2.1 compliance checks:

- All OCP-IP Functional Verification Workgroup members, including participants from:

MIPS Technologies, Inc.

Sonics, Inc.

Synopsys, Inc.

Texas Instruments, Inc.

TransEDA

YOGITECH, SPA

- All reviewers who participated in the General Member Review

2 Compliance Checks Tables

2.1 Introduction

The following table 'Table of Checks' gives at a glance the complete list of OCP compliance checks. The checks are divided into six main categories:

1. Dataflow Signal Checks
2. Dataflow Phase Checks
3. Dataflow Burst Checks
4. Dataflow Transfer Checks
5. Dataflow ReadEx Checks
6. Sideband Checks

For each compliance check, the protocol version, a name and a short description are given. For an OCP 2.0 interface, only the 2.0 checks are relevant. For an OCP 2.1 interface, both the 2.0 and the 2.1 checks are relevant.

The compliance check names have been created using the following template:

<hierarchy>_<check type>_<critical signal>_<extra details>

In which:

- <hierarchy>** : signal, request, datahs, response, burst, transfer, rdex
<check type> : valid, hold, value, exact, phase_order, lock_release, sequence, order, reorder
<critical signal> : (optional) any OCP signal name that is impacted by the compliance check
<extra details> : (optional) a short additional explanation

2.2 Table of Checks

1. Dataflow Signal Checks		
Version	Name	Short Description
2.0	signal_valid_ <signal> when_reset_inactive MCmd MDataValid MThreadBusy SDataThreadBusy SResp SThreadBusy	Check every cycle that <signal> is not X or Z when reset inactive
2.0	request_valid_ <signal> MAddr MAddrSpace MAAtomicLength MBurstLength MBurstPrecise MBurstSeq MBurstSingleReq MByteEn MConnID MReqLast MThreadID SCmdAccept	Check that <signal> is not X or Z during the request phase
2.0	datahs_valid_ <signal> MDataByteEn MDataLast MDataThreadID SDataAccept	Check that <signal> is not X or Z during the datahandshake phase
2.0	response_valid_ <signal> MRespAccept SRespLast SThreadID	Check that <signal> is not X or Z during the response phase
2.1	request_valid_MTagInOrder	When tags>1 and taginorder=1, check that MTagInOrder is not X or Z during the request phase
2.1	response_valid_STagInOrder	When tags>1 and taginorder=1, check that STagInOrder is not X or Z during the response phase
2.1	request_valid_MTagID_when_MTagInOrder_zero	When tags>1, for tagged transactions [MTagInOrder=0], check that MTagID is not X or Z during the request phase
2.1	datahs_valid_MDataTagID_when_MTagInOrder_zero	When tags>1 and datahandshake=1. for tagged write transactions [MTagInOrder=0], check that MDataTagID is not X or Z during the datahandshake phase
2.1	response_valid_STagID_when_STagInOrder_zero	When tags>1, for tagged transactions [STagInOrder=0], check that STagID is not X or Z during the response phase

2. Dataflow Phase Checks		
Version	Name	Short Description
2.0	request_exact_SThreadBusy	Request phase is illegal if SThreadBusy and exact
2.0	request_hold_ <signal> MAddr MAddrSpace MAAtomicLength MBurstLength MBurstPrecise MBurstSeq MBurstSingleReq MByteEn MCmd MConnID MData MDataInfo MReqInfo MReqLast MThreadID	<signal> must hold for the request phase
2.0	request_value_MCmd_ <command> BCST RDL WRC RD RDEX WR WRNP	MCmd <command> is illegal if <parameter> is 0
2.0	request_value_MAddr_word_aligned	MAddr must be OCP word aligned (LSB bits zero)
2.0	request_value_MAAtomicLength_0x0	MAAtomicLength == 0 is an illegal value
2.0	request_value_MBurstSeq_ <sequence> DLFT1 DLFT2 INCR STRM UNKN WRAP XOR	MBurstSeq <sequence> is illegal if <parameter > is 0
2.0	request_value_MBurstSeq_0x7	MBurstSeq == 0x7 is an illegal value
2.0	request_value_MBurstLength_0x0	MBurstLength == 0 is an illegal value
2.0	request_value_MByteEn_force_aligned	MByteEn must be a 'legal' pow2 (see patterns) if force_aligned
2.0	request_value_MThreadID	MThreadID must be < threads
2.0	datahs_exact_SDataThreadBusy	Data phase is illegal if SDataThreadBusy and exact
2.0	datahs_hold_ <signal> MData MDataByteEn MDataInfo MDataThreadID MDataValid MDataLast	<signal> must hold for the datahandshake phase
2.0	datahs_value_MDataByteEn_force_aligned	MDataByteEn must be a 'legal' pow2 (see patterns) if force_aligned
2.0	datahs_value_MDataThreadID	MDataThreadID must be < threads

2.0	response_exact_MThreadBusy	Response phase is illegal if MThreadBusy and exact
2.0	response_hold_ <signal> SData SDataInfo SResp SRespInfo SRespLast SThreadID	<signal> must hold for the response phase
2.0	response_value_SResp_FAIL_without_WRC	SResp can only be FAIL for a WRC cmd
2.0	response_value_SThreadID	SThreadID must be < threads
2.1	request_hold_MTagInOrder	When tags>1 and taginorder=1, MTagInOrder must hold for the request phase
2.1	response_hold_STagInOrder	When tags>1 and taginorder=1, STagInOrder must hold for the response phase
2.1	request_hold_MTagID_when_MTagInOrder_zero	When tags>1, for tagged transactions [MTagInOrder=0], MTagID must hold for the request phase
2.1	datahs_hold_MDataTagID_when_MTagInOrder_zero	When tags>1 and datahandshake=1, for tagged write transactions [MTagInOrder=0], the MDataTagID must hold for the datahandshake phase
2.1	response_hold_STagID_when_STagInOrder_zero	When tags>1, for tagged transactions [STagInOrder=0], STagID must hold for the response phase
2.1	request_value_MTagID_when_MTagInOrder_zero	When tags>1, for tagged transactions [MTagInOrder=0], MTagID must be < tags
2.1	datahs_value_MDataTagID_when_MTagInOrder_zero	When tags>1 and datahandshake=1, for tagged write transactions [MTagInOrder=0], MDataTagID must be < tags
2.1	response_value_STagID_when_STagInOrder_zero	When tags>1, for tagged transactions [STagInOrder=0], STagID must be < tags
2.1	datahs_order_MDataTagID_when_MTagInOrder_zero	When tags>1 and datahandshake=1, for tagged write transactions [MTagInOrder=0], the datahandshake phase must observe the same order as the request phase (MTagID - MDataTagID)
2.1	response_reorder_STagID_tag_interleave_size	Responses that are part of the same transaction must stay together up to the tag_interleave_size
2.1	response_reorder_STagID_overlapping_addresses	Responses to requests with target overlapping addresses must not be re-ordered with respect to another

3. Dataflow Burst Checks		
Version	Name	Short Description
2.0	burst_hold_MBurstLength_precise	MBurstLength must hold for precise bursts
2.0	burst_hold_ <signal> MAddrSpace MAtomicLength MBurstPrecise MBurstSeq MBurstSingleReq MCmd MConnID MReqInfo SRespInfo	<signal> must hold for all appropriate phases of the entire burst
2.0	burst_hold_ <signal> _STRM MByteEn MDataByteEn	<signal> must hold for STRM bursts
2.0	burst_phase_order_reqdata_together	If reqdata_together, the request and datahandshake phases of the first transfer in a SRMD burst must begin and end together
2.0	burst_sequence_MAddr_INCR	Check MAddr sequence for INCR burst (++)
2.0	burst_sequence_MAddr_STRM	Check MAddr sequence for STRM bursts (constant)
2.0	burst_sequence_MAaddr_WRAP	Check MAddr sequence for WRAP bursts (++,wrap)
2.0	burst_sequence_MAddr_XOR	Check MAddr sequence for XOR bursts (see spec)
2.0	burst_value_ <signal> _<sequence> MByteEn STRM MDataByteEn DFLT2	<signal> must have at least 1 byte enabled for <sequence>
2.0	burst_value_ <signal> _SINGLE MReqLast MDataLast SRespLast	<signal> must be 1 for a SINGLE transfer
2.0	burst_value_MAddr_INCR_burst_aligned	INCR bursts with burst_aligned MAddr must be aligned with the burst length
2.0	burst_value_MAddr_INCR_no_wrap	INCR burst MAddr must never wrap around the address space of the OCP interface
2.0	burst_value_MBurstLength_ <sequence> WRAP XOR	<sequence> bursts must have a pow2 burst length
2.0	burst_value_MBurstLength_INCR_burst_aligned	INCR bursts with burst_aligned MBurstLength must be pow2
2.0	burst_value_MBurstPrecise_ <sequence> WRAP XOR	<sequence> bursts must be precise
2.0	Burst_value_MBurstPrecise_INCR_burst_aligned	INCR bursts with burst_aligned must be precise
2.0	Burst_value_MBurstPrecise_SRMD	SRMD burst must be precise
2.0	Burst_value_MBurstSeq_UNKN_SRMD	If MBurstSingleReq is asserted, an MBurstSeq == UNKN is illegal (not predictable by slave)

2.0	burst_value_MCmd_ <command> RDEX RDL WRC	<command> can not be part of a burst
2.0	burst_value_MDataLast_ <mode> MRMD SRMD	MDataLast must be 1 for the last datahandshake phase of a <mode> burst
2.0	burst_value_MReqLast_MRMD	MReqLast must ONLY be 1 for the last request of a MRMD burst
2.0	burst_value_MReqLast_SRMD	MReqLast must be 1 when MBurstSingleReq == 1 for a SRMD burst
2.0	burst_value_SRespLast_ <mode> MRMD SRMD	SRespLast must be 1 for the last response phase for a <mode> burst
2.1	burst_hold_MTagID_when_MTagInOrder_zero	When tags>1, when MTagInOrder == 0, MTagID must remain constant for all transfers of a burst
2.1	burst_hold_MTagInOrder	When tags>1 and taginorder=1, MTagInOrder must remain constant for all transfers of a burst

4. Dataflow Transfer Checks		
Version	Name	Short Description
2.0	transfer_phase_order_datahs_before_request_begin	Datahandshake phase cannot begin before request phase begins
2.0	transfer_phase_order_datahs_before_request_end	Datahandshake phase cannot end before request phase ends
2.0	transfer_phase_order_response_before_request_begin	Response phase cannot begin before request phase begins
2.0	transfer_phase_order_response_before_request_end	Response phase cannot end before request phase ends
2.0	transfer_phase_order_response_before_datahs_begin	Response phase cannot begin before datahandshake phase begins
2.0	transfer_phase_order_response_before_datahs_end	Response phase cannot end before datahandshake phase ends
2.0	transfer_phase_order_response_before_last_datahs_begin_SRMD_wr	Response phase cannot begin before last datahandshake phase begins (SRMD writes)
2.0	transfer_phase_order_response_before_last_datahs_end_SRMD_wr	Response phase cannot end before last datahandshake phase ends (SRMD writes)

5. Dataflow ReadEx Checks		
Version	Name	Short Description
2.0	rdex_hold_ <signal> MAddr MAddrSpace	<signal> must hold for a RDEX + WR/WRNP
2.0	rdex_hold_ <signal> MByteEn MDataByteEn)	<signal> must hold for a RDEX + WR/WRNP
2.0	rdex_lock_release_no WR/WRNP	A RDEX must be followed by a WR/WRNP on the same thread
2.0	rdex_lock_release_no_burst_allowed	The unlocking WR/WRNP following a RDEX cannot be part of a burst

6. Sideband Checks		
Version	Name	Short Description
2.0	signal_valid_ <signal> MReset_n SReset_n	Check every cycle that <signal> is not X or Z
2.0	signal_valid_ <signal> _when_reset_inactive ControlBusy ControlWr MError SError SInterrupt StatusBusy StatusRd	Check every cycle that <signal> is not X or Z when reset inactive
2.0	signal_hold_ <signal> _16_cycles MReset_n SReset_n	If active, must stay at least 16 cycles
2.0	signal_hold_Control_after_reset	Control must be held steady the cycle after reset is de-asserted
2.0	signal_hold_Control_2_cycles	Control can only transition at most every other cycle
2.0	signal_hold_Control_controlbusy_active	Control must not transition while ControlBusy is active
2.0	signal_hold_ControlWr_after_reset	ControlWr must not assert during the cycle after reset becomes active
2.0	signal_value_ControlWr_control_transitioned	ControlWr must assert when Control transitions
2.0	signal_value_ControlWr_controlbusy_active	ControlWr must not assert when ControlBusy is active
2.0	signal_hold_ControlWr_2_cycles	ControlWr can only assert at most every other cycle
2.0	signal_value_ControlBusy	ControlBusy may only start to be asserted in a cycle after ControlWr is asserted or when reset transitions to inactive
2.0	signal_hold_StatusRd_2_cycles	StatusRd can only assert at most every other clock cycle
2.0	signal_value_StatusRd_statusbusy_active	StatusRd must not assert when StatusBusy is active

2.3 Activation Table

The following table puts together the essential parameters that are needed for each check to fire. The following assumptions are made with respect to this table.

1. The user of the document and the following tables is aware of the OCP protocol and understands that a combination of these parameters can lead to an illegal configuration.
2. Every check in the tables below will only talk about the minimum parameters needed for each check to be fired. Each configuration needs the parameters defined for each check in this table and also every other parameter needed to make it a legal configuration to start with. For example, a check for INCR bursts would at the minimum need some command (read_enable, write_enable, etc.) parameters defined to test the check.

1. Dataflow Signal Checks		
Version	Name	Activation Parameters
2.0	signal_valid_ <signal> when_reset_inactive MCmd MDataValid MThreadBusy SDataThreadBusy SResp SThreadBusy	- datahandshake mthreadbusy sdatathreadbusy resp stthreadbusy
2.0	request_valid_ <signal> MAddr MAddrSpace MAtomicLength MBurstLength MBurstPrecise MBurstSeq MBurstSingleReq MByteEn MConnID MReqLast MThreadID SCmdAccept	addr addrspace atomiclength burstlength burstprecise burstseq burstsinglereq byteen connid reqlast threads > 1 cmdaccept
2.0	datahs_valid_ <signal> MDataByteEn MDataLast MDataThreadID SDataAccept	mdatabyteen datalast datahandshake & threads > 1 dataaccept
2.0	response_valid_ <signal> MRespAccept SRespLast SThreadID	respaccept resplast resp & threads > 1
2.1	request_valid_MTagInOrder	Taginorder
2.1	response_valid_STagInOrder	resp & taginorder
2.1	request_valid_MTagID_when_MTagInOrder_zero	tags > 1
2.1	datahs_valid_MDataTagID_when_MTagInOrder_zero	datahandshake & tags > 1
2.1	response_valid_STagID_when_STagInOrder_zero	resp & tags > 1

2. Dataflow Phase Checks		
Version	Name	Activation Parameters
2.0	request_exact_SThreadBusy	stthreadbusy & stthreadbusy_exact
2.0	request_hold_ <signal> MAddr MAddrSpace MAAtomicLength MBurstLength MBurstPrecise MBurstSeq MBurstSingleReq MByteEn MCmd MConnID MData MDataInfo MReqInfo MReqLast MThreadID	cmdaccept & addr cmdaccept & addrspace cmdaccept & atomiclength cmdaccept & burstlength cmdaccept & burstprecise cmdaccept & burstseq cmdaccept & burstsinglereq cmdaccept & byteen cmdaccept cmdaccept & connid cmdaccept & mdata & !datahandshake cmdaccept & mdatainfo & !datahandshake cmdaccept & reqinfo cmdaccept & reqlast cmdaccept & threads > 1
2.0	request_value_MCmd_ <command> BCST RDL WRC RD RDEX WR WRNP	!broadcast_enable !rdlwr_enable !rdlwr_enable !read_enable !readex_enable !write_enable !writenonpost_enable
2.0	request_value_MAddr_word_aligned	Addr
2.0	request_value_MAtomicLength_0x0	Atomiclength
2.0	request_value_MBurstSeq_ <sequence> DLFT1 DLFT2 INCR STRM UNKN WRAP XOR	burstlength & !burstseq_dflt1_enable burstlength & !burstseq_dflt2_enable burstlength & !burstseq_incr_enable burstlength & !burstseq_strm_enable burstlength & !burstseq_unkn_enable burstlength & !burstseq_wrap_enable burstlength & !burstseq_xor_enable
2.0	request_value_MBurstSeq_0x7	Burstlength & burstseq
2.0	request_value_MBurstLength_0x0	Burstlength
2.0	request_value_MByteEn_force_aligned	byteen & force_aligned
2.0	request_value_MThreadID	threads > 1
2.0	datahs_exact_SDataThreadBusy	sdatathreadbusy & sdatathreadbusy_exact & datahandshake
2.0	datahs_hold_ <signal> MData MDataByteEn MDataInfo MDataThreadID MDataValid MDataLast	dataaccept & mdata & datahandshake mdatabyteen & dataaccept & datahandshake dataaccept & mdatainfo & datahandshake datahandshake & threads > 1 & dataaccept dataaccept & datahandshake datalast & dataaccept & datahandshake
2.0	datahs_value_MDataByteEn_force_aligned	Mdatabyteen & datahandshake & force_aligned
2.0	datahs_value_MDataThreadID	datahandshake & threads > 1
2.0	response_exact_MThreadBusy	mthreadbusy & mthreadbusy_exact & resp

2.0	response_hold_ <signal> SData SDataInfo SResp SRespInfo SRespLast SThreadId	respaccept & sdata & resp & (read_enable readex_enable rdllwrc_enable) respaccept & sdatainfo respaccept & resp respaccept & resp & respinfo respaccept & resp & respplast respaccept & resp & threads > 1
2.0	response_value_SResp_FAIL_without_WRC	resp & rdllwrc_enable
2.0	response_value_SThreadId	resp & threads > 1
2.1	request_hold_MTagInOrder	cmdaccept & taginorder & tags > 1
2.1	response_hold_STagInOrder	resp & respaccept & taginorder & tags > 1
2.1	request_hold_MTagID_when_MTagInOrder_zero	cmdaccept & tags > 1
2.1	datahs_hold_MDataTagID_when_MTagInOrder_zero	datahandshake & dataaccept & tags > 1
2.1	response_hold_STagID_when_STagInOrder_zero	resp & respaccept & tags > 1
2.1	request_value_MTagID_when_MTagInOrder_zero	tags > 1
2.1	datahs_value_MDataTagID_when_MTagInOrder_zero	datahandshake & tags > 1
2.1	response_value_STagID_when_STagInOrder_zero	resp & tags > 1
2.1	datahs_order_MDataTagID_when_MTagInOrder_zero	burstlength & datahandshake & tags > 1
2.1	response_reorder_STagID_tag_interleave_size	burstsinglereq & resp & tags > 1
2.1	response_reorder_STagID_overlapping_addresses	resp & tags > 1 & (addr addrspace byteen)

3. Dataflow Burst Checks		
Version	Name	Activation Parameters
2.0	Burst_hold_MBurstLength_precise	burstlength
2.0	Burst_hold_ <signal> MAddrSpace MAtomicLength MBurstPrecise MBurstSeq MBurstSingleReq MCmd MConnID MReqInfo SRespInfo	burstlength & addrspace burstlength & atomiclength burstlength & burstprecise burstseq & burstlength burstlength & burstsinglereq burstlength burstlength & connid burstlength & reqinfo burstlength & respinfo
2.0	Burst_hold_ <signal> _STRM MByteEn MDataByteEn	burstlength & burstseq_strm_enable & byteen & burstseq burstlength & burstseq_strm_enable & datahandshake & mdatabyteen & burstseq
2.0	Burst_phase_order_reqdata_together	reqdata_together & datahandshake
2.0	Burst_sequence_MAddr_INCR	burstlength & addr & burstseq_incr_enable & burstseq
2.0	Burst_sequence_MAddr_STRM	burstlength & addr & burstseq_strm_enable & burstseq
2.0	Burst_sequence_MAddr_WRAP	burstlength & burstseq_wrap_enable & addr & burstseq
2.0	Burst_sequence_MAddr_XOR	burstlength & burstseq_xor_enable & addr & burstseq
2.0	burst_value_ <signal>_ <sequence> MByteEn STRM MDataByteEn STRM MByteEn DFLT2 MDataByteEn DFLT2	burstlength & burstseq_strm_enable & byteen & burstseq burstseq & burstseq_strm_enable & mdatabyteen & datahandshake burstlength & burstseq_dflt2_enable & byteen & burstseq burstseq & burstseq_dflt2_enable & mdatabyteen & datahandshake
2.0	burst_value_ <signal>_SINGLE MReqLast MDataLast SRespLast	reqlast datalast & mdata resp & reslast
2.0	Burst_value_MAddr_INCR_burst_aligned	burstlength & addr & burstseq_incr_enable & burst_aligned & burstseq
2.0	Burst_value_MAddr_INCR_no_wrap	burstlength & burstseq_incr_enable & addr
2.0	burst_value_MBurstLength_ <sequence> WRAP XOR	burstlength & burstseq & burstseq_wrap_enable burstlength & burstseq_xor_enable & burstseq
2.0	Burst_value_MBurstLength_INCR_burst_aligned	burstlength & burstseq_incr_enable & burst_aligned & burstseq
2.0	burst_value_MBurstPrecise_ <sequence> WRAP XOR	burstprecise & burstseq_wrap_enable & burstlength & burstseq burstprecise & burstseq_xor_enable & burstlength & burstseq
2.0	Burst_value_MBurstPrecise_INCR_burst_aligned	burstaligned & burstprecise & burstseq_incr_enable & burstseq

2.0	Burst_value_MBurstPrecise_SRMD	burstprecise & burstsinglereq
2.0	Burst_value_MBurstSeq_UNKN_SRMD	burstsinglereq & burstreq & burstseq_unkn_enable
2.0	burst_value_MCmd_ <command> RDEX RDL WRC	burstlength & readex_enable burstlength & rdwrc_enable burstlength & rdwrc_enable
2.0	burst_value_MDataLast_ <mode> MRMD SRMD	datalast & mdata datalast & mdata
2.0	Burst_value_MReqLast_MRMD	reqlast
2.0	Burst_value_MReqLast_SRMD	Reqlast & burstsinglereq
2.0	burst_value_SRespLast_ <mode> MRMD SRMD	resplast & resp resplast & resp & burstsinglereq
2.1	burst_hold_MTagID_when_MTagInOrder_zero	burstlength & tags > 1
2.1	burst_hold_MTagInOrder	burstlength & tags > 1 & taginorder

4. Dataflow Transfer Checks		
Version	Name	Activation Parameters
2.0	transfer_phase_order_datahs_before_request_begin	datahandshake
2.0	transfer_phase_order_datahs_before_request_end	datahandshake
2.0	transfer_phase_order_response_before_request_begin	resp
2.0	transfer_phase_order_response_before_request_end	resp
2.0	transfer_phase_order_response_before_datahs_begin	resp & datahandshake
2.0	transfer_phase_order_response_before_datahs_end	resp & datahandshake
2.0	transfer_phase_order_response_before_last_datahs_begin_SRMD_wr	resp & datahandshake & burstsinglereq
2.0	transfer_phase_order_response_before_last_datahs_end_SRMD_wr	resp & datahandshake & burstsinglereq

5. Dataflow ReadEx Checks		
Version	Name	Activation Parameters
2.0	rdex_hold_ <signal> MAddr MAddrSpace	readex_enable & addr readex_enable & addrspace
2.0	rdex_hold_ <signal> MByteEn MDataByteEn)	readex_enable & byteen readex_enable & mdatabyteen & datahandshake
2.0	rdex_lock_release_no WR/WRNP	readex_enable
2.0	rdex_lock_release_no_burst_allowed	burstlength & readex_enable

6. Sideband Checks		
Version	Name	Activation Parameters
2.0	signal_valid_ <signal> MReset_n SReset_n	mreset sreset
2.0	signal_valid_ <signal> _when_reset_inactive ControlBusy ControlWr MError SError SInterrupt StatusBusy StatusRd	controlbusy controlwr merror serror interrupt statusbusy statusrd
2.0	signal_hold_ <signal> _16_cycles MReset_n SReset_n	mreset sreset
2.0	signal_hold_Control_after_reset	control
2.0	signal_hold_Control_2_cycles	control
2.0	signal_hold_Control_controlbusy_active	controlbusy
2.0	signal_hold_ControlWr_after_reset	controlwr
2.0	signal_value_ControlWr_control_transitioned	control & controlwr
2.0	signal_value_ControlWr_controlbusy_active	controlbusy & controlwr
2.0	signal_hold_ControlWr_2_cycles	controlwr
2.0	signal_value_ControlBusy	controlwr & controlbusy
2.0	signal_hold_StatusRd_2_cycles	statusrd
2.0	signal_value_StatusRd_statusbusy_active	statusrd & statusbusy

3 Compliance Checks Details

3.1 Introduction

This part of the document contains a detailed description of all compliance checks. Each entry has been created using the following template:

<compliance_check_name>

Nomenclature

Protocol version	Indicates to which OCP protocol version the check applies: <i>2.0, 2.1</i>
Protocol hierarchy	Indicates the check hierarchy: <i>request, response, datahandshake, burst, transfer, readEx, reset activity, control, or status</i>
Signal group	Indicates the signals type involved: <i>dataflow – basic bignals, simple extensions, burst extensions, tag extensions, thread extensions</i> <i>sideband – control or busy</i>
Critical signals	Indicates the OCP signal(s) impacted by the check: <i>any OCP signal name</i>
Assertion type	Gives the kind of check performed on the signals: <i>X,Z , hold, value, or ordering</i>

Description

Gives the detailed description of the compliance check.

References

Gives the related pages and sections in the OCP 2.0 / 2.1 specification documents.

3.2 Dataflow Signals Checks

3.2.1 signal_valid_<signal>_when_reset_inactive

Nomenclature

Protocol version	2.0
Protocol hierarchy	Reset activity
Signal group	Dataflow
Critical signals	MCmd, MDataValid, MThreadBusy, SDataThreadBusy, SResp, SThreadBusy
Assertion type	X, Z

Description

When reset is inactive, the following signals should never have an X or Z value on the rising edge of the OCP clock:

MCmd	MDataValid	MThreadBusy
SDataThreadBusy	SResp	SThreadBusy

References

OCP Specification, Release 2.0 references
p166, section "Signal Testing"

OCP Specification, Release 2.1 references
p194, section "Signal Testing"

3.2.2 request_valid_<signal>

Nomenclature

Protocol version	2.0
Protocol hierarchy	Request phase
Signal group	Dataflow
Critical signals	MAddr, MAddrSpace, MAtomicLength, MBurstLength, MBurstPrecise, MBurstSeq, MBurstSingleReq, MByteEn, MConnID, MReqLast, MThreadID, SCmdAccept
Assertion type	X, Z

Description

The following signals should never have an X or Z value on the rising edge of the OCP clock during a request phase:

MAddr	MAddrSpace	MAtomicLength
MBurstLength	MBurstPrecise	MBurstSeq
MBurstSingleReq	MByteEn	MConnID
MReqLast	MThreadID	SCmdAccept

The following exception applies: if datahandshake=1 and mdatabyteen=1 then MByteEn can be invalid for write accesses during the request phase

References

OCP Specification, Release 2.0 references
p166, section "Signal Testing"

OCP Specification, Release 2.1 references
p194, section "Signal Testing"

3.2.3 datahs_valid_<signal>

Nomenclature

Protocol version	2.0
Protocol hierarchy	Datahandshake
Signal group	Dataflow
Critical signals	MDataByteEn, MDataLast, MDataThreadID, SDataAccept
Assertion type	X, Z

Description

The following signals should never have an X or Z value on the rising edge of the OCP clock during a datahandshake phase:

MDataByteEn	MDataLast	MDataThreadID
SDataAccept		

References

OCP Specification, Release 2.0 references
p166, section "Signal Testing"

OCP Specification, Release 2.1 references
p194, section "Signal Testing"

3.2.4 response_valid_<signal>

Nomenclature

Protocol version	2.0
Protocol hierarchy	Response
Signal group	Dataflow
Critical signals	MRespAccept, SRespLast, SThreadID
Assertion type	X, Z

Description

The following signals should never have an X or Z value on the rising edge of the OCP clock during a response phase:

MRespAccept SRespLast SThreadID

References

OCP Specification, Release 2.0 references
p166, section "Signal Testing"

OCP Specification, Release 2.1 references
p194, section "Signal Testing"

3.2.5 request_valid_MTagInOrder

Nomenclature

Protocol version	2.1
Protocol hierarchy	Request
Signal group	Dataflow – tag extensions
Critical signals	MTagInOrder
Assertion type	X, Z

Description

MTagInOrder should not be X/Z during the request phase.

References

OCP Specification, Release 2.1 references
p154, section “Tags”

3.2.6 response_valid_STagInOrder

Nomenclature

Protocol version	2.1
Protocol hierarchy	Response
Signal group	Dataflow – tag extensions
Critical signals	STagInOrder
Assertion type	X, Z

Description

STagInOrder should not be X/Z during the response phase.

References

OCP Specification, Release 2.1 references
p154, section “Tags”

3.2.7 request_valid_MTagID_when_MTagInOrder_zero

Nomenclature

Protocol version	2.1
Protocol hierarchy	Request
Signal group	Dataflow – tag extensions
Critical signals	MTagID, MTagInOrder
Assertion type	X, Z

Description

If MTagInOrder is 0 during the request phase, MTagID should not be X/Z during the request phase.

References

OCP Specification, Release 2.1 references
p154, section “Tags”

3.2.8 datahs_valid_MDataTagID_when_MTagInOrder_zero

Nomenclature

Protocol version	2.1
Protocol hierarchy	Datahandshake
Signal group	Dataflow – tag extensions
Critical signals	MDataTagID, MTagInOrder
Assertion type	X, Z

Description

If datahandshake is active and MTagInOrder is 0 (during the request phase), MDataTagID should not be X/Z during the datahandshake phase.

References

OCP Specification, Release 2.1 references
p154, section “Tags”

3.2.9 response_valid_STagID_when_STagInOrder_zero

Nomenclature

Protocol version	2.1
Protocol hierarchy	Response
Signal group	Dataflow – tag extensions
Critical signals	STagID, STagInOrder
Assertion type	X, Z

Description

If STagInOrder is 0 during the response phase, STagID should not be X/Z during the response phase.

References

OCP Specification, Release 2.1 references
p154, section “Tags”

3.3 DataFlow Phase Checks

3.3.1 request_exact_SThreadBusy

Nomenclature

Protocol version	2.0
Protocol hierarchy	Request
Signal group	Dataflow – thread extensions
Critical signals	MCmd
Assertion type	Value

Description

If `sthreadbusy_exact = 1`, when a given slave thread is busy, the master must stay idle on this thread.

References

- OCP Specification, Release 2.0 references
 - p37, section “Ungrouped Signals”
 - p169, section “Request Phase Checks”
- OCP Specification, Release 2.1 references
 - p39, section “Ungrouped Signals”
 - p156, section “Threads”
 - p197, section “Request Phase Checks”

3.3.2 request_hold_<signal>

Nomenclature

Protocol version	2.0
Protocol hierarchy	Request
Signal group	Dataflow
Critical signals	MAddr, MCmd, MData, MAddrSpace, MByteEn, MDataInfo, MReqInfo, MAtomicLength, MBurstLength, MBurstPrecise, MBurstSeq, MBurstSingleReq, MReqLast, MConnID, MThreadID
Assertion type	Hold

Description

Once a request phase has begun, the following signals may not change their value until the OCP slave has accepted the request.

Basic Signals

MAddr
MCmd
MData

Simple Extensions

MAddrSpace
MByteEn
MDataInfo
MReqInfo

Burst Extensions

MAtomicLength
MBurstLength
MBurstPrecise
MBurstSeq
MBurstSingleReq
MReqLast

Thread Extensions

MConnID
MThreadID

The following exception applies: If datahandshake=1 and mdatabyteen=1 then MByteEn can change for write accesses during the request phase.

References

OCP Specification, Release 2.0 references
p167, section "Signal Retraction Testing"

OCP Specification, Release 2.1 references
p138, section "Request Phase"
p195, section "Signal Retraction Testing"

3.3.3 request_value_MCcmd_<command>

Nomenclature

Protocol version	2.0
Protocol hierarchy	Request
Signal group	Dataflow – basic signals
Critical signals	MCcmd
Assertion type	Value

Description

The following <Command> is illegal if the corresponding <Parameter> is set to 0.

Command	Parameter
BCST	broadcast_enable
RD	read_enable
RDEX	readex_enable
RDL	rdlwr_enable
WR	write_enable
WRC	rdlwr_enable
WRNP	writenonpost_enable

References

OCP Specification, Release 2.0 references
p168, section “Request Phase Checks”

OCP Specification, Release 2.1 references
p196, section “Request Phase Checks”

3.3.4 request_value_MAddr_word_aligned

Nomenclature

Protocol version	2.0
Protocol hierarchy	Request
Signal group	Dataflow – basic signals
Critical signals	MAddr
Assertion type	Value

Description

Signal MAddr must be OCP word aligned as follows:

data_width = 16 and MAddr[0] = 0
data_width = 32 and MAddr[1:0] = 0
data_width = 64 and MAddr[2:0] = 0
data_width = 128 and MAddr[3:0] = 0

References

- OCP Specification, Release 2.0 references p169, section “Request Phase Checks”
- OCP Specification, Release 2.1 references p197, section “Request Phase Checks”

3.3.5 request_value_MAtomicLength_0x0

Nomenclature

Protocol version	2.0
Protocol hierarchy	Request
Signal group	Dataflow – burst extensions
Critical signals	MAtomicLength
Assertion type	Value

Description

During a request phase, MAtomicLength must be non zero.

References

OCP Specification, Release 2.0 references
p18, section “Burst Extensions”

OCP Specification, Release 2.1 references
p19, section “Burst Extensions”

3.3.6 request_value_MBurstSeq_<sequence>

Nomenclature

Protocol version	2.0
Protocol hierarchy	Request
Signal group	Dataflow – burst extensions
Critical signals	MBurstSeq
Assertion type	Value

Description

The following <Burst type> is illegal if its corresponding <Parameter > is set to 0.

Burst type	Parameter
DLFT1	burstseq_dflt1_enable
DLFT2	burstseq_dflt2_enable
INCR	burstseq_incr_enable
STRM	burstseq_strm_enable
UNKN	burstseq_unkn_enable
WRAP	burstseq_wrap_enable
XOR	burstseq_xor_enable

References

OCP Specification, Release 2.0 references
 p47, section “Optional Burst Sequences”
 p170, section “Burst Checks”

OCP Specification, Release 2.1 references
 p51, section “Optional Burst Sequences”
 p198, section “Burst Checks”

3.3.7 request_value_MBurstSeq_0x7

Nomenclature

Protocol version	2.0
Protocol hierarchy	Request
Signal group	Dataflow – burst extensions
Critical signals	MBurstSeq
Assertion type	Value

Description

During a request phase, the MBurstSeq signal cannot take value 0x7.

References

OCP Specification, Release 2.0 references
p196, section “Burst Checks”

OCP Specification, Release 2.1 references
p198, section “Burst Checks”

3.3.8 request_value_MBurstLength_0x0

Nomenclature

Protocol version	2.0
Protocol hierarchy	Request
Signal group	Dataflow – burst extensions
Critical signals	MBurstLength
Assertion type	Value

Description

During a request phase, the MBurstLength signal cannot take value 0x0.

References

OCP Specification, Release 2.0 references
p172, section “Burst Checks”

OCP Specification, Release 2.1 references
p172, section “Burst Checks”

3.3.9 request_value_MByteEn_force_aligned

Nomenclature

Protocol version	2.0
Protocol hierarchy	Request
Signal group	Dataflow – simple extensions
Critical signals	MByteEn
Assertion type	Value

Description

If *force_aligned=1*, the Byte Enable values during a request phase are restricted to the following patterns for *data_width ≥ 32*:

data_width=32 and MByteEn has one of the following values:

```
0001
0010
0100
1000
0011
1100
1111
0000
```

data_width=64 and MByteEn has one of the following values:

```
00000001
00000010
00000100
00001000
00010000
00100000
01000000
10000000
00000011
00001100
00110000
11000000
00001111
11110000
11111111
00000000
```

data_width=128 and MByteEn has one of the following values:

```
0000000000000001
0000000000000010
00000000000000100
00000000000001000
```

```
000000000010000
000000000010000
000000001000000
000000001000000
0000000100000000
0000001000000000
0000010000000000
0000100000000000
0001000000000000
0010000000000000
0010000000000000
0100000000000000
0000000000000011
0000000000001100
0000000000110000
0000000011000000
0000011000000000
0000110000000000
0011000000000000
1100000000000000
0000000000001111
0000000011110000
0000111100000000
1111000000000000
0000000011111111
1111111100000000
1111111111111111
0000000000000000
```

The following exception applies: If datahandshake=1 and mdatabyteen=1 then MByteEn can change for write accesses during the request phase.

References

- OCP Specification, Release 2.0 references
 - p48, section "Byte Enable Patterns"
 - p168, section "Request Phase Checks"
- OCP Specification, Release 2.1 references
 - p51, section "Byte Enable Patterns"
 - p197, section "Request Phase Checks"

3.3.10 request_value_MThreadID

Nomenclature

Protocol version	2.0
Protocol hierarchy	Request
Signal group	Dataflow – thread extensions
Critical signals	MThreadID
Assertion type	Value

Description

MThreadID value is always < threads.

References

OCP Specification, Release 2.0 references
p169, section “Request Phase Checks”

OCP Specification, Release 2.1 references
p197, section “Request Phase Checks”

3.3.11 datahs_exact_SDataThreadBusy

Nomenclature

Protocol version	2.0
Protocol hierarchy	Datahandshake
Signal group	DataFlow – thread extensions
Critical signals	MDataValid
Assertion type	Value

Description

If `sdatathreadbusy_exact = 1`, when a given slave data thread is busy, the master must not present a data phase on this thread.

References

OCP Specification, Release 2.0 references
169, section “Datahandshake Phase”

OCP Specification, Release 2.1 references
p197, section “Datahandshake Phase”

3.3.12 datahs_hold_<signal>

Nomenclature

Protocol version	2.0
Protocol hierarchy	Datahandshake
Signal group	Dataflow
Critical signals	MData, MDataValid, MDataByteEn, MDataInfo, MDataLast, MDataThreadID
Assertion type	Hold

Description

Once a datahandshake phase has begun, the following signals may not change their value until the OCP slave has accepted the data.

Basic Signals

MData
MDataValid

Simple Extensions

MDataByteEn
MDataInfo

Burst Extensions

MDataLast

Thread Extensions

MDataThreadID

References

- OCP Specification, Release 2.0 references
p167, section "Signal Retraction Testing"
- OCP Specification, Release 2.1 references
p196, section "Signal Retraction Testing"

3.3.13 datahs_value_MDataByteEn_force_aligned

Nomenclature

Protocol version	2.0
Protocol hierarchy	Datahandshake
Signal group	Dataflow – simple extensions
Critical signals	MDataByteEn
Assertion type	Value

Description

If *force_aligned=1*, the Data Byte Enable values during a datahandshake phase are restricted to the following patterns for *data_width* ≥ 32:

data_width=32 and MDataByteEn has one of the following values:

```
0001
0010
0100
1000
0011
1100
1111
0000
```

data_width=64 and MDataByteEn has one of the following values:

```
00000001
00000010
00000100
00001000
00010000
00100000
01000000
10000000
00000011
00001100
00110000
11000000
00001111
11110000
11111111
00000000
```

data_width=128 and MDataByteEn has one of the following values:

```
00000000000000000001
00000000000000000010
00000000000000000100
00000000000000001000
```

000000000010000
0000000000100000
00000000001000000
000000000010000000
0000000100000000
0000001000000000
0000010000000000
0000100000000000
0001000000000000
0010000000000000
00100000000000000
0100000000000000
0000000000000011
0000000000001100
0000000000110000
0000000011000000
0000001100000000
0000110000000000
0011000000000000
1100000000000000
0000000000001111
0000000011110000
0000111100000000
1111000000000000
0000000011111111
1111111100000000
1111111111111111
0000000000000000

References

- OCP Specification, Release 2.0 references
p168, section "Datahandshake Phase"
- OCP Specification, Release 2.1 references
p197, section "Datahandshake Phase"

3.3.14 datahs_value_MDataThreadID

Nomenclature

Protocol version	2.0
Protocol hierarchy	Datahandshake
Signal group	Dataflow – thread extensions
Critical signals	MDataThreadID
Assertion type	Value

Description

MDataThreadID value must be < threads.

References

OCP Specification, Release 2.0 references
p20, section “Thread Extensions”

OCP Specification, Release 2.1 references
p23, section “Thread Extensions”

3.3.15 response_exact_MThreadBusy

Nomenclature

Protocol version	2.0
Protocol hierarchy	Response
Signal group	Dataflow – thread extensions
Critical signals	SResp
Assertion type	Value

Description

If mthreadbusy_exact = 1, when a given master thread is busy, the slave must not present a response on that thread.

References

OCP Specification, Release 2.0 references
p169, section “Response Phase”

OCP Specification, Release 2.1 references
p197, section “Response Phase”

3.3.16 response_hold_<signal>

Nomenclature

Protocol version	2.0
Protocol hierarchy	Response
Signal group	Dataflow
Critical signals	SData, SResp, SDataInfo, SRespInfo, SRespLast, SThreadID
Assertion type	Hold

Description

Once a response phase has begun, the following signals may not change their value until the master has accepted the response.

Basic Signals

SData
SResp

Simple Extensions

SDataInfo
SRespInfo

Burst Extensions

SRespLast

Thread Extensions

SThreadID

References

- OCP Specification, Release 2.0 references p167, section "Signal Retraction Testing"
- OCP Specification, Release 2.1 references p196, section "Signal Retraction Testing"

3.3.17 response_value_SResp_FAIL_without_WRC

Nomenclature

Protocol version	2.0
Protocol hierarchy	Response
Signal group	Dataflow – basic signals
Critical signals	SResp
Assertion type	Value

Description

The FAIL response can occur only on a WRC request.

References

- OCP Specification, Release 2.0 references
 - p40, section “Transfer Effects”
 - p169, section “Transfer-Based Checks”
- OCP Specification, Release 2.1 references
 - p42, section “Transfer Effects”
 - p198, section “Transfer-Based Checks”

3.3.18 response_value_SThreadID

Nomenclature

Protocol version	2.0
Protocol hierarchy	Response
Signal group	Dataflow – thread extensions
Critical signals	SThreadID
Assertion type	Value

Description

SThreadID value must be < threads.

References

OCP Specification, Release 2.0 references
p169, section “Response Phase”

OCP Specification, Release 2.1 references
p197, section “Response Phase”

3.3.19 request_hold_MTagInOrder

Nomenclature

Protocol version	2.1
Protocol hierarchy	Request
Signal group	Dataflow – tag extensions
Critical signals	MTagInOrder
Assertion type	Hold

Description

If taginorder = 1, the MTagInOrder signal cannot change until accepted by the OCP slave (SCmdAccept = 1).

References

OCP Specification, Release 2.1 references
p195, section “Signal Retraction Testing”, Table 43

3.3.20 response_hold_STagInOrder

Nomenclature

Protocol version	2.1
Protocol hierarchy	Response
Signal group	Dataflow – tag extensions
Critical signals	STagInOrder
Assertion type	Hold

Description

If taginorder = 1, the STagInOrder signal cannot change until it is accepted by the master (MRespAccept = 1).

References

OCP Specification, Release 2.1 references
p196, section “Signal Retraction Testing”, Table 45

3.3.21 request_hold_MTagID_when_MTagInOrder_zero

Nomenclature

Protocol version	2.1
Protocol hierarchy	Request
Signal group	Dataflow – tag extensions
Critical signals	MTagID, [MTagInOrder]
Assertion type	Hold

Description

If tags > 1, the MTagID signal cannot change until accepted by the OCP slave (SCmdAccept = 1).

References

OCP Specification, Release 2.1 references
p195, section “Signal Retraction Testing”, Table 43

3.3.22 datahs_hold_MDataTagID_when_MTagInOrder_zero

Nomenclature

Protocol version	2.1
Protocol hierarchy	Datahandshake
Signal group	Dataflow – tag extensions
Critical signals	MDataTagID, [MTagInOrder]
Assertion type	Hold

Description

When tags > 1, during a datahandshake phase corresponding to a non in-order request phase (MTagInOrder = 0), the MDataTagID signal cannot change value until accepted by the OCP slave (SDataAccept = 1).

References

OCP Specification, Release 2.1 references
p196, section “Signal Retraction Testing”, Table 44

3.3.23 response_hold_STagID_when_STagInOrder_zero

Nomenclature

Protocol version	2.1
Protocol hierarchy	Response
Signal group	Dataflow – tag extensions
Critical signals	STagID, [STagInOrder]
Assertion type	Hold

Description

If tags > 1, the STagID signal cannot change until it is accepted by the master (MRespAccept = 1).

References

OCP Specification, Release 2.1 references
p196, section “Signal Retraction Testing”, Table 45

3.3.24 request_value_MTagID_when_MTagInOrder_zero

Nomenclature

Protocol version	2.1
Protocol hierarchy	Request
Signal group	Dataflow – tag extensions
Critical signals	MTagID, [MTagInOrder]
Assertion type	Value

Description

The MTagID signal must always be < tags.

References

OCP Specification, Release 2.1 references
p197, section “Request Phase Checks”, bullet 8

3.3.25 datahs_value_MDataTagID_when_MTagInOrder_zero

Nomenclature

Protocol version	2.1
Protocol hierarchy	Datahandshake
Signal group	Dataflow – tag extensions
Critical signals	MDataTagID, [MTagInOrder]
Assertion type	Value

Description

The MDataTagID value must always be < tags.

References

OCP Specification, Release 2.1 references
p197, section “Datahandshake Phase Checks”, bullet 5

3.3.26 response_value_STagID_when_STagInOrder_zero

Nomenclature

Protocol version	2.1
Protocol hierarchy	Response
Signal group	Dataflow – tag extensions
Critical signals	STagID, [STagInOrder]
Assertion type	Value

Description

The STagID signal must always be < tags.

References

OCP Specification, Release 2.1 references
p197, section “Response Phase”, bullet 2

3.3.27 datahs_order_MDataTagID_when_MTagInOrder_zero

Nomenclature

Protocol version	2.1
Protocol hierarchy	Datahandshake
Signal group	Dataflow – tag extensions
Critical signals	MDataTagID, [MTagInOrder]
Assertion type	Data_order

Description

When datahandshake = 1, for tagged write transactions, the datahandshake phase must observe the same order as the request phase.

References

OCP Specification, Release 2.1 references
p48, section “Ordering Restrictions”, first paragraph

3.3.28 response_reorder_STagID_tag_interleave_size

Nomenclature

Protocol version	2.1
Protocol hierarchy	Response
Signal group	Dataflow – tag extensions
Critical signals	STagID
Assertion type	Reorder

Description

The slave must ensure that responses that are part of the same transaction stay together up to the tag_interleave_size. When tags > 1, the tag_interleave_size parameterizes the interleaving permitted for responses associated with packing burst sequences.

References

OCP Specification, Release 2.1 references
p48, section “Ordering Restrictions”, third paragraph
p53, section “Burst Interleaving with Tags”

3.3.29 response_reorder_STagID_overlapping_addresses

Nomenclature

Protocol version	2.1
Protocol hierarchy	Response
Signal group	Dataflow – tag extensions
Critical signals	STagID
Assertion type	Reorder

Description

Responses to requests that target overlapping addresses (as determined by MAddrSpace, MAddr and MByteEn) must not be re-ordered with respect to another. Note that this property does not need take into account the value of MTagInOrder.

References

OCP Specification, Release 2.1 references
p10, section “Tags”, third paragraph
p49, section “Ordering Restrictions”, fourth paragraph

3.4 Dataflow Burst Checks

3.4.1 burst_hold_MBurstLength_precise

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	DataFlow – burst extensions
Critical signals	MBurstLength
Assertion type	Hold

Description

For precise bursts, MBurstLength must hold its value during all request phases of the entire burst.

References

- OCP Specification, Release 2.0 references
p44, section “Constant Fields in Bursts”
p171, section “Burst Checks”
- OCP Specification, Release 2.1 references
p47, section “Constant Fields in Bursts”
p199, section “Burst Checks”

3.4.2 burst_hold_<signal>

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – burst extensions
Critical signals	MAddrSpace, MAtomicLength, MBurstPrecise, MBurstSeq, MBurstSingleReq, MCmd, MConnID, MReqInfo, (SRespInfo)
Assertion type	Hold

Description

The following signals must hold the same value on all request phases of the entire burst:

MAddrSpace
 MAtomicLength
 MBurstPrecise
 MBurstSeq
 MBurstSingleReq
 MCmd
 MConnID
 MReqInfo

The hold requirements for SRespInfo in a burst are different for the 2.0 versus 2.1 specifications.

In the OCP 2.0 specification p44 it is stated that: “SRespInfo must be held steady by the slave for every transfer in a burst”.

In the OCP 2.1 specification p47 it is stated that: “If possible, slaves should hold SRespInfo steady for every transfer in a burst”

Description

OCP Specification, 2.0 references
 p44, section “Constant Fields in Bursts”
 p171, section “Burst Checks”

OCP Specification, Release 2.1 references
 p47, section “Constant Fields in Bursts”
 p199, section “Burst Checks”

3.4.3 burst_hold_<signal>_STRM

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – simple extensions
Critical signals	MByteEn, MDataByteEn
Assertion type	Hold

Description

For streaming bursts, MByteEn / MDataByteEn must hold the same value on all request / datahandshake phases of the entire burst.

References

OCP Specification, Release 2.0 references
p43, section “Byte Enable Restrictions”
p172, section “Burst Checks”

OCP Specification, Release 2.1 references
p46, section “Byte Enable Restrictions”
p200, section “Burst Checks”

3.4.4 burst_phase_order_reqdata_together

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – basic signals
Critical signals	MCmd, MDatavalid
Assertion type	Ordering

Description

For Single Request Multiple Data bursts, if reqdata_together = 1, the master must present the request and first write data in the same cycle, and the slave must accept the request and the first write data in the same cycle.

References

- OCP Specification, Release 2.0 references p50, section “Phase Options”
- OCP Specification, Release 2.1 references p54, section “Phase Options”

3.4.5 burst_sequence_MAddr_INCR

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – basic signals
Critical signals	MAddr
Assertion type	Ordering

Description

Within an incrementing burst, the address increases for each new master request by the OCP word size.

References

OCP Specification, Release 2.0 references
p171, section “Burst Checks”

OCP Specification, Release 2.1 references
p200, section “Burst Checks”

3.4.6 burst_sequence_MAddr_STRM

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – basic signals
Critical signals	MAddr
Assertion type	Ordering

Description

Within a streaming burst, the address remains constant on all request phases of the burst.

References

OCP Specification, Release 2.0 references
p171, section “Burst Checks”

OCP Specification, Release 2.1 references
p199, section “Burst Checks”

3.4.7 burst_sequence_MAddr_WRAP

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – basic signals
Critical signals	MAddr
Assertion type	Ordering

Description

Within a wrapping burst, the address increases for each new master request by the OCP word size, and wraps on the burst length x OCP word size.

References

- OCP Specification, Release 2.0 references p171, section “Burst Checks”
- OCP Specification, Release 2.1 references p200, section “Burst Checks”

3.4.8 burst_sequence_MAddr_XOR

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow –basic signals
Critical signals	MAddr
Assertion type	Ordering

Description

Within an XOR burst, the address increases for each new OCP master request as follows:

BASE is the lowest byte address in the burst, which must be aligned with the total burst size.

FIRST_OFFSET is the byte offset (from BASE) of the first transfer in the burst.

CURRENT_COUNT is the count of current transfer in the burst starting at 0. WORD_SHIFT is the log2 of the OCP word size in bytes.

The current address of the transfer is $BASE | (FIRST_OFFSET \wedge (CURRENT_COUNT \ll WORD_SHIFT))$.

References

OCP Specification, Release 2.0 references
p42, section “Burst Address Sequence”

OCP Specification, Release 2.1 references
p45, section “Burst Address Sequence”

3.4.9 burst_value_<signal>_<sequence>

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – simple extensions
Critical signals	MByteEn, MDataByteEn
Assertion type	Value

Description

When mdatabyteen = 0, during STRM or DFLT2 bursts, MByteEn should never take the value 0.

When mdatabyteen = 1, during read-type STRM or DFLT2 bursts, MByteEn should never take the value 0.

When mdatabyteen = 1, during write-type STRM or DFLT2 bursts, MDataByteEn should never take value 0.

References

OCP Specification, Release 2.0 references
p43, section “Byte Enable Restrictions”

OCP Specification, Release 2.1 references
p46, section “Byte Enable Restrictions”

3.4.10 burst_value_<signal>_SINGLE

Nomenclature

Protocol version	2.0
Protocol hierarchy	Single
Signal group	Dataflow – burst extensions
Critical signals	MReqLast, MDataLast, SRespLast
Assertion type	Value

Description

Signal MDataLast must be 1 for the datahandshake phase of a single write-type request.

Signal MReqLast must be 1 for any single request.

Signal SRespLast must be 1 for the response to a single request.

References

OCP Specification, Release 2.0 references
p45, section “MReqLast, MDataLast, SRespLast”
p170, section “Burst Checks”

OCP Specification, Release 2.1 references
p45, section “MReqLast, MDataLast, SRespLast”
p198, section “Burst Checks”

3.4.11 burst_value_MAddr_INCR_burst_aligned

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – basic signals
Critical signals	MAddr
Assertion type	Value

Description

When *burst_aligned=1*, the first burst request of an incrementing burst must have its address aligned. The equation/example below indicates which MAddr bits must be 0.

Equation

$$\text{MAddr}[(\text{size}-1)+\text{BL}:0] = 0$$

Where:

size = $\log_2(\text{bytes}(\text{data_width}))$ for $\text{data_width} > 1$ byte

BL = $\log_2(\text{burst_length})$ for $\text{burst_length} > 1$

Example

For an interface with $\text{data_width}=32$, $\text{size}=2$ and:

$\text{burst_length} = 2$: $\text{MAddr}[2:0] = 0$

$\text{burst_length} = 4$: $\text{MAddr}[3:0] = 0$

References

OCP Specification, Release 2.0 references
 p48, section “Burst Alignment”
 p171, section “Burst Checks”

OCP Specification, Release 2.1 references
 p51, section “Burst Alignment”
 p199, section “Burst Checks”

3.4.12 burst_value_MAddr_INCR_no_wrap

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – basic signals
Critical signals	MAddr
Assertion type	Value

Description

An incrementing burst can never cross the address space boundary.

References

OCP Specification, Release 2.0 references
p171, section “Burst Checks”

OCP Specification, Release 2.1 references
p199, section “Burst Checks”

3.4.13 burst_value_MBurstLength_<sequence>

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – burst extensions
Critical signals	MBurstLength
Assertion type	Value

Description

The length of a wrapping or XOR burst must be a power of two.

References

OCP Specification, Release 2.0 references
p42, section “Burst Address Sequence”
p171, section “Burst Checks”

OCP Specification, Release 2.1 references
p45, section “Burst Address Sequence”
p199, section “Burst Checks”

3.4.14 burst_value_MBurstLength_INCR_burst_aligned

Nomenclature

Protocol version	2.0
Protocol hierarchy	Bust
Signal group	Dataflow – burst extensions
Critical signals	MBurstLength
Assertion type	Value

Description

When burst_aligned = 1, the length of an incrementing burst must be a power of two.

References

OCP Specification, Release 2.0 references
p48, section “Burst Alignment”
p171, section “Burst Checks”

OCP Specification, Release 2.1 references
p48, section “Burst Alignment”
p199, section “Burst Checks”

3.4.15 burst_value_MBurstPrecise_<sequence>

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – burst extensions
Critical signals	MBurstPrecise
Assertion type	Value

Description

Wrapping and XOR bursts can be issued only as precise bursts.

References

OCP Specification, Release 2.0 references
p42, section “Burst Address Sequence”
p171, section “Burst Checks”

OCP Specification, Release 2.1 references
p45, section “Burst Address Sequence”
p198, section “Burst Checks”

3.4.16 burst_value_MBurstPrecise_INCR_burst_aligned

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – burst extensions
Critical signals	MBurstPrecise
Assertion type	Value

Description

When burst_aligned = 1, incrementing bursts can be issued only as precise bursts.

References

OCP Specification, Release 2.0 references
p42, section “Burst Address Sequence”
p171, section “Burst Checks”

OCP Specification, Release 2.1 references
p42, section “Burst Address Sequence”
p198, section “Burst Checks”

3.4.17 burst_value_MBurstPrecise_SRMD

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – burst extensions
Critical signals	MBurstPrecise
Assertion type	Value

Description

Single Request Multiple Data transfers can be issued only as precise bursts.

References

- OCP Specification, Release 2.0 references
p45, section “Single Request / Multiple Data Bursts”
- OCP Specification, Release 2.1 references
p47, section “Single Request / Multiple Data Bursts”

3.4.18 burst_value_MBurstSeq_UNKN_SRMD

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – Burst Extensions
Critical signals	MBurstSeq
Assertion type	Value

Description

An unknown burst sequence (value UNKN) is illegal during a Single Request Multiple Data transfer.

References

OCP Specification, Release 2.0 references
p35, section “Phases in a Transfer”

OCP Specification, Release 2.1 references
p30, section “Signal Configuration”

3.4.19 burst_value_MCcmd_<command>

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – basic signals
Critical signals	MCcmd
Assertion type	Value

Description

The RDEX, RDL, and WRC commands cannot be part of a burst.

References

OCP Specification, Release 2.0 references
p42, section “Burst Definition”
p170, section “Burst Checks”

OCP Specification, Release 2.1 references
p44, section “Burst Definition”
p198, section “Burst Checks”

3.4.20 burst_value_MDataLast_ <mode>

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – burst extensions
Critical signals	MDataLast
Assertion type	Value

Description

The MDataLast signal must be 0 for all datahandshake phases in a write-type burst (either a MRMD or SRMD), except on the last one where it must be 1.

References

- OCP Specification, Release 2.0 references
 - p45, section “MReqLast, MDataLast, SRespLast”
 - p171, section “Burst Checks”
- OCP Specification, Release 2.1 references
 - p47, section “MReqLast, MDataLast, SRespLast”
 - p198, section “Burst Checks”

3.4.21 burst_value_MReqLast_MRMD

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – burst extensions
Critical signals	MReqLast
Assertion type	Value

Description

The MReqLast signal is 0 for all request phases of a MRMD burst, except on the last one where it must be 1.

References

- OCP Specification, Release 2.0 references
 - p45, section “MReqLast, MDataLast, SRespLast”
 - p172, section “Burst Checks”
- OCP Specification, Release 2.1 references
 - p47, section “MReqLast, MDataLast, SRespLast”
 - p199, section “Burst Checks”

3.4.22 burst_value_MReqLast_SRMD

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – burst extensions
Critical signals	MReqLast
Assertion type	Value

Description

The signal MReqLast must be 1 for any single request (SRMD being active or not).

References

OCP Specification, Release 2.0 references
p45, section “MReqLast, MDataLast, SRespLast”
p172, section “Burst Checks”

OCP Specification, Release 2.1 references
p47, section “MReqLast, MDataLast, SRespLast”
p199, section “Burst Checks”

3.4.23 burst_value_SRespLast_<mode>

Nomenclature

Protocol version	2.0
Protocol hierarchy	Burst
Signal group	Dataflow – burst extensions
Critical signals	SRespLast
Assertion type	Value

Description

The signal SRespLast must be 0 for all response phases in a MRMD burst, except on the last one where it must be 1.

The signal SRespLast must be 1 for any single response (with SRMD active or not).

References

- OCP Specification, Release 2.0 references
 - p45, section “MReqLast, MDataLast, SRespLast”
 - p172, section “Burst Checks”
- OCP Specification, Release 2.1 references
 - p47, section “MReqLast, MDataLast, SRespLast”
 - p199, section “Burst Checks”

3.4.24 burst_hold_MTagID_when_MTagInOrder_zero

Nomenclature

Protocol version	2.1
Protocol hierarchy	Burst
Signal group	Dataflow – tag extensions
Critical signals	MTagID, [MTagInOrder]
Assertion type	Hold

Description

The MTagID signal must remain constant for all transfers of a burst when MTagInOrder is zero. The master cannot interleave requests [or datahandshake] phases with different tags within a transaction. Note that this check should only focus on the request phase. The datahandshake phase is covered by phase property “datahs_order_MDataTagID_when_MTagInOrder_zero”. This last property checks that the datahandshake phase observes the same order as the request phase.

References

OCP Specification, Release 2.1 references
p48, section “Ordering Restrictions”, first paragraph

3.4.25 burst_hold_MTagInOrder

Nomenclature

Protocol version	2.1
Protocol hierarchy	Burst
Signal group	Dataflow – tag extensions
Critical signals	MTagInOrder
Assertion type	Hold

Description

The MTagInOrder signal must remain constant for all transfers of a burst.

References

OCP Specification, Release 2.1 references
p48, section "Ordering Restrictions", first paragraph

3.5 DataFlow Transfer Checks

3.5.1 transfer_phase_order_datahs_before_request_begin

Nomenclature

Protocol version	2.0
Protocol hierarchy	Transfer
Signal group	Dataflow – basic signals
Critical signals	MDataValid, MCmd
Assertion type	Ordering

Description

For each thread, for each transaction tag, a datahandshake phase cannot begin before the associated request phase begins, but can begin in the same clock cycle.

References

- OCP Specification, Release 2.0 references
p35, section “Phase Ordering Within a Transfer”
- OCP Specification, Release 2.1 references
p38, section “Phase Ordering Within a Transfer”

3.5.2 transfer_phase_order_datahs_before_request_end

Nomenclature

Protocol version	2.0
Protocol hierarchy	Transfer
Signal group	Dataflow – basic signals
Critical signals	MDataValid, MCmd
Assertion type	Ordering

Description

For each thread, for each transaction tag, a datahandshake phase cannot end before the associated request phase ends, but can end in the same clock cycle.

References

- OCP Specification, Release 2.0 references
p35, section “Phase Ordering Within a Transfer”
- OCP Specification, Release 2.1 references
p38, section “Phase Ordering Within a Transfer”

3.5.3 transfer_phase_order_response_before_request_begin

Nomenclature

Protocol version	2.0
Protocol hierarchy	Transfer
Signal group	Dataflow – basic signals
Critical signals	MCmd, SResp
Assertion type	Ordering

Description

For each thread, for each transaction tag, a response phase cannot begin before the associated request phase begins, but can begin in the same clock cycle.

References

- OCP Specification, Release 2.0 references
 - p35, section “Phase Ordering Within a Transfer”
 - p169, section “Response Phase”
 - p169, section “Transfer-Based Checks”
- OCP Specification, Release 2.1 references
 - p38, section “Phase Ordering Within a Transfer”
 - p197, section “Response Phase”
 - p197, section “Transfer-Based Checks”

3.5.4 transfer_phase_order_response_before_request_end

Nomenclature

Protocol version	2.0
Protocol hierarchy	Transfer
Signal group	Dataflow – basic signals
Critical signals	MCmd, SResp
Assertion type	Ordering

Description

For each thread, for each transaction tag, a response phase cannot end before the associated request phase ends, but can end in the same clock cycle.

References

- OCP Specification, Release 2.0 references
 - p35, section “Phase Ordering Within a Transfer”
 - p169, section “Response Phase”
 - p169, section “Transfer-Based Checks”
- OCP Specification, Release 2.1 references
 - p38, section “Phase Ordering Within a Transfer”
 - p197, section “Response Phase”
 - p197, section “Transfer-Based Checks”

3.5.5 transfer_phase_order_response_before_datahs_begin

Nomenclature

Protocol version	2.0
Protocol hierarchy	Transfer
Signal group	Dataflow – basic signals
Critical signals	MDataValid, SResp
Assertion type	Ordering

Description

For each thread, for each transaction tag, when datahandshake = 1, the response phase cannot begin before the associated datahandshake begins, but can begin in the same clock cycle.

References

- OCP Specification, Release 2.0 references
 - p35, section “Phase Ordering Within a Transfer”
 - p169, section “Response Phase”
 - p169, section “Transfer-Based Checks”
- OCP Specification, Release 2.1 references
 - p38, section “Phase Ordering Within a Transfer”
 - p197, section “Response Phase”
 - p197, section “Transfer-Based Checks”

3.5.6 transfer_phase_order_response_before_datahs_end

Nomenclature

Protocol version	2.0
Protocol hierarchy	Transfer
Signal group	Dataflow – basic signals
Critical signals	MDataValid, SResp
Assertion type	Ordering

Description

For each thread, for each transaction tag, when datahandshake = 1, the response phase cannot end before the associated datahandshake ends, but can end in the same clock cycle.

References

- OCP Specification, Release 2.0 references
 - p35, section “Phase Ordering Within a Transfer”
 - p169, section “Response Phase”
 - p169, section “Transfer-Based Checks”
- OCP Specification, Release 2.1 references
 - p38, section “Phase Ordering Within a Transfer”
 - p197, section “Response Phase”
 - p197, section “Transfer-Based Checks”

3.5.7 **transfer_phase_order_response_before_last_datahs_begin_SRMD_wr**

Nomenclature

Protocol version	2.0
Protocol hierarchy	Transfer
Signal group	Dataflow – basic signals
Critical signals	MDataValid, SResp
Assertion type	Ordering

Description

For each thread, for each transaction tag, with a write-type SRMD, the response phase cannot begin before the last datahandshake phase begins, but it can begin in the same clock cycle.

References

- OCP Specification, Release 2.0 references
 - p35, section “Phase Ordering Within a Transfer”
 - p169, section “Response Phase”
 - p169, section “Transfer-Based Checks”
- OCP Specification, Release 2.1 references
 - p38, section “Phase Ordering Within a Transfer”
 - p197, section “Response Phase”
 - p197, section “Transfer-Based Checks”

3.5.8 transfer_phase_order_response_before_last_datahs_end_SRMD_wr

Nomenclature

Protocol version	2.0
Protocol hierarchy	Transfer
Signal group	Dataflow – basic signals
Critical signals	MDataValid, SResp
Assertion type	Ordering

Description

For each thread, for each transaction tag, with a write-type SRMD, the response phase cannot end before the last datahandshake phase ends, but it can end in the same clock cycle.

References

- OCP Specification, Release 2.0 references
 - p35, section “Phase Ordering Within a Transfer”
 - p169, section “Response Phase”
 - p169, section “Transfer-Based Checks”
- OCP Specification, Release 2.1 references
 - p38, section “Phase Ordering Within a Transfer”
 - p197, section “Response Phase”
 - p197, section “Transfer-Based Checks”

3.6 DataFlow ReadEx Checks

3.6.1 rdex_hold_<signal>

Nomenclature

Protocol hierarchy	ReadEx
Signal group	Dataflow – Basic Signals
Critical signals	MAddr, MAddrSpace
Assertion type	Hold

Description

The unlocking command following a ReadEx must retain the same address and address space values.

References

- OCP Specification, Release 2.0 references
 - p39, section “Transfer Effects”
 - p42, section “Burst Definition”
 - p172, section “Read Exclusive Transaction Checks”
- OCP Specification, Release 2.1 references
 - p42, section “Transfer Effects”
 - p44, section “Burst Definition”
 - p200, section “Read Exclusive Transaction Checks”

3.6.2 rdex_hold_ <signal>

Nomenclature

Protocol hierarchy	ReadEx
Signal group	Dataflow – Simple Extensions
Critical signals	MByteEn, MDataByteEn
Assertion type	Hold

Description

When mdatabyteen = 0, the unlocking command following a ReadEx must retain the same MByteEn value.

When mdatabyteen = 1, the unlocking command following a ReadEx must retain for MDataByteEn the value given to MByteEn during the ReadEx command. If MByteEn is absent, MDataByteEn must be all 1's.

References

- OCP Specification, Release 2.0 references
 - p39, section "Transfer Effects"
 - p42, section "Burst Definition"
 - p172, section "Read Exclusive Transaction Checks"
- OCP Specification, Release 2.1 references
 - p42, section "Transfer Effects"
 - p44, section "Burst Definition"
 - p200, section "Read Exclusive Transaction Checks"

3.6.3 rdex_lock_release_no_WR/WRNP

Nomenclature

Protocol hierarchy	ReadEx
Signal group	Dataflow – Basic Signals
Critical signals	MCmd
Assertion type	Ordering

Description

If a ReadEx is issued on an address on a particular thread, no other request with the same address can be issued on any other thread until the ReadEx is unlocked.

The command following the ReadEx on the same thread must be a write command (WR or WRNP). This command unlocks the ReadEx.

References

OCP Specification, Release 2.0 references
p39, section "Transfer Effects"
p42, section "Burst Definition"

OCP Specification, Release 2.1 references
p42, section "Transfer Effects"
p44, section "Burst Definition"

3.6.4 rdex_lock_release_no_burst_allowed

Nomenclature

Protocol hierarchy	ReadEx
Signal group	Dataflow – Basic Signals
Critical signals	MBurstLength
Assertion type	Value

Description

The unlocking command following a RDEX must have MBurstLength = 1.

References

- OCP Specification, Release 2.0 references
 - p39, section "Transfer Effects"
 - p42, section "Burst Definition"
 - p172, section "Read Exclusive Transaction Checks"
- OCP Specification, Release 2.1 references
 - p42, section "Transfer Effects"
 - p44, section "Burst Definition"
 - p200, section "Read Exclusive Transaction Checks"

3.7 Sideband Checks

3.7.1 signal_valid_<signal>

Nomenclature

Protocol version	2.0
Protocol hierarchy	Reset Activity
Signal group	Sideband – Reset
Critical signals	MReset, SReset
Assertion type	X, Z

Description

Signals MReset_n and SReset_n are never X or Z.

References

OCP Specification, Release 2.0 references
p166-167

OCP Specification, Release 2.1 references
p194

3.7.2 signal_valid_ <signal> when_reset_inactive

Nomenclature

Protocol version	2.0
Protocol hierarchy	Reset Activity
Signal group	Sideband
Critical signals	ControlBusy, ControlWr, MError, SError, SInterrupt, StatusBusy, StatusRd
Assertion type	X, Z

Description

When reset is inactive, the following signals should never have an X or Z value on the rising edge of the OCP clock:

ControlBusy	ControlWr	MError
SError	SInterrupt	
StatusBusy	StatusRd	

References

OCP Specification, Release 2.0 references
p166-167

OCP Specification, Release 2.1 references
p194

3.7.3 signal_hold_<signal>_16_cycles

Nomenclature

Protocol version	2.0
Protocol hierarchy	Reset Activity
Signal group	Sideband – Reset
Critical signals	MReset, SReset
Assertion type	Hold

Description

If they are active, signals MReset_n and SReset_n must stay active at least 16 consecutive cycles.

References

- OCP Specification, Release 2.0 references
 - p38, section “Reset”
 - p172, section “Reset Checks”
- OCP Specification, Release 2.1 references
 - p40, section “Reset”
 - p200, section “Reset Checks”

3.7.4 signal_hold_Control_after_reset

Nomenclature

Protocol version	2.0
Protocol hierarchy	Control
Signal group	Sideband – Control
Critical signals	Control
Assertion type	Hold

Description

The Control signal must be held steady the first cycle after reset is de-asserted.

References

- OCP Specification, Release 2.0 references
 - p38, section "Status and Control"
 - p172, section "Control Checks"
- OCP Specification, Release 2.1 references
 - p41, section "Status and Control"
 - p200, section "Control Checks"

3.7.5 signal_hold_Control_2_cycles

Nomenclature

Protocol version	2.0
Protocol hierarchy	Control
Signal group	Sideband – Control
Critical signals	Control
Assertion type	Hold

Description

The Control signal must be held steady for a full cycle after the cycle in which it has transitioned.

References

OCP Specification, Release 2.0 references
p38, section "Status and Control"
p172, section "Control Checks"

OCP Specification, Release 2.1 references
p41, section "Status and Control"
p200, section "Control Checks"

3.7.6 signal_hold_Control_ControlBusy_active

Nomenclature

Protocol version	2.0
Protocol hierarchy	Control
Signal group	Sideband – Control
Critical signals	Control
Assertion type	Value

Description

If the ControlBusy signal was sampled active at the end of the previous cycle, the Control signal must not transition in the current cycle.

References

OCP Specification, Release 2.0 references
p38, section "Status and Control"
p172, section "Control Checks"

OCP Specification, Release 2.1 references
p41, section "Status and Control"
p200, section "Control Checks"

3.7.7 signal_hold_ControlWr_after_reset

Nomenclature

Protocol version	2.0
Protocol hierarchy	Control
Signal group	Sideband – Control
Critical signals	ControlWr
Assertion type	Hold

Description

The ControlWr signal must not be asserted in the cycle following a reset.

References

OCP Specification, Release 2.0 references
p172, section “Control Checks”

OCP Specification, Release 2.1 references
p200, section “Control Checks”

3.7.8 signal_value_ControlWr_Control_transitioned

Nomenclature

Protocol version	2.0
Protocol hierarchy	Control
Signal group	Sideband – Control
Critical signals	ControlWr
Assertion type	Value

Description

If signal Control transitions in a cycle, signal ControlWr must be driven active on that cycle.

References

- OCP Specification, Release 2.0 references
 - p38, section "Status and Control"
 - p172, section "Control Checks"
- OCP Specification, Release 2.1 references
 - p41, section "Status and Control"
 - p200, section "Control Checks"

3.7.9 signal_value_ControlWr_ControlBusy_active

Nomenclature

Protocol version	2.0
Protocol hierarchy	Control
Signal group	Sideband – Control
Critical signals	ControlWr
Assertion type	Value

Description

The ControlWr signal must not be asserted if ControlBusy is active.

References

OCP Specification, Release 2.0 references
p172, section “Control Checks”

OCP Specification, Release 2.1 references
p200, section “Control Checks”

3.7.10 signal_hold_ControlWr_2_cycles

Nomenclature

Protocol version	2.0
Protocol hierarchy	Control
Signal group	Sideband – Control
Critical signals	ControlWr
Assertion type	Hold

Description

The ControlWr signal must not remain asserted for two consecutive cycles.

References

OCP Specification, Release 2.0 references
p172, section “Control Checks”

OCP Specification, Release 2.1 references
p200, section “Control Checks”

3.7.11 signal_value_ControlBusy

Nomenclature

Protocol version	2.0
Protocol hierarchy	Control
Signal group	Sideband – Control
Critical signals	ControlBusy
Assertion type	Value

Description

The ControlBusy signal can only be asserted in a cycle after the ControlWr signal is asserted or after the reset transitions to inactive.

References

OCP Specification, Release 2.0 references
p173, section “Control Checks”

OCP Specification, Release 2.1 references
p200, section “Control Checks”

3.7.12 signal_hold_StatusRd_2_cycles

Nomenclature

Protocol version	2.0
Protocol hierarchy	Status
Signal group	Sideband – Status
Critical signals	StatusRD
Assertion type	Hold

Description

If the StatusRd signal was asserted in the previous cycle, it must not be asserted in the current cycle.

References

OCP Specification, Release 2.0 references
p39, section “Status and Control”
p173, section “Status Checks”

OCP Specification, Release 2.1 references
p41, section “Status and Control”
p201, section “Status Checks”

3.7.13 signal_value_StatusRd_StatusBusy_active

Nomenclature

Protocol version	2.0
Protocol hierarchy	Status
Signal group	Sideband – Status
Critical signals	StatusRD
Assertion type	Value

Description

The StatusRd signal must not be asserted while StatusBusy is asserted.

References

OCP Specification, Release 2.0 references
p173, section “Status Checks”

OCP Specification, Release 2.1 references
p201, section “Status Checks”

4 Functional Coverage

4.1 Introduction

The functional coverage approach described in this chapter is bottom-up, meaning the analysis starts at the signal level and goes up to the transaction level. Note that the transfer level has been skipped for reasons highlighted in paragraph 4.3. Along this path several coverage types are used.

The coverage types used at signal level are:

- Toggle coverage
- State coverage
- Meta coverage

The coverage types used at transaction level are:

- Cross coverage
- Meta coverage

Toggle Coverage

Toggle coverage provides baseline information that a system is connected properly, and that higher level coverage or compliance failures are not simply the result of connectivity issues. Toggle coverage answers the question: “Did a bit change from a value of 0 to 1 and back from 1 to 0?” It does not indicate that every value of a multi-bit vector was seen. It simply measures that all the individual bits of a multi-bit vector did toggle. Note that in certain cases, not all bits can toggle. A system which only supports RD commands, (“010”) for example, will only need toggle coverage on MCcmd bit1. MCcmd bit0 and bit2 will always be ‘0’. Therefore they must be filtered from the MCcmd toggle coverage.

State Coverage

State coverage applies to signals which are a minimum of two (2) bits wide. In most cases, the states (also commonly referred to as coverage bins) can be easily identified as all possible combinations of the signal. For example, for the SResp signal, the states could be “00” (IDLE), “01” (DVA), “10” (FAIL) and “11” (ERR). If the state space is too large, an intelligent classification of the states must be made. In the case of the MAddr signal for example, a possible choice of the coverage bins could be one (1) bin to cover the lower address range, one (1) bin to cover the upper address range and one (1) bin to cover all other intermediary addresses.

Meta Coverage

Meta coverage is collecting second-order coverage data. Possible meta coverage measurements include: accept backpressure delays, threadbusy backpressure delays and inter-phase delays. Meta coverage information is particularly useful to flag excessive latencies (possibly indicating dead-locks) and to evaluate the OCP backpressure mechanisms (accept / threadbusy).

Cross Coverage

Cross coverage measures the activity of one (1) or multiple categories. A category is defined at the transaction level. It typically groups multiple OCP signals to form a more abstract, higher-level view of a particular aspect of the OCP protocol. The most pertinent category example is the ‘transTypes’. This category combines the MCcmd, MBurstLength and MBurstSingleReq signals into the higher-level ‘transTypes’ category. Cross coverage on one category, for example the ‘transTypes’ category, indicates which kind of transactions were applied to the system under test (e.g., MRMD-RD-4, SINGLE-WRNP, etc.). Cross coverage on multiple categories, for example the ‘transTypes’ and ‘transResults’ categories, not only provides information about the transactions applied to the system, but also on their results. In essence, cross coverage measures the types of transactions passing through a system.

4.2 Signal Level

Table 4-1 on the next page summarizes the OCP functional coverage approach for the signal level. The table maps all the OCP signals (non-sideband) into phase groups (req / datahs / resp) and provides coverage information in the two outermost right columns. Each coverage type field is colored either in green or in gold. Green fields are mandatory for functional coverage. Gold fields are optional for functional coverage.

'Level 1 – Baseline' Coverage Type

The purpose of the 'level 1 – baseline' column is to establish a solid baseline for the signal level functional coverage. Hence it contains only mandatory coverage. The coverage type used at 'level 1 - baseline' is toggle coverage. Toggle coverage will provide a minimum level of confidence to the verification engineer that the device under test is alive and properly connected to the rest of the system. It proves as well that no OCP signals are stuck-at-0 or stuck-at-1. Note that in certain cases, filters should be applied to the toggle coverage to exclude coverage of bits, which can never toggle (refer to the MCcmd example in paragraph 4.1).

'Level 2' Coverage Type

The 'level 2' coverage type column defines coverage in addition to the basic toggle coverage of the 'level 1 - baseline' column. Possible coverage types are state or meta. State coverage will define states (bins) for a multi-bit vector to provide a higher level of abstraction. Meta coverage will cover accept / threadbusy backpressure delays.

The 'level 2' column contains a mix of green (mandatory) and gold (optional) fields.

Mandatory fields must be covered. They are with their respective coverage type:

- MAddr/MCcmd/SResp : state coverage
- MAddrSpace/MByteEn/MDataByteEn : state coverage
- MAtomicLength/MBurstLength/MBurstSeq : state coverage
- MTagID/MDataTagID/STagID : state coverage
- MThreadID/MDataThreadID/SthreadID : state coverage

Optional column 'level 2' fields and their coverage types are:

- MData/SData : state coverage
- MDataValid : meta coverage
- SCmdAccept/SDataAccept/MRespAccept : meta coverage
- MReqInfo/MDataInfo/SDataInfo/SRespInfo : state coverage
- MConnID : state coverage
- SThreadBusy/SDataThreadBusy/MThreadBusy : meta coverage

Signals which are only one (1) bit wide only have toggle coverage:

- MBurstPrecise/MBurstSingleReq
- MReqLast/MDataLast/SRespLast
- MTagInOrder/STagInOrder

Table 4-2 outlines possibilities for signal level meta coverage. For each phase group (req / datahs / resp), two (2) meta coverage types are identified: accept backpressure delay and threadbusy backpressure delay. As indicated, other interesting meta coverage types could be identified; those presented are the most trivial.

Table 4-1. Signal Level Functional Coverage

Signal Group	Signal	Phase Groups			Coverage Type	
		Req	Datahs	Resp	Level 1 Baseline	Level 2
Basic	MAddr	X			Toggle	State
	MCmd	X			Toggle	State
	MData	X			Toggle	State
	MDataValid		X		Toggle	Meta
	MRespAccept			X	Toggle	Meta
	SCmdAccept	X			Toggle	Meta
	SData			X	Toggle	State
	SDataAccept		X		Toggle	Meta
SResp			X	Toggle	State	
Simple	MAddrSpace	X			Toggle	State
	MByteEn	X			Toggle	State
	MDataByteEn		X		Toggle	State
	MDataInfo		X		Toggle	State
	MReqInfo	X			Toggle	State
	SDataInfo			X	Toggle	State
SRespInfo			X	Toggle	State	
Burst	MAtomicLength	X			Toggle	State
	MBurstPrecise	X			Toggle	
	MBurstLength	X			Toggle	State
	MBurstSeq	X			Toggle	State
	MBurstSingleReq	X			Toggle	
	MDataLast		X		Toggle	
	MReqLast	X			Toggle	
SRespLast			X	Toggle		
Tag	MDataTagID		X		Toggle	State
	MTagID	X			Toggle	State
	MTagInOrder	X			Toggle	
	STagID			X	Toggle	State
	STagInOrder			X	Toggle	
Thread	MConnID	X			Toggle	State
	MDataThreadID		X		Toggle	State
	MThreadBusy			X	Toggle	Meta
	MThreadID	X			Toggle	State
	SDataThreadBusy		X		Toggle	Meta
	SThreadBusy	X			Toggle	Meta
SThreadID			X	Toggle	State	

Table 4-2. Signal Level Meta Coverage Examples

Phase Group	Meta Coverage Types	Coverage Details
Request phase	accept backpressure delay	MCmd – SCmdAccept delay
	threadbusy backpressure delay	SThreadBusy backpressure delay (per bit).
Datahs phase	accept backpressure delay	MDataValid – SDataAccept delay
	threadbusy backpressure delay	SDataThreadBusy backpressure delay (per bit)
Response phase	accept backpressure delay	SResp – MRespAccept delay
	threadbusy backpressure delay	MThreadBusy backpressure delay (per bit)

4.3 Transfer Level

As indicated in the introduction paragraph 4.1, the transfer level for functional coverage is being skipped and the transaction level will be discussed next. The underlying motivations are:

- The most obvious order for the OCP functional coverage definition would have been to follow the OCP hierarchy: signal, phase, transfer, transaction. However, such reasoning does not work well for SRMD bursts. SRMD bursts can be constructed as 1 req + n datahs + 1 resp. As such, the transfer concept does not 100% apply because the number of phases per transfer is not constant. Since it is desirable to have a uniform functional coverage definition, which applies to all OCP transactions (MRMD / SRMD / SINGLE), it makes sense to skip the transfer level.
- Even if the transfer level was included, there are no valuable coverage points. The combination of phases into transfers is a pure protocol-check related matter. However, meta coverage to measure inter-phase delays could be interesting and is therefore discussed at the transaction level.

4.4 Transaction Level

'TransTypes' Concept

Before discussing coverage at the transaction level, clarification is required concerning the process of getting from the signal level to the transaction level. In essence, signals are combined into phases which are then combined into transactions. Table 4-3 below summarizes this process.

Table 4-3. TransTypes

MBurstSingleReq	MCmd	Phases			Enabling Condition	
		Req	Datahs	Resp	Datahandshake	Writeresp_enable
0 MRMD SINGLE transfer	RD/RDEX/RDL	L		L		
	WR/BCST	L			0	0
	WR/BCST/ WRNP/WRC	L		L	0	1
	WR/BCST	L	L		1	0
	WR/BCST/ WRNP/WRC	L	L	L	1	1
1 SRMD SINGLE transfer	RD	1		L		
	WR/BCST	1	L		1	0
	WR/BCST/ WRNP	1	L	1	1	1

Notes for Table 4-3 above:

1. Table 4-3 is based on the 'phases in a transfer' paragraph in the Protocol Semantics chapter of the OCP 2.0/2.1 specifications.
2. It shows how phases are combined into SINGLE transfers, MRMD bursts and SRMD bursts. Note that SINGLE transfers can be de-generated from either MRMD or SRMD bursts.
3. 'L' stands for the MBurstLength [one (1) in the case of a SINGLE transfer].
4. The outcome of this table is referred to from now on as the 'transTypes'.
5. The 'transTypes' are controlled by the following parameters:
 - a) Three (3) signals: MBurstSingleReq, MCmd and MBurstLength ('L')
 - b) Two (2) configuration parameters: datahandshake and writeresp_enable
6. RDEX, RDL and WRC commands only apply to SINGLE transfers.

7. RD, RDEX and RDL are not controlled by the datahandshake and writeresp_enable configuration parameters.
8. The possible 'transTypes' are:
 - a) Seven (7) SINGLE transfers : RD/WR/BCST/WRNP/RDEX/RDL/WRC
 - b) Four (4) MRMD bursts : RD/WR/BCST/WRNP
 - c) Four (4) SRMD bursts : RD/WR/BCST/WRNP
9. The following example illustrates this.
 If MBurstSingleReq supports values '0' and '1' and
 If MCmd only supports RD,WR,WRNP,RDEX and
 If datahandshake == '1' and writeresp_enable == '1'
 then the 'transTypes' will be:
 - a) SINGLE transfers, RD/WR/WRNP/RDEX
 - b) MRMD bursts, RD/WR/WRNP
 - c) SRMD bursts, RD/WR/WRNP

'Category' Concept

A category:

- Groups one (1) or more OCP signals.
- Serves as a building block for cross coverage.
- Represents a higher level view of the OCP protocol, such that intelligent crosses can be made of one (1) or several categories.

Table 4-4 below lists and describes the proposed categories:

Table 4-4. Categories

#	Category Name	Category Description
1	transTypes	Category containing the transaction types based on Table 4-3
2	transTargets	Category containing the transaction targets (MAddr / MAddrSpace)
3	transResults	Category containing the transaction results (SResp)
4	transBurstProps	Category containing the burst properties (AtomicLength / MBurstPrecise / MBurstSeq)
5	transByteens	Category containing the transaction byte enables (MByteEn / MDataByteEn)
6	flowThreads	Category containing the flows as function of the ThreadID
7	flowTagTypes	Category containing the flows as function of the TagInOrder / TagID

Notes for Table 4-4 above:

1. The 'transTypes' were discussed in detail in Table 4-3.
2. The 'transBurstProps' category may be split into multiple categories enabling a higher granularity for cross coverage.
3. The 'flowTagTypes' category combines the TagInOrder and TagID signals.
 The tag type is encoded as follows:
 - a) If TagInOrder, then tag type == tag-in-order (1 enumerated value)
 - b) If not, then tag type == the TagID (multiple enumerated values)
 If the TagID range is too large, sub-ranges should be defined. As such, the tag type will have 1 + x enumerated values.

Mapping Signals into Categories

Table 4-5 below shows the OCP signals (non-sideband) mapped into the categories described in the previous section. Only signals which were not optional in the 'level 2' column of Table 4-1, and which are not MReqLast / MDataLast / SRespLast, are mapped.

Table 4-5. Signal Mapping into Categories

Signal	Categories						
	1	2	3	4	5	6	7
	transTypes	transTargets	transResults	transBurstProps	transByteens	flowThreads	flowTransTypes
MAddr		X					
MCmd	X						
SResp			X				
MAddrSpace		X					
MByteEn					X		
MDataByteEn					X		
MAtomicLength				X			
MBurstPrecise				X			
MBurstLength	X						
MBurstSeq				X			
MBurstSingleReq	X						
MDataTagID							X
MTagID							X
MTagInOrder							X
STagID							X
STagInOrder							X
MDataThreadID						X	
MThreadID						X	
SThreadID						X	

Cross Coverage of One (1) Category (=> optional)

As mentioned in the Introduction chapter, cross coverage can be applied to just one (1) category. Obviously, this kind of cross coverage only makes sense if a category contains more than one (1) signal. As such, the 'transResults' and 'transByteens' categories are excluded from this type of cross coverage.

Cross coverage of one (1) category can be useful to measure, for example, what kind of 'transTypes' flowed through a design regardless of the signals contained in other categories (for example, the 'transResults'). However, a more profound coverage can be obtained by applying crosses among several categories as outlined in the next paragraph. Therefore, cross coverage of one category is considered optional, and cross coverage on multiple categories (see below) is considered mandatory.

Cross Coverage on Multiple Categories (=> mandatory)

Cross coverage can also be applied to combinations of the previously defined categories. Theoretically, many crosses are possible (2^{exp7} or 128 in total), but only some will make sense for a specific OCP interface configuration and design architecture.

The crosses between the 'transTypes' category and one (1) or more other categories are considered mandatory. They will establish a solid base for cross coverage at the transaction level. Table 4-6 below shows some of the mandatory crosses. It is clear that these mandatory crosses form a sub-set of the theoretical possibilities. It is up to the user to declare additional crosses which exclude the 'transTypes' category, but are important for the system under test. Such crosses are considered optional.

Table 4-6. Examples of Mandatory Crosses of Multiple Categories (Including 'transType')

Categories							Cross Description
1	2	3	4	5	6	7	
transTypes	transTargets	transResults	transBurstProps	transByteens	Flow Threads	flowTagTypes	
X	X						Cross all transaction types with all targets
X					X		Cross all transaction types for all threads
X		X					Cross all transaction types with all transaction results
X			X	X			Cross all transaction types for all bursts with all byte enable patterns
X		X	X				Cross all transaction types for all bursts with the transaction results

Meta Coverage

Table 4-7 outlines possibilities for the transaction level meta coverage. Three (3) meta coverage types are identified: accept backpressure delays relative to the position in a transaction, threadbusy backpressure delays relative to the position in a transaction and several inter-phase delays. As indicated, other interesting meta coverage types could be identified; those presented are the most trivial ones.

Table 4-7. Transaction Level Meta Coverage Examples

Meta Coverage Types	Coverage Details
inter-phase delays	req to req / datahs to datahs / resp to resp delays
	req to datahs / req to resp / datahs to resp delays
	1 st req accepted delay / last req accepted delay for MRMD bursts
accept backpressure delays relative to the position in the transaction	Measure when accept backpressure occurs in a transaction
threadbusy backpressure delays relative to the position in the transaction	Measure when threadbusy backpressure occurs in a transaction

4.5 Sideband Signals Coverage

Toggle Coverage

Similar to the dataflow signals discussed before, toggle coverage must be applied to each individual bit of the sideband signals to establish a solid coverage baseline. It is mandatory.

State Coverage

Sideband signals which consist of multiple bits can optionally have state coverage similar to the dataflow signals.

Meta Coverage

Meta coverage may be added for the control and status signals.

Some examples of meta coverage that might be added for the control signals handshake are:

- The delay in between two ControlWr signal assertions.
- The length of the ControlBusy signal assertion.
- The ControlBusy assertion relative to the previous ControlWr assertion.

Some examples of meta coverage that might be added for the status signals handshake are:

- The delay in between two StatusRd signal assertions.
- The length of the StatusBusy signal assertion.

4.6 Naming Conventions

This paragraph deals with naming conventions for functional coverage. It's divided into three (3) main sections: signal level (dataflow signals), transaction level (dataflow signals) and sideband signals. All naming conventions are illustrated with some concrete examples.

Signal Level (Dataflow Signals)

Naming template:

signal_<coverage type>_<signal name | meta name>_<bin>

In which:

<coverage type> : toggle
state
meta
<signal name> : OCP signal
<meta name> : SCmdAcceptDelay
SDataAcceptDelay
MRespAcceptDelay
SThreadBusyDelay
SDataThreadBusyDelay
MThreadBusyDelay
<bin> : if enumerated types are defined in OCP use them
for example: SResp in [ERR,DVA,FAIL,IDLE]
else be free to choose a clear name

Examples:

signal_toggle_MAddr_bit0_0to1
signal_state_MByteEn_allOnes
signal_meta_SThreadBusyDelay_2

Transaction Level (Dataflow Signals)

Naming template:

```
trans_<coverage type>_<cross name | meta name>[_<bin>]
```

In which:

```
<coverage type> : cross
                  meta
<cross name>    : for cross coverage of 1 category:
                  transTypes
                  transTargets
                  transBurstProps
                  flowThreads
                  flowTagTypes
                  for cross coverage of multiple categories:
                  trans_<list of ...>_flow_<list of ...>
                    Types           Threads
                    Results         TagTypes
                    Targets
                    BurstProps
                    Byteens
<meta name>    : ScmdAcceptDelay
                  SDataAcceptDelay
                  MRespAcceptDelay
                  SThreadBusyDelay
                  SDataThreadBusyDelay
                  MThreadBusyDelay
                  ReqReqPhaseDelay
                  ReqRespPhaseDelay
                  ...
[<bin>]        : The bin naming is optional for cross coverage.
                  In most cases the bins will automatically be chosen by the verification tool
                  itself. However if the cross includes signals which have specific OCP
                  enumerated values defined (as DVA for SResp), it's advisable to use them.
```

Examples:

```
trans_cross_transTypes
trans_cross_trans_TypesResults
trans_cross_flow_ThreadsTagTypes
trans_cross_trans_TypesResults_flow_Threads
trans_meta_ReqReqPhaseDelay_4
```

Sideband Signals

Naming template:

sideband_<coverage type>_<signal name | meta name>_<bin>

In which:

<coverage type> : toggle
state
meta
<signal name> : OCP signal
<meta name> : ControlWrControlWrDelay
ControlBusyDuration
ControlWrControlBusyDelay
StatusRdStatusRdDelay
StatusRdDuration
<bin> : be free to choose a clear name

Examples:

sideband_toggle_ControlWr
sideband_state_Control_3
sideband_meta_StatusRdDuration_5

OCP-IP Association
3855 SW 153rd Drive
Beaverton, OR 97006
Ph: 503-619-0560
Fax: 503-644-6708
admin@ocpip.org
www.ocpip.org

The Overriding Document On OCP Certification Is the Current OCP Specification

