Worker Interface Profiles (WIP)

Functional Specification

Includes the OCP Profiles for:

- Worker Control Interface (WCI)
- Worker Streaming Interface (WSI)
- Worker Message Interface (WMI)
- Worker Memory Interface (WMemI)
- Worker Time Interface (WTI)

Authors: Jim Kulp, Shepard Siegel

25

Revision History

Revision	Description of Change	By	Date
0.01	Creation	ssiegel	2006-10-12
0.02	Progressive update	ssiegel	2006-10-17
0.03	Progressive update	ssiegel	2006-10-17
0.04	WCI Focus	ssiegel	2006-11-08
0.05	WCI Gloss, Start in on WSI	ssiegel	2006-11-09
0.06	WSI Focus	ssiegel	2006-11-10
0.07	WSI Re-Focus (rewrite)	ssiegel	2006-11-13
0.08	WCI, WSI Gloss	ssiegel	2006-11-14
0.09	WMemI Rough-out 6.1 and 6.2	ssiegel	2006-11-15
0.10	WMemI Rough-out 6.3	ssiegel	2006-11-16
0.11	WMemI continue	ssiegel	2006-11-17
0.12	WMemI choices 6.5	ssiegel	2006-11-21
0.13	Add to overview, start WMI section 5.1	ssiegel	2006-11-22
0.14	WMI continue	ssiegel	2006-11-28
0.15	WMI continue, gloss 5.1, removed unnecessary details	ssiegel	2006-11-29
0.16	WMI continue, introduce “Worker Choices” Section	ssiegel	2006-11-30
0.17	Reformat with x.2 “Worker Choices”	ssiegel	2006-12-01
0.18	Refine terminology “attributes”	ssiegel	2006-12-02
0.19	Incorporate Jim’s v.17 comments	ssiegel	2006-12-04
0.20	Incorporate Jim’s v.19 comments	ssiegel	2006-12-05
0.21	WMemI round-out section 6.1 and 6.2	ssiegel	2006-12-05
0.22	WMemI gave into OCP 8-bits per Byte Rule	ssiegel	2006-12-06
0.23	Gloss section WMemI section, incorporate Jim’s v22 feedback	ssiegel	2006-12-07
0.24	Introduce “Immediate” attribute to WMemI as per team meeting	ssiegel	2006-12-11
0.25	Gloss for first document check-in	ssiegel	2006-12-14
0.26	Add WCI State Machine and Control Operations, Jim’s feedback	ssiegel	2006-12-19
0.27	WMemI feedback	ssiegel	2006-12-20
0.28	WMI opcode and message length fix, first checkin-candidate	ssiegel	2006-12-21
0.29	Incorporate Changes from first specification review	ssiegel	2006-12-28
0.30	Continued glossing	ssiegel	2006-12-29
0.31	Remove imprecise SR-MD from WMemI	ssiegel	2007-01-10
0.32	Incorporate feedback from formal design review, part one	ssiegel	2007-02-13
0.33	Incorporate feedback from formal design review, part two	ssiegel	2007-03-02
0.34	Refine explanation of Message and Burst lengths	ssiegel	2007-03-07
0.35	Minor changes in advance of glossing review	ssiegel	2007-03-09
0.36	Rename to “Worker Interface Profiles”	ssiegel	2007-03-14
0.37	Burst Precise discussion	ssiegel	2007-03-15
0.38	WCI: Detail the writeresp_enable behavior	ssiegel	2007-05-22
0.39	Clarify/Correct use of SResp Error (ERR) Response	ssiegel	2007-05-25
0.40	Tan Shade PreciseBurstsOnly in WSI and WMI, detail difference	ssiegel	2007-06-08
0.41	Added stronger NDA mark to cover page and footer	ssiegel	2007-06-22
0.42	Added observation on framing and MDataLast to WSI (page 42)	ssiegel	2007-06-28
0.43	Added Reset observation	ssiegel	2007-07-23
0.43 (cont.)	Attribute massaging for Precise/Imprecise Burst attributes	ssiegel	2007-07-26
0.44	Attribute massaging for Precise/Imprecise Burst attributes	ssiegel	2007-07-27
0.44 (cont.)	WCI force_aligned defect corrected	ssiegel	2007-08-01
0.45	WCI MFlag response rules clarified	ssiegel	2007-08-06
0.46	WCI nit, WMemI signals, Burst attributes changed	ssiegel	2007-08-07
0.47	Remove BurstCapable attribute from WMemI	ssiegel	2007-08-08
0.48	Remove MaxIdleCycles and MaxIdleRules	ssiegel	2007-08-31
0.48 (cont.)	Add Continuous attribute (WSI, WMI, and WMemI)	ssiegel	2007-09-04
0.49	Refine Continuous attribute (WSI, WMI, and WMemI) from feedback	ssiegel	2007-09-06
0.50	First inclusion of the schema information	ssiegel	2007-09-10
0.51	Incorporate Shawn’s markup to v.47 (part 1)	ssiegel	2007-09-10
0.52	WCI discussion embellished	ssiegel	2007-09-11
0.53	Corrected wording for WMemI dataaccept configuration paramater	ssiegel	2007-10-26
0.54	Corrected over-determined case for WSI attributes wrt MessageLengthIsExplicit	ssiegel	2007-11-29
0.55	Added WSI RequestCanLead attribute	ssiegel	2007-11-30
0.56	Renamed WSI RequestCanLead attribute to EarlyRequest	ssiegel	2007-12-03
0.57	Added WSI MaxMessageBytes, RemainderEOM Attribute	ssiegel	2008-02-04
0.58	Refined RemainderEOM attribute to support zero-length transfers	ssiegel	2008-03-31
0.61	Major Editorial Update, plus revert RemainderEOM to byte enables in WSI	jkulp	2008-08-19
0.66	Continue Major Editorial edit plus factoring data interface attributes, and more precise endian support, for review of sections 1-6	Jkulp	2008-10-06
0.72	Finish major editorial update, plus more consistency, OCP compliance, abortable attribute on WSI, simplify WMI, fix heading levels in XML schema section	Jkulp	2008-10-21
1.0	Enter final comments, all typos and clarifications. Add MReqLast to WMemI	jkulp	2008-11-13
1.1	OpenCPI initial version.	Jkulp	2009-10-14
1.2	Added WTI, and clarified areas based on automation (0.77)	jkulp	2010-01-20
1.3	Some typos, property metadata details, WSI abort to DataInfo, reset/clock clarifications	jkulp	2010-06-12

26

27

Table of Contents

1 References

When the term "OCP Specification" is used in this document, the reference is to version 2.2 of the OCP Specification, Document Revision 1.0. The OCP specification and supporting information are available from www.ocpip.org .

Table 1 - Table of Reference Documents

Title	Published By	Link
Open Core Protocol Specification version 2.2	OCP-IP	Public URL: http://www.ocpip.org/socket/ocpspec
IEEE Standard VHDL Language Reference Manual Std. 1076-2002	IEEE	Public URL: http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/7863/21658/01003477.pdf

1.1 Definitions

Application – IP is either application of infrastructure. Application IP blocks are called workers in this document.

"(interface) Attribute" – The WIP attributes of an interface determine the OCP configuration parameters and thus the signals in the interface.

"(worker) Configuration Properties" – Named storage locations of a worker that may be read or written. Their values indicate or control aspects of the worker's operation. Reading and writing these property values may or may not have side effects on the operation of the worker. Configuration properties with side effects can be used for custom worker control. Each worker may have its own, possibly unique, set of configuration properties. They may include hardware resource such registers, memory, and state.

WIP – Worker Interface Profiles – The interface profiles defined in this document.

"(application) Container" – A collection of "infrastructure" IP configured to "contain" a set of application workers, such that the two combined represent a complete FPGA design. I.e. a "donut" of IP with chip pins on the outside, and the application workers residing in the "hole". Interfaces that cross the inside container bounds (between the hole and the donut) are between application and infrastructure. Interfaces that stay within the hole are between application workers. The outside of the container represents off-chip interfaces.

"(worker) Control Operations" – A fixed set of control operations that every worker has. The control aspect is a common control model that allows all workers to be managed without having to customize the management infrastructure software for each

piece of IP, while the aforementioned configuration properties are used to specialize components.

"Implementation Attribute" – An attribute related to a particular implementation (design) of a worker, for one of its interfaces. I.e. one that is not necessarily common across a set of implementations of the same high level interface definition.

Infrastructure – IP is either application or infrastructure. IP that is not application is infrastructure.

"OCP Configuration Parameter" – An OCP-defined parameter used to configure the signals of an OCP interface.

"Opcode" – Synonymous with "message type" in the context of this document. Opcodes allow multiple message types to be sent across worker data interfaces.

"Profile" – A name for a subset of the universe of possible values of OCP configuration parameters with a specific and well-defined set of choices, intended to facilitate and regularize common usage. See OCP chapter 11 "OCP Profiles".

"Protocol Attribute" – A higher-level attribute more related to "what" is communicated than "how". Protocol attributes for communication between connected data interfaces that are independent of implementation.

"Worker" – A concrete implementation (and possibly runtime instance) of a component, generally existing within a container. "Application IP blocks" (components) are termed "workers" in this document.

"Wxx" – A shorthand abbreviation for any of the four interface profiles defined herein (WCI, WSI, WMI, WMemI).

2 Introduction

2.1 Purpose

The purpose of this document is to introduce and describe a collection of Worker Interface Profiles (WIP or WIPs)[1] and how to use them to produce WIP-compliant IP. It specifies metadata and interfaces used by tools and RTL IP developers, using underlying OCP and XML standards. These interfaces are relevant to both application and infrastructure/platform developers, but this document is focused on application developers, with many infrastructure requirements inferred. I.e., by stating what application IP (worker) developers may do, it implies what infrastructure developers must "cope" with, and support.

[1] Both "WIP" and "WIPs" are acceptable acronyms. As the mnemonic is shorthand for "Worker Interface Profile**s**", they both refer to a *collection* of profiles. This is the WIP functional spec. It describes and defines the four WIPs.

Standardized interfaces are understood to separate or decouple the concerns on each side of the interface (e.g. application and infrastructure). This decoupling increases opportunity for reuse, simplifies overall system design, and helps enable team design flows. By hiding details that are not important to the other side of the interface, the groundwork is set for deploying solutions where application and infrastructure components may be changed without impacting the other.

2.2 Requirements

- Well-Defined Adaptability
- Well-Defined Interoperability
- Independent from Specific Implementations
- HDL Language-Agnostic, HDL Language-Neutral
- Semantics hold under Hierarchy
- Must support compliance with SCA/CP289

2.3 Goals

This document defines, for WIP users, *which* choices can be made by the IP (core, worker) designer about Control, Data, and Memory interfaces, *how* those choices are described, and the implications of those choices on OCP, protocol, timing, and signal details.

Goal 1: **Well-Defined Interoperability:** The WIPs are defined to achieve interoperability when connecting the implementations of two designers who make independent choices within the categories of Control, Data, and Memory interfaces. This interoperability is achieved in several ways, in order of desirability from most-desirable to least-desirable:

- direct (glue-less) signal connection
- using tie-off/ignore rules when the signals do not exist on the other side
- using well-defined adapter glue (gasket logic added between the IP cores)
- using advice to inform one side of the choices of the other (implies generated/parameterized IP)

Well defined in this case means, given the interface choices made by both sides, there is a well defined way of connecting them such that they interoperate. If there are no choices (i.e. there is exactly one interface that must be used in all situations) then this is easy. If choices are provided for the convenience or performance of the implementations, interoperability is more challenging. This tradeoff is a key force in the design of the WIPs: make it easy to design workers, make it possible for them to interoperate.

Goal 2: **Bias Towards the Application:** The interfaces described here are intended to be used either between different application IP blocks, or between application IP blocks and the infrastructure blocks that manage the platform. It is a specific goal of the WIPs to bias the freedom of choice and ease of design towards the application-side and have the infrastructure-side of the platform *cope and adapt* to these choices. Thus the application developer's job is easier, and the platform developer's job is harder (and

thus the platform is more valuable). This asymmetry recognizes that there are more applications than platforms.

Goal 3: **Clearly Defined Choices:** The expression of the choices will be crisp and clear, such that design artifacts (such as headers and entity declarations) may be programmatically generated without ambiguity. Each interface profile (WIP) will have *attributes* that precisely and unambiguously determine OCP configuration parameters and OCP signals used as a consequence of the chosen attribute values.

Goal 4: **Completeness for Application IP:** The profiles described here will capture *most* requirements for the external interfaces for application IP blocks. Excluded are interfaces that contain predetermined or constrained functionality, such as a hardcore, pin, or legacy IP.

2.4 Non-Goals

This document does not intend to prescribe any specific implementation of a particular interface.

This document does not address Transaction Level Modeling (TLM) and other verification concerns.

This document is not intended as a usage tutorial for either the WIPs or the underlying OCP semantics. It is recognized that separate documents for application developers are needed to reduce the learning and knowledge required to use the WIPs. In particular, using the WIPs means using a subset of OCP. Using only this document, the reader must read and understand much more of the complete OCP specification than is fundamentally required to use the WIPs.

2.5 Overview

The WIPs are OCP profiles, fully compliant with the OCP specification as well as a thin layer of additional metadata and semantic definition, so as to provide utility for their use in Control, Data, Memory, and Time interface patterns. All the WIPs taken together use a modest subset of OCP.

The diagram below shows the taxonomy of the worker (component) interfaces described in this document. Each interface is defined by choosing one of the 4 profiles, and choosing values for the attributes defined for each profile.

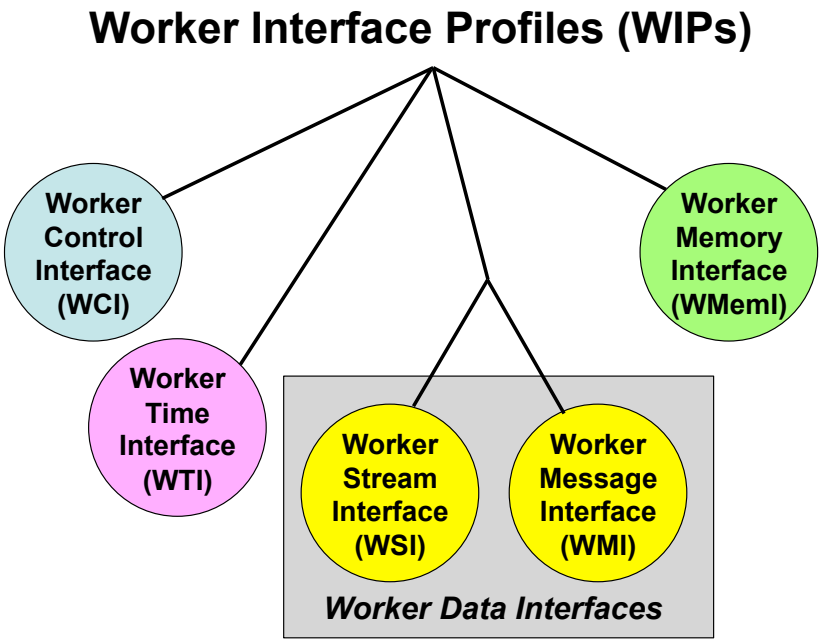


Figure 1 – Worker Interface Profile Taxonomy

The term "worker" and "component" are used interchangeably in this document; however worker is generally more concrete and closer to an implementation and component is more abstract and closer to a functionally specified entity. The WIP specification presents a worker- centric bias making it easy for the designer to express the choices from the WIP patterns (i.e. control, data, and memory). Worker authors (component implementers) should find that the communication patterns presented here cover substantially all of their core connectivity needs (when they are not already externally specified). They should use this document as the specification for that connectivity.

When the other side of an interface is not connected to a co-located worker (another part of the same application residing in the same FPGA), it is connected to "infrastructure IP". Infrastructure IP includes logic that adapts the outside world to the WIP interfaces and visa versa. Infrastructure IP may also include IP that adapts WIP profiles to each other, when certain profile choices are made on each side that is not directly connectable. Infrastructure IP creators must be aware that they take on an additional burden and value, as they must in general accommodate all possible worker implementation choices *per profile* for a particular interface, not just one worker's implementation choice.

The application (worker) and infrastructure are two sides of the WIP solution; both are aspects of IP development. Both application and infrastructure providers using these WIP profiles of OCP-IP will use this document as the normative spec.

An illustrative example:

The following diagram shows two (application) workers inside an FPGA. To be a complete FPGA design this application must be surrounded by a container that connects to the external hardware (pins and external devices). In this particular example

of deploying two of an application's workers on a single FPGA, there are eight OCP links, identified by a numbered circle, each with a master and slave.

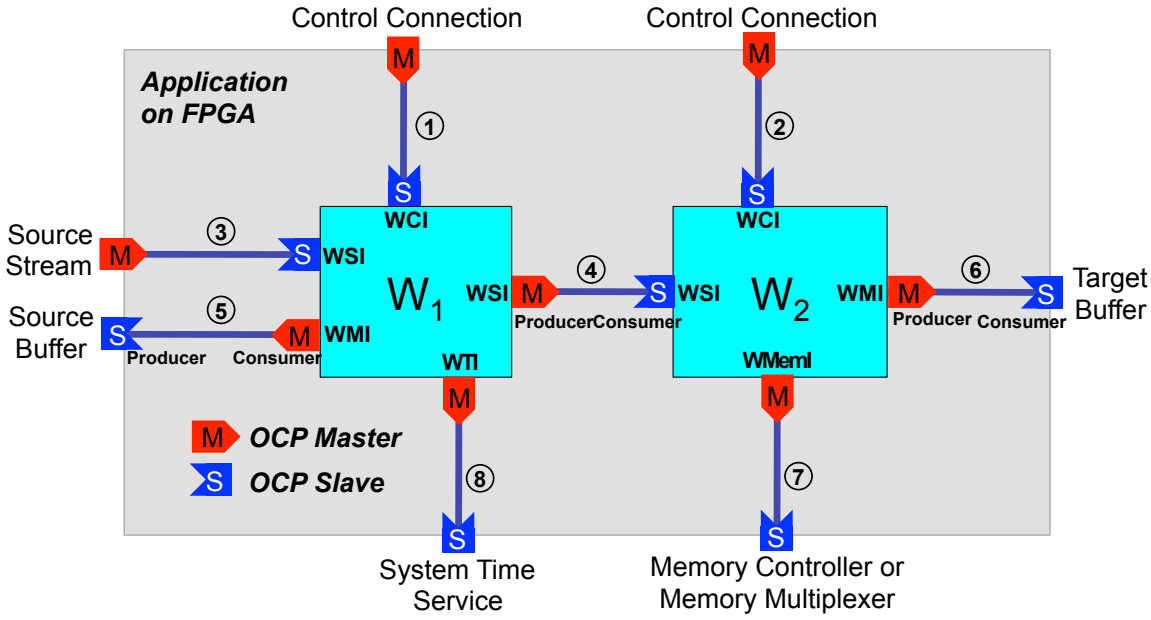


Figure 2 – Two Workers in an FPGA

Links 1 and 2 show examples of the Worker Control Interface (WCI). The WCI provides *control and configuration* for each worker. Each worker in an application will have exactly one WCI slave interface.

WCIs are used to implement the "control plane". Workers (components of the application executing inside an FPGA) are controlled and configured by external entities, usually software.

Links 3 and 4 show examples of the Worker Stream Interface (WSI). The WSI provides unidirectional FIFO streaming with flow control. Workers may have 0 or more WSIs. Consumer WSIs are always slaves. WSI producers are always masters. Link 3 talks to a worker in another chip somewhere else, with help from the container. Link 4 is local.

Links 5 and 6 show examples of the Worker Message Interface (WMI). The WMI provides the ability for workers to actively produce or consume data from buffers with the capability for random addressing and buffer reuse. Workers may have 0 or more WMIs. A worker's WMI interfaces are always masters, regardless if they are producing or consuming. Both links in this case talk to workers in other chips, with help from the container.

WSIs and WMIs are used to implement the "data plane". This is the way workers communicate with each other.

Link 7 shows an example of the Worker Memory Interface (WMemI). The WMemI provides the ability for a worker to have a direct or multiplexed link to local memory such as bulk-attached SRAM or DRAM, or embedded BRAM or MRAM. The worker's WMemI is the master and the infrastructure's WMemI is the slave.

Link 8 shows an example of the Worker Time Interface (WTI). The WTI provides the ability for a worker to access the current time (time of day, UTC etc.). The worker's WTI is the master and infrastructure's WTI is the slave.

2.5.1 Worker vs. Infrastructure and OCP Master/Slave Roles

The table below lists each of the four WIPs and their OCP roles.

Table 2 - Interface OCP Master/Slave Roles

Interface	Worker OCP Role	Infrastructure OCP Role
Worker Control Interface (WCI)	Slave	Master
Worker Streaming Interface (WSI)	Stream Producer is Master Stream Consumer is Slave	
Worker Message Interface (WMI)	Master	Slave
Worker Memory Interface (WMemI)	Master	Slave
Worker Time Interface (WTI)	Master	Slave

Recall that the OCP Master side of an interface is the side that is generating OCP requests. The figure below shows two workers ($\mathbf{W_1}$ and $\mathbf{W_2}$) communicating messages via WMI, which requires in interposed infrastructure module ($\mathbf{I_A}$) that acts as WMI slave on both sides.

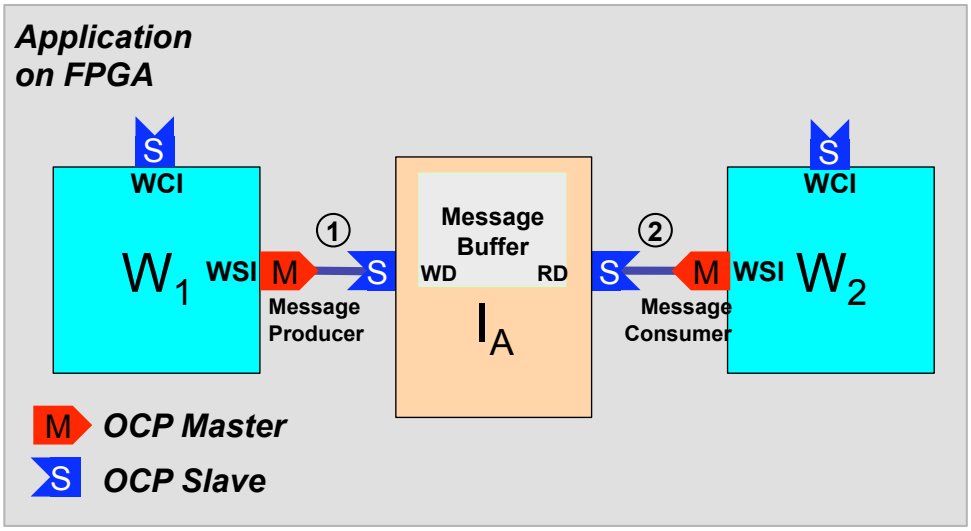


Figure 3 - Two WMI Application Workers Communicating via a buffer

2.6 Document Organization

For each of the four interfaces described, there are sub-sections as follows to describe the interface profile. **X** is the particular section describing a profile. **Www** is the name of the profile.

X.1 Www Motivation – This section explains the purpose of the profile.

X.2 Www Overview – This section provides a high level description of the profile and how it may be used.

X.3 Www Attributes – The available choices, from a worker author's perspective, to use this particular profile. These *WIP attributes* determine the OCP configuration *parameters* and the OCP signals. These attribute choices are a simpler and higher level set of choices than the lower level details of the multiplicity of the OCP configuration parameters. Thus there are tools which automate the generation of OCP configuration parameters and signals from these simpler WIP attributes, according to this specification. Developers simply specify the attributes, use the tools, and implement based on the generated IP definitions and skeletons generated by the tools.

All attributes have a default value, although some attributes are not used if other attributes are set to specific values.

The types used in the worker profile attribute tables are specific XML Schema data types such as boolean or unsignedInt. The XML namespace prefix (usually xsd:) is omitted from the types for readability.

Observation: In some frameworks, there may be a high-level component description of the overall component and its interfaces (frequently called "ports"). This may be an Interface Definition Language (IDL) or electronic datasheet such as SPIRIT/IP-XACT. In such a case, some of the values for these attributes may be derived or inferred from such a high level representation.

Every interface has a Clock attribute to indicate if the interface requires an OCP clock that is different from the OCP clock applied to the WCI. When it is different from the WCI clock, it must be specified as one of the clocks in the Clocks global attribute for the worker. Every interface has a ClockSignal attribute, whose default is the standard OCP name of the clock signal for the interface (with appropriate prefix).

Attribute Name	Type (per XML Schema)	Default Value	Description
Clock	*string*	*Same as WCI*	*The name of the clock to be used at this interface*
ClockSignal	*String*	*Empty*	*The name of the clock signal to be driven at this interface when it is not the OCP standard name (Clk) with appropriate prefix.*

X.4 Www OCP Profile/Configuration Parameters - The worker author does not specify the OCP configuration parameters directly. They are either constant over the profile; or they are strictly derived from the simpler, condensed WIP attributes. This section provides details as to how the OCP configuration parameters are strictly derived from the choice of WIP profile attribute values. This mapping from WIP attributes to OCP configuration parameters is automated by transformation tools that accept the WIP

attributes as inputs and produce OCP configuration files and/or VHDL entity/component definitions as output.

The OCP Configuration Parameter Table lists parenthetically in the description column the attribute(s) that determine the value of each OCP configuration parameter. This summarizes the map from attributes to OCP configuration parameters.

This section and table can be skipped when readers simply want to know how OCP signals are derived from WIP attributes, since the table under the next section (Www OCP Signals), describes this mapping (attributes to signals) directly, without reference to the OCP configuration parameters.

The values in the table are the explicit values and default-overrides to the OCP configuration parameter defaults, based on attribute values (or constant for the profile).

X.5 Www OCP Signals - The worker developer does not specify the OCP signals used. The profile attributes determine the OCP configuration parameters, which strictly determine the OCP signals, per the OCP specification.

Because the OCP interface semantics are defined by the OCP specification, there is no need for this document to describe any OCP signal timings or protocols. As a result of the worker interface attribute selections, there can be additional semantic meaning, layered above the OCP behavior,for certain signals as described by the next section (Www Semantics).

Clock signals are not listed in the master-to-slave or slave-to-master tables. Refer to section 2.7 on OCP Interface Clocking.

The OCP Signal table for the profile shows which signals may be included, when they are included, and what their width is. All of these are determined by the WIP attribute values (and the OCP configuration parameters that are determined by the attribute values).

X.6 Www Semantics – This subsection describes any added meaning to the signals or metadata that are not inferred from the OCP configuration parameters and thus the OCP specification.

X.7 Www Infrastructure Coping – Interface specific guidance on how the infrastructure needs to cope with the application worker chosen attributes. (E.g. generics, generators, and requirements).

Use cases and examples not part of the normative specification are located outside the profile definition sections.

2.7 OCP Interface Clocking

This section describes the clocking of the OCP interfaces of workers

All OCP interaction is between a single master and slave. The clock for each OCP interface is in general driven neither by master or slave[2]. It is driven by the

[2] "The Clk and EnableClk signals are special in that they are supplied by a third (external) entity that is neither of the two modules connected through the OCP Interface". (OCP specification Pp33)

infrastructure. For this reason, the clock signal has been omitted from each profile's Wxx OCP Signal section, even though there is a clear association between clocks and interfaces. Thus WIP-compliant workers do not generate or emit clocks.

Every interface on a worker is assigned to a particular ***Named Clock Domain*** *(sometimes abbreviated to "clock")*. This capability enables the worker author to express which interfaces share, or do not share, synchronization. The default, if no such clock domain is mentioned for an interface, is that the interface uses the same clock as the WCI for the worker (the control interface). The worker metadata is used to specify the assignment of interfaces to clocks.

Worker implementation logic must be consistent with the stated choices. If an interface is assigned to a particular clock domain, it must *respect* that clock domain.

> Example 1: If a worker consumes a WSI stream on one interface, delays it through a register, and outputs it on another WSI interface; these two interfaces must be placed in the same named clock domain. If the worker author designed the logic so that the output interface was designed for asynchronous operation, then the two interfaces *could* be in the same named domain, or in different domains.
>
> Example 2: If a worker implements a fixed N:1 decimator (or 1:M interpolator) then the clocks may be in different named domains with additional specific attributes, which may describe their (rational) relationships.

Rather than have each interface on a worker carry a clock name attribute, all the clocks associated with a particular worker are collected together in the section on ***Worker Global Attributes***. Then each interface can reference the appropriate clock in its own "Clock" attribute.

2.8 Reset Propagation and Behavior

This section describes the reset behavior and propagation rules for a worker.

Every application worker must have exactly one WCI (Worker Control Interface). That WCI interface is connected to the "control system" infrastructure (typically controlled by software), not another worker. The control system controls the operational state of the worker and configures it. The configuration and control behavior is discussed below in the WCI section.

There are special semantics for the (OCP) reset on the WCI:

> When WCI reset is asserted (MReset_n='0') as *input* to the WCI (slave) interface of a worker, the worker shall propagate the reset as an *output* to all other interfaces on that worker (as OCP MReset_n or SReset_n as appropriate for each interface).
>
> The WCI MReset_n input signal is in the clock domain of the WCI (per OCP). It is the responsibility of the worker to properly propagate the reset indication to the

other interfaces in the appropriate clock domains at each interfaces (data, memory, time etc.).

For all data interfaces on a worker:

Data interfaces have both reset inputs and reset outputs. All worker implementations must insure that reset inputs (or outputs propagated from WCI) at their data interfaces force all OCP output signals at that interface to a proper reset state per OCP.

The simplest (and least helpful for application debugging) worker implementations will have the default "false" value of the "ResetWhileSuspended" attribute of its WCI. This means that the worker is not prepared to retain any particular functionality when it sees reset inputs at its data interfaces (other than obeying OCP rules for output signals at the interface under reset). The control system will know this attribute setting, and consider the worker non-functional if any adjacent (data-connected) workers are reset (via WCI). The worker will be fully reset via WCI reset before any other actions are taken.

The "ResetWhileSuspended" attribute of a worker (when true) indicates this more advanced behavior: the worker will support the Stop control operation and, *while suspended*, accept data interface resets. Such resets (from adjacent workers) will not render the worker non-functional and thus allow for configuration property operations while in this suspended state even though some data interfaces are in a reset state. After the asserted data interface resets are deasserted, such workers may be resumed via the "start" control operation.

Such a suspended worker receiving a reset on a data interface should interpret the meaning as a "*message-level abort*", and bring the associated interface to an idle, neutral state, while attempting to minimize the loss of state in the rest of the worker. This may be performed by "disposing of" or "aborting" a message in-flight across a particular interface at the time of reset and preparing that particular interface to properly operate anew following the deassertion of the reset input.

The intent of this per-data-interface reset is to allow some workers to be entirely reset (e.g. if they are locked or behaving badly), while still allowing access to the state of adjacent (and not as badly behaved) workers. In some cases this is simply for debugging, but there are cases where the control system for a certain application may be able to recover from unresponsive workers by suspending adjacent ones (via **stop**), and resetting the "bad" one. The extent to which a worker implementation allows for suspended states and data-interface resets is implementation defined. I.e. some worker implementations may not recover (be resumable) after having data interfaces reset while in the suspended state.

Example: A 3:1 message merge worker receiving a reset on one of its message input interfaces should, if buffering, avoid appending the partially received message to the output, and should understand that any remaining input on that interface will not arrive. It should prepare for new, valid, input following the reset event.

OCP Reset Policy Applies. See the OCP section on the sideband signal reset; but specifically:

- Reset asserts asynchronously, but de-asserts synchronously.
- Reset shall be asserted for a minimum duration of 16 cycles of the clock domain to which the reset signal is synchronous.
- Workers must be able to reset in this 16-cycle duration.

Observation: Although system reset will assert asynchronously and de-assert synchronously; this does not preclude an implementation from using a fully synchronous reset scheme, if so desired. Such an implementation would need to be aware that reset-assertion could occur without a relationship to any clock edge and may need to synchronized appropriately. WIP-compliant workers must accomplish reset in 16 WCI clocks.

2.9 Worker Global Attributes

Worker global attributes contain information that is common across the worker, and not specific to an interface.

2.9.1 Worker Global Attribute Summary

Table 3 – Worker Global Attribute Table

Attribute Name	Type (per XML Schema)	Description
Name	*WIP:Name*	*The name of the worker[TBD: extra constraints]*
Clocks	*WIP:Clock*	*Declaration of a clock for the worker*
Endian	*WIP:Endian*	*WCI endian and global endian default*

2.9.2 Worker Global Attribute Details

Name	*WIP:Ident*	*The name of the worker*

Name – The name of the worker.

In VHDL, the worker Name attribute must match (ignoring case) the entity name for top-level entity (design unit) for the RTL worker component being described. In Verilog, it should match the module name.

Clocks	*WIP:Clock*	*Declaration of a clock for the worker*

Clocks – A set of all the implementation's *Named Clock Domains*. Each interface may reference these clocks to indicate that the interface operates in the domain of that clock. Interfaces with no such association are assumed to operate with the same clock as the WCI interface. Interfaces may also declare that they "own" a clock, which simply means that the clock signal will be specifically be associated with the signals of that interface (and use the standard OCP naming for that signal). Interfaces may also simply reference other interfaces explicitly, meaning that they use the same clock as the referenced interfaces. If a clock is not "owned" by an interface, it requires its own explicitly specified name. If no clocks are mentioned at all, a single clock, "owned" by

the WCI, will be defined and used for all interfaces. See the XML metadata section for syntactic details.

Constraints (min/max rates etc.) per clock are TBD in this version of this specification.

Endian	*WIP:Endian*	*WCI endian and global endian default*

Endian – A string indicating the endianness of the worker's WCI interface behavior and the default endian behavior for all data interfaces.

The default value (when this attribute is unspecified) is the simple case when workers:

- have no configuration property values of sizes other than the data path of the WCI (32 bits)
- accept or produce messages on any data interface that are only a (byte) multiple of the width of that interface
- process data values not smaller than the width of the data interface.

If any of these are *not* true, the implementer must specify a non-default value, as defined here:

Attribute Value	**Description**
neutral	No inherent endianness — the default as described above
big	Big-endian
little	Little-endian
static	The implementation supports both endian via a static configuration input.
dynamic	Runtime supports both, accepted dynamically (on release of WCI reset) from WCI MFlag[1]

WCI MFlag[1]	**Description**
0	Little-Endian
1	Big-Endian

2.10 Signal Prefix Rules and Guidelines

This section describes the rules and guidelines used to generate unique signal names for an RTL component based on interface name and interface profile selected.

2.10.1 Signal Prefix Rule

Signals shall be named by forming a signal name that is the concatenation of a unique interface identifier and the OCP signal name separated by the underscore ('_') character.

For example:

```
MyUniqueInterfaceName_MCmd
```

2.10.2 Signal Prefix Guideline

A default practice of the above rule involves the concatenation of the following sub-strings, separated by the underscore ('_') character:

1. The string "ocp" to indicate behavior per the OCP spec.

2. The abbreviation for the interface/profile type {WCI|WSI|WMI|WMemI}
3. The interface name
4. The OCP signal name

For example:

```
ocp_wsi_chan1_MCmd
```

2.10.3 Signal Prefix Example

The VHDL code snippet below shows an example use of the signal prefix naming guideline. This is part of a worker entity declaration. The annotated sections indicate:

1. There is no special “types” or “definitions” package required for the WIPs.
2. These signals are not explicitly part of any interface. They may or may not be used with a particular WIP.
3. Infrastructure with a WSI named “AD0”.
4. Infrastructure with a WSI named “AD1”.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

library WORK;
use WORK.vita49_types.all;

entity app is
  port (
    -- Clock and reset signals...
    dataClkDiv2      : in std_logic;    -- 137.5 MHz
    hsdl_clk         : in std_logic;    -- 62.5 MHz
    local_clk        : in std_logic;    -- 33 MHz
    dataClkDiv2_rstn : in std_logic;    -- Reset synchronized to dataClkDiv2
    hsdl_clk_rstn    : in std_logic;    -- Reset synchronized to hsdl_clk
    local_clk_rstn   : in std_logic;    -- Reset synchronized to local_clk

    -- A/D Data Interface
    ocp_wsi_ad0_MBurstLength  : in  std_logic_vector(1 downto 0);
    ocp_wsi_ad0_MBurstPrecise : in  std_logic;
    ocp_wsi_ad0_MCmd          : in  std_logic_vector(2 downto 0);
    ocp_wsi_ad0_MData         : in  std_logic_vector(63 downto 0);
    ocp_wsi_ad0_MDataLast     : in  std_logic;
    ocp_wsi_ad0_MDataValid    : in  std_logic;
    ocp_wsi_ad0_MReset_n      : in  std_logic;
    ocp_wsi_ad0_SThreadBusy   : out std_logic;
    ocp_wsi_ad0_SReset_n      : out std_logic;

    ocp_wsi_ad1_MBurstLength  : in  std_logic_vector(1 downto 0);
    ocp_wsi_ad1_MBurstPrecise : in  std_logic;
    ocp_wsi_ad1_MCmd          : in  std_logic_vector(2 downto 0);
    ocp_wsi_ad1_MData         : in  std_logic_vector(63 downto 0);
    ocp_wsi_ad1_MDataLast     : in  std_logic;
    ocp_wsi_ad1_MDataValid    : in  std_logic;
    ocp_wsi_ad1_MReset_n      : in  std_logic;
    ocp_wsi_ad1_SThreadBusy   : out std_logic;
    ocp_wsi_ad1_SReset_n      : out std_logic;
```

Note that entity declarations can be automatically generated using WIP-compliant tools, based on the XML format of the worker’s attributes described in the WIP XML Schema section below.

3 Worker Control Interface (WCI)

3.1 WCI Motivation

The Worker Control Interface's primary goal is to make control plane support simple for worker developers by ***insulating them from platform-specific interconnection networks***, busses, bridges and endpoints that may exist between a generic control software and the worker. It is the interface used to control and configure workers by the "control system", which is usually software-based, with intervening infrastructure IP and hardware between the control software and worker IP.

The diagram below illustrates the separation of concerns between Application Software Developer, Platform Supplier, and the worker control interface (WCI) described in this section.

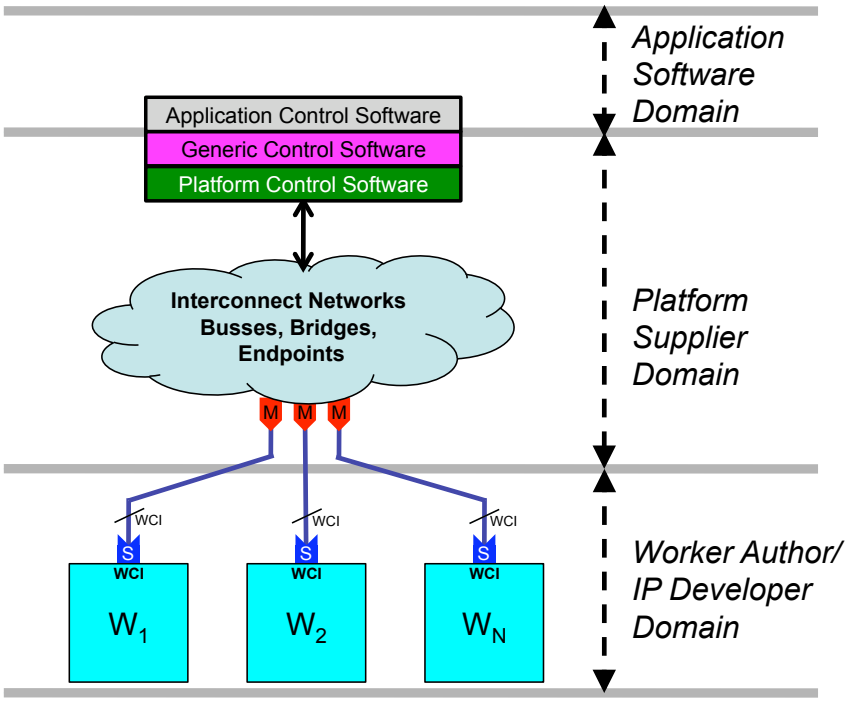


Figure 4 – Worker Control Interface – Separation of Concerns

3.2 WCI Overview

The WCI is how workers are controlled and configured, and exactly one WCI exists in every worker. Worker authors make certain choices for their WCI, such as the types and extent of run-time configurable properties that are to be read or written. The choices, as in all WIP interfaces, are specified in simple XML-based "metadata". This metadata then can be used to generate (for any implementation language) the interface definition used for a particular WCI-based interface.

A set of workers, each with a WCI, may at some point be collected together to be part of an application, running together, collocated, within a single FPGA. The infrastructure provider can generate Control Plane IP (CPIP) for a set of workers, each having a WCI slave. CPIP refers to an implementation of the infrastructure-side logic used to control

the workers. The CPIP is used by control software to access the WCI of a group of workers in the FPGA.

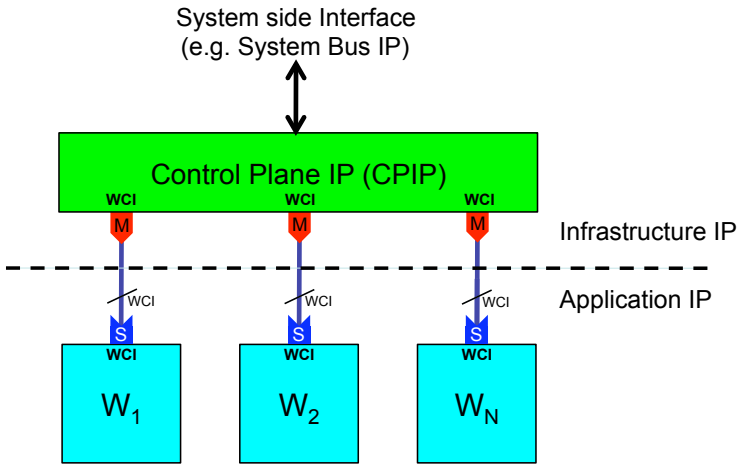


Figure 5 - Relation of CPIP to Workers

3.2.1 Control and Configuration

WCI transactions are either “control” or “configuration”.

Control refers to a fixed set of “control operations” that every worker component, explicitly or implicitly, must support. There are seven specific control operations. For some workers, the action associated with a particular WCI Control Operation may be no action at all. Note that while (application) workers must all be uniformly controllable according to this control scheme, infrastructure IP may be configurable using a subset of WCI when it is not appropriate for them to be controllable as application IP.

This control scheme is based on a number of other similar systems, and is intended to be both generally useful and also support enough controllability to be compliant with other component standards. The general model is that workers transition through various states based on the control operations. It is the responsibility of control software to keep track of the states and issue the control operations in the correct sequence. OCP transactions on the WCI interface perform (request) the control operations. Here is a summary of the control operations, which are detailed below.

632 **Table 4 – WCI Fixed Control Operations – Function Overview**

WCI Control Operation	Function Overview
Initialize	Perform any post-reset initializations, before configuration.
Start	Enter the operational state; start doing the operational work.
Stop	Suspend work such that operation can be resumed.
Release	Shut down operations, undoing "initialize".
Test	Run component-level built-in-test, if available.
BeforeQuery	Ensure configuration properties are consistent prior to accessing.
AfterConfig	Act on multiple interdependent configuration property settings.

633 **Configuration** (**Properties)** refers to the set of and access to writeable and readable
634 properties, with and without side effects, within the worker. This is the typical "register
635 file" configuration pattern in many IP designs. Components may also implement their
636 own worker-specific specialized control schemes by read and/or write side effects within
637 the configuration property space. WCI provides the interface to access a worker's
638 configuration properties at run time.

639 The worker is responsible for the mechanism, which might be an array of registers with
640 write-enable logic and read-back multiplexers. If required or desired by a given worker
641 implementation, properties may be smaller than 32 bits (i.e. 8 or 16), and "packed" with
642 multiple 8 or 16-bit properties in the same 32 bit word. All properties must appear in the
643 configuration "space" on their natural boundaries (see the "force_aligned" semantic in
644 the OCP specification). If a property value is larger than 32 bits, then when the control
645 software changes it, the first 32-bit word of the property value is written *last*, so that the
646 worker knows when the entire value has been updated via the WCI.

647 ### 3.2.2 WCI Control Operations

648 In order to keep workers as simple as possible in common cases:

649 • It is the responsibility of software to sequence these operations properly, so workers
650 do not need to check for invalid control operations (issued in the wrong state or in
651 the wrong order)

652 • Metadata known to control software can indicate whether operations are
653 implemented at all (except start, which *must be implemented*). See the
654 ControlOperations attribute below.

655 The operations are:

656 • **(**Implicit**) Reset** via the WCI's Mreset_n signal

657 • *After loading or unsuccessful release, a hard reset to known state*

658 • *Infrastructure/control system automatically asserts reset upon loading*

659 • *Worker must be ready to accept **Initialize** operation after reset is deasserted*

660 • *Worker can count on reset being asserted for 16 cycles, per OCP*

- *If there is any possibility that 16 cycles are not enough, use **initialize** to perform initializations that take more time.*
- *WCI Reset has special meaning for other interfaces. It is described in section 2.9 on Reset Propagation and Behavior.*
- *Workers must be reusable via a reset assertion multiple times after a bitstream load.*

- Initialize
  - *Needed when additional initialization is required that is not done during reset.*
  - *Allows worker to perform initial work not under reset, to achieve a known ready-to-run state*
  - *Appropriate place for worker to perform runtime initialization (e.g. programmatic initialization of a LUT).*
  - *Worker can internally set initial/default values of properties (if not done via **Reset**)*
  - *Worker can't do one-time work based on software-configured properties. They are not valid yet since control software has not set them yet. Control software will not perform property access until **Initialize** completes.*
- Start:
  - *Transition to operational - to doing the real work based on property values and data (at data interfaces).*
  - *Must do any one-time initializations that depend on software-configured property values, since **Initialize** cannot do this (properties are not set yet).*
  - *Starting operation is based on property values set after **Initialize** completes*
  - *No OCP commands on **data** interfaces are allowed before this operation. Thus slaves must assert OCP SThreadBusy upon reset and not deassert it until **start**. Masters must not issue commands until **start**.*
- Stop
  - *Make the worker inactive/paused/suspended, such that it can be resumed (via **Start**); maintain resources/state/properties for examination.*
  - *When this operation is complete, data interfaces are "between messages". This is a "graceful" suspension, not an immediate destructive stop or abort.*
  - *Worker is suspended, data interfaces can be reset by connected workers (if implementation is capable of it)*
- Release
  - *Can be no-op if all un-initialization must be done under **Reset**: it returns failure in this case.*
  - *Shut down/abort operations, discard any state (state is now undefined)*
  - *If release fails, worker is unusable until reset.*
  - *May be called after any operation, but not immediately after deassertion of reset.*

- BeforeQuery
  - *After **Initialize**, **Start**, or **Stop**, informs worker that a group of related property queries (configuration reads) will be made and their values should be consistent.*
  - *Used to enable coherent access to multiple property values.*
  - *Only necessary when property values depend on each other and need to be properly updated to be meaningful/correct.*
- AfterConfig
  - *After **Initialize**, **Start**, or **Stop,** informs worker that a group of related property changes have been made.*
  - *Informs worker that changes to a set of interrelated property values can be processed.*
- Test
  - *After **Initialize**, used to run component-specific built-in tests, parameterized by current configuration property settings.*
  - *In a given system or application, the overall system built-in-test can be easily parameterized for each worker by specifying the configuration property settings prior to running "test", and the expected configuration property settings after running "test".*
  - *Simple workers that have no tests can indicate it in metadata.*

3.2.2.1 Simple workers can be very simple for control operations

For the simplest workers, only the **Start** operation is mandatory. The implications are:

- **Initialize** is not needed – all initialization is performed during reset, and after reset is deasserted, the worker can accept configuration accesses.
- **Release** is not needed, and the implementation forces control software to reset the worker (which will propagate the reset to adjacent workers). This reduces system debugging and reloading flexibility, since this “software reset” is not possible.
- **Stop** is not implemented, and thus the worker cannot be paused/suspended at all. This forces whole applications to not be stoppable and thus degrades application and system-level debugging.
- **Test** is not implemented: there is no built-in test capability.
- **BeforeQuery** and **AfterConfig** operations are not needed if there are no interdependent property values that must be used atomically and/or consistently.

Thus simple, but useful, workers are only required to implement the **Start** operation; all others can be specified as unimplemented in metadata.

3.2.3 WCI States

Since control software is required to issue control operations correctly, in the appropriate sequence as defined above, these states are not normative, but are use to further clarify the use of control operations.

- State**: Exists**: after deassertion of reset or after successful **release** operation, ready for **initialize**.
  - *The worker is loaded (if necessary), not necessarily fully initialized, configuration properties are not valid, property access is not valid, but reset is deasserted.*
  - *Only the **initialize** operation will be issued. If successful the worker is considered to be in the "initialized" state.*
- State: **Initialized:** after successful **initialize** operation**,** ready to **start** (or **release**)
  - *The worker has been completed any run-time initialization, doing whatever is necessary or appropriate after reset was deasserted.*
  - *The worker is in a known, stable, initial state "ready to start doing work", but not operational.*
  - *Configuration properties can be read and written (SW software can see the result of initialization, and can configure the worker prior to normal operation). Thus the **BeforeQuery** or **AfterConfig** operations maintain this state.*
- State: **Operating:** after successful **Start** operation**,** ready to **Stop** (or **Release**)
  - *The worker is doing normal work based on properties and data interfaces; it is "processing".*
  - *Configuration properties can be read or written, including **BeforeQuery** or **AfterConfig** operations*
- State: **Suspended:** after successful **Stop** operation, ready to **Start** (or **Release**)
  - *The worker is suspended; data interfaces can be reset by connected workers (if the implementation is capable of it).*
  - *Configuration properties can be read or written, including **BeforeQuery** or **AfterConfig** operations*
- State: **Unusable**: after unsuccessful **Release** operation
  - *In this state the only action to take is to assert WCI reset.*

3.2.4 WCI Errors

WCI errors are indicated via the OCP SResp signal taking the ERR value for control operations. Metadata indicates whether errors are transient - whether retrying operations (other than **release**) makes any sense for an implementation.

- All **release** errors indicate unusable worker circuit state requiring reset before re-use
- If Initialize, Start, Stop, BeforeQuery or AfterConfig return an error
  - *If metadata (the **Retry** attribute) says retries are possible, try again.*
  - *If not, (or after unsuccessful retries) then **Release** may be attempted.*
- After a successful **Start** operation, worker may assert the attention signal (see semantics for Sflag[0] below) to indicate that an error has occurred or attention is required
  - *What this means is worker dependent, but generic software knows to inform users or invoke attention handlers for the worker.*

- *Should be cleared by worker during **Release** operation (and reset).*

3.3 WCI Attributes

Worker authors specify their WCI implementation choices via the following attributes, from which all OCP configuration parameters and signals are derived.

Each attribute is labeled indicating whether the attribute value relates to the worker's interface (and thus applies to all implementations), or whether the attribute value is specific to a particular implementation of the same interface common to all related implementations. The metadata formats (XML Schema described below) separates attributes that are about high level interface behavior (e.g. properties) from those that are implementation-specific.

The Clock and ClockSignal attributes, described in section 2.6, are applicable to WCI.

3.3.1 WCI Worker Attribute Summary

Table 5 – WCI Worker Canonical Attribute Table

Attribute Name	**Type** (per XML Schema)	**Interface vs. Implementation**	**Default Value**	**Description** (OCP configuration parameters affected by this choice)
SizeOfConfigSpace	*unsignedInt*	*Int*	*0*	*Size (in Bytes) of worker's configurable property space*
WritableConfigProperties	*boolean*	*Int*	*false*	*TRUE: there are writable configuration properties*
ReadableConfigProperties	*boolean*	*Int*	*false*	*TRUE: there are readable configuration properties*
Sub32bitConfigProperties	*boolean*	*Int*	*false*	*TRUE: there are properties that are smaller than 4-Byte values*
ControlOperations	*string*	*Imp*	*empty*	*List of supported control operations (start is assumed and mandatory)*
ResetWhileSuspended	*boolean*	*Imp*	*false*	*TRUE: the worker will remain functional when adjacent interface resets when SUSPENDED*

3.3.2 WCI Worker Attribute Details

This subsection contains the per-attribute details of choices available for the WCI.

SizeOfConfigSpace	*unsignedInt*	*Int*	*0*	*Size (in Bytes) of worker's configurable property space*

This attribute, when non-zero, indicates the size in bytes of the configuration property space. When this attribute is zero, it indicates that there are no configurable properties on this worker.

Observation: Because the WCI physical OCP data path is 4 bytes wide, this value is typically a multiple of four. However, this is not a requirement; this attribute may be any unsignedInt less than or equal to 2^16 (0x10000), 64KB.

WritableConfigProperties	*boolean*	*Int*	*false*	*TRUE: there are writable configuration properties*

This attribute indicates the worker has writable configuration properties. If FALSE all configuration properties are read-only.

ReadableConfigProperties	*boolean*	*Int*	*false*	*TRUE: there are readable configuration properties*

This attribute indicates the worker has readable configuration properties. If FALSE, all configuration properties are only writable.

Sub32bitConfigProperties	*boolean*	*Int*	*false*	*TRUE: there are properties that are smaller than 4-Byte values*

This attribute indicates the worker contains configuration properties that are less than 32-bit (4-byte) values. When TRUE, OCP byte enable logic is provided to allow for 1-byte, and aligned 2-byte and 4-byte reads and writes. When FALSE, there is no byte enable logic and all accesses are 4-bytes.

The following attributes are metadata only and do not affect the OCP configuration

ControlOperations	*string*	*Imp*	*empty*	*List of supported control operations (start is assumed and mandatory)*
ResetWhileSuspended	*boolean*	*Imp*	*false*	*TRUE: the worker will remain functional when adjacent interface resets when SUSPENDED*

These two attributes provide information to the generic control software about the behavior of the workers in response to certain conditions. All default to FALSE to keep simple workers simple.

The ControlOperations attribute indicates (to control software) which of the optional control operations are implemented by this worker. Start is required, but all others are optional. Lack of support for **Stop** and **Release** degrades system debugging capability, and is thus not recommended.

The ResetWhileSuspended attribute indicates that the worker supports data interface resets when in the suspended state. This essentially means that the worker can be stopped/suspended, adjacent connected workers can be reset, and this worker can resume operation without itself requiring a complete WCI reset. The use-case is the ability to reset an adjacent broken worker without losing all the state and configuration of this worker. This attribute helps to define three levels of support for stoppable workers, with increasing value for system control and debugging:

- No support at all: “Stop” is not in the list in the ControlOperations attribute.
- Stoppable without supporting reset of adjacent workers: ResetWhileSuspended is FALSE
- Stoppable and able to remain configurable and resumable (via Start) when adjacent workers are reset.

3.4 WCI OCP Profile/Configuration Parameters

The WCI OCP profile is the aspect of the WCI that has to do with OCP configuration parameters and signal configuration. This section describes the concrete, complete, and unambiguous transformation from WCI WIP attributes described above, to the OCP configuration parameter inferred from those attributes. This transformation is implemented in automation tools.

The table below lists only those parameters that are different from the OCP configuration parameter defaults (per the OCP specification).

834 **Table 6 – WCI OCP Configuration Parameters**

OCP Configuration Parameter	Value	Description (How OCP parameter is determined if not constant, and any tieoffs)
addr_wdth	*[5..N]*	*If SizeofConfigSpace <= 32, N = 5 Else N = ceil(log2(SizeofConfigSpace - 1))*
addr_space addrspace_wdth	*{0|1}*	*If SizeofConfigSpace > 0: 1 Else: 0*
byteen	*{0|1}*	*If Sub32BitConfigProperties = TRUE: 1 Else: 0*
cmdaccept	*0*	
data_wdth	*{0|32}*	*If SizeofConfigSpace > 0 : 32 Else: 0*
force_aligned	*{0|1}*	*If Sub32BitConfigProperties = TRUE: 1 Else: 0*
mdata	*{0|1}*	*If WritableConfigProperties = TRUE: 1 Else: 0*
mflag	*1*	
mflag_wdth	*2*	
mreset	*1*	
sdata	*{0|1}*	*If ReadableConfigProperties = TRUE: 1 Else: 0*
sflag sflag_wdth	*1*	
sthreadbusy sthreadbusy_exact sthreadbusy_pipelined	*1*	*Note: For pipelined, proactive flow control (backpressure) by way of the SThreadBusy signal*
write_enable	*{0|1}*	*If WritableConfigProperties = TRUE: 1 Else: 0*
writeresp_enable		

835 MAddr[1:0] will always be zero (per OCP) and should be ignored. Sub-32bit write
836 indication, when configured, is indicated only by the byte enables.

837 Observation: WCI is an exception among the four profiles listed herein in that it is the
838 only one to define the OCP configuration parameter writeresp_enable to '1' (when there
839 are writable properties). This is because WCI requires that configuration writes provide
840 the capability for success versus failure indication. The use of the writeresp_enable
841 OCP behavior provides this capability with the minimal implementation. All WCI writes
842 require a SResp response, either accept/success (DVA) or error/failure (ERR).

843 • Accept (DVA) is interpreted as a worker configuration property being committed, with
844 any side effects that might produce errors complete.

845 • Error (ERR) is interpreted as a worker explicitly failing to accept a configuration
846 property write.

847 Each worker may define its own meaning of the write response beyond the indication of
848 success or failure. This can be a range of behavior from "write value stored" to "side-
849 effect of configuration property write action completed".

850 Observation: WCI is not pipelined (datahandshake=0) and is single threaded.

3.5 WCI OCP Signals

Signal configuration is unambiguously determined by the OCP interface configuration parameters (derived from WIP attributes as defined above) and as called out in the "OCP Signal Configuration Parameter" table in the OCP specification. The tables below show the signals from master-to-slave and slave-to-master. The positional order of signals is alphabetical.

Light yellow shading of the description indicates where an OCP signal has added WCI semantics. Details described in following section.

Table 7 – WCI Signals Driven by the Master, to the Slave

OCP Signal	When included	Width	Usage
MAddr	*always*	*If SizeofConfigSpace <= 32, N = 5 Else N = ceil(log2(SizeofConfigSpace-1))*	*Encodes control operation [4:2] and property word addr[x:2]. [1:0]==0, per OCP, and ignored*
MAddrSpace	*SizeofConfigSpace != 0*	*1*	*0=Control Transaction, 1=Config Transaction*
MByteEn	*Sub32BitConfigProperties == TRUE*	*4*	*Byte enables for configuration writes*
MCmd	*always*	*3*	*Read/write command: MCmd[2]==0*
MData	*WritableConfigProperties ==TRUE*	*32*	*Configuration write data*
MFlag	*always*	*2*	*MFlag[0] = 1: Force control operation to terminate with error MFlag[1]:0/1=Little/Big-Endian*
MReset_n	*always*	*1*	*0 = Reset Entire Worker*

Table 8 – WCI Signals Driven by the Slave, to the Master

OCP Signal	When included	Width	Usage
SData	*ReadableConfigProperties == TRUE*	*32*	*Configuration read data*
SFlag	*always*	*1*	*SFlag[0]:1 = Attention*
SResp	*always*	*2*	*Command response*
SThreadBusy	*always*	*1*	*0/1 = Ready/not-read for Cmd*

In the simplest cases where a worker has no configurable properties, is fixed endian, does not assert "attention", and takes no significant time (clocks) for configuration or control operations, there are eight non-constant, non-ignored signals: MAddr[4:2], MCmd[1], MReset_n, SResp[1:0], SThreadBusy.

3.6 WCI Semantics

This section describes specific semantic signal meanings in WCI that are layered on top of the fundamental OCP signals. They are organized by function and then the signals affected.

3.6.1 Indicating Control versus Configuration

WCI transactions are either "control" or "configuration" as described in a previous section. The signal MAddrSpace[0], when configured into the interface, indicates a

"Control" versus "Configuration" command. The value 0 indicates control operations. The value 1 indicates configuration property accesses (when they are needed).

Workers that do not advertise any properties would not have this signal in their interface (SizeofConfigSpace == 0). They may assume all WCI transactions are control transactions.

3.6.2 Encoding of Control

Control operations are invoked by OCP read commands qualified with MAddrSpace[0]='0' as described previously. Writes to the control operation AddrSpace will never occur. The specific control operation is encoded on the MAddr[4:2] signals as follows.

Table 9 – WCI Control Encoding (MCmd=Read; MAddrSpace='0')

Signal	Value	Control Operation
MAddr[4:2]	*B'000 (0)*	*Initialize*
	B'001 (1)	*Start*
	B'010 (2)	*Stop*
	B'011 (3)	*Release*
	B'100 (4)	*Test*
	B'101 (5)	*BeforeQuery*
	B'110 (6)	*AfterConfig*
	B'111 (7)	*RESERVED*

3.6.3 Control Response

WCI control operations are implemented as OCP Read commands to enforce end-to-end signaling. There is no read data returned on SData for control operations. Note that the SData signals are not configured in workers that have no readable properties. However, the worker is always required to return either SResp=DVA for success or SResp=ERR for error in response to a control (or configuration) operation.

The worker's return on SResp of either DVA or ERR signals the completion of the operation. In some cases these control responses may be immediate; in others there may be a delay, perhaps due to acquiring a resource or serially executing a long sequence.

Workers should avoid having excessively long delays for control response. They are "in transition" until such time as either DVA or ERR are returned. Access to configuration properties will not be issued during the time between control operation issuance and response, owing to the single threaded interface. This is intentional: no configuration property changes are allowed during control operation execution.

Workers must return a response ERR in response to a Worker Control Operation Interrupt (MFlag[0]) when that master-to-slave interrupt assertion is made while the request response is pending (see below for details).

Note that the "read side effect" aspect of this control scheme is defined only at the boundary between the infrastructure IP of the control system, and the worker. This does *not* imply that such a read operation is directly initiated by software on some external bus or interconnect. The mechanism by which software invokes a control operation on a worker is determined by the infrastructure IP (e.g. CPIP), not this WCI behavior. Thus the typical problems of read side effects (from bus retries or fetch-

ahead) are not relevant here. The design decision was based on worker implementation simplicity and minimizing signal requirements.

3.6.3.1 Retry

If a worker's error response is a transient condition that could be resolved by repeating a command, the worker author may indicate as much in metadata by setting the Retry attribute to TRUE.

3.6.4 Worker Control Operation Interrupt on the MFlag[0] signal

To handle the potential for a worker not responding to a control operation, the signal MFlag[0] is always configured in WCI. The rising edge of MFlag[0]='1' forces a *timely completion* to an outstanding control operation. The WCI master sets MFlag[0]='1' where it remains asserted. It is cleared sometime after a SResp (error or success) is detected by system software. This allows a worker to be interruptible without implementing any timeout scheme itself.

Rule: When processing a control operation, a worker must provide a response within 16 clock cycles from receiving MFlag[0]='1' (similar to the 16 cycle reset requirement of OCP). This timely response may be either the normal response to the outstanding control operation, or an error response. It must not cause an "extra" response.

Observation: If a worker never takes longer than 16 clock cycles to respond to a control operation response, it has no reason to monitor MFlag[0] and can ignore it.

MFlag[0] is signaled synchronously with the WCI interface clock and is out-of-band with respect to commands and responses, per the OCP specification, .

3.6.5 Worker "Endianness" Indicated on the MFlag[1] signal

Workers that have the capability of operating either little-endian or big-endian are called *bi-endian* workers. The signal driven by the infrastructure on the MFlag[1] signal indicates to such a worker if it is in a little- or big-endian environment. The worker only has this signal if the global Endian attribute is set to the "dynamic".

WCI MFlag[1]	Description
0	Environment is Little-Endian
1	Environment is Big-Endian

3.6.6 Worker Attention on the SFlag[0] signal

The signal SFlag[0] provides the capability for a worker to ask for attention. Assertion of SFlag[0]='1' alerts the control system that the worker requests attention. The worker holds it asserted as a level. Worker attention is cleared by some worker-defined action.

The semantics of this attention (i.e. why a given worker would assert this) is worker-specific, and thus generic controlling software would know what to do with it. A typical response would be for a worker-specific handler to read certain configuration property values to find out "what happened that requires attention". Otherwise generic software can simply report the error condition. Examples include "a new high water mark to some measured value has been reached", or "an overrun condition has been detected".

After this, some writable property might acknowledge receipt of the information causing the attention indication and thus clear it.

SFlag[0] is signaled synchronously with the WCI interface clock, but “out of band” with respect of commands and responses, per the OCP specification.

3.6.7 Configuration Property Access

“Configuration Properties” represent the collection of accessible locations inside a worker that can be read or written. Hardware engineers may think of configuration properties as the “control/status register description” of the worker IP. These locations don’t necessarily have “register” storage. They can be anything abstracted though load/store semantics and may have worker-specific side effects.

The default choice is for all properties to be at least 32 bits in size, the full data width of the WCI OCP interface. In the case where the worker provides sub-32-bit properties, individual 1-byte or 2 byte locations may be read or written, using the OCP byte enables (note any low order address bits - per OCP).

Access to configuration properties occurs when MCmd=RD or MCmd=WR and MAddrSpace[0]=’1’. In this case the word address of the configuration property is signaled on MAddr[N:2]. Note that when configured, SData and/or MData must use data_wdth=32. The MAddr[1:0] shall always be zero, per OCP[3].

3.6.8 Reset

Reset has special meaning for WCI. It is described in section 2.9 on Reset Propagation and Behavior

3.7 WCI Infrastructure Coping

The infrastructure is the master of the WCI interface, and must support the choices made by the worker implementing the WCI slave. The WCI attributes are largely used to add certain features as required by the implementer. The attribute choices that must be handled by the infrastructure IP acting as master of the WCI interface are:

- Presence or absence of configuration parameters, implying presence or absence of data and address space signals.
- Readability or writability of properties, implying presence or absence of the data signals in each direction.
- Presence of sub32bit properties, implying presence or absence of byte enables.
- Size of the usable configuration address space, determining the width of the address signals, and the range of value address values.

WCI masters, which are infrastructure IP, should include all the optional signals. For signals that are optional for the worker, the table below describes how the infrastructure

[3] MAddr is a byte address that must be aligned to the OCP word size (`data_wdth`). If the OCP word size is larger than a single byte, the aggregate is addressed at the OCP word-aligned address and the ***lowest order address bits are hardwired to 0***. (from OCP specification, page 15)

(WCI master) accommodates their absence on the worker (or uses OCP tieoff rules to accommodate them):

Table 10 – Infrastructure Handling of Optional WCI Signals

OCP Signal	When Signal would be absent at the worker	How missing signals are handled
MAddr	*High order [5+] may be missing, depending on SizeofConfigSpace*	*OCP Tieoff rules apply: No connection is made, extra MAddr bits are only driven to the default tieoff: 0. This should be checked in the test environment.*
MAddrSpace	*SizeofConfigSpace == 0*	*OCP Tieoff rules apply: No connection is made, MAddrSpace only driven to the default tieoff: 0 (indicating control). This should be checked in the test environment.*
MByteEn	*Sub32BitConfigProperties == FALSE*	*OCP Tieoff rules apply: No connection is made, MByteEn only driven to the default tieoff: all 1s (indicating full 32 bit words). This should be checked in the test environment.*
MData	*WritableConfigProperties ==FALSE*	*OCP Tieoff rules apply: No connection is made, MData will never be driven at all. This should be checked in the test environment.*
SData	*ReadableConfigProperties == FALSE*	*OCP Tieoff rules apply: No connection is made, SData will never be driven at all.*

4 Worker Data Interfaces

The Worker Streaming Interface (WSI) and Worker Message Interface (WMI) are both worker data interface profiles that enable message-oriented, data-plane communication. These interfaces are used by workers to produce or consume streams of data messages to or from other workers, whether collocated in the same device or executing elsewhere (possibly in a different type of device, including other workers implemented in software). This is the WIP data plane.

From the point of view of an application worker, it is talking to another worker in the application or some source or sink I/O or network endpoint at the “edge” of the application. However, the other side of the interface may be infrastructure IP in these cases:

- The other application worker is local, but needs an adapter for the two data interfaces to interoperate. Example: one worker producing on a WSI, and the other consuming on a WMI.
- This worker is at the “edge” of the application, and the data is being sent to or received from some external source or I/O device. Example: a signal is being received from an A/D converter, and the worker is receiving the output of the A/D via the infrastructure IP that deals with the A/D hardware.
- The other worker is remote, in another device, requiring infrastructure IP to forward messages across some interconnect or fabric. Example: a worker is producing data for a consuming worker in a different processor connected by a fabric or bus attached to the local processor or FPGA.

4.1 Data Protocol between Workers: the Data Plane Protocol

Worker Data Interfaces convey data content between workers according to a defined simple data protocol described by metadata attributes. Certain protocol aspects (e.g. explicit length, or message type) are supported and defined in this specification to “flatten” the overall protocol stack: simple protocol aspects that are efficiently implementable at this level of interface (based on OCP) are supported here. The definition and format of actual message payloads are not defined here. This data plane protocol is independent of:

- the actual width (in wires/signals) of the interface
- the chosen data interface profile (WSI or WMI)
- the implementation-specific attributes specified by the worker implementer.

The protocol is defined as a sequence of fixed or variable size messages, each of which may have a “message type” or “payload type” called an opcode. Messages contain data values, which may or may not be same size within a message. When all the data values in all types of messages in the protocol are the same size, the infrastructure can provide interoperability support between diverse endianness. By knowing the message granularity (the least common multiple of all message sizes), the actual signal interface can be simplified to the minimum necessary (e.g. eliminate byte enables).

The simplest case of the "protocol" is a sequence of single-data-value messages of a single message type. This is the way to describe such a continuous stream of data values:

- There is one type of message (and thus no need for an explicit "opcode").
- The message length is fixed as the size of a single data value (and thus no need for an explicit message length, e.g. if the data values are single precision IEEE floats, all messages are 32 bits/4 bytes).

In the more complex case all these attributes may be used to specify the "data protocol":

- Size in bits of the smallest data values in all messages (default to 8). Note that the width of the data path is never allowed to be less than this.
- Maximum message length in data values (default is 1).
- Whether messages are variable length or are all the same (maximum) length. (default is fixed length)
- The granularity of message sizes in number of data values (default: 1 data value, example of complex single float: 2 values).
- How many different message types (also called "opcodes") (default: 1)

The concept of "data value" is used to specify interfaces that are not byte oriented. If the data message is byte oriented, then MaxMessageValues is simply the maximum message length in bytes. If all defaults are used, all messages are of the same type, and are one byte long. Streams of single values with no real message boundaries are specified simply by specifying the DataValueWidth attribute to the size of the data values (in bits). Thus the concept of "length" of messages is always in units of data values.

These attributes are specified for each of the worker's data interfaces, independent of choices of WSI or WMI, or other implementation-specific attributes. A worker "implementer" then chooses either WSI (streaming, fifo-style) or WMI (buffered or more complex cases) to implement the (producer or consumer) data interface. Thus the message attributes above are about the messages produced or consumed, and not about the WIP profile used, or the implementation choices within that profile.

4.2 Worker Data Interfaces Attribute Summary

The WIP data *protocol* attributes are described in the following table. While these attributes are valid for all data interface profiles (WSI and WMI), the way the attributes affect actual OCP signals in the interface are different and specified in the section for each profile. The "producer/consumer" role attribute of the interface is also here since it is also common to all data interface profiles. These attributes are repeated in the attribute tables for the data interface protocols.

1059 **Table 11 – Worker Data Interface Protocol Attributes**

Attribute Name	Type (per XML Schema)	Default Value	Description
DataValueWidth	*unsignedInt*	*8 (octets)*	*Number of bits in the smallest data value in any message.*
DataValueGranularity	*unsignedInt*	*1*	*The number of data values in all messages are a multiple of this value.*
DiverseDataSizes	*boolean*	*false*	*TRUE: the data value sizes in messages vary, in which case automatic endian transformation is not possible. FALSE: the data values in all messages are all the same size: DataValueWidth*
MaxMessageValues	*unsignedInt*	*1*	*Maximum number of values transferred in the longest message over all opcodes.*
NumberOfOpcodes	*unsignedInt*	*1*	*The number of different message types in the protocol*
Producer	*boolean*	*false*	*TRUE: the interface role is PRODUCER; FALSE: it is CONSUMER*
VariableMessageLength	*boolean*	*false*	*TRUE: the message length may vary, FALSE: message length is fixed.*
ZeroLengthMessages	*boolean*	*false*	*TRUE: some message types may be zero length FALSE: all messages contain at least one data value.*

1060 ## *4.3 Worker Data Interfaces Attribute Details*

1061 This subsection contains the per-attribute details for data interfaces.

DataValueWidth	*unsignedInt*	*8 (octets)*	*Number of bits in the smallest data value in any message.*

1062 This attribute indicates the smallest unit of data, in bits, in the message payloads, and
1063 thus is the "bit granularity" of the message payloads. When the actual interface width is
1064 known, this value helps in determining things like byte enables and endian packing.
1065 The default is 8 for typical byte oriented payloads. The actual width of the data
1066 interfaces are not allowed to be smaller than this. This term was used in preference to
1067 "ByteWidth" (used in WMemI) since it has more semantics than typically associated with
1068 the term "byte".

DataValueGranularity	*unsignedInt*	*1*	*The number of data values in all messages are a multiple of this value.*

1069 This attribute further constrains the length of messages, indicating that the size of all
1070 messages are this multiple of the basic data value size. This value also helps in byte
1071 enable and endian packing determination for interfaces.

1072 An example might be that the data stream consists of pairs of 32 bit single float data
1073 values (thus the interface width cannot be less than 32), thus all messages are a
1074 multiple of two such values or 64 bits. The interface is allowed to be 32 bits or wider. If
1075 it is 64 bits wide, then all messages are full words. If it is 128 bits wide, then the last
1076 128-bit word of a message may only be half full.

DiverseDataSizes	*boolean*	*false*	*TRUE: the data value sizes in messages vary, in which case automatic endian transformation is not possible. FALSE: the data values in all messages are all the same size: DataValueWidth*

1077 This attribute indicates whether message payloads consist of different sized data values
1078 or not. Then false, it indicates uniform data value sizes and thus enables infrastructure-
1079 supported endian adaptation.

MaxMessageValues	*unsignedInt*	*1*	*Maximum number of values transferred in the longest message over all opcodes.*

The maximum (or fixed) size of messages, expressed in data values, each of which is DataValueWidth in size. If not specified, the default is that all messages are single data values. This value can be zero, indicating that all "messages" have no payload at all.

NumberOfOpcodes	*unsignedInt*	*1*	*The number of different message types in the protocol*

This attribute specified the number of different message types in the protocol. Opcodes are conveyed separate from message data, to indicate the message type. Typically different message types have different payload layouts. The default value of 1 is used when there is only one type of message in the stream.

Producer	*boolean*	*false*	*TRUE: the interface role is PRODUCER; FALSE: it is CONSUMER*

This attribute indicates whether the interface of the worker is a producer (TRUE) or consumer (FALSE).

VariableMessageLength	*boolean*	*false*	*TRUE: the message length may vary, FALSE: message length is fixed.*

This attribute indicates whether all messages are the same size (FALSE, the default), or whether they can have different lengths (TRUE).

ZeroLengthMessages	*boolean*	*false*	*TRUE: some message types may be zero length FALSE: all messages contain at least one data value.*

This attribute, when TRUE, indicates that the stream of messages may contain messages of zero length, requiring support for this in the actual interface. If all messages are zero length, and there is one opcode, the protocol is simply a stream of events with no content. When this attribute is true, it forces byte enables to be included in WSI.

4.4 Worker Data Interface OCP Profiles

Beyond the attributes above, there are several aspects of the OCP configuration and signal semantics that are common across Worker Data Interfaces: reset behavior and flow control.

4.4.1 Worker Data Interface Reset Behavior

Workers in the same FPGA communicate with each other via data (plane) interfaces. According to the reset propagation rules mentioned earlier, every worker must propagate its master reset (an input which is on its single WCI), to all of its data interfaces (as an output). Thus a worker that is being reset via its WCI asserts reset on all its data interfaces, using the master or slave OCP reset signal according to the master or slave role of its data interface.

Conversely, a worker that is stoppable and suspended, or has just come out of reset itself (after its WCI reset has been deasserted), must accept a reset input on its data ports, indicating that its adjacent worker is being reset via its *WCI*. Such reset inputs at data interfaces will not be asserted when the worker's WCI commands it to **initialize** or **start**.

4.4.2 Worker Data Interface Flow Control

Worker data interfaces all use OCP pipelined, non-blocking flow control (see “Flow Control Options” in the OCP specification). Thus flow control is accomplished without combinatorial paths crossing the interface more than once in a cycle. It also means that when the interface is flow-controlled it is *between* transactions, not in the middle of a transaction. This allows a worker master to take unilateral actions (such as abandoning a transaction) based on being blocked (perhaps for some amount of time), without affecting the other side of the interface and without backing out of a transaction.

5 Data Widths and Byte Enables in WSI, WMI, and WMemI profiles

In these three profiles, the width of the data path is defined in the WIP attributes by the implementer. This width is specified by setting the DataWidth attribute, whose default value (for WSI and WMI) is the value of the DataValueWidth protocol attribute described above. Also, all these profiles define a minimum unit of data such that the data path width must be a multiple of that minimum width. We call this minimum width the ByteWidth of the interface, which of course is commonly = 8 (for octets). For Worker Data Interfaces, we use a synonym: "DataValueWidth", to connote that this is related to message payloads (i.e. the width of the data values in the payload). Since the WMemI profile is not associated with message payloads, we simply use "ByteWidth" for it. Due to the requirements of OCP, and to keep simple cases simple for the implementer, we treat three cases differently for all these profiles:

- DataWidth is the same as ByteWidth
- DataWidth is a multiple of ByteWidth, and ByteWidth is 8 (bytes are octets)
- DataWidth is a multiple of ByteWidth, and ByteWidth is greater than 8

There is no support for ByteWidths less than 8 when ByteWidth is not the same as DataWidth. Byte enables are only relevant to writes in these profiles.

5.1 DataWidth equals ByteWidth

This simple case sets the normal OCP data field (the data_wdth configuration parameter for the MData and SData signals). There is no need for byte enables, and no need for the complexity of the OCP "datainfo" signals. There is no WIP or OCP constraint on the value of DataWidth.

5.2 DataWidth is a multiple of ByteWidth, and ByteWidth is 8 (bytes are octets)

This case is fairly simple and common, and well supported by OCP. Write byte enables are enabled in the interface unless, for WSI, the message size granularity implies that they would never be needed (i.e. all words are always "full", not partial).

OCP dictates that each byte enable signal corresponds to exactly 8 bits of the OCP MData path, and that path must be a multiple of 8 bits if byte enables are used.

5.3 DataWidth is a multiple of ByteWidth, and ByteWidth is greater than 8

To properly handle this case, OCP dictates that the data be split into two separate signal fields, {M,S}data and {M,S}datainfo. 8 bits of each byte are packed into the OCP {M,S}Data field, and the remaining bits of each byte are packed into the {M,S}DataInfo field. For example, if the WIP logical DataWidth was 36, and ByteWidth was 9, then the OCP MData field would carry logical bits <34:27><25:18><16:9><7:0>, and the OCP MDataInfo path would carry logical bits <35><26><17><8>.

Thus the worker implementer (and possibly automation tools) might map the MData and MDataInfo fields into a single 36 bit Data field internal to the implementation for clarity and convenience.

6 Worker Streaming Interface (WSI)

See the section on Worker Data Interfaces for information that applies to all data-plane WIP.

6.1 WSI Motivation

The most common and simple way for one data processing IP block to communicate with another (with flow control) is through a parallel interface with simple FIFO semantics, typically with data and data-strobe signals from the producer and "ready" or "accept" signal from the consumer. The WIP profiles must certainly support this common, lean, simple case. The WSI is intended to satisfy this requirement, configuring OCP to meet this objective.

6.2 WSI Overview

The worker streaming interface is designed to be a simple, FIFO sequential method for workers to exchange a stream of messages. In WSI, workers producing data are always OCP masters, while workers consuming data are always OCP slaves. In cases where the workers are collocated (in the same device), this allows the possibility of "glue-less" or "gasket-less" signal connection between workers when both implementers choose the same WIP attributes.

WSI is purposefully "feature-constrained" for simplicity. In additional to the data protocol attributes described earlier, the worker implementer makes these choices:

- How wide (in bits) is the streaming data interface; i.e. how many bits does one wish to produce or consume in a single OCP clock cycle?
- Does the worker wish to support precise bursts? (i.e. is the message length known at the start of the message?).
- Do the message OCP requests need to come ahead of (in advance of) the OCP message data itself? (especially in the case of the first request for the first data word of the message).
- Are idle cycles permitted in the middle of the message?
- Can a frame be aborted and discarded after it has started transmission?

WSI uses basic OCP burst semantics to represent a message or "frame":

- The start of a burst is the start of a message
- Burst length is message length (in transfer words of the data width of the interface)
- Precise OCP bursts may be used when message length is known
- Imprecise OCP burst may be used when the producer cannot indicate exact message length at the start of a message
- The end of the burst is the end of a message

While a "WIP Message" is an abstraction used in both the WSI and WMI profiles and an "OCP Burst" is defined by the OCP specification, WSI maps WIP messages to OCP bursts, 1-to-1. WMI does not.

Workers using the WSI may want the ability to signal as early as possible, before data, the impending arrival of a message. The EarlyRequest attribute is used to enable this optimal latency behavior. Masters (producers) are free to insert intra-message idle data cycles either by driving MCmd=IDLE or (when EarlyRequest = TRUE) by de-asserting MDataValid, unless the Continuous attribute is set.

When ImpreciseBursts are being used, and the Abortable attribute is set, the producer/master can abort a frame, indicating to the slave that the message should be discarded. This enables the typical cut-through optimization in some cases.

In all cases the default OCP burstsinglereq=0 semantics apply: there will be exactly as many valid request beats as there are valid data words. WSI only allows FIFO access to data. Use WMI when random or out-of-order access to message contents is required, either as producer or consumer.

6.3 WSI Attributes

Worker authors using a WSI specify their implementation choices via the following attributes (as well as the data protocol attributes above), from which all OCP configuration parameters and signals are derived.

The Clock and ClockSignal attributes, described in section 2.6, are applicable to WSI.

Since WSI is a data plane interface, the worker data interface protocol attributes are applicable, as described in section 4.2. These are: DataValueWidth, DataValueGranularity, DiverseDataSizes, MaxMessageValues, NumberOfOpcodes, Producer, VariableMessageLength, and ZeroLengthMessages.

6.3.1 WSI Attribute Summary

Table 12 – WSI Attribute Table

Attribute Name	Type (per XML Schema)	Default Value	Description
Continuous	*boolean*	*false*	*TRUE: within a burst, there shall be no idle cycles.* *FALSE: there may be idle cycles within a burst*
DataWidth	*unsignedInt*	*DataValue-Width*	*Physical width in bits/wires of the data path.*
ByteWidth:DERIVED FROM OTHER ATTRIBUTES	*unsignedInt*	*none*	*Smallest data that the interface needs to handle:* *If DataValueWidth*DataValueGranularity is a multiple of DataWidth and ZeroLengthMessage = FALSE:* *ByteWidth = DataWidth (thus no byte enables)* *Else:* *ByteWidth = DataValueWidth*
ImpreciseBurst	*boolean*	*false*	*TRUE: Imprecise OCP bursts will be used* *FALSE: Imprecise OCP bursts will not be used*
PreciseBurst	*boolean*	*false*	*TRUE: Precise OCP bursts will be used* *FALSE: Precise OCP bursts will not be used*
Abortable	*boolean*	*false*	*TRUE: Unfinished messages can be aborted* *FALSE: All messages are transmitted in full.*
EarlyRequest	*boolean*	*false*	*TRUE: Start-of-message can occur before data* *FALSE: First data will accompany start of message.*

6.3.2 WSI Attribute Details

This subsection contains the per-attribute details of choices available for a WSI.

Continuous	*boolean*	*false*	*TRUE: within a burst, there shall be no idle cycles.* *FALSE: there may be idle cycles within a burst*

The Continuous attribute indicates *continuous* data bursts between master and slave, meaning that once burst data transfer begins, data is transferred on every cycle without interruption, either by the master (no idle cycles) or the slave (no flow control via SThreadBusy). This attribute only refers to *intra*-burst continuity and implies nothing about *inter*-burst behavior.

Continuous=TRUE indicates, for producers: "Once I get started and allowed by flow control at the start of a burst, I will produce all the data in one continuous burst with no idle cycles and cannot accept any flow control (backpressure)". For consumers: "Once I receive the first data in a burst, I require and will accept all the data in one continuous burst, without idle cycles, and without asserting SThreadBusy until after the last request in the burst."

This attribute does not change the OCP configuration or any of the signals in the interface. It expresses two aspects of the data transfer: *data will fill every cycle,* and no *data flow-control will occur within the burst*. This attribute is common to the WSI, WMI, and WMemI profiles.

DataWidth	*unsignedInt*	*DataValue-Width*	*Physical width in bits/wires of the data path.*

The DataWidth attribute is the physical width (word size) of the interface in bits. It must be a multiple of the DataValueWidth attribute. If not specified, the default value is the DataValueWidth attribute of the protocol used over this interface. When the DataValueWidth*DataValueGranularity is smaller than the DataWidth, and not a multiple of 8, OCP requires the data path to be split between two OCP fields (MData and MDataInfo). Octet-oriented or word-oriented interfaces never have this complexity. When DataValueWidth*DataValueGranularity is smaller than DataWidth, byte enables are required in the interface.

This attribute is common to the WSI, WMI, and WMemI profiles, except that for the WMemI profile, the default value is 8.

ByteWidth:DERIVED	*unsignedInt*	*none*	*Smallest data that the interface needs to handle:* *If DataValueWidth*DataValueGranularity is a multiple of DataWidth and ZeroLengthMessage = FALSE:* *ByteWidth = DataWidth (thus no byte enables)* *Else:* *ByteWidth = DataValueWidth*

ByteWidth expresses the size in bits of the smallest data value that the interface needs to handle, and is used to enable byte enable functionality in WSI, WMI, and WMemI.

For WSI, ByteWidth is a derived based on data protocol attributes. It is not specified directly. For WSI, when ByteWidth is less than DataWidth, byte enables are enabled to indicate the actual valid data in the last word messages (and, when ZeroLengthMessages = TRUE, that there is no data in the last word).

Thus byte enables are only required in WSI when the messages may have zero length, or may contain a number of values that do not fill the last DataWidth word of the message.

ImpreciseBurst	*boolean*	*false*	*TRUE: Imprecise OCP bursts will be used* *FALSE: Imprecise OCP bursts will not be used*
PreciseBurst	*boolean*	*false*	*TRUE: Precise OCP bursts will be used* *FALSE: Precise OCP bursts will not be used*

These burst attributes indicate what sort of burst transactions will be issued by the worker, as OCP master, or accepted by the worker as OCP slave. These attributes are common to WSI, WMI and WMemI WIPs.

For WSI and WMI, one, but not both of these attributes must be TRUE.

Abortable	*boolean*	*false*	*TRUE: Unfinished messages can be aborted* *FALSE: All messages are transmitted in full.*

Abortable specifies that the producer may abort a message before it is fully transmitted, and that the consumer can handle and respect such an action. This is an important optimization when both sides are able to do this without extra buffering. It is analogous to the cut-through routing optimization in switches. The semantic is similar to the "discontinue" action in the Xilinx LocalLink Interface. It is optional since it has a modest cost in interface complexity, and can only be TRUE when ImpreciseBurst is TRUE.

EarlyRequest	*boolean*	*false*	*TRUE: Start-of-message can occur before data* *FALSE: First data will accompany the start of message.*

The attribute EarlyRequest is an implementation choice that allows, when TRUE, the WSI to convey message arrival (a.k.a. "start of message") in advance of the first data in the message. In general, this is a latency improvement: you can start a message earlier.

For consumers, use EarlyRequest=TRUE in situations where the early indication of message arrival has value, typically to reduce latency. For producers, pass along this information if it is available: i.e. of the producer knows about a message before it has the first data of the message, it should pass this potentially latency-improving information to the consumer.

When EarlyRequest = FALSE, the first data word of a message is required to be available at the same time as the first OCP request of the message. Secondarily, data is supplied at the same time as the OCP request for *all* words of the message.

When EarlyRequest=TRUE, the interface uses the OCP defined behavior of datahandshake=1 to decouple the request from the data. This allows the producing-master to send OCP requests ahead of (or at the same time as), the corresponding data.

Since OCP datahandshake behavior is thus optional, single-request-multiple-data (SRMD) bursts are not allowed for WSI. All bursts are always MRMD.

6.4 WSI OCP Profile/Configuration Parameters

The WSI OCP profile is the aspect of the WSI that has to do with OCP configuration parameters and signal configuration. This section describes the concrete, complete, and unambiguous transformation from WSI WIP attributes described above, to the OCP configuration parameters inferred from those attributes. This transformation is implemented in automation tools.

The table below lists only those parameters that are different from the OCP configuration parameter defaults (per the OCP specification).

1292 **Table 13 – Worker Stream Interface OCP Configuration Parameters**

OCP Configuration Parameter	Value	Description (How OCP parameter is determined if not constant, and any tieoffs)
addr	*0*	*Note: No addresses, FIFO-like*
burstlength	*1*	
burstlength_wdth	*{P}*	*If PreciseBurst = TRUE:* *NWords = ceil(MaxMessageValues*DataValueWidth/DataWidth)* *P = max(2, floor(log2(N)) + 1)* *Else :* *P = 2;*
burstprecise	*0*	*If PreciseBurst = FALSE:* *tieoff MPreciseBurst to 0 (OCP default tieoff is 1)*
cmdaccept	*0*	*Note: The SThreadBusy signal is used for backpressure.*
datahandshake	*{0|1}*	*If EarlyRequest = FALSE: 0, Else: 1*
datalast	*{0|1}*	*If EarlyRequest = FALSE: 0, Else: 1*
reqlast	*1*	
data_wdth	*{Q}*	*If ByteWidth != DataWidth and ByteWidth != 8:* *Q = 8 * DataWidth/ByteWidth* *Else:* *Q = DataWidth*
byteen	*{0:1}*	*If ByteWidth != DataWidth || ZeroLengthMessages = TRUE: 1, Else: 0*
mdatainfo	*{0|1}*	*If ByteWidth != DataWidth and ByteWidth != 8: 1* *Else: 0*
mdatainfo_wdth	*{R}*	*If ByteWidth != DataWidth and ByteWidth != 8:* *R = DataWidth – (8 * DataWidth/ByteWidth)* *Else* *R = 0* *If Abortable: R = R + 1*
mdatainfobyte_wdth	*{S}*	*If ByteWidth != DataWidth and ByteWidth != 8:* *S = ByteWidth - 8*
mreset	*1*	*Note: producer may reset the connection*
read_enable	*0*	*Note: Push only stream – No reads*
sdata		
resp	*0*	
sreset	*1*	*Note: consumer may reset the connection*
reqinfo	*{0|1}*	*If NumberOfOpcodes > 1: 1, Else: 0*
reqinfo_wdth	*{T}*	*T = Ceil(log2(NumberOfOpcodes))*
sthreadbusy	*1*	*Note: SThreadBusy signal supports pipelined, proactive, flow-control backpressure from slave to master.*
sthreadbusy_exact		
sthreadbusy_pipelined		

1293 ### *6.5 WSI OCP Signals*

1294 Signal configuration is determined by the above interface configuration parameters as
1295 called out in the “OCP Signal Configuration Parameter” table in the OCP specification.
1296 The tables below show the signal from master-to-slave and slave-to-master. The
1297 positional order of signals is alphabetical.

1298 Light yellow shading of the description indicates where an OCP signal has added WSI
1299 semantics. Details described in following section.

1300 **Table 14 – WSI OCP Signals Driven by the Producer/Master**

OCP Signal	When Included	Width	Usage
MBurstLength	*always*	*If PreciseBurst = TRUE:* *max(2,floor(log2(ceil(* *MaxMessageValues** *DataValueWidth/* *DataWidth)))+1)* *Else:* *2;*	*When PreciseBurst = TRUE, the number of words in the burst, constant over the burst (OCP Precise Burst).* *When PreciseBurst = FALSE,* *= b'01' for the last request of the burst,* *= b'10' otherwise (OCP Imprecise Burst)*
MByteEn	*ByteWidth != DataWidth || ZeroLength-Messages= TRUE*	*DataWidth/ByteWidth*	*WSI Specific Semantics (if included):* *Must be all 1s in all requests but last one in the burst. When ZeroMessageLength= TRUE, then they can be all zeroes in the last or only request in the burst.*
MCmd	*always*	*3*	*Transfer Command; only writes allowed. Only one bit is not constant.*
MData	*If DataWidth != 0*	*If ByteWidth != DataWidth and ByteWidth != 8:* *8 *DataWidth/ByteWidth* *Else:* *DataWidth*	
MDataInfo	*If (ByteWidth != DataWidth and ByteWidth != 8) or Abortable*	*DataWidth – (8 * DataWidth/ByteWidth) + (Abortable ? 1 : 0)*	*Extra write data bits not allowed in MData per OCP* *WSI Specific Semantic:* *If Abortable, MSB indicates ABORT in last data of imprecise burst*
MDataLast	*EarlyRequest = TRUE*	*1*	*Indicates last beat of data*
MDataValid	*EarlyRequest = TRUE*	*1*	*Qualifies active data beats*
MReqInfo	*NumberOf-Opcodes > 1*	*Ceil(log2(NumberOfOpcodes))*	*WSI Specific Semantics:* *Message Opcode indication*
MReqLast	*always*	*1*	*Indicates last request of bursts.*
MReset_n	*always*	*1*	*Push Reset Master-to-Slave, producer to consumer (Active Low)*

1301 **Table 15 – WSI OCP Signals Driven by the Consumer/Slave**

OCP Signal	When included	Width	Usage
SReset_n	*always*	*1*	*0 = Push Reset Slave-to-Master (Active Low)*
SThreadBusy	*always*	*1*	*Allows slave to provide backpressure, preventing new requests*

1302 ## *6.6 WSI Semantics*

1303 This section describes specific semantic signal meanings in WSI that are layered on top
1304 of the fundamental OCP signal semantics. The semantics stay within OCP compliance,
1305 so that even though they are additional behavioral constraints, they represent valid
1306 OCP-defined behavior.

1307 ### 6.6.1 Opcode Indication on MReqInfo

1308 When there is more than one message op-code, the MReqInfo signal is configured into
1309 the interface. This extra information about the message, its opcode, must be constant
1310 over the message/burst.

6.6.2 Messages (or "frames") are OCP Bursts

Messages are transmitted as OCP bursts, with the start and end of a message being the first and last word of a burst. When PreciseBurst = TRUE, the interface is configured statically for precise bursts (MPreciseBurst tied off to 1 in the master – the OCP default, ignored in the slave), otherwise it is configured for imprecise bursts (MPreciseBurst tied off to 0 in the master, ignored in the slave). In all cases the master asserts MReqLast with the last request of the burst. This means that slaves, whether precise or imprecise, can use this direct mechanism to determine the last request in the burst in all cases, without counting. The master of an imprecise burst must also drive the value of "0b01" in the MBurstLength field of the last request of the burst for OCP compliance, but this is in fact redundant with MReqLast.

When the EarlyRequest attribute = TRUE, the end of a message can also be determined (precise or imprecise) with the MDataLast signal associated with the data. When EarlyRequest = TRUE, the first request can be supplied before the first data is available, to improve message startup latency in some cases.

Interfaces configured with MaxMessageValues = 1 essentially have single word bursts, and the MBurstLength signal is constant (0b01).

6.6.3 Byte enables

Byte enables are configured and used (when necessary) to indicate partial word content in the last word of the message/burst. When they are included, they must be all ones for all words in the message except the last. They are only included when needed, when it is known from the data protocol attributes that messages may have lengths that are not a multiple of DataWidth or are zero.

Also, the use of byte enables must be consistent with the endianness of the data interface. If the interface is big endian, then partial words will have low order byte enables set to zero as appropriate (e.g. if there is one valid byte, it must be the high order byte). If the interface is little endian, then partial words will have high order byte enables set to zero as appropriate (e.g. if there is one valid byte, it must be the low order byte). A typical example would be when input data was stored into an internal buffer, using the byte enables. At this point the worker's endianness would determine which bytes in memory are valid, since the byte enables are only valid during the write to internal memory, and are not necessarily captured anywhere.

If the data protocol requires support for zero length messages (ZeroLengthMessages), byte enables are also included and may be all zero in the last or only word of a zero length message. Note that enabling ZeroLengthMessages in a WSI *also* enables the last word of non-zero-length messages to be empty. In particular, this allows a producer to signal the end of a message *without* data (and thus without knowing if any word is in fact the last one). This is useful for a producer what wants to pass every word of data to the consumer as soon as it is available, without needing to wait until it knows whether that word is in fact the last one or not (in the case of imprecise bursts).

Thus the byte enable logic of a WSI is constrained beyond OCP, and all WSI transactions are OCP compliant.

6.6.4 Aborting Messages

When the interface is using ImpreciseBursts, and the Abortable attribute is set, a feature is enabled allowing the master to terminate a burst and indicate that the message is being "aborted", which indicates to the slave that the message should not be processed, and should be discarded. This indication is placed on MDataInfo[N], which is sampled by the slave with the data (MData) in the last data beat of an imprecise burst. N is the MSB of this field.

This feature is useful to reduce latency and buffering in some cases. An example is a "framer" worker that is extracting frames with CRC checks from a continuous stream. If it starts receiving a frame, it doesn't know whether the frame is valid until the CRC is checked. Since the common case is that the CRC is valid, it wants to forward the beginning of the frame to its output as soon as possible and not delay transmission (and avoid buffering) waiting until the CRC. In the rare case that the CRC is bad, it wants to abort the frame and start scanning for a new frame. If the downstream worker is already buffering the frame (e.g. when it is waiting to DMA a whole frame over a bus), then it can implement the "abort" feature with no extra buffering, since it is buffering already.

If the downstream worker cannot handle aborting a message, then a buffer can be inserted between the two workers.

6.7 WSI Infrastructure Coping

Infrastructure IP supporting WSI interfaces (attached to application IP) falls into three categories:

- Adapter IP for interposing between workers that do not have directly interoperable data plane interfaces, but are otherwise logically, locally, connected.
- Communications IP when workers are not in the same device and need the data messages conveyed across some inter-chip interconnection or fabric (e.g. PCI, RapidIO, LVDS parallel, Aurora serial, XAUI serial, etc.
- I/O IP when workers are receiving or sending IP to an I/O device (sink or source of data) directly attached to the device (e.g. ADC/DAC).

The first category (adapters) does the most "coping" as that is their job: to enable interoperability when worker developers make different attribute choices. Essentially adapters convert between choices like:

- Early Request or not
- Fixed or Variable size messages
- Precise or Imprecise Bursts
- Different width data paths
- Continuous data or not.
- Abortable or not.
- Different Endian

Due to the design of the WIPs, interoperability is well defined in nearly all cases, and trivial to accomplish (by the infrastructure) in many cases. Burst conversions

sometimes are simple logic, sometimes require counting, and sometimes require buffering.

Table 16 – WSI Slave Infrastructure Handling of Optional WSI Signals

OCP Signal	When Signal would be absent at the master	How missing signals are handled
MByteEn	*ByteWidth = DataWidth*	*OCP Tieoff rules apply: No connection is made, MByteEn connected to the default tieoff: all 1s (indicating full words)*
MDataLast	*EarlyRequest==FALSE*	*Adapter: MDataLast = MReqLast*
MDataValid	*EarlyRequest==FALSE*	*Adapter: MDataValid = MCmd[0]*
MDataInfo[MSB]	*Abortable = FALSE*	*OCP Tieoff rules apply: Abort never happens, MDataInfo[MSB] = 0*
MReqInfo	*NumberOfOpcodes=1*	*OCP Tieoff rules apply: No connection is made, MReqInfo tied to the default tieof: 0 (indicating opcode == 0). Infrastructure slaves should generally support/convey 256 opcodes (8 bits).*

Table 17 – WSI Master Infrastructure Handling of Optional WSI Signals

OCP Signal	When Signal would be absent at the slave	How missing signals are handled
MByteEn	*ByteWidth = DataWidth*	*OCP Tieoff rules apply: No connection is made, MByteEn always driven to the default tieoff: all 1s (indicating full words) This should be checked in test environment.*
MDataLast	*EarlyRequest==FALSE*	*Adapter: MDataLast is not connected*
MDataValid	*EarlyRequest==FALSE*	*Adapter: Request info (MCmd[0], MReqLast, MBurstLength,MByteEn) must be captured and delayed until MDataValid. MDataValid is not connected*
MDataInfo[MSB]	*Abortable = FALSE*	*OCP Tieoff tules apply: Abort never asserted, MDataInfo[MSB] = 0*
MReqInfo	*NumberOfOpcodes=1*	*OCP Tieoff rules apply: No connection is made, MReqInfo tied to the default tieof: 0 (indicating opcode == 0). Infrastructure slaves should generally support 256 opcodes (8 bits).*

These tables above assume an infrastructure that supports all features (byte enables, early request/data handshake, and opcodes). Conversion between precise and imprecise burst lengths is slightly more complicated, but well defined.

Note that when infrastructure is conveying messages between application workers, interoperability can be determined even when the infrastructure supports features that the application worker does not. A good example is if an infrastructure master/producer generically supports 256 opcodes, but the application slave/consumer does not. Here interoperability checking is based on knowing that the upstream application worker (wherever it is), also doesn't produce opcodes, so the infrastructure master will never in fact drive the opcode signals to anything other than the tieoff. If the infrastructure was a data source rather than an interconnect between workers, then interoperability would be based on the infrastructure itself. If a application consumer did not support aborts, the infrastructure must be configured to never generate them.

6.7.1 Infrastructure Best Practice for Master/Producers

Infrastructure IP acting as WSI producer/master should:

- use PreciseBurst = TRUE unless this adds buffering or latency.
- use EarlyRequest = FALSE unless this adds latency (i.e. when the infrastructure IP actually does know that a request should start before it has the data).
- use VariableMessageLength = FALSE when it knows that all messages it produces will be the same size.
- Use Continuous = TRUE when it knows that bursts it produces will never have idle cycles.
- Use Abortable = TRUE if the data source supports aborting (defer buffering until needed)

6.7.2 Infrastructure Best Practice for Slave/Consumers

- Support both precise and imprecise bursts, use MReqLast rather than MBurstLength, unless burst length is in fact required to be known early (in which case an adapter should add the imprecise-to-precise conversion when needed). Using both precise and imprecise bursts is not legal for an application worker producer in any case, but an infrastructure consumer may indeed support both.
- Support EarlyRequest = TRUE.
- Support VariableMessageLength = TRUE unless its function in fact requires fixed message sizes.
- Use Continuous = FALSE, allowing intra-burst idle cycles on input.
- Use Abortable = TRUE if the data sink does, or if buffering is already taking place

7 Worker Message Interface (WMI)

See the earlier section on Worker Data Interfaces for information that applies to this and other data-plane WIPs.

7.1 WMI Motivation

A worker data interface more powerful than WSI, the Worker Message Interface (WMI) is a more complete and complex interface. It is used in preference to WSI only when the worker implementation needs its additional capabilities. Infrastructure supporting WIP should support workers whether they are implemented using WSI or WMI at any worker data interface.

This description is not really standalone; it builds on WSI and adds capabilities.

7.2 WMI Overview

Key features of the WMI beyond what WSI provides include:

- The worker is always the OCP master, the active agent, whether consuming message input or producing message output.
- WMI is not based on the "messages are bursts" model of WSI, since WMI provides random access to message contents, and each such access can be its own burst.
- Random addressing within the content of a message is enabled. This generally implies message buffering by the infrastructure (slave).
- Explicit indication that the worker has finished processing (consuming or producing) a message is available, independent of access to message contents.
- Ability to configure a worker-consumer WMI for write-back, to allow the worker to write to (modify) received message in-place to allow in-place processing to reduce buffer requirements.
- Ability to configure a worker-producer WMI to read-back, to allow the worker to read from (examine) outbound message in-place to allow in-place processing to reduce buffer requirements.
- Ability to abort/discard an input message without accessing all of it.
- Ability to abort/discard an output message without sending it.

Workers that need any of these capabilities use WMI in preference to WSI, and thus generate addresses to access the contents of the message.

WMI provides the worker, as an OCP master, with a view of the current buffer, whether producing or consuming. The slave interface on the other side is infrastructure that might be interfaced to local or remote memory and might include the logic for DMA buffer management.

7.2.1 WMI Consuming versus Producing:

With WMI, when a worker is a message-consumer, it is an OCP master and must read, or “pull”, data from a slave. The infrastructure supplies this slave message-producer as shown below on the left side of the link marked 1.

When a worker is a message-producer, it is always an OCP master and must write, or "push", data to a slave. The infrastructure supplies this slave message-consumer interface as shown below on the right side of the link marked 2.

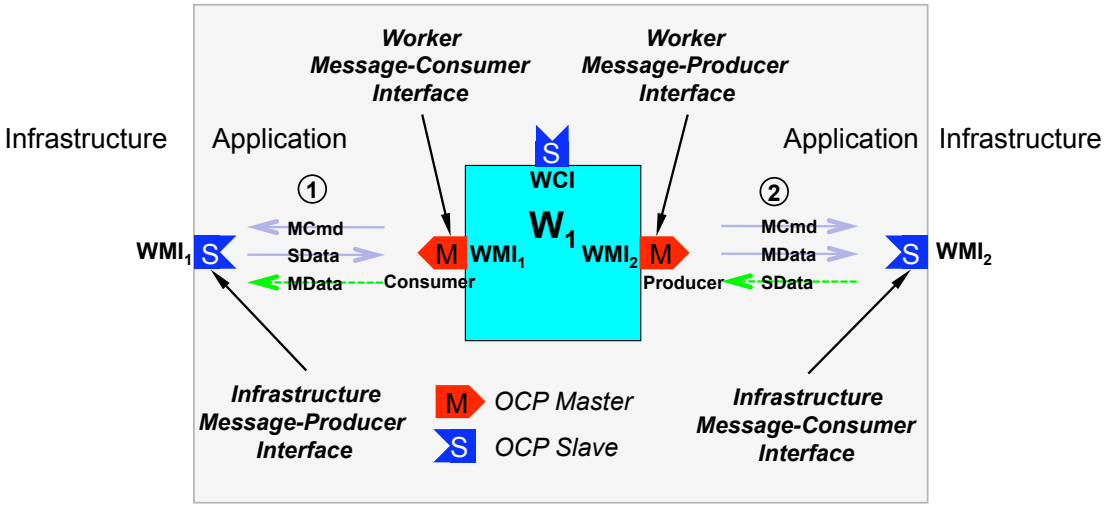


Figure 6 - Signals on Worker with two WMI interfaces

For both worker-consuming and worker-producing, WMI provides the symmetric capabilities of slave buffer availability (to process on input, to fill on output) and a "done with message" indication.

For the worker-consumer:

- Slave/infrastructure indicates source buffer availability (source buffer ready to consume)
- Master/worker indicates "Done with Message" (discard/recycle the consumed source buffer)

For the worker-producer:

- Slave/infrastructure indicates target buffer availability (target buffer ready to fill with message)
- Master/worker indicates "Done with Message" (transmit the finalized contents of the output buffer)

Although the attributes are identical for consumer and producer, the OCP configuration parameters and signals differ owing to the fact that the worker is in an OCP Master role in both cases.

7.2.2 Indication of message length

Since in WMI messages are *not* bursts, the length of a message must be indicated independent of burst length (unless VariableMessageLength = FALSE). For a WMI consumer, both the opcode and the message length are supplied separately from data using the OCP SFlag signals. For the WMI producer, the opcode and message length are indicated on the OCP MFlag signals (again, unless VariableMessageLength = FALSE). The exact semantics are discussed below.

7.2.3 Indication of done-with-message

Since the consumer can randomly access the contents of a message, there is no implicit way for the slave to know when the consumer is done processing the message. Thus, during any request to access the current message (input or output), the OCP MReqInfo[0] signal is used to indicate whether that access represents the last access required to complete the processing (input or output) of the message. This indicates, for input, that the message can be discarded and the buffer reused, and for output, that the buffer can be made available to transmit to the directly (locally) or indirectly (remote) consuming worker or output device ("downstream").

Since the worker may not know that it is done processing a message until after it has performed all required data accesses for the contents of a message, there is a provision for making a request that accesses no data at all, but simply allows the done-with-message indication with no overhead for accessing memory. The exact semantics are discussed below. This indication is the same whether producing or consuming.

This scheme enables two optimizations: no *extra access* (OCP transaction) is required to complete the message, and no *data delay* is required to wait until it is known that an access will be the last one. The mechanism is identical for consuming and producing.

7.2.4 Indication of buffer available

When the worker is "between" messages (before the first one, or after indicating "done" with the previously processed one), the OCP SThreadBusy signal acts as the indication that a buffer is available, for both consuming and producing. This indication is the same whether producing or consuming. It uses OCP semantics directly.

7.2.5 Message content addressing

WMI supports the worker, as OCP master, accessing message contents via random access reads and writes. Address 0 is always the first word of the "current" message. There is an additional "talkback" option, which allows the worker to write to input buffers or read from output buffers. This option enables read/write access to buffers (rather than read-only input buffers and write-only output buffers), with the associated hardware cost, but can reduce overall buffer requirements.

7.3 WMI Worker Attributes

Worker authors using a WMI specify their implementation choices via the following attributes (as well as the data protocol attributes above), from which all OCP configuration parameters and signals are derived. Light green shading of the description indicates where the attribute has the same meaning as in WSI.

The Clock and ClockSignal attributes, described in section 2.6, are applicable to WMI.

Since WMI is a data plane interface, the worker data interface protocol attributes are applicable, as described in section 4.2. These are: DataValueWidth, DataValueGranularity, DiverseDataSizes, MaxMessageValues, NumberOfOpcodes, Producer, VariableMessageLength, and ZeroLengthMessages.

7.3.1 WMI Attribute Summary

Table 18 – WMI Attribute Table

Attribute Name	Type (per XML Schema)	Default Value	Description
Continuous	*boolean*	*false*	*TRUE: within a burst, there shall be no idle cycles.* *FALSE: there may be idle cycles within a burst*
DataWidth	*unsigned Int*	*DataValue-Width*	*Physical width in bit/wires of data path*
ByteWidth	*unsigned Int*	*DataWidth*	*Set to non-default to have byte enables for writes to buffers*
ImpreciseBurst	*boolean*	*false*	*TRUE: Imprecise OCP bursts will be used* *FALSE: Imprecise OCP bursts will not be used*
PreciseBurst	*boolean*	*false*	*TRUE: Precise OCP bursts will be used* *FALSE: Precise OCP bursts will not be used*
TalkBack	*boolean*	*false*	*TRUE: Wants/needs to write input or read back output buffers* *FALSE: Will only read input or write output buffers*

7.3.2 WMI Worker Attribute Details

This subsection contains the per-attribute details of choices available for a WMI on a worker.

Continuous	*boolean*	*false*	*TRUE: within a burst, there shall be no idle cycles.* *FALSE: there may be idle cycles within a burst*

See WSI Attribute Details.

DataWidth	*unsigned Int*	*DataValue-Width*	*Physical width in bit/wires of data path (same as WSI)*

See WSI Attribute Details.

ByteWidth	*unsigned Int*	*DataWidth*	*Set to non-default to have byte enables for writes to buffers*

ByteWidth expresses the size in bits of the smallest data value that the interface needs to handle, and is used to enable byte enable functionality in WSI, WMI, and WMemI.

If the WMI, when it is writing to buffers, requires byte enable support, set ByteWidth to a value that divides into DataWidth. Normally this would be DataValueWidth, but that is not required. Note that byte enables in WMI are never needed to express message lengths and are not used for reads.

ImpreciseBurst	*boolean*	*false*	*TRUE: Imprecise OCP bursts will be used* *FALSE: Imprecise OCP bursts will not be used*
PreciseBurst	*boolean*	*false*	*TRUE: Precise OCP bursts will be used* *FALSE: Precise OCP bursts will not be used*

See WSI Attribute Details.

For WMI:

- There is no relationship between bursts and messages.
- One, but not both, of these attributes must be TRUE (same as WSI).
- Maximum message length determines maximum burst length, but bursts can be used to access any subset of the current message.

TalkBack	*boolean*	*false*	*TRUE: Wants/needs to write input or read back output buffers* *FALSE: Will only read input or write output buffers*

When TRUE, this attribute configures a particular WMI interface to have both read and write data capability. For a consumer, this attribute enables write-back to the input buffer, in addition to the default message read capability. For a producer, this attribute enables read-back from the output buffer, in addition to the default message write capability. When FALSE, input buffers can only be read, and output buffers can only be written.

This capability can obviate the need for workers to have extra internal buffers or use external memories.

7.4 WMI OCP Profile/Configuration Parameters

The WMI OCP profile is the aspect of the WMI that has to do with OCP configuration parameters and signal configuration. This section describes the concrete, complete, and unambiguous transformation from WMI WIP attributes described above, to the OCP configuration parameters inferred from those attributes. This transformation is implemented in automation tools.

The table below lists only those parameters that are different from the OCP configuration parameter defaults (per the OCP specification).

1575 **Table 19 – Worker Message Interface OCP Configuration Parameters**

OCP Configuration Parameter	Value	Description (How OCP parameter is determined if not constant, and any tieoffs)
addr_wdth	*{M}*	*N = ceil(MaxMessageValues*DataValueWidth/DataWidth)* *If (N <= 1) M = 0;* *Else M = ceil(log2(N))) + max(0,ceil(log2(OCP data_wdth))-3)*
addr_space, addrspace_wdth	*1*	
burstlength	*1*	
burstlength_wdth	*{P}*	*If PreciseBurst = TRUE:* *N = ceil(MaxMessageValues*DataValueWidth/DataWidth)* *If (N < 4) P = 2* *Else P = floor(log_2(N)) + 1* *Else:* *P = 2*
PreciseBurst	*0*	*If ImpreciseBurst = TRUE, tieoff MPreciseBurst = 0 (OCP default tieoff is 1)*
cmdaccept	*0*	*Note: The SThreadBusy signal is used everywhere for backpressure.*
datahandshake	*1*	
datalast	*1*	
data_wdth	*{Q}*	*If ByteWidth != DataWidth and ByteWidth != 8:* *Q = 8 * DataWidth/ByteWidth* *Else:* *Q = DataWidth*
mdatabyteen	*{0:1}*	*If (Producer = TRUE or Talkback = TRUE) and ByteWidth != DataWidth: 1, Else: 0*
mdata	*{0|1}*	*If Producer = TRUE or TalkBack = TRUE: 1, Else: 0*
mdatainfo	*{0|1}*	*If ByteWidth != DataWidth and ByteWidth != 8: 1* *Else: 0*
mdatainfo_wdth	*{R}*	*If ByteWidth != DataWidth and ByteWidth != 8:* *R = DataWidth – (8 * DataWidth/ByteWidth)*
mdatainfobyte_wdth	*{S}*	*If ByteWidth != DataWidth and ByteWidth != 8:* *S = ByteWidth - 8*
mflag	*1*	*If Producer = TRUE and (NumberOfOpcodes>1 or VariableMessageLength = TRUE): 1*
mflag_wdth	*{T}*	*If Producer = TRUE and (NumberOfOpcodes>1 or VariableMessageLength = TRUE): 1* *T = 8 + ceil(log2(MaxMessageValues))*
mreset	*1*	
read_enable	*{0|1}*	*If Producer = FALSE or TalkBack = TRUE: 1, Else: 0*
reqlast	*1*	
resp	*{0|1}*	*If Producer = FALSE or TalkBack = TRUE: 1, Else: 0*
sdata	*{0|1}*	*If Producer = FALSE or TalkBack = TRUE: 1, Else: 0*
sreset	*1*	
reqinfo	*1*	
reqinfo_wdth	*1*	
sflag	*1*	*If Producer = FALSE and (NumberOfOpcodes > 1 or VariableMessageLength = TRUE): 1*
sflag_wdth	*{U}*	*If Producer = FALSE and (NumberOfOpcodes > 1 or VariableMessageLength = TRUE):* *U = 8 + ceil(log2(MaxMessageValues))*
sthreadbusy *sthreadbusy_exact* *sthreadbusy_pipelined*	*1*	*Enables the SThreadBusy signal to support **Request** flow-control backpressure from slave to master.*
sdatathreadbusy *sdatathreadbusy_exact* *sdatathreadbusy_pipelined*	*{0|1}*	*If Producer = TRUE or TalkBack = TRUE: 1* *Else: 0*
write_enable	*{1|0}*	*If Producer = TRUE or TalkBack = TRUE: 1, Else: 0*

1576 ## *7.5 WMI OCP Signals*

Signal configuration is determined by the interface configuration parameters and as called out in the “OCP Signal Configuration Parameter” table in the OCP specification. The tables below show the signal from master-to-slave and slave-to-master. The positional order of signals is alphabetical.

Light yellow shading of the description indicates where an OCP signal has had WMI semantics added. Details described in following section.

1583 **Table 20 – WMI OCP Signals Driven by the Worker/Master**

OCP Signal	When Included	Width	Usage
MAddr	*always*	*See addr_wdth parameter above*	*Current message access address when MAddrSpace='1'*
MAddrSpace	*always*	*1*	*WMI Semantics Added: 0= Normal Transaction 1= No Data access for this command*
MBurstLength	*always*	*See burstlength_wdth parameter above*	*Standard OCP burst access: Constant burst length for Precise B'01' for last request for Imprecise*
MCmd	*always*	*3*	*Only one bit is not constant unless TalkBack= TRUE*
MData	*If Producer = TRUE or TalkBack = TRUE*	*If ByteWidth != DataWidth and ByteWidth != 8: 8 *DataWidth/ByteWidth Else: DataWidth*	*Write Data included when Producer=TRUE or MessageTalkBack = TRUE*
MDataByteEn	*If (Producer = TRUE or TalkBack = TRUE) and ByteWidth != DataWidth*	*DataWidth/ ByteWidth*	*Qualify bytes in writes to current buffer*
MDataInfo	*If (Producer = TRUE or TalkBack = TRUE) and ByteWidth != DataWidth and ByteWidth != 8*	*DataWidth – (8 * DataWidth/ByteWidth)*	*Extra write data not allowed in MData per OCP*
MDataLast	*always*	*1*	*Indicates last word of data in burst*
MDataValid	*always*	*1*	*Qualifies active write-data*
MFlag	*If Producer = TRUE and (NumberOfOpcodes>1 or VariableMessageLength = TRUE)*	*8 + ceil(log2(MaxMessageValues+1))*	*WMI Specific Semantics: Opcode and Message Length*
MReqInfo	*always*	*1*	*WMI Semantics Added: Bit-0 = Done-with-Message (DWM) prevents SThreadBusy = FALSE*
MReqLast	*always*	*1*	
MReset_n	*always*	*1*	*0 = Reset Master-to-Slave. (Asserted in response to WCI reset)*

1584 **Table 21 – Worker Message Interface Signals Driven by the Infrastructure/Slave**

OCP Signal	When Included	Width	Usage
SData	*If Producer = FALSE or TalkBack = TRUE*	*DataWidth*	*Read Data*
SDataThreadBusy	*If Producer = TRUE or TalkBack = TRUE*	*1*	*Allows slave to provide pipelined, proactive write-data* ***Datahandshake*** *backpressure*
SFlag	*If Producer = FALSE and (NumberOfOpcodes > 1 or VariableMessageLength = TRUE)*	*8 + ceil(log2(MaxMessageValues+1))*	*WMI Specific Semantics: Opcode and Message Length*
SReset_n	*always*	*1*	*0 = Reset Slave-to-Master (Active Low)*

SResp	*If Producer=FALSE or TalkBack = TRUE*	*FIXED*	*Qualifies active read-data transfer words within a message*
SRespLast	*If (ImpreciseBurst or PreciseBurst) and (Producer = FALSE or TalkBack = TRUE)*	*1*	*Indicates last beat of read data in a burst*
SThreadBusy	*always*	*1*	*WMI Specific Semantics: When between messages, when FALSE, it signifies: New Message Available*

7.6 WMI Semantics

This section describes specific semantic signal meanings in WMI that are layered on top of the fundamental OCP signals. They are organized by function and then the signals affected.

7.6.1 "Done with Message" (DWM) Indicated on MReqInfo[0]

MReqInfo[0] adds information to the current OCP command request (to read or write message contents) to indicate that the data access associated with this command is the final access of the associated message. It is sent in this fashion to eliminate the need for an extra OCP request to indicate this.

This mechanism is used for both consuming and producing workers. The associated data access can be suppressed (possibly reducing latency) by using the no-data indication via MAddrSpace[0] described next.

7.6.2 No-Data Indication on MAddrSpace[0]

WMI transactions initiated by a worker that include the DWM indication, either consumer or producer, may also indicate that the OCP command, read or write, has "No Data" by setting the MAddrSpace[0] signal to '1'. This allows a master to send to the slave the Done-with-Message (DWM) on MReqInfo[0] without any read or write side effects or overheads. Thus when the worker decides it is done with the current message, but requires no additional data accesses, it can issue the DWM request with no data access required by the infrastructure.

Normal transactions to read and write "Data" require that MAddrSpace be set to '0'. Setting MAddrSpace[0] = 1 when MReqInfo[0] = 0 should not be done.

7.6.3 Buffer Availability Indicated on SThreadBusy

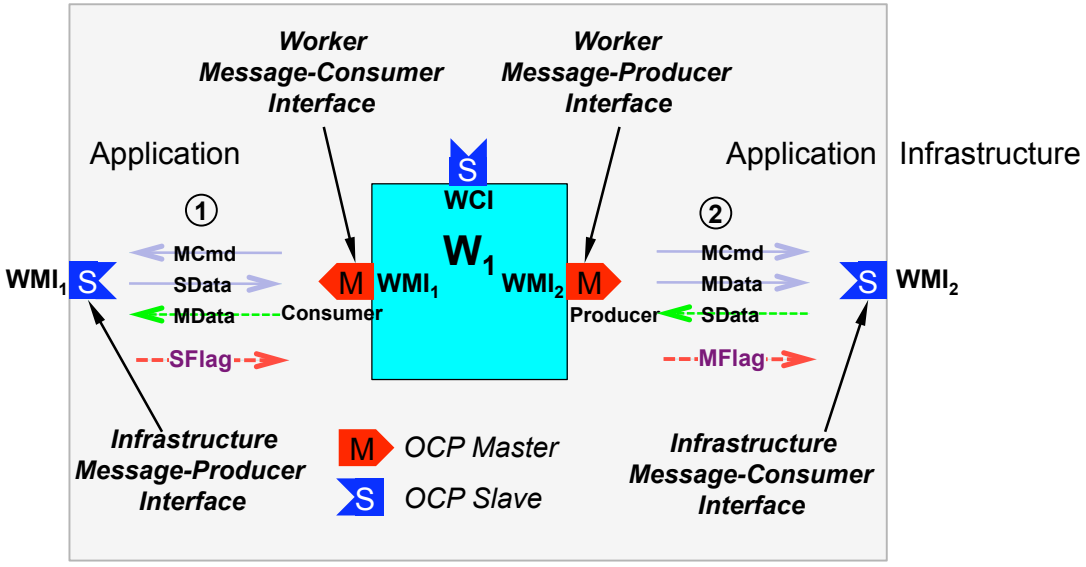
When between messages (either after reset or after the master indicates "done with message"), SThreadBusy is used to indicate the availability of a source data buffer to read (worker consumer) or an output data buffer to write (worker producer). An asserted SThreadBusy[0] signal blocks the worker from issuing an OCP request. When not between messages, SThreadBusy is simply the backpressure for reading or writing message data. For a consumer, the infrastructure side (slave) may allow access as data arrives (for improved latency), and thus latency could depend on the address of that first access to a (new) message.

As an additional constraint on OCP, the master *cannot* assert a request in the cycle after the DWM is asserted, in order to allow the slave to assert the (pipelined)

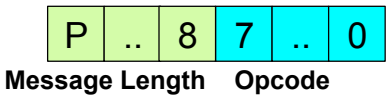
SThreadBusy for at least one cycle between messages. Thus the worker must not issue a request based on the normal pipelined SThreadBusy from the slave *or* the MReqInfo[0] in the previous cycle. For the purpose of this non-blocking flow control, both signals are sampled at the same clock: at the end of the cycle before the request is to be asserted.

7.6.4 Opcode and Message Size Move Downstream on {M|S}Flag

Opcodes and message size flow downstream. They are driven together on MFlag and SFlag as shown in the following diagram. A worker message-consumer *observes* these on SFlag and a worker message-producer *drives* them on MFlag. They are only configured when NumberOfOpcodes > 1 or VariableMessageLength = TRUE.



Encoding of Opcode and Message Length on both SFlag and Mflag:



P = Number of Message Length bits + 7

Figure 7 WMI Use of SFlag/MFlag

When the field is configured, the opcode is driven on {S|M}Flag[7:0]. When VariableMessageLength = TRUE the message length is driven on {S|M}Flag[P:8], where P = ceil(log2(MaxMessageValues+1)) + 7.

For a worker *consumer*, this information is driven on SFlag by the infrastructure and the values will be valid for a given message starting in the first cycle when SThreadBusy is not asserted (after reset or the previous DWM), through the cycle when the worker asserts DWM (MReqInfo[0] = 1). Thus the worker may not need to capture them.

For a worker *producer*, this information is driven on MFlag by the worker and the values must be valid in the (single) same cycle as the request with DWM asserted.

7.7 WMI Infrastructure Coping

The key options chosen by the worker for WMI are data width, burst capabilities, and, like WSI, the continuous attribute. Some of these attributes allow for infrastructure optimization, such as when the TalkBack option is FALSE, more buffer sharing is possible (multiple consumers could share the same input buffer).

Table 22 – WMI Slave Infrastructure Handling of Optional WMI Master Signals

OCP Signal	When Signal would be absent at the master	How missing signals are handled
MData	*If Producer = FALSE and TalkBack = FALSE*	*No issue - infrastructure is prepared for TalkBack for a consumer, but worker doesn't use it. OCP tieoff rules apply, signals will never be driven.*
MDataByteEn	*If (ByteWidth = DataWidth or Producer = FALSE and Talkback = FALSE)*	*OCP Tieoff rules apply: No connection is made, MDataByteEn connected to the default tieoff: all 1s (indicating full words)*
MFlag	*NumberOfOpcodes=1 and VariableMessageLength = FALSE*	*OCP Tieoff rules apply: No connection is made, MFlag tied to the default tieof: 0 (indicating opcode == 0). Infrastructure slaves should generally support 256 opcodes (8 bits). Fixed message length would be the tieoff for bits [P:8]*

7.7.1 Infrastructure Best Practice for Slave Producers (feeding Worker Consumers)

- Support CONTINUOUS when the buffer memory can support it.
- Support both precise and imprecise bursts, use MReqLast rather than MBurstLength.
- Support Byte Enables on Writes.
- Support VariableMessageLength = TRUE.
- Support 8 bits of opcode.
- Have a variant IP to support TalkBack (since it might be costly)

7.7.2 Infrastructure Best Practice for Slave Consumers (fed by Worker Producers)

- Support CONTINUOUS when the buffer memory can support it.
- Support both precise and imprecise bursts, use MReqLast rather than MBurstLength.
- Support Byte Enables on Writes.
- Support VariableMessageLength = TRUE.
- Support 8 bits of opcode.
- Have a variant IP to support TalkBack (since it might be costly)

[Issue: packing non-power-of-two for DMA: map each byte to round up to 2^N?]

8 Worker Memory Interface (WMemI)

8.1 WMemI Motivation

WMemI is an interface that allows workers to access memory in a technology independent fashion, with enough choices to make workers simple and also allow for high performance use of local memories. It exists to enable workers to express their needs for FPGA-attached (or FPGA-internal) memory based on their own requirements and preferences. Since explicitly buffered data plane communication is handled by WMI, WMemI is focused on memory usage for non-communication purposes such as internal history buffers and LUTs.

Since memory usage ranges from latency-optimized non-burst LUT-style random word access, to throughput optimized burst-style sequential access, the WMemI attribute choices enable this range of behavior.

The intended purpose of WMemI is *not* for a worker to communicate via memory to any other worker, since that is the job of WSI or WMI, which may have underlying memory used for buffering. An example of this is shown earlier in the WMI section describing master-to-master adaptation.

8.2 WMemI Overview

Typical FPGA-attached memory scenarios include both SRAM and DRAM[4]. WMemI balances between abstracting enough away so that workers may be portable across heterogeneous memory architectures; vs. exposing enough of the micro-architectural details when “ultimate performance” is required.

Workers may have zero or more WMemIs, each with their own attributes.

The figure below shows in isolation a (WMemI) interface (1) between Worker1 and MemController1. The worker specifies the WMemI attributes it desires at master interface (1M), such as data width and memory capacity. The infrastructure satisfies those requests on the other side on slave interface (1S). The worker side of a WMemI is always the OCP master.

[4] This memory might be “on-chip”, such as rich memory (e.g. BRAM or MRAM) resource found on contemporary FPGAs. Or it may, most commonly, be “FPGA- attached”, such as a SRAM or DRAM controller connected to a FPGA. It could also be a remote memory resource. Thus infrastructure can supply the type of memory that is available and appropriate, without the application worker making the choice explicit.

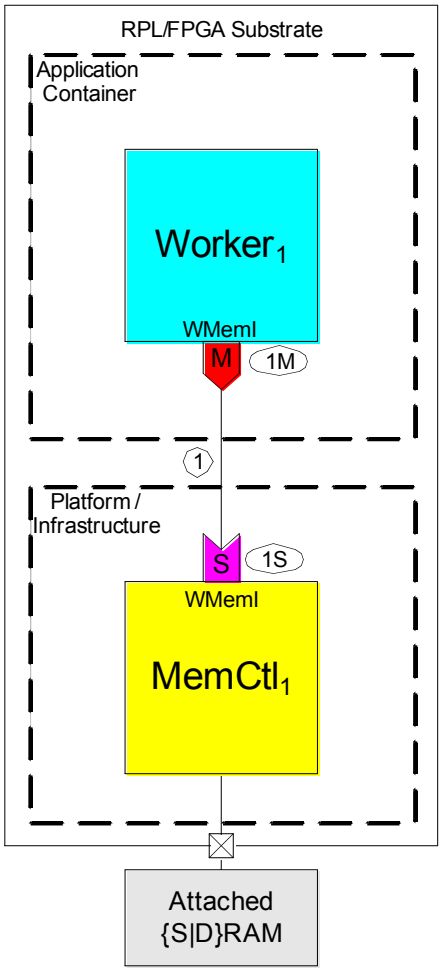


Figure 8 - Master/Slave Roles for WMemI

The non-worker side of a WMemI interface does not have to be a direct connection to a memory controller; it could be other logic, such as a memory multiplexer that provides multiple memory views to multiple workers as shown below:

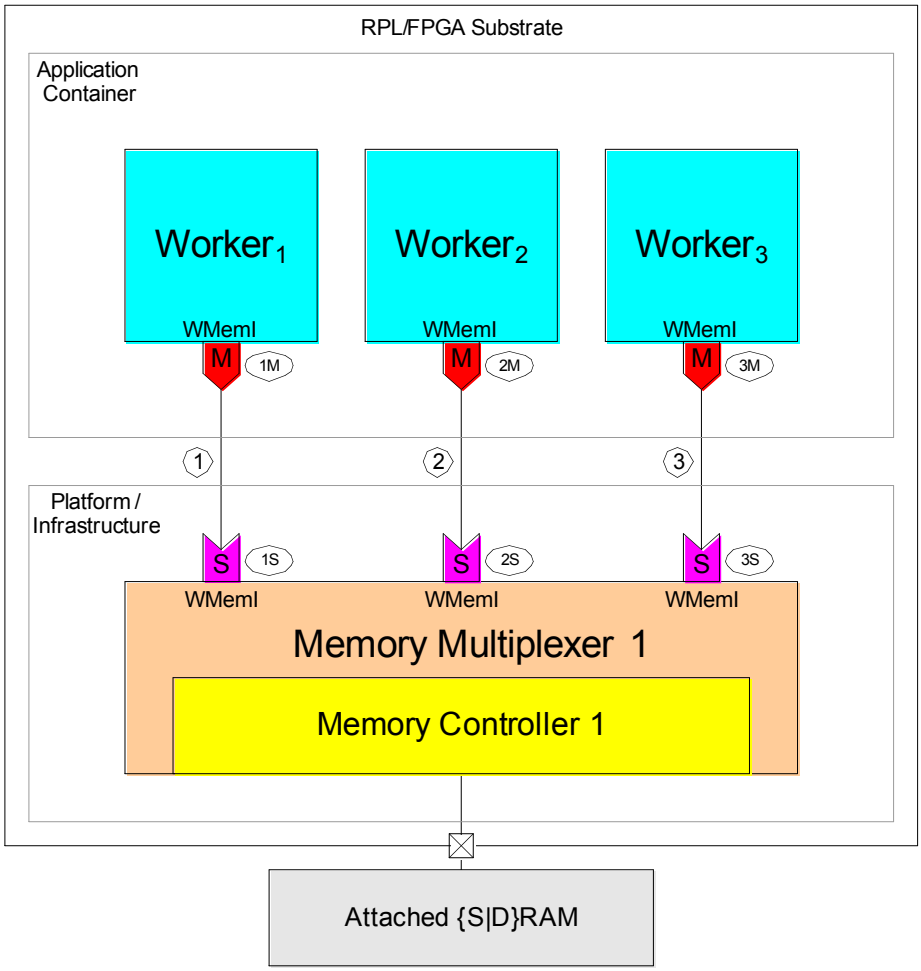


Figure 9 - Memory Multiplexer with WMemI

Each of the workers shown above may have different FPGA-attached memory requirements, and would express those as unique WMemI attributes associated with each WMemI. Because of these differences the three WMemI interfaces may have different OCP configuration parameters. Thus, in the general case, the worker masters (1M, 2M, 3M) are loosely constrained for data-width and capacity; and the infrastructure slaves (1S, 2S, 3S) must conform or be adapted to them as the worker attributes are generating the OCP configuration parameters. Furthermore, the memory multiplexer must adapt these potentially disparate requests to a common access method for the memory controller.

The most significant choice made in specifying a WMemI is the use of bursts, which adds complexity to the interface (e.g. pipelined requests) and determines several significant differences in signal configuration (e.g. how byte enables are specified).

The key choices made when a worker specifies a WMemI are:

- ***Burst usage***: does the worker want burst-style (throughput optimized) access or SRAM-style latency-optimized single word access? How long will bursts be?
- ***Burst style***: will the worker issue precise or imprecise bursts? Or *both*?
- ***Intra-burst flow control***: will the worker handle flow control within a burst (after it is started)? Will it use (issue) flow control against the read data in a burst read?
- ***Memory width and size***: how much memory is required, and how wide should it be? Should it support bytes of some width (byte enables on writes)?

WMemI is specified using blocking request flow control (where the master issues the request and waits for it to be accepted) rather than the non-blocking flow control used in the other profiles (where the master waits for the not-busy indication before it can issue the request). This is because memory behavior (and the delay in processing requests) frequently depends on the address associated with the request (i.e. same bank/row or not).

8.3 WMemI Worker Attributes

Worker authors using a WMemI specify their implementation choices via the following attributes, from which all OCP configuration parameters and signals are derived. Light green shading of the description indicates where the attribute has the same meaning as in WMI.

The Clock and ClockSignal attributes, described in section 2.6, are applicable to WMemI.

8.3.1 WMemI Attribute Summary

Table 23 – WMemI Worker Canonical Attribute Table

Attribute Name	Type (per XML Schema)	Default value	Description (OCP configuration parameters affected by this choice)
DataWidth	*unsignedInt*	*8*	*Physical width in bits/wires of data path similar to as WSI/WMI, must be a multiple of ByteWidth*
ByteWidth	*unsignedInt*	*8*	*Number of bits controlled by a write-byte-enable signal, must divide evenly into Data Width. If same as DataWidth, no byte enables are configured.*
ImpreciseBurst	*boolean*	*false*	*TRUE: Imprecise OCP bursts will be used FALSE: Imprecise OCP bursts will not be used*
PreciseBurst	*boolean*	*false*	*TRUE: Precise OCP bursts will be used FALSE: Precise OCP bursts will not be used*
MemoryWords	*unsignedInt*	*none*	*The amount of data memory (in words of DataWidth size) that are available, starting at address zero.*
MaxBurstLength	*unsignedInt*	*none*	*Maximum burst length, if either ImpreciseBurst or PreciseBurst is TRUE.*
WriteDataFlowControl	*boolean*	*false*	*TRUE: worker/master supports Write Data Flow Control from memory/slave during write bursts*
ReadDataFlowControl	*boolean*	*false*	*TRUE: worker/master requires memory/slave to support Read Data Flow Control during bursts*

8.3.2 WMemI Attribute Details

This subsection contains the per-attribute details of choices available for a WMemI on a worker.

ImpreciseBurst	*boolean*	*false*	*TRUE: Imprecise OCP bursts will be used FALSE: Imprecise OCP bursts will not be used*
PreciseBurst	*boolean*	*false*	*TRUE: Precise OCP bursts will be used FALSE: Precise OCP bursts will not be used*
MaxBurstLength	*unsignedInt*	*1*	*Maximum burst length, if either ImpreciseBurst or PreciseBurst are TRUE.*

These attributes are as described under the WMI profile, except that it is allowed to configure a WMemI such that the master (worker) may issue ***both*** precise and imprecise bursts, or neither. MaxBurstLength (only for WMemI) is the maximum number of words that will be transmitted in a burst. (WMI infers this from the maximum message length).

DataWidth	*unsignedInt*	*8*	*Physical width in bits/wires of data path similar to as WSI/WMI, must be a multiple of ByteWidth*
ByteWidth	*unsignedInt*	*8*	*Number of bits controlled by a write-byte-enable signal, must divide evenly into Data Width and be >= 8*
MemoryWords	*unsignedInt*	*none*	*The amount of data memory (in words of DataWidth size) that are available, starting at address zero.*

DataWidth and ByteWidth are as in WSI and WMI (data path width in bits). ByteWidth expresses the byte enable capabilities for writes. There are no byte enable capabilities in WSI, WMI or WMemI for reads. Memory is sized in words, with MemoryWords expressing the amount of memory available (or required), starting at address zero. MemoryWords implies the width of the address path. When ByteWidth equals DataWidth, no byte enables are used.

WriteDataFlowControl	*boolean*	*false*	*TRUE: worker/master supports Write Data Flow Control from memory/slave during write bursts*
ReadDataFlowControl	*boolean*	*false*	*TRUE: worker/master requires memory/slave to support Read Data Flow Control during bursts*

The WriteDataFlowControl attribute indicates if the worker will support and accept intra-burst flow control on memory writes. The default (FALSE) indicates that the worker cannot support any intra-burst backpressure, and that once a burst requests is accepted, all the data in the burst will be accepted. Only valid when bursts are used.

The ReadDataFlowControl attribute indicates if the worker will assert intra-burst flow control on burst reads. The default (FALSE) indicates that the worker will not assert any intra-burst backpressure, and that it is prepared to accept all the data it requests. Only valid when bursts are used.

8.4 WMemI OCP Profile/Configuration Parameters

The WMemI OCP profile is the aspect of the WMemI that has to do with OCP configuration parameters and signal configuration. This section describes the concrete, complete, and unambiguous transformation from WMemI WIP attributes described above, to the OCP configuration parameters inferred from those attributes. This transformation is implemented in automation tools.

The table below lists only those parameters that are different from the OCP configuration parameter defaults (per the OCP specification).

1760 **Table 24 – Worker Memory Interface OCP Configuration Parameter Settings**

OCP Configuration Parameter	Value	Description (How OCP parameter is determined if not constant, and any tieoffs)
addr_wdth	*{M}*	*M = ceil(log2(MemorWords)) + ceil(log2(DataWidth/ByteWidth))*
burstlength	*{0|1}*	*If PreciseBurst = TRUE or ImpreciseBurst = TRUE: 1* *Else: 0*
burstlength_wdth	*{N}*	*If PreciseBurst = TRUE:* *N = floor(log₂(max(2,MaxBurstLength))) + 1* *Else If ImpreciseBurst = TRUE:* *N = 2*
burstprecise	*{0|1}*	*If PreciseBurst = TRUE and ImpreciseBurst = TRUE: 1* *Else: 0* *If PreciseBurst = FALSE and ImpreciseBurst = TRUE: {tie_off 0}* *(note OCP default tieoff is 1)*
burstsinglereq	*{0|1}*	*If PreciseBurst = TRUE and ImpreciseBurst = TRUE: 1* *Else: 0* *If PrecisesBurst = TRUE and ImpreciseBurst = FALSE: {tie_off 1}* *(note OCP default tieoff is 0)*
dataaccept	*{0|1}*	*If (PreciseBurst = TRUE or ImpreciseBurst = TRUE) and* *WriteDataFlowControl = TRUE: 1, Else: 0*
datahandshake	*{0|1}*	*If PreciseBurst = TRUE or ImpreciseBurst = TRUE: 1, Else: 0*
datalast	*{0|1}*	*If PreciseBurst = TRUE or ImpreciseBurst = TRUE: 1, Else: 0*
data_wdth	*N*	*If ByteWidth = DataWidth or ByteWidth = 8 (no bytes or bytes=octets)* *N = DataWidth* *Else:* *N = DataWidth/ByteWidth * 8*
mbyteen	*1*	*If PreciseBurst = FALSE and ImpreciseBurst = FALSE and* *ByteWidth != DataWidth: 1, Else: 0*
mdatabyteen	*1*	*If (PreciseBurst = TRUE or ImpreciseBurst = TRUE) and* *ByteWidth != DataWidth: 1, Else: 0*
mdatainfo	*{0|1}*	*If ByteWidth != DataWidth and ByteWidth != 8: 1, Else: 0*
mdatainfo_wdth	*{Q}*	*If ByteWidth != DataWidth and ByteWidth != 8:* *Q = DataWidth – (8 * DataWidth/ByteWidth)*
mdatainfobyte_wdth	*{R}*	*If ByteWidth != DataWidth and ByteWidth != 8:* *R = ByteWidth - 8*
mreset	*1*	
respaccept	*{0|1}*	*If (PreciseBurst = TRUE or ImpreciseBurst = TRUE) and* *ReadDataFlowControl=TRUE: 1, Else: 0*
reqlast	*{0|1}*	*If PreciseBurst = TRUE or ImpreciseBurst = TRUE: 1, Else: 0*
resplast	*{0|1}*	*If PreciseBurst = TRUE or ImpreciseBurst = TRUE: 1, Else: 0*
sdatainfo	*{0|1}*	*If ByteWidth != DataWidth and ByteWidth != 8: 1, Else: 0*
sdatainfo_wdth	*{S}*	*If ByteWidth != DataWidth and ByteWidth != 8:* *S = DataWidth – (8 * DataWidth/ByteWidth)*
sdatainfobyte_wdth	*{T}*	*If ByteWidth != DataWidth and ByteWidth != 8:* *T = ByteWidth - 8*

1761 ## *8.5 WMemI OCP Signals*

1762 Signal configuration is determined by the interface configuration parameters and as
1763 called out in the “OCP Signal Configuration Parameter” table in the OCP specification.
1764 The tables below show the signals from master-to-slave and slave-to-master. The
1765 positional order of signals is alphabetical.

1766 **Table 25 – Worker Memory Interface Signals Driven by the Master, to the Slave**

OCP Signal	When Included	Width	Usage
MAddr	*always*	*ceil(log2(MemorWords)) + ceil(log2(DataWidth/ByteWidth))*	*Memory Address, with byte address lines always 0*
MBurstLength	*If PreciseBurst or ImpreciseBurst*	*If PreciseBurst: floor(log2(max(2, MaxBurstLength))) + 1 Else If ImpreciseBurst: 2*	*Burst Length*
MPreciseBurst	*If PreciseBurst and ImpreciseBurst*	*1*	*Indicates precise vs imprecise bursts.*
MBurstSingleReq	*If PreciseBurst and ImpreciseBurst*	*1*	*Indicates Single request for all data transfers in the burst*
MCmd	*always*	*3*	*Transfer Command, READ or WRITE*
MData	*always*	*If ByteWidth != DataWidth and ByteWidth != 8: 8 *DataWidth/ByteWidth Else: DataWidth*	*Write Data*
MByteEn	*!PreciseBurst and !ImpreciseBurst and ByteWidth != DataWidth*	*DataWidth/ByteWidth*	*Per-lane write byte enable when bursts and pipelining are not enabled: a request phase signal*
MDataByteEn	*(PreciseBurst or ImpreciseBurst) and ByteWidth != DataWidth*	*DataWidth/ByteWidth*	*Per-lane write byte enable when bursts and pipelining are enabled: a data handshake phase signal*
MDataInfo	*If ByteWidth != DataWidth and ByteWidth != 8*	*DataWidth – (8 * DataWidth/ByteWidth)*	*Extra write data not allowed in MData per OCP*
MDataLast	*PreciseBurst or ImpreciseBurst*	*1*	*Indicates last write data in a burst*
MReqLast	*PreciseBurst or ImpreciseBurst*	*1*	*Indicates last request in a burst*
MDataValid	*PreciseBurst or ImpreciseBurst*	*1*	*Qualifies active write-data within bursts*
MReset_n	*always*	*1*	*Reset Master-to-Slave. (Asserted in response to WCI reset asserted)*
MRespAccept	*(PreciseBurst or ImpreciseBurst) and ReadDataFlowControl*	*1*	*Indicates master has accepted the read data. Allows master to throttle read data it has requested.*

1767 **Table 26 – Worker Memory Interface Signals Driven by the Slave, to the Master**

OCP Signal	When Included	Width	Usage
SCmdAccept	*always*	*1*	*Indicates slave has accepted the transfer command*
SData	*always*	*If ByteWidth != DataWidth and ByteWidth != 8: 8 *DataWidth/ByteWidth Else: DataWidth*	*Read Data*
SDataAccept	*(PreciseBurst or ImpreciseBurst) and WriteData-FlowControl*	*1*	*Indicates slave has accepted write data word.*
SDataInfo	*If ByteWidth != DataWidth and ByteWidth != 8*	*DataWidth – (8 * DataWidth/ByteWidth)*	*Extra read data not allowed in SData per OCP*
SResp	*always*	*1*	*Qualifies active read-data words*

SRespLast	*PreciseBurst or ImpreciseBurst*	*1*	*Indicates last beat of read data in a burst*

8.6 WMemI Semantics

This section describes specific semantic signal meanings in WMemI that are layered on top of the fundamental OCP signals. They are organized by function and then the signals affected.

There are no special semantics with WMemI, but there are additional restrictions.

8.6.1 Two basic modes of WMemI: bursts or not

When neither type of burst is enabled by the WMemI attributes, the interface is simplified to have no data handshake phase, and thus no pipelining of requests before write data. The worker always presents data with the request, doesn't deal with bursts, and uses the MByteEn signals for write byte enables.

When either type of burst is enabled by the WMemI attributes, the interface is configured with pipelined requests, bursts, and the (data handshake phase) MDataByteEn signals are used for write byte enables.

8.6.2 Use only Single-Request Multiple-Data for Precise Bursts

When the interface supports precise bursts, they must be issued with a single request, which is the semantic when the MBurstSingleReq signal is true (or tied true). Thus WMemI masters must not issue precise bursts with multiple requests. When the interface supports imprecise bursts, OCP requires using multiple-request-multiple-data bursts. When both precise bursts and imprecise bursts are supported, the MBurstSingleReq signal is configured (as well as the MPreciseBurst signal) and they both always have the same value: if the burst is precise, a single request is used, and if the burst is imprecise, multiple requests are used.

8.6.3 Intra-burst write data flow control

With WMemI, the SCmdAccept signal is always enabled and used to accept requests. The WriteDataFlowControl attribute, which can only be TRUE when bursts are used, enables the dataaccept OCP configuration parameter and thus the SDataAccept signal. In this case every data word asserted with MDataValid, must be held until accepted via the SDataAccept signal being asserted by the slave.

When WriteDataFlowControl is FALSE, the dataaccept parameter is zero, and the MDataAccept signal is not in the interface. In this case, precise and imprecise bursts may work differently since imprecise may throttle write data via SCmdAccept within the burst.

Regardless of the setting of WriteDataFlowControl, if the first write data word in the burst (first data handshake phase in the burst) is presented (via MDataValid) *before* SCmdAccept is asserted (which indicates the end of the first request phase), that data (and the associated MDataValid) must be held constant until SCmdAccept is asserted.

8.6.3.1 Precise Bursts with no Write Flow Control.

After SCmdAccept is asserted for the single request for the precise burst, any data words are simply presented in a single cycle with MDataValid asserted, and the slave must/will capture the data. Masters issuing precise bursts when WriteDataFlowControl is FALSE are assured that all data asserted after the cycle in which SCmdAccept is asserted will be accepted by the slave immediately.

8.6.3.2 Imprecise Bursts with no Write Flow Control.

In WMemI imprecise bursts use multiple requests for the burst. Thus, per OCP, the data associated with a request may be asserted starting in the same cycle as the request, but can also be arbitrarily delayed. As mentioned above, data asserted (with MDataValid) before the associated SCmdAccept must be held until SCmdAccept is asserted. Data asserted during or after the associated SCmdAccept assertion must/will be captured by the slave in that cycle.

Masters issuing imprecise bursts must obey the SCmdAccept behavior for every request in the burst. Thus even when WriteDataFlowControl is FALSE, the master must support request flow control throughout the burst.

8.6.4 Intra-burst read data flow control

The WMemI ReadDataFlowControl attribute configures the interface to provide this feature by way of the MRespAccept signal, which causes the slave to hold read data until accepted by the master. This option is only used when bursts are enabled, and is typically only used for precise bursts, since intra-burst read flow control can be accomplished by the worker/master via throttling its own issuance of read requests.

8.7 WMemI Infrastructure Issues

WMemI infrastructure IP acts as a slave to worker interfaces specified with the WMemI attributes above. There are three significant issues in supporting worker choices: burst support (implying datahandshake), intra-burst flow control (for precise bursts), and data/byte widths.

8.7.1 Burst support

If memory IP supports both precise bursts and imprecise bursts then it will be able to satisfy any worker burst behavior. The major challenge/cost is in adding the dynamic behavior to optimize the imprecise case where the controller does not know how much data will be in the burst. In both cases, the memory IP does not know how many intra-burst idle cycles there will be (either in requests or data), and is usually optimized for the continuous case.

When the worker has no burst support, and thus no data handshake phase, tie-offs on the burst-related signals (imprecise burst with burst length of 1) should suffice.

If the memory IP has no burst support, then a slightly more complex adapter is required, to align request and data phases, and, for precise bursts, an address counter/generator is required.

8.7.2 Intra-burst flow control

When a worker does not support write data flow control, all data in the burst must be captured in the cycle it is asserted (via MDataValid), up to the maximum burst size. This may require adding buffering to the memory IP (or adding an adapter with such memory). If the worker implements read data flow control, the memory IP may require extra buffering when it cannot apply backpressure to the underlying memory mechanisms.

When there are burst size mismatches, without flow control, extra buffering is also required (i.e. when the worker writes a burst without flow control, larger than the buffering available in the memory IP).

8.7.3 Data/byte widths

There are many cases of width mismatches, but a common one is when the memory itself has more bits in width than the worker requires or wants to use. The simple case is to simply ignore the extra bits (sometimes the extra bits-per-byte when memory bytes are actually > 8 bits wide). Most Memory IP should work when the worker wants simple power-of-2 memory widths and 8-bit bytes.

9 Worker Time Interface (WTI)

9.1 WTI Motivation

WTI is an interface that allows workers to access the current system time (TOD, “now”). This allows any worker in the system to know what time it is, to some system-defined accuracy (depending on the implementation of that service across the system). Many applications require accurate and precise timestamping, time measurement, and time scheduling. The interface must enable time to be provided as accurately as technically possible (depending on infrastructure capabilities), and be conveniently accessible from worker logic. The interface should also enable the time service to be controlled in order to reduce power when the worker does not need it.

This interface should provide “time” as a service to workers, much like the WMemI provides memory as a service to workers. Implementations of this service are characterized by accuracy across the chip substrate (workers spread across a chip) as well as across chips on a circuit board or across circuit boards in a system or systems with GPS and/or network connections. The interface does not define accuracy, but allows workers to specify precision.

Future versions of this specification may allow workers to indicate required accuracy as a quality of service.

9.2 WTI Overview

The format of the time value made available to the worker is a fixed point binary value corresponding to GPS time. When the worker’s functionality requires access to time, the author specifies, via WIP attributes, the number of binary digits required on both sides of the radix point. With 32 bits to the right of the radix point, the low order bit would represent 0.23283 ns. This format, when using 32 bits on each side of the radix point, is the same as the format used in NTP (Network Time Protocol). GPS is a time scale based on atomic clocks that advances monotonically, unlike UTC or UNIX/POSIX time, which does not (due to handling of leap seconds). GPS time zero started at 00:00:00 UTC on January 6, 1980. Conversion to UTC or UNIX/POSIX requires a table of leap seconds. GPS is chosen rather than UTC or POSIX so that workers will never deal with leap seconds when performing time calculations. There have been 24 leap seconds since 1972.

This (OCP) profile specifies that the infrastructure is the master, and the worker is the slave, with the time constantly “written” from the infrastructure to the worker. Thus the time is constantly and trivially available as long as the worker requests it (via very simple flow control). Workers can ignore flow control and OCP signal tieoff rules will enable time to be always available.

As in all WIP interfaces on a worker, the worker author chooses the clock domain in which the interface operates (the domain of the OCP clock signal associated with that interface). The worker author would typically choose the same clock domain as one of

its data input and/or output interfaces, but if it is specified to be in its own clock domain, it will likely be in a clock domain of the timekeeping infrastructure itself.

The WTI is configured (per OCP) such that the time is delivered via (qualified by) an OCP "write" command on every clock cycle when the time value is valid and the worker allows, via flow control (SThreadBusy), the time value to be presented. This flow control aspect of WTI allows the worker to indicate to the infrastructure when it actually needs to know the time, enabling the infrastructure the possibility to make power optimizations when the worker does not need the time value. The presence of the OCP "write" command indicates valid time. The absence of the OCP "write" command indicates that time is not available (not valid).

So to know time, the worker author specifies a WTI, and the number of data bits on both sides of the radix point.

9.3 WTI Worker Attributes

Worker authors using a WTI specify their implementation choices via the following attributes, from which all OCP configuration parameters and signals are derived.

The Clock and ClockSignal attributes, described in section 2.6, are applicable to WTI.

The default attribute settings provide GPS time in integer seconds.

9.3.1 WTI Attribute Summary

Table 27 – WTI Worker Canonical Attribute Table

Attribute Name	Type (per XML Schema)	Default value	Description (OCP configuration parameters affected by this choice)
SecondsWidth	*unsignedInt*	*32*	*Physical width in bits/wires of of the data path used to convey whole seconds of GPS time.*
FractionWidth	*unsignedInt*	*0*	*Physical width in bits/wires of of the data path used to convey fractional seconds of GPS time. The MSB is the first bit to the right of the radix point.*
AllowUnavailable	*boolean*	*false*	*Indicates, when true, that the worker is prepared to deal with the time value being unavailable.*

9.3.2 WTI Attribute Details

This subsection contains the per-attribute details of choices available for a WTI on a worker.

The two attributes, SecondsWidth, and FractionWidth, specify the number of bits to the left and right, respectively, of the radix point, of the fixed point binary value made available to the worker, on the OCP data signals (MData). The data path width is the sum of these two attributes.

9.4 WTI OCP Profile/Configuration Parameters

The WTI OCP profile is the aspect of the WTI that has to do with OCP configuration parameters and signal configuration. This section describes the concrete, complete, and unambiguous transformation from WTI WIP attributes described above, to the OCP

configuration parameters inferred from those attributes. This transformation is implemented in automation tools.

The table below lists only those parameters that are different from the OCP configuration parameter defaults (per the OCP specification).

Table 28 – Worker Time Interface OCP Configuration Parameter Settings

OCP Configuration Parameter	Value	Description (How OCP parameter is determined if not constant, and any tieoffs)
addr	*0*	
cmdaccept	*0*	
data_wdth	*W*	*W = SecondsWidth + FractionWidth*
mreset	*0*	
read_enable	*0*	
resp	*0*	
sdata	*0*	
sreset	*1*	*Worker must propagate its WCI reset*
sthreadbusy	*1*	*Normal tieoff applies: signal is always deasserted*
sthreadbusy_exact	*1*	*Well defined sthreadbusy semantics per OCP*
sthreadbusy_pipelined	*1*	*SthreadBusy at a clock permits command in next cycle.*

9.5 WTI OCP Signals

Signal configuration is determined by the interface configuration parameters and as called out in the "OCP Signal Configuration Parameter" table in the OCP specification. The tables below show the signals from master-to-slave and slave-to-master. The positional order of signals is alphabetical.

Table 29 – Worker Time Interface Signals Driven by the Master, to the Slave

OCP Signal	When Included	Width	Usage
MCmd	*always*	*3*	*MCmd == WRITE when time is available on MData, else IDLE. IDLE only before "start" or if AllowUnavailabe == true.*
MData	*always*	*SecondsWidth + FractionWidth*	*Time value, fixed point*

Table 30 – Worker Time Interface Signals Driven by the Slave, to the Master

OCP Signal	When Included	Width	Usage
SReset_n	*always*	*1*	*Indicates worker is in reset (from WCI). Must be propagated from WCI's MReset_n, in clock domain of this WTI.*
SThreadBusy	*always*	*1*	*Indicates when deasserted, that worker wants time in next cycle.*

9.6 WTI Semantics

This section describes specific semantic signal meanings in WTI that are layered on top of the fundamental OCP signals. They are organized by function and then the signals affected.

9.6.1 Indicating to the worker when time is valid and available

The MCmd signal field having the value “WRITE”, indicates that the data is valid (on the MData signals), which in the case of WTI indicates when the time value is available and valid. A worker that has set the “AllowUnavailable” attribute to “false” (the default), indicates that it expects time to always be available before becoming operational (before the “start” control operation is issued to the worker via its WCI), and thus can assume this value is always “WRITE” before it receives the “start” control operation.

9.6.2 Indicating to the infrastructure when time is not required

If the worker would like to indicate to the infrastructure when a valid time value is *not* needed, it can assert the SThreadBusy signal. According to the OCP specification, this disallows a WRITE command. In the case of WTI, the infrastructure can use this indication to disable certain timekeeping activities to save power. The OCP tieoff for this signal is deasserted, so it can be ignored for workers that choose not to have such an indication.

9.6.3 The simplest case

With all attributes set to default values, and the worker not caring to indicate when it does not need the timevalue, the worker simple samples the MData signals with the clock indicated for the WTI. SThreadBusy will tie off to deasserted, and MCmd will always have the “WRITE” value. Other than sampling the MData signals, the worker must assert the SReset_n signal (in the WTI’s clock domain), when the MReset_n signal is asserted on the worker’s WCI.

9.7 WTI Infrastructure Issues

WTI infrastructure IP acts as a master to worker interfaces specified with the WTI attributes above. There are two significant issues in supporting worker behavior: time being unavailable, and the worker indicating that time is not needed.

9.7.1 Time not needed

The WTI infrastructure (master) must respect the SThreadBusy signal (with the exact, pipelined behavior per OCP) by not asserting the WRITE value on the MCmd signals unless the SThreadBusy signal from the worker is deasserted in the previous cycle. The infrastructure ***may*** take advantage of this indication of SThreadBusy to reduce activity/power in the infrastructure since the worker is explicitly indicating that it does not need a valid time value in the following cycle.

9.7.2 Time not available

When a worker cannot tolerate time being unavailable (when its AllowUnavailable attribute is false), then the control system will ensure that time is indeed available before issuing the “start” control operation to the worker’s WCI interface. Furthermore, the infrastructure must be made aware (through implementation-defined mechanisms), that the worker cannot tolerate time becoming unavailable, and thus must inform the control system (again through implementation-defined mechanisms), when time becomes *unavailable*.

10 WIP XML Schema

10.1 Introduction

This section describes the WIP schema for users creating XML documents that describe WIP-based IP modules. It is the format of the WIP metadata.

10.2 Quick XML Introduction

XML documents are files that contain information formatted according to XML (Extensible Markup Language) syntax and structure according to a particular application-specific "XML schema". XML information itself is formatted into Elements, Attributes, and Textual Content. The WIP XML Schema does not allow Textual Content at this time. XML Elements take two forms. The simpler one is when an element has no child (embedded) elements and no Textual Content. It looks like this (for element of type , "xyz", with attribute "abc"):

```
<xyz abc="123"/>
```

Thus the element begins with the "<" character and the element type, and terminated with the "/>" characters. Attributes have values in double-quotes. Any white space, indentation, or new lines can be inserted for readability between the element name and attributes. Thus the above example could also be:

```
<xyz
   abc="123"
   />
```

When the element has child elements (in this case element of type "ccc" with attribute "cat"), it looks like:

```
<xyz abc="123">
   <ccc cat="345"/>
</xyz>
```

So, an XML Schema defines which Elements, Attributes, and child Elements the document may contain. Every XML document (in this context) has a single top level element that must be structured (attributes and sub-elements) according to the WIP XML schema.

An element can be entered directly (as above) or entered by referring to a separate file that contains that element. So the example above might have a file "ccc1.xml" containing:

```
<ccc cat="345"/>
```

And then a top level file called "xyz1.xml" containing:

```
<xyz abc="123">
   <xi:include href="ccc1.xml"/>
</xyz>
```

However, the schema specifies which elements are allowed to be top-level elements in any file.

There are various built-in attributes that are part of the XML system itself and elements (such as the xi:include element above), and are required in certain cases. In particular, the top level element of a top level file (not an included file) must have these built-in, or "boilerplate" attributes:

To specify the schema according to which this document is structured:

```
xmlns:x="http://www.w3.org/2001/XMLSchema-instance"
x:schemaLocation="http://www.mc.com/CPI WIP-schema1.xsd"
xmlns="http://www.mc.com/CPI"
```

To allow the use of the "xi:include" feature above, the top level element that will have inclusions inside it must contain the attribute:

```
xmlns:xi="http://www.w3.org/2001/XInclude"
```

If all these boilerplate attributes were included in the example above, it would look like:

```
<xyz
     xmlns:x="http://www.w3.org/2001/XMLSchema-instance"
     x:schemaLocation="http://www.omg.org/CPI WIP-schema1.xsd"
     xmlns="http://www.omg.org/CPI"
     xmlns:xi="http://www.w3.org/2001/XInclude"
  abc="123">
  <xi:include href="ccc1.xml"/>
</xyz>
```

The schema reference attributes allow the files to be processed with very generic tools, since the files contain the reference to the schema they are following. If they are not included, then they must be supplied as command-line arguments on the tools that process these files. With these attributes included any XML processor can process the files without being told about the schema.

Note that the default value of all boolean attributes is false unless otherwise noted.

10.3 Top Level Element: Component Specification

A component specification is the XML element describing a component at a high level *without implementation choices*. It describes enough about how the component can be integrated into an application to ensure possible WIP-based interoperability and interchangeability among different implementations. Many such implementations are possible given this *component specification*. In particular, a component specification can usually be completely derived from a higher level component model such as that defined in the various software-defined radio component standards such as SCA and the OMG's "PIM/PSM for software-based communication", as well as the generic component standards such as the ones in UML v2 or CCM v4.

The component specification contains component-global attributes, control plane aspects and data plane aspects. A component specification is contained in the XML element whose type is "ComponentSpec", which should be a top level element in a file, structured as:

```
<ComponentSpec
     xmlns:x="http://www.w3.org/2001/XMLSchema-instance"
     x:schemaLocation="http://www.mc.com/CPI WIP-schema1.xsd"
     xmlns="http://www.mc.com/CPI"
   ---attributes---
     >
   ---child elements---
</ComponentSpec>
```

In the various examples below the boilerplate attributes will be omitted.

10.3.1 Attributes of a Component Specification

10.3.1.1 Name

The “Name” attribute of the component specification is constrained to be acceptable as an identifier in several contexts, including VHDL. It identifies the component specification as a whole. It is case *insensitive*: when in a library or application, two different component specifications cannot have the same name, when compared in a case-insensitive way.

10.3.1.2 NoControl

The “NoControl” attribute of the component specification is a boolean attribute that indicates, when true, that components using this specification have no control plane/interface at all. This is not allowed for application components but is specified for certain infrastructure components.

10.3.2 Control Plane Aspects of a Component Specification

A component’s control plane specification is essentially a description of its configuration properties used to parameterize its operation or to retrieve operational statistics and resulting scalar values based on the component’s operation. There are two different ways to specify this: the summary or the details. The Properties element of a component specification enumerates the name and type and behavior of each configuration property supported by implementations of this specification. The alternative PropertySummary element simply provides the information necessary to define the (WCI) control interface for the component, without providing the details of individual configuration properties. If the Properties element is present, the property summary information is inferred from the enumerated property descriptions.

Both the Properties and PropertySummary elements may be in separate files and referenced using the <xi:include href=”<file>”/> syntax. This is common when PropertySummary or Properties elements are shared among multiple component specifications.

10.3.2.1 Properties Element

The Properties element has no attributes, but consists of a list of Property elements. The ordering of the Property elements is important: implementations are required to respect the order of the listed properties. Anywhere in the list of Property elements in the overall Properties element, you can be include additional Properties elements using

the <xi:include href="<file>"/> syntax. This allows a list of properties to be shared as a subset of the properties of different components.

10.3.2.1.1 Property Element

A Property element describes one configuration property. It has attributes, but no sub-elements. It always occurs as a sub-element of the Properties element. A Property element describes the name and data type of a configuration property, and there is a "struct" data type to allow properties to consist of a structure of simpler scalar data members. [As of the current WIP specification, there is no support for recursive structures] Furthermore, each property can be an array or sequence of its data type. Consistent with the CORBA IDL specification, the term "array" refers to a fixed number of data values of the specified type, whereas the term "sequence" refers to a variable number of data values, up to a specified maximum length. The attributes of the Property Element are listed here:

10.3.2.1.1.1 Name attribute

The Name attribute is the case insensitive name of the property. A set of properties cannot have properties whose names differ only in case. Mixed case property names can be used for readability. When a Properties element includes (via xi:include) other Properties elements there is still only one flat case-insensitive name space of properties for the component.

10.3.2.1.1.2 Type attribute

The Type property specifies the data type of the property. The legal types (case insensitive) are: Bool, Char, Double, Float, Short, Long, UChar, ULong, UShort, LongLong, ULongLong, and String. The "Char", "Short", "Long", and "LongLong" types represent 8, 16, 32, and 64 integer values respectively. The "Float" and "Double" types are consistent with 32 and 64 bit IEEE floating point types, and the "String" type is a null terminated string. When the "Type" attribute has the "String" value, the "StringLength" attribute must also be supplied, to indicate the maximum length of the string property values, excluding the terminating null character (consistent with the ISO-C strlen function). If no Type attribute is present in the Property element, the type "ULong" is used.

The "ArrayLength" attribute is used when the property is a fixed length array of the indicated type. The "SequenceLength" attribute is used when the property is a variable length sequence of the indicated type.

Then the type is "Struct", the Property element has Property sub-elements that indicate the types of the members of the struct value. No struct members can be of type "Struct". The SequenceLength and ArrayLength attributes do apply to "Struct" properties.

Thus all types have a maximum length. There are no unbounded length property data types.

10.3.2.1.1.3 StringLength attribute

The StringLength attribute is used when the Type attribute is “String”, and indicates the maximum length null-terminated string value that this property can hold. The null is not included in the this length.

10.3.2.1.1.4 ArrayLength attribute

The presence of this attribute indicates that the property values are a fixed length array of the type specified in the Type attribute, and that fixed length is indicated in the value of this ArrayLength attribute.

10.3.2.1.1.5 SequenceLength attribute

The presence of this attribute indicates that the property values are a variable, but bounded, sequence of the type specified in the Type attribute, and that maximum length is indicated in the value of this SequenceLength attribute. Thus this property has the specified maximum length, and always contains a current length, up to that limit.

10.3.2.1.1.6 Readable attribute

This attribute, which defaults to true, indicates whether this property can be read by control software. If set to false, attempts to read the property value at any time after instantiation will result in an error.

10.3.2.1.1.7 Writable attribute

This attribute, which defaults to true, indicates whether this property can be written by control software. If set to false, attempts to write the property value at any time after instantiation will result in an error.

10.3.2.1.1.8 Initial attribute

This attribute, which defaults to true, indicates whether this property can be specified at the time of instantiation as an initial value, by control software. If set to false, attempts to specify the property at instantiation time will result in an error.

10.3.2.1.1.9 Volatile attribute

This attribute, which defaults to true, indicates whether this property’s value will change as a consequence of the component’s execution, without it being written by control software. When true, it indicates to control software that the values set at instantiation time or when the value is written, cannot be cached.

10.3.2.2 Property Summary Element

The property summary contains information about the configuration properties of the component that are relevant to the control interface, based on the WCI profile. It is called a “summary” since there is normally more information about configuration properties (the “details”) that are neither relevant to nor required by the component’s control interface. This property summary is expressed as an XML element whose type is “PropertySummary”, and is a child element of a ComponentSpec. [A later version of this specification will include a PropertyDetails element that can be used in place of this when appropriate. The property detail will enumerate each configuration property value and its attributes to aid in generating more convenience items in the VHDL skeletons.]

The property summary includes the following attributes, as documented in the WIP WCI profile:

10.3.2.2.1 SizeOfConfigSpace attribute

The type of this attribute is of type unsignedInt <= 2^20 with a default value of zero.

10.3.2.2.2 WritableConfigProperties attribute of type boolean

The type of this attribute is of type boolean.

10.3.2.2.3 ReadableConfigProperties attribute

The type of this attribute is of type boolean.

10.3.2.2.4 Sub32BitConfigProperties attribute

The type of this attribute is of type boolean.

Thus an example is:

```
<ComponentSpec
   ---attributes---
      >
   <PropertySummary SizeOofConfigSpace="20"/>
   ---other child elements---
</ComponentSpec>
```

If there is no PropertySummary, then all the related attributes take on default values for the ComponentSpec, as if "<PropertySummary/>" was included. This implies that the specified component has no configuration properties as all.

10.3.3 Data Plane Aspects of a Component Specification

10.3.3.1 Data Interface Specification Element

The component specification defines data plane interfaces through the use of the "DataInterfaceSpec" element. It specifies the direction of the interface (producer or consumer) and the message-level protocol used at that interface. In particular it does *not* specify the WIP data profile (WSI, WMI) for that interface as that is an implementation choice.

The data interface specification element has several attributes and one child element: the Protocol Summary

10.3.3.1.1 Name attribute

This attribute specifies the name of the interface for this component. The value of the name attribute is a string that is constrained to be valid in various languages like OCP configuration files and VHDL. It must be unique (case insensitive) within the component specification.

10.3.3.1.2 Producer attribute

This boolean attribute is as specified in the WIP data plane profiles, and indicates whether data is produced at this worker interface or consumed at this interface. A value of “true”, indicates a producer interface.

10.3.3.2 Protocol Summary element

The protocol summary element, which is a child element of the data interface specification element, specifies attributes of the message protocol used at this interface. We use the term “summary” since this element only contains the interface-related summary aspects of the message protocol, not the details (analogous to the PropertySummary above). [In a future version of this specification we will define a ProtocolDetails element that can be used in place of this]. The ProtocolSummary is a separate element since it will likely be reused across a variety of components and interfaces and thus may be referred to in a separate common file via the “xi:include” mechanism mentioned above.

The attributes of the ProtocolSummary element are attributes defined for the data interfaces in the WIP profiles (WSI and WMI). They may be omitted if they have default values as defined in the WIP profiles. The attributes are described in the worker data interface section above.

10.3.3.2.1 DataValueWidth

The type of this attribute is of type unsignedInt.

10.3.3.2.2 DataValueGranularity

The type of this attribute is of type unsignedInt.

10.3.3.2.3 DiverseDataSizes

The type of this attribute is of type boolean.

10.3.3.2.4 MaxMessageValues attribute

The type of this attribute is of type unsignedInt.

10.3.3.2.5 NumberOfOpcodes

The type of this attribute is of type unsignedInt.

10.3.3.2.6 VariableMessageLength

The type of this attribute is of type boolean.

10.3.3.2.7 ZeroLengthMessages

The type of this attribute is of type boolean.

10.3.4 Component Specification Examples

```
<ComponentSpec Name="K1spec"
   <PropertySummary
      SizeofConfigSpace=”8” WritableConfigProperties=”true”/>
   <DataInterfaceSpec Name="lvds_tx" Producer="true">
      <ProtocolSummary
         MaxMessageValues="1024" VariableMessageLength="true"/>
   </DataInterfaceSpec>
   <DataInterfaceSpec Name="lvds_rx" Producer="false">
      <ProtocolSummary
         MaxMessageValues="1024" VariableMessageLength="true"/>
   </DataInterfaceSpec>
</ComponentSpec>
<ComponentSpec Name="S1spec"
   <PropertySummary
      SizeofConfigSpace="256"
      WritableConfigProperties="true”
      ReadableConfigProperties="true"/>
   <DataInterfaceSpec Name="sfpdp_prod" Producer="true">
      <ProtocolSummary
         MaxMessageValues="1024" VariableMessageLength="true"/>
   </DataInterfaceSpec>
   <DataInterfaceSpec Name="sfpdp_cons" Producer="false">
      <ProtocolSummary
         MaxMessageValues="1024" VariableMessageLength="true"/>
   </DataInterfaceSpec>
</ComponentSpec>
```

10.4 Top Level Element: Component Implementation Description

A component implementation description contains information provided by someone creating an implementation based on a component specification. It includes or references a component specification, and then describes implementation information about a particular implementation of that specification. Thus, after the attributes, it must either include as a child element a complete ComponentSpec, or include one by reference, for example, if the “fastcore” implementation of the “corespec1” specification referenced the component specification found in the “corespec1.xml” file:

```
<HdlImplementation
      xmlns:x="http://www.w3.org/2001/XMLSchema-instance"
      x:schemaLocation="http://www.omg.org/CPI WIP-schema1.xsd"
      xmlns="http://www.omg.org/CPI"
      xmlns:xi="http://www.w3.org/2001/XInclude"
   Name=”fastcore”
   ---other attributes---
   >
   <xi:include href=”corespec1.xml”/>
   ---other child elements---
</HdlImplementation>
```

10.4.1 Attributes of an HDL Component Implementation

10.4.1.1 Name

The “Name” attribute of the component implementation is constrained to be acceptable as an identifier in several contexts, including VHDL. It identifies the component specification as a whole. It is also used at the “core” name in OCP.

10.4.2 Control Plane Aspects of an HDL Component Implementation

10.4.2.1 Control Interface element

The control interface child element (ControlInterface) of the component implementation description (ComponentImplementation) specifies implementation aspects of the worker’s control interface (its interface based on the WCI profile). If this element is not present, then the PropertySummary must also not be present in the referenced Component Specification, and indicates that this worker has no control interface at all. This is only allowed for infrastructure cores, not application cores. There can be zero or one control interface elements per component implementation. The attributes of the control interface are:

10.4.2.1.1 Name attribute

This attribute specifies the name used for this WCI-based interface, which defaults to “control” if not specified. The string value of this attribute is constrained to be a valid identifier in both OCP and VHDL.

10.4.2.1.2 ControlOperations attribute

This attribute is defined as a WCI profile attribute, and contains a comma-separated list of strings identifying the implemented control operations (the start operation must be included, and the default value is “start”). o

10.4.2.1.3 ResetWhileSuspended attribute

This boolean attribute is defined as a WCI profile attribute.

10.4.2.1.4 Clock

This attribute specifies the name of the clock domain used at this interface. It must match the name of a clock child element of the component implementation. It may also refer to the name of another interface, in which case the clock for this interface will use the clock specified at that other interface. If not specified, the WCI has its own clock input signal, named according to the prefix rules, associated with the other signals at this interface.

10.4.2.2 Clock element

The clock child element (Clock) of the component implementation description (ComponentImplementation) specifies a clock domain (clock input) to the implementation. It may be used by any interfaces, or be independent of any of those interfaces. There can be zero or more clock elements in the component

implementation. If there are none, then each interface (control or data) is assumed to have the same clock as the WCI, behaving per OCP. The attributes of the clock element are:

10.4.2.2.1 Name attribute

This attribute specifies the name used for this clock, which defaults to “clock” if not specified.

10.4.2.2.2 Signal attribute

This attribute specifies the name of the signal used inside the implementation for this clock. If the clock is “owned” by an interface (i.e. the “MyClock” attribute of that interface element is true), then this attribute must not be specified. The string value of this attribute is constrained to be a valid identifier in both OCP and VHDL.

[future versions of this specification are expected to include constraints for the clocks required by an implementation]

10.4.3 Data Plane Aspects of an HDL Component Implementation

10.4.3.1 Stream Interface element

The stream interface child element (StreamInterface) of the component implementation description (ComponentImplementation) specifies a data interface that uses the streaming WIP profile (WSI). It references a DataInterfaceSpec by its Name attribute. Thus the Name attribute of the StreamInterface element must match the Name attribute of a DataInterfaceSpec element of the ComponentSpec. The Stream Interface element adds implementation-specific information about the interface initially defined in that DataInterfaceSpec.

10.4.3.1.1 Name attribute

This attribute specifies the name used to reference the DataInterfaceSpec in the ComponentSpec.

10.4.3.1.2 Clock attribute

This attribute specifies the name of the clock domain used at this interface. It must match the name of a clock child element of the component implementation or refer to the name of another interface, in which case the clock for this interface will use the clock specified at that other interface. If not specified (and the MyClock attribute is not true), then this interface uses the same clock as the WCI.

10.4.3.1.3 MyClock attribute

This Boolean attribute when true specifies that the clock used at this interface will be associated with the other OCP signals at this interface, and be named with the prefix rules for signals at this interface. It specifies that the clock is “owned” by this interface. If this attribute is true, and the “Clock” attribute is not specified, then this interface has its own clock, named with the other signals of this interface.

10.4.3.1.4 DataWidth attribute

As specified for the WSI.

10.4.3.1.5 PreciseBurst attribute

As specified for the WSI.

10.4.3.1.6 ImpreciseBurst attribute

As specified for the WSI.

10.4.3.1.7 Continuous attribute

As specified for the WSI.

10.4.3.1.8 Abortable attribute

As specified for the WSI.

10.4.3.1.9 EarlyRequest attribute

As specified for the WSI.

10.4.3.2 Message Interface element

The message interface child element (MessageInterface) of the component implementation description (ComponentImplementation) specifies a data interface that uses the messaging WIP profile (WMI). It references a DataInterfaceSpec by its Name attribute. Thus the Name attribute of the MessageInterface element must match the Name attribute of a DataInterfaceSpec element of the ComponentSpec. The Message Interface element adds implementation-specific information about the interface initially defined in that DataInterfaceSpec.

10.4.3.2.1 Name attribute

This attribute specifies the name used to reference the DataInterfaceSpec in the ComponentSpec.

10.4.3.2.2 Clock attribute

See StreamInterface Clock attribute above.

10.4.3.2.3 MyClock attribute

See StreamInterface MyClock attribute above.

10.4.3.2.4 DataWidth attribute

As specified in the WIP data profiles.

10.4.3.2.5 ByteWidth attribute

As specified in the WIP data profiles.

10.4.3.2.6 PreciseBurst attribute

As specified in the WIP data profiles.

10.4.3.2.7 ImpreciseBurst attribute

As specified in the WIP data profiles.

10.4.3.2.8 Continuous attribute

As specified in the WIP data profiles.

10.4.3.2.9 TalkBack attribute

As specified in the WIP data profiles.

10.4.4 Memory Aspects of a Component Implementation

10.4.4.1 Memory Interface element

The memory interface child element (MemoryInterface) of the component implementation description (ComponentImplementation) specifies a memory interface that uses the memory WIP profile (WMemI).

10.4.4.1.1 Name attribute

This attribute specifies the name used for the memory interface (default is “memory”).

10.4.4.1.2 Clock attribute

See StreamInterface Clock attribute above.

10.4.4.1.3 MyClock attribute

See StreamInterface MyClock attribute above.

10.4.4.1.4 DataWidth attribute

As specified in the WIP memory profiles.

10.4.4.1.5 ByteWidth attribute

As specified in the WIP memory profiles.

10.4.4.1.6 PreciseBurst attribute

As specified in the WIP data profiles.

10.4.4.1.7 ImpreciseBurst attribute

As specified in the WIP data profiles.

10.4.4.1.8 MemoryWords

As specified in the WIP data profiles.

10.4.4.1.9 MaxBurstLength attribute

As specified in the WIP data profiles.

10.4.4.1.10 WriteDataFlowControl attribute

As specified in the WIP data profiles.

10.4.4.1.11 ReadDataFlowControl attribute

As specified in the WIP data profiles.

10.4.5 Time Aspects of a Component Implementation

10.4.5.1 Time Interface element

The time interface child element (InterfaceInterface) of the component implementation description (ComponentImplementation) specifies a time interface that uses the time WIP profile (WTI).

10.4.5.1.1 Name attribute

This attribute specifies the name used for the time interface (default is “time”).

10.4.5.1.2 Clock attribute

See StreamInterface Clock attribute above.

10.4.5.1.3 MyClock attribute

See StreamInterface MyClock attribute above.

10.4.5.1.4 SecondsWidth attribute

As specified in the WTI profile.

10.4.5.1.5 FractionWidth attribute

As specified in the WTI profile.

10.4.5.1.6 AllowUnavailable attribute

As specified in the WTI profile.