1 # Worker Interface Profiles (WIP)

2 ## Functional Specification

3

4

5

6 Includes the OCP Profiles for:

7 Worker Control Interface (WCI)

8 Worker Streaming Interface (WSI)

9 Worker Message Interface (WMI)

10 Worker Memory Interface (WMemI)

11 Worker Time Interface (WTI)

12

13

14

15

16

17

18

19

20

21

22

23

24 Authors: Jim Kulp, Shepard Siegel

25

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 several areas based on automation implementaiton	jkulp	2010-01-xx

26

27

Table of Contents

1   References

165 When the term "OCP Specification" is used in this document, the reference is to version
166 2.2 of the OCP Specification, Document Revision 1.0. The OCP specification and
167 supporting information are available from www.ocpip.org .

168 **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

169 *1.1   Definitions*

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

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

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

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

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

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

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

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

197 **Infrastructure** – IP is either application or infrastructure. IP that is not application is
198 infrastructure.

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

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

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

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

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

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

2   Introduction

2.1   Purpose

216 The purpose of this document is to introduce and describe a collection of Worker
217 Interface Profiles (WIP or WIPs)[1] and how to use them to produce WIP-compliant IP. It
218 specifies metadata and interfaces used by tools and RTL IP developers, using
219 underlying OCP and XML standards. These interfaces are relevant to both application
220 and infrastructure/platform developers, but this document is focused on application
221 developers, with many infrastructure requirements inferred. I.e., by stating what
222 application IP (worker) developers may do, it implies what infrastructure developers
223 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.

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

2.2 Requirements

231 • Well-Defined Adaptability

232 • Well-Defined Interoperability

233 • Independent from Specific Implementations

234 • HDL Language-Agnostic, HDL Language-Neutral

235 • Semantics hold under Hierarchy

236 • Must support compliance with SCA/CP289

2.3 Goals

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

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

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

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

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

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

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

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

275   *2.4   Non-Goals*

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

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

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

286   *2.5   Overview*

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

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

Worker Interface Profiles (WIPs)

294
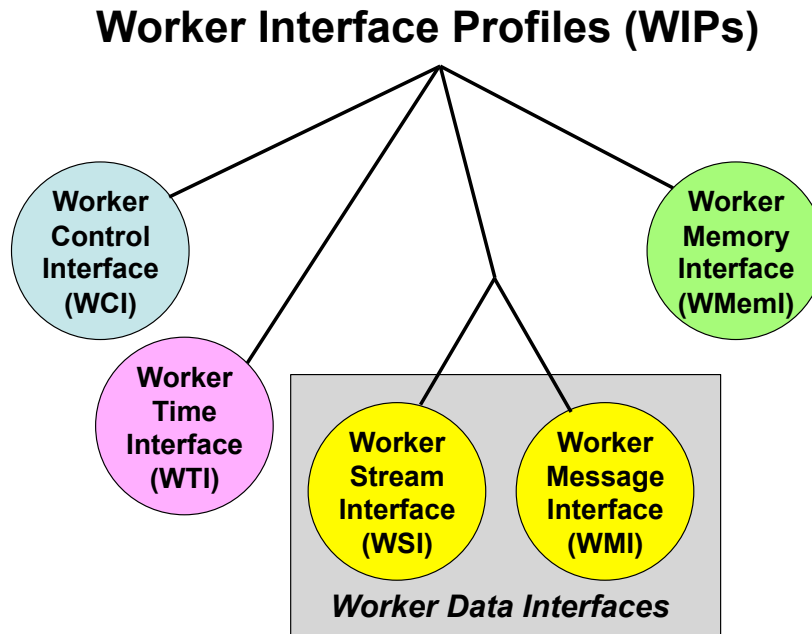
Figure 1 – Worker Interface Profile Taxonomy

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

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

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

317 An illustrative example:

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

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

323
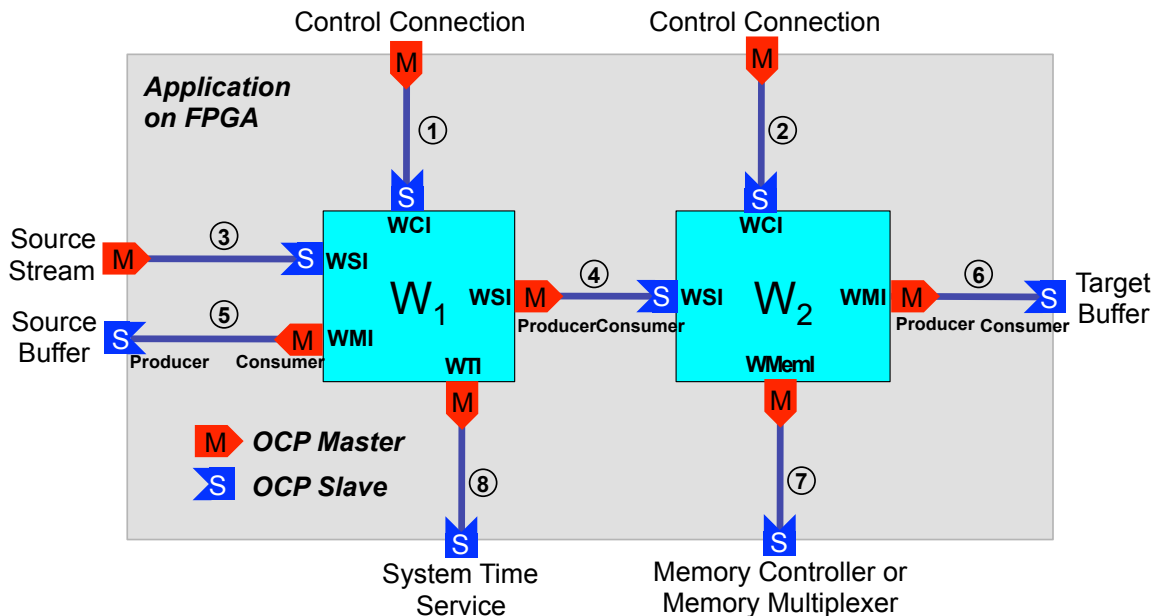
324 **Figure 2 – Two Workers in an FPGA**

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

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

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

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

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

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

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

351 2.5.1 Worker vs. Infrastructure and OCP Master/Slave Roles

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

353 **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

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

358
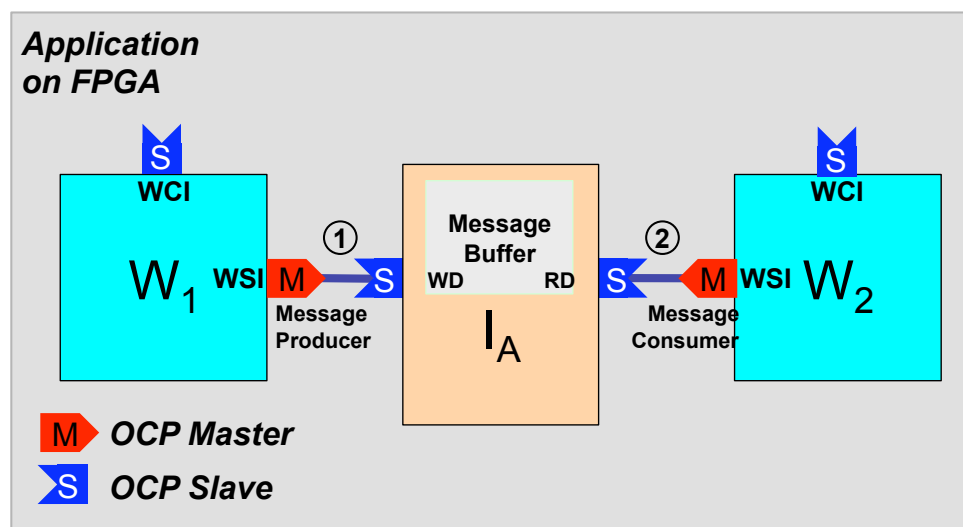
359 **Figure 3 - Two WMI Application Workers Communicating via a buffer**

360   *2.6   Document Organization*

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

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

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

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

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

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

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

385   Every interface has a Clock attribute to indicate if the interface requires an OCP clock
386   that is different from the OCP clock applied to the WCI. When it is different from the
387   WCI clock, it must be specified as one of the clocks in the Clocks global attribute for the
388   worker. Every interface has a ClockSignal attribute, whose default is the standard OCP
389   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.*

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

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

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

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

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

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

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

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

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

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

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

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

429 *2.7   OCP Interface Clocking*

430 This section describes the clocking of the OCP interfaces of workers

431 All OCP interaction is between a single master and slave. The clock for each OCP
432 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)

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

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

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

444   Example 1: If a worker consumes a WSI stream on one interface, delays it
445   through a register, and outputs it on another WSI interface; these two interfaces
446   must be placed in the same named clock domain. If the worker author designed
447   the logic so that the output interface was designed for asynchronous operation,
448   then the two interfaces *could* be in the same named domain, or in different
449   domains.

450   Example 2: If a worker implements a fixed N:1 decimator (or 1:M interpolator)
451   then the clocks may be in different named domains with additional specific
452   attributes, which may describe their (rational) relationships.

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

2.8   Reset Propagation and Behavior

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

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

464 There are special semantics for the reset on WCI:

465   When WCI reset is asserted (MReset_n='0') on the WCI (slave) interface of a
466   worker, the worker shall propagate reset to all other interfaces on that worker (as
467   MReset_n or SReset_n as appropriate for that interface).

468   The WCI MReset_n signal is in the clock domain of the WCI. It is the
469   responsibility of the worker to properly propagate the reset indication to any other
470   clock domains at its other interfaces (data, memory, time etc.).

471 For all data interfaces on a worker:

472 All worker implementations must insure that reset inputs at their data interfaces
473 force all OCP output signals at that interface to a proper reset state per OCP.

474 The simplest (and least helpful for application debugging) worker
475 implementations will not support the "Stop" control operation, and will simply
476 need to reset their data interfaces. They will be assumed non-functional and will
477 be fully reset via WCI reset before any other actions are taken.

478 Worker implementations may support the Stop control operation, but also not be
479 functional after receiving data interface reset assertions. They will also be
480 assumed non-functional and be fully reset via WCI reset before any other actions
481 are taken.

482 The "ResetWhileSuspended" attribute of a worker (when true) indicates the most
483 helpful behavior: the worker will support the Stop control operation and accept
484 data interface resets after being suspended by the Stop control operation. Such
485 resets (from adjacent workers) while suspended, will not render the worker non-
486 functional and thus allow for configuration property operations while in this
487 suspended state even though data interfaces are in the reset state. After the
488 asserted data interface resets are deasserted, such workers may be resumed via
489 the "start" control operation.

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

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

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

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

512 • Reset asserts asynchronously, but de-asserts synchronously.

513 • Reset shall be asserted for a minimum duration of 16 cycles of the clock domain to
514   which the reset signal is synchronous.

515 • Workers must be able to reset in this 16-cycle duration.

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

2.9 *Worker Global Attributes*

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

525   2.9.1 Worker Global Attribute Summary

526   **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

527   2.9.2 Worker Global Attribute Details

528

Name	WIP:Ident	The name of the worker

529   Name – The name of the worker.

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

Clocks	WIP:Clock	Declaration of a clock for the worker

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

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

Endian	WIP:Endian	WCI endian and global endian default

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

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

548 • have no configuration property values of sizes other than the data path of the WCI
549   (32 bits)

550 • accept or produce messages on any data interface that are only a (byte) multiple of
551   the width of that interface

552 • process data values not smaller than the width of the data interface.

553 If any of these are *not* true, the implementer must specify a non-default value, as
554 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]

555

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

556 *2.10 Signal Prefix Rules and Guidelines*

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

559 2.10.1 Signal Prefix Rule

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

563 For example:

564   MyUniqueInterfaceName_MCmd

565 2.10.2 Signal Prefix Guideline

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

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

569   2. The abbreviation for the interface/profile type {WCI|WSI|WMI|WMemI}

570   3. The interface name

571   4. The OCP signal name

572   For example:

573
```
ocp_wsi_chan1_MCmd
```

574   2.10.3 Signal Prefix Example

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

577   1. There is no special "types" or "definitions" package required for the WIPs.

578   2. These signals are not explicitly part of any interface. They may or may not be
579   used with a particular WIP.

580   3. Infrastructure with a WSI named "AD0".

581   4. Infrastructure with a WSI named "AD1".

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
                                                        1
library WORK;
use WORK.vita49_types.all;

entity app is
  port (                                                       2
    -- 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);     3
    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);     4
    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;
```

582

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

586   3   Worker Control Interface (WCI)

587   *3.1   WCI Motivation*

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

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

597
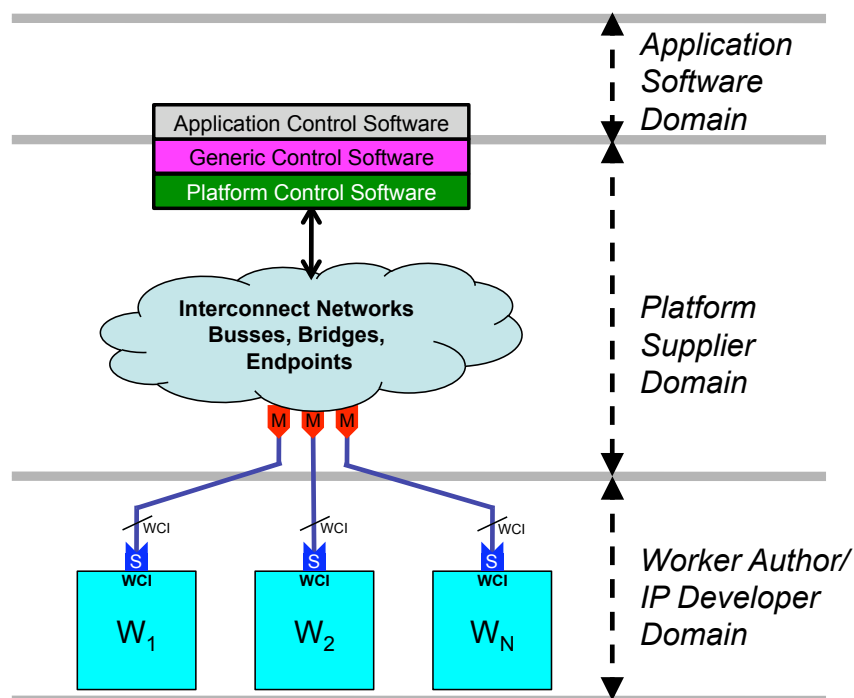
598   **Figure 4 – Worker Control Interface – Separation of Concerns**

599   *3.2   WCI Overview*

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

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

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

System side Interface
(e.g. System Bus IP)

Control Plane IP (CPIP)

WCI   WCI   WCI

M   M   M

Infrastructure IP

WCI   WCI   WCI

Application IP

S   S   S

WCI   WCI   WCI

W_1 W_2 W_N

612
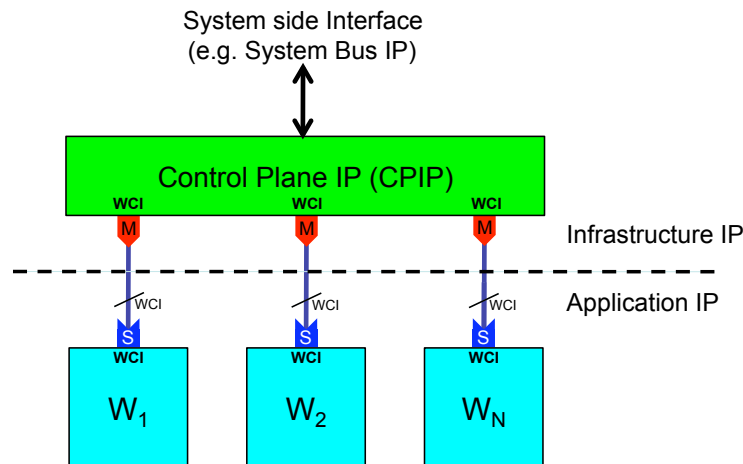
Figure 5 - Relation of CPIP to Workers

613

614 3.2.1 Control and Configuration

615 WCI transactions are either "control" or "configuration".

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

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

629   **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.

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

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

644   3.2.2 WCI Control Operations

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

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

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

652   The operations are:

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

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

655   •   Infrastructure/control system automatically asserts reset upon loading

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

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

658
659
- If there is any possibility that 16 cycles are not enough, use **initialize** to perform initializations that take more time.

660
661
- WCI Reset has special meaning for other interfaces. It is described in section 2.9 on *Reset Propagation and Behavior*.

662
663
- Workers must be reusable via a reset assertion multiple times after a bitstream load.

664
- Initialize

665
- Needed when additional initialization is required that is not done during reset.

666
667
- Allows worker to perform initial work not under reset, to achieve a known ready-to-run state

668
669
- Appropriate place for worker to perform runtime initialization (e.g. programmatic initialization of a LUT).

670
671
- Worker can internally set initial/default values of properties (if not done via **Reset**)

672
673
674
- 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.

675
- Start:

676
677
- Transition to operational - to doing the real work based on property values and data (at data interfaces).

678
679
- Must do any one-time initializations that depend on software-configured property values, since **Initialize** cannot do this (properties are not set yet).

680
- Starting operation is based on property values set *after* **Initialize** completes

681
- Stop

682
683
- Make the worker inactive/paused/suspended, such that it can be resumed (via **Start**); maintain resources/state/properties for examination.

684
685
- When this operation is complete, data interfaces are "between messages". This is a "graceful" suspension, not an immediate destructive stop or abort.

686
687
- Worker is suspended, data interfaces can be reset by connected workers (if implementation is capable of it)

688
- Release

689
690
- Can be no-op if all un-initialization must be done under **Reset**: it returns failure in this case.

691
- Shut down/abort operations, discard any state (state is now undefined)

692
- If release fails, worker is *unusable until reset.*

693
- May be called after any operation, but not immediately after deassertion of reset.

694
- BeforeQuery

695
696
- 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.

697
- Used to enable coherent access to multiple property values.

698
699

- Only necessary when property values depend on each other and need to be properly updated to be meaningful/correct.

700 • AfterConfig

701
702

- After **Initialize**, **Start**, or **Stop,** informs worker that a group of related property changes have been made.

703
704

- Informs worker that changes to a set of interrelated property values can be processed.

705 • Test

706
707

- After **Initialize**, used to run component-specific built-in tests, parameterized by current configuration property settings.

708
709
710
711

- 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".

712

- Simple workers that have no tests can indicate it in metadata.

713 *3.2.2.1 Simple workers can be very simple for control operations*

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

715
716

- **Initialize** is not needed – all initialization is performed during reset, and after reset is deasserted, the worker can accept configuration accesses.

717
718
719

- **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.

720
721
722

- **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.

723

- **Test** is not implemented: there is no built-in test capability.

724
725

- **BeforeQuery** and **AfterConfig** operations are not needed if there are no interdependent property values that must be used atomically and/or consistently.

726
727

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

728 3.2.3   WCI States

729
730
731

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.

732
733

- State**: Exists**: after deassertion of reset or after successful **release** operation, ready for **initialize**.

734
735

- The worker is loaded (if necessary), not necessarily fully initialized, configuration properties are not valid, property access is not valid, but reset is deasserted.

736
737

- Only the **initialize** operation will be issued. If successful the worker is considered to be in the "initialized" state.

738 • State: **Initialized:** after successful **initialize** operation**,** ready to **start** (or **release**)

739 • The worker has been completed any run-time initialization, doing whatever is
740 necessary or appropriate *after reset was deasserted*.

741 • The worker is in a known, stable, initial state "ready to start doing work", but not
742 operational.

743 • Configuration properties can be read and written (SW software can see the result
744 of initialization, and can configure the worker prior to normal operation). Thus the
745 **BeforeQuery** or **AfterConfig** operations maintain this state.

746 • State: **Operating:** after successful **Start** operation**,** ready to **Stop** (or **Release**)

747 • The worker is doing normal work based on properties and data interfaces; it is
748 "processing".

749 • Configuration properties can be read or written, including **BeforeQuery** or
750 **AfterConfig** operations

751 • State: **Suspended:** after successful **Stop** operation, ready to **Start** (or **Release**)

752 • The worker is suspended; data interfaces can be reset by connected workers (if
753 the implementation is capable of it).

754 • Configuration properties can be read or written, including **BeforeQuery** or
755 **AfterConfig** operations

756 • State: **Unusable**: after unsuccessful **Release** operation

757 • In this state the only action to take is to assert WCI reset.

758 3.2.4 WCI Errors

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

762 • All **release** errors indicate unusable worker circuit state requiring reset before re-use

763 • If Initialize, Start, Stop, BeforeQuery or AfterConfig return an error

764 • If metadata (the **Retry** attribute) says retries are possible, try again.

765 • If not, (or after unsuccessful retries) then **Release** may be attempted.

766 • After a successful **Start** operation, worker may assert the attention signal (see
767 semantics for Sflag[0] below) to indicate that an error has occurred or attention is
768 required

769 • What this means is worker dependent, but generic software knows to inform
770 users or invoke attention handlers for the worker.

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

772 *3.3 WCI Attributes*

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

775 Each attribute is labeled indicating whether the attribute value relates to the worker's
776 interface (and thus applies to all implementations), or whether the attribute value is

777 specific to a particular implementation of the same interface common to all related
778 implementations. The metadata formats (XML Schema described below) separates
779 attributes that are about high level interface behavior (e.g. properties) from those that
780 are implementation-specific.

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

782 ### 3.3.1 WCI Worker Attribute Summary

783 **Table 5 – WCI Worker Canonical Attribute Table**

Attribute Name	Type (per XML Schema)	Interface vs. Imple-mentation	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

784 ### 3.3.2 WCI Worker Attribute Details

785 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

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

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

WritableConfigProperties	boolean	Int	false	TRUE: there are writable configuration properties

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

ReadableConfigProperties	boolean	Int	false	TRUE: there are readable configuration properties

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

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

796 This attribute indicates the worker contains configuration properties that are less than
797 32-bit (4-byte) values. When TRUE, OCP byte enable logic is provided to allow for 1-

798 byte, and aligned 2-byte and 4-byte reads and writes. When FALSE, there is no byte
799 enable logic and all accesses are 4-bytes.

800 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 |

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

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

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

815 • No support at all: "Stop" is not in the list in the ControlOperations attribute.

816 • Stoppable without supporting reset of adjacent workers: ResetWhileSuspended is
817 FALSE

818 • Stoppable and able to remain configurable and resumable (via Start) when adjacent
819 workers are reset.

820 ### *3.4 WCI OCP Profile/Configuration Parameters*

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

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

828
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, $\quad$ N = 5 Else $\quad$ 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 writeresp_enable	{0|1}	If WritableConfigProperties = TRUE: 1 Else: 0

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

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

837 • Accept (DVA) is interpreted as a worker configuration property being committed, with
838 $\quad$ any side effects that might produce errors complete.

839 • Error (ERR) is interpreted as a worker explicitly failing to accept a configuration
840 $\quad$ property write.

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

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

845 ### 3.5 WCI OCP Signals

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

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

853
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

854
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

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

859 ### 3.6 WCI Semantics

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

863 ### 3.6.1 Indicating Control versus Configuration

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

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

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

871 ### 3.6.2 Encoding of Control

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

876 **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

877 ### 3.6.3 Control Response

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

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

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

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

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

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

903 *3.6.3.1 Retry*

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

907 3.6.4 Worker Control Operation Interrupt on the MFlag[0] signal

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

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

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

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

922 3.6.5 Worker "Endianness" Indicated on the MFlag[1] signal

923 Workers that have the capability of operating either little-endian or big-endian are called
924 *bi-endian* workers. The signal driven by the infrastructure on the MFlag[1] signal
925 indicates to such a worker if it is in a little- or big-endian environment. The worker only
926 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

927 3.6.6 Worker Attention on the SFlag[0] signal

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

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

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

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

941 3.6.7 Configuration Property Access

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

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

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

955 3.6.8 Reset

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

958 **_3.7   WCI Infrastructure Coping_**

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

963 • Presence or absence of configuration parameters, implying presence or absence of
964   data and address space signals.

965 • Readability or writability of properties, implying presence or absence of the data
966   signals in each direction.

967 • Presence of sub32bit properties, implying presence or absence of byte enables.

968 • Size of the usable configuration address space, determining the width of the address
969   signals, and the range of value address values.

970 WCI masters, which are infrastructure IP, should include all the optional signals. For
971 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)

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

974 **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.*

975

976 ## 4   Worker Data Interfaces

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

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

987 • The other application worker is local, but needs an adapter for the two data
988   interfaces to interoperate. Example: one worker producing on a WSI, and the other
989   consuming on a WMI.

990 • This worker is at the "edge" of the application, and the data is being sent to or
991   received from some external source or I/O device. Example: a signal is being
992   received from an A/D converter, and the worker is receiving the output of the A/D via
993   the infrastructure IP that deals with the A/D hardware.

994 • The other worker is remote, in another device, requiring infrastructure IP to forward
995   messages across some interconnect or fabric. Example: a worker is producing data
996   for a consuming worker in a different processor connected by a fabric or bus
997   attached to the local processor or FPGA.

998 ### 4.1   Data Protocol between Workers: the Data Plane Protocol

999 Worker Data Interfaces convey data content between workers according to a defined
1000 simple data protocol described by metadata attributes. Certain protocol aspects (e.g.
1001 explicit length, or message type) are supported and defined in this specification to
1002 "flatten" the overall protocol stack: simple protocol aspects that are efficiently
1003 implementable at this level of interface (based on OCP) are supported here. The
1004 definition and format of actual message payloads are not defined here. This data plane
1005 protocol is independent of:

1006 • the actual width (in wires/signals) of the interface

1007 • the chosen data interface profile (WSI or WMI)

1008 • the implementation-specific attributes specified by the worker implementer.

1009 The protocol is defined as a sequence of fixed or variable size messages, each of which
1010 may have a "message type" or "payload type" called an opcode. Messages contain
1011 data values, which may or may not be same size within a message. When all the data
1012 values in all types of messages in the protocol are the same size, the infrastructure can
1013 provide interoperability support between diverse endianness. By knowing the message
1014 granularity (the least common multiple of all message sizes), the actual signal interface
1015 can be simplified to the minimum necessary (e.g. eliminate byte enables).

1016 The simplest case of the "protocol" is a sequence of single-data-value messages of a
1017 single message type. This is the way to describe such a continuous stream of data
1018 values:

1019 • There is one type of message (and thus no need for an explicit "opcode").

1020 • The message length is fixed as the size of a single data value (and thus no need for
1021 an explicit message length, e.g. if the data values are single precision IEEE floats,
1022 all messages are 32 bits/4 bytes).

1023 In the more complex case all these attributes may be used to specify the "data
1024 protocol":

1025 • Size in bits of the smallest data values in all messages (default to 8). Note that the
1026 width of the data path is never allowed to be less than this.

1027 • Maximum message length in data values (default is 1).

1028 • Whether messages are variable length or are all the same (maximum) length.
1029 (default is fixed length)

1030 • The granularity of message sizes in number of data values (default: 1 data value,
1031 example of complex single float: 2 values).

1032 • How many different message types (also called "opcodes") (default: 1)

1033 The concept of "data value" is used to specify interfaces that are not byte oriented. If
1034 the data message is byte oriented, then MaxMessageValues is simply the maximum
1035 message length in bytes. If all defaults are used, all messages are of the same type,
1036 and are one byte long. Streams of single values with no real message boundaries are
1037 specified simply by specifying the DataValueWidth attribute to the size of the data
1038 values (in bits). Thus the concept of "length" of messages is always in units of data
1039 values.

1040 These attributes are specified for each of the worker's data interfaces, independent of
1041 choices of WSI or WMI, or other implementation-specific attributes. A worker
1042 "implementer" then chooses either WSI (streaming, fifo-style) or WMI (buffered or more
1043 complex cases) to implement the (producer or consumer) data interface. Thus the
1044 message attributes above are about the messages produced or consumed, and not
1045 about the WIP profile used, or the implementation choices within that profile.

1046 *4.2 Worker Data Interfaces Attribute Summary*

1047 The WIP data *protocol* attributes are described in the following table. While these
1048 attributes are valid for all data interface profiles (WSI and WMI), the way the attributes
1049 affect actual OCP signals in the interface are different and specified in the section for
1050 each profile. The "producer/consumer" role attribute of the interface is also here since it
1051 is also common to all data interface profiles. These attributes are repeated in the
1052 attribute tables for the data interface protocols.

1053

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.*

1054 ## *4.3 Worker Data Interfaces Attribute Details*

1055 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.*

1056 This attribute indicates the smallest unit of data, in bits, in the message payloads, and
1057 thus is the "bit granularity" of the message payloads. When the actual interface width is
1058 known, this value helps in determining things like byte enables and endian packing.
1059 The default is 8 for typical byte oriented payloads. The actual width of the data
1060 interfaces are not allowed to be smaller than this. This term was used in preference to
1061 "ByteWidth" (used in WMemI) since it has more semantics than typically associated with
1062 the term "byte".

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

1063 This attribute further constrains the length of messages, indicating that the size of all
1064 messages are this multiple of the basic data value size. This value also helps in byte
1065 enable and endian packing determination for interfaces.

1066 An example might be that the data stream consists of pairs of 32 bit single float data
1067 values (thus the interface width cannot be less than 32), thus all messages are a
1068 multiple of two such values or 64 bits. The interface is allowed to be 32 bits or wider. If
1069 it is 64 bits wide, then all messages are full words. If it is 128 bits wide, then the last
1070 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*

1071 This attribute indicates whether message payloads consist of different sized data values
1072 or not. Then false, it indicates uniform data value sizes and thus enables infrastructure-
1073 supported endian adaptation.

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

1074 The maximum (or fixed) size of messages, expressed in data values, each of which is
1075 DataValueWidth in size. If not specified, the default is that all messages are single data
1076 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

1077 This attribute specified the number of different message types in the protocol. Opcodes
1078 are conveyed separate from message data, to indicate the message type. Typically
1079 different message types have different payload layouts. The default value of 1 is used
1080 when there is only one type of message in the stream.

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

1081 This attribute indicates whether the interface of the worker is a producer (TRUE) or
1082 consumer (FALSE).

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

1083 This attribute indicates whether all messages are the same size (FALSE, the default), or
1084 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.

1085 This attribute, when TRUE, indicates that the stream of messages may contain
1086 messages of zero length, requiring support for this in the actual interface. If all
1087 messages are zero length, and there is one opcode, the protocol is simply a stream of
1088 events with no content. When this attribute is true, it forces byte enables to be included
1089 in WSI.

1090 *4.4 Worker Data Interface OCP Profiles*

1091 Beyond the attributes above, there are several aspects of the OCP configuration and
1092 signal semantics that are common across Worker Data Interfaces: reset behavior and
1093 flow control.

1094 4.4.1 Worker Data Interface Reset Behavior

1095 Workers in the same FPGA communicate with each other via data (plane) interfaces.
1096 According to the reset propagation rules mentioned earlier, every worker must
1097 propagate its master reset (an input which is on its single WCI), to all of its data
1098 interfaces (as an output). Thus a worker that is being reset via its WCI asserts reset on
1099 all its data interfaces, using the master or slave OCP reset signal according to the
1100 master or slave role of its data interface.

1101 Conversely, a worker that is stoppable and suspended, or has just come out of reset
1102 itself (after its WCI reset has been deasserted), must accept a reset input on its data
1103 ports, indicating that its adjacent worker is being reset via its *WCI*. Such reset inputs at
1104 data interfaces will not be asserted when the worker's WCI commands it to **initialize** or
1105 **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.

1114   **5   Data Widths and Byte Enables in WSI, WMI, and WMemI profiles**

1115   In these three profiles, the width of the data path is defined in the WIP attributes by the
1116   implementer (by setting the DataWidth attribute). Also, all these profiles define a
1117   minimum unit of data such that the data path width must be a multiple of that minimum
1118   width. We call this minimum width the ByteWidth of the interface, which of course is
1119   commonly = 8 (for octets). For Worker Data Interfaces, we use a synonym:
1120   "DataValueWidth", to connote that this is related to message payloads (i.e. the width of
1121   the data values in the payload. Since the WMemI profile is not associated with message
1122   payloads, we simply use "ByteWidth" for it. Due to the requirements of OCP, and to
1123   keep simple cases simple for the implementer, we treat three cases differently for all
1124   these profiles:

1125   •   DataWidth is the same as ByteWidth

1126   •   DataWidth is a multiple of ByteWidth, and ByteWidth is 8 (bytes are octets)

1127   •   DataWidth is a multiple of ByteWidth, and ByteWidth is greater than 8

1128   There is no support for ByteWidths less than 8 when ByteWidth is not the same as
1129   DataWidth. Byte enables are only relevant to writes in these profiles.

1130   *5.1   DataWidth equals ByteWidth*

1131   This simple case sets the normal OCP data field (the data_wdth configuration
1132   parameter for the MData and SData signals). There is no need for byte enables, and no
1133   need for the complexity of the OCP "datainfo" signals. There is no WIP or OCP
1134   constraint on the value of DataWidth.

1135   *5.2   DataWidth is a multiple of ByteWidth, and ByteWidth is 8 (bytes are octets)*

1136   This case is fairly simple and common, and well supported by OCP. Write byte enables
1137   are enabled in the interface unless, for WSI, the message size granularity implies that
1138   they would never be needed (i.e. all words are always "full", not partial).

1139   OCP dictates that each byte enable signal corresponds to exactly 8 bits of the OCP
1140   MData path, and that path must be a multiple of 8 bits if byte enables are used.

1141   *5.3   DataWidth is a multiple of ByteWidth, and ByteWidth is greater than 8*

1142   To properly handle this case, OCP dictates that the data be split into two separate
1143   signal fields, {M,S}data and {M,S}datainfo. 8 bits of each byte are packed into the OCP
1144   {M,S}Data field, and the remaining bits of each byte are packed into the {M,S}DataInfo
1145   field. For example, if the WIP logical DataWidth was 36, and ByteWidth was 9, then the
1146   OCP MData field would carry logical bits <34:27><25:18><16:9><7:0>, and the OCP
1147   MDataInfo path would carry logical bits <35><26><17><8>.

1148   Thus the worker implementer (and possibly automation tools) might map the MData and
1149   MDataInfo fields into a single 36 bit Data field internal to the implementation for clarity
1150   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.

1190 Workers using the WSI may want the ability to signal as early as possible, before data,
1191 the impending arrival of a message. The EarlyRequest attribute is used to enable this
1192 optimal latency behavior. Masters (producers) are free to insert intra-message idle data
1193 cycles either by driving MCmd=IDLE or (when EarlyRequest = TRUE) by de-asserting
1194 MDataValid, unless the Continuous attribute is set.

1195 When ImpreciseBursts are being used, and the Abortable attribute is set, the
1196 producer/master can abort a frame, indicating to the slave that the message should be
1197 discarded. This enables the typical cut-through optimization in some cases.

1198 In all cases the default OCP burstsinglereq=0 semantics apply: there will be exactly as
1199 many valid request beats as there are valid data words. WSI only allows FIFO access
1200 to data. Use WMI when random or out-of-order access to message contents is
1201 required, either as producer or consumer.

6.3 WSI Attributes

1203 Worker authors using a WSI specify their implementation choices via the following
1204 attributes (as well as the data protocol attributes above), from which all OCP
1205 configuration parameters and signals are derived.

1206 The Clock and ClockSignal attributes, described in section 2.6, are applicable to WSI.

1207 Since WSI is a data plane interface, the worker data interface protocol attributes are
1208 applicable, as described in section 4.2. These are: DataValueWidth,
1209 DataValueGranularity, DiverseDataSizes, MaxMessageValues, NumberOfOpcodes,
1210 Producer, VariableMessageLength, and ZeroLengthMessages.

1211 ### 6.3.1 WSI Attribute Summary

1212 <div align="center">**Table 12 – WSI Attribute Table**</div>

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*	*8*	*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 the start of message.*

1213 ### 6.3.2 WSI Attribute Details

1214 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

1215 The Continuous attribute indicates *continuous* data bursts between master and slave,
1216 meaning that once burst data transfer begins, data is transferred on every cycle without
1217 interruption, either by the master (no idle cycles) or the slave (no flow control via
1218 SThreadBusy). This attribute only refers to *intra*-burst continuity and implies nothing
1219 about *inter*-burst behavior.

1220 Continuous=TRUE indicates, for producers: "Once I get started and allowed by flow
1221 control at the start of a burst, I will produce all the data in one continuous burst with no
1222 idle cycles and cannot accept any flow control (backpressure)". For consumers: "Once I
1223 receive the first data in a burst, I require and will accept all the data in one continuous
1224 burst, without idle cycles, and without asserting SThreadBusy until after the last request
1225 in the burst."

1226 This attribute does not change the OCP configuration or any of the signals in the
1227 interface. It expresses two aspects of the data transfer: *data will fill every cycle,* and no
1228 *data flow-control will occur within the burst*. This attribute is common to the WSI, WMI,
1229 and WMemI profiles.

DataWidth	unsignedInt	8	Physical width in bits/wires of the data path.

1230 The DataWidth attribute is the physical width (word size) of the interface in bits. It must
1231 be a multiple of the DataValueWidth attribute. When the
1232 DataValueWidth*DataValueGranularity is smaller than the DataWidth, and not a multiple
1233 of 8, OCP requires the data path to be split between two OCP fields (MData and
1234 MDataInfo). Octet-oriented or word-oriented interfaces never have this complexity.
1235 When DataValueWidth*DataValueGranularity is smaller than DataWidth, byte enables
1236 are required in the interface.

1237 This attribute is common to the WSI, WMI, and WMemI profiles.

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

1238 ByteWidth expresses the size in bits of the smallest data value that the interface needs
1239 to handle, and is used to enable byte enable functionality in WSI, WMI, and WMemI.

1240 For WSI, ByteWidth is a derived based on data protocol attributes. It is not specified
1241 directly. For WSI, when ByteWidth is less than DataWidth, byte enables are enabled to
1242 indicate the actual valid data in the last word messages (and, when
1243 ZeroLengthMessages = TRUE, that there is no data in the last word).

1244 Thus byte enables are only required in WSI when the messages may have zero length,
1245 or may contain a number of values that do not fill the last DataWidth word of the
1246 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

1247 These burst attributes indicate what sort of burst transactions will be issued by the
1248 worker, as OCP master, or accepted by the worker as OCP slave. These attributes are
1249 common to WSI, WMI and WMemI WIPs.

1250 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.

1251 Abortable specifies that the producer may abort a message before it is fully transmitted,
1252 and that the consumer can handle and respect such an action. This is an important
1253 optimization when both sides are able to do this without extra buffering. It is analogous
1254 to the cut-through routing optimization in switches. The semantic is similar to the
1255 "discontinue" action in the Xilinx LocalLink Interface. It is optional since it has a modest
1256 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.

1257 The attribute EarlyRequest is an implementation choice that allows, when TRUE, the
1258 WSI to convey message arrival (a.k.a. "start of message") in advance of the first data in
1259 the message. In general, this is a latency improvement: you can start a message
1260 earlier.

1261 For consumers, use EarlyRequest=TRUE in situations where the early indication of
1262 message arrival has value, typically to reduce latency. For producers, pass along this
1263 information if it is available: i.e. of the producer knows about a message before it has
1264 the first data of the message, it should pass this potentially latency-improving
1265 information to the consumer.

1266 When EarlyRequest = FALSE, the first data word of a message is required to be
1267 available at the same time as the first OCP request of the message. Secondarily, data
1268 is supplied at the same time as the OCP request for *all* words of the message.

1269 When EarlyRequest=TRUE, the interface uses the OCP defined behavior of
1270 datahandshake=1 to decouple the request from the data. This allows the producing-
1271 master to send OCP requests ahead of (or at the same time as), the corresponding
1272 data.

1273 Since OCP datahandshake behavior is thus optional, single-request-multiple-data
1274 (SRMD) bursts are not allowed for WSI. All bursts are always MRMD.

6.4   WSI OCP Profile/Configuration Parameters

1276 The WSI OCP profile is the aspect of the WSI that has to do with OCP configuration
1277 parameters and signal configuration. This section describes the concrete, complete,
1278 and unambiguous transformation from WSI WIP attributes described above, to the OCP
1279 configuration parameters inferred from those attributes. This transformation is
1280 implemented in automation tools.

1281 The table below lists only those parameters that are different from the OCP
1282 configuration parameter defaults (per the OCP specification).

1283

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:* *N = ceil(MaxMessageValues*DataValueWidth/DataWidth)* *If (N < 4) P = 2* *else P = floor(log$_2$(N)) + 1* *If PreciseBurst =FALSE:* *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: 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	{0|1}	*If Abortable: 1, Else: 0*
mflag_wdth	{0|1}	*If Abortable: 1*
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(log$_2$(NumberOfOpcodes))*
sthreadbusy	1	*Note: SThreadBusy signal supports pipelined, proactive, flow-control backpressure from slave to master.*
sthreadbusy_exact		
sthreadbusy_pipelined		

1284 ## 6.5 WSI OCP Signals

1285 Signal configuration is determined by the above interface configuration parameters as
1286 called out in the "OCP Signal Configuration Parameter" table in the OCP specification.
1287 The tables below show the signal from master-to-slave and slave-to-master. The
1288 positional order of signals is alphabetical.

1289 Light yellow shading of the description indicates where an OCP signal has added WSI
1290 semantics. Details described in following section.

1291   **Table 14 – WSI OCP Signals Driven by the Producer/Master**

OCP Signal	When Included	Width	Usage
MBurstLength	always	If PreciseBurst = TRUE: $ceil(log_2(MaxMessageValues*$ $DataValueWidth/$ $DataWidth))$ 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	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	DataWidth – (8 * DataWidth/ByteWidth)	Extra write data not allowed in MData per OCP
MDataLast	EarlyRequest = TRUE	1	Indicates last beat of data
MDataValid	EarlyRequest = TRUE	1	Qualifies active data beats
MFlag	Abortable= TRUE	1	WSI Specific Semantics: Abort Frame (in last request in burst)
MReqInfo	NumberOf-Opcodes > 1	$Ceil(log_2(NumberOfOpcodes))$	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)

1292   **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

1293   ***6.6   WSI Semantics***

1294 This section describes specific semantic signal meanings in WSI that are layered on top
1295 of the fundamental OCP signal semantics. The semantics stay within OCP compliance,
1296 so that even though they are additional behavioral constraints, they represent valid
1297 OCP-defined behavior.

1298 6.6.1 Opcode Indication on MReqInfo

1299 When there is more than one message op-code, the MReqInfo signal is configured into
1300 the interface. This extra information about the message, its opcode, must be constant
1301 over the message/burst.

1302 6.6.2 Messages (or "frames") are OCP Bursts

1303 Messages are transmitted as OCP bursts, with the start and end of a message being
1304 the first and last word of a burst. When PreciseBurst = TRUE, the interface is
1305 configured statically for precise bursts (MPreciseBurst tied off to 1 in the master – the
1306 OCP default, ignored in the slave), otherwise it is configured for imprecise bursts
1307 (MPreciseBurst tied off to 0 in the master, ignored in the slave). In all cases the master
1308 asserts MReqLast with the last request of the burst. This means that slaves, whether
1309 precise or imprecise, can use this direct mechanism to determine the last request in the
1310 burst in all cases, without counting. The master of an imprecise burst must also drive
1311 the value of "0b01" in the MBurstLength field of the last request of the burst for OCP
1312 compliance, but this is in fact redundant with MReqLast.

1313 When the EarlyRequest attribute = TRUE, the end of a message can also be
1314 determined (precise or imprecise) with the MDataLast signal associated with the data.
1315 When EarlyRequest = TRUE, the first request can be supplied before the first data is
1316 available, to improve message startup latency in some cases.

1317 Interfaces configured with MaxMessageValues = 1 essentially have single word bursts,
1318 and the MBurstLength signal is constant (0b01).

1319 6.6.3 Byte enables

1320 Byte enables are configured and used (when necessary) to indicate partial word content
1321 in the last word of the message/burst. When they are included, they must be all ones
1322 for all words in the message except the last. They are only included when needed,
1323 when it is known from the data protocol attributes that messages may have lengths that
1324 are not a multiple of DataWidth or are zero.

1325 Also, the use of byte enables must be consistent with the endianness of the data
1326 interface. If the interface is big endian, then partial words will have low order byte
1327 enables set to zero as appropriate (e.g. if there is one valid byte, it must be the high
1328 order byte). If the interface is little endian, then partial words will have high order byte
1329 enables set to zero as appropriate (e.g. if there is one valid byte, it must be the low
1330 order byte). A typical example would be when input data was stored into an internal
1331 buffer, using the byte enables. At this point the worker's endianness would determine
1332 which bytes in memory are valid, since the byte enables are only valid during the write
1333 to internal memory, and are not necessarily captured anywhere.

1334 If the data protocol requires support for zero length messages (ZeroLengthMessages),
1335 byte enables are also included and may be all zero in the last or only word of a zero
1336 length message. Note that enabling ZeroLengthMessages in a WSI *also* enables the
1337 last word of non-zero-length messages to be empty. In particular, this allows a
1338 producer to signal the end of a message *without* data (and thus without knowing if any
1339 word is in fact the last one). This is useful for a producer what wants to pass every
1340 word of data to the consumer as soon as it is available, without needing to wait until it
1341 knows whether that word is in fact the last one or not (in the case of imprecise bursts).

1342 Thus the byte enable logic of a WSI is constrained beyond OCP, and all WSI
1343 transactions are OCP compliant.

1344 6.6.4 Aborting Messages

1345 When the interface is using ImpreciseBursts, and the Abortable attribute is set, a feature
1346 is enabled allowing the master to terminate a burst and indicate that the message is
1347 being "aborted", which indicates to the slave that the message should not be processed,
1348 and should be discarded.   This indication is placed on MFlag[0], which is sampled by
1349 the slave in the request phase.

1350 This feature is useful to reduce latency and buffering in some cases. An example is a
1351 "framer" worker that is extracting frames with CRC checks from a continuous stream. If
1352 it starts receiving a frame, it doesn't know whether the frame is valid until the CRC is
1353 checked. Since the common case is that the CRC is valid, it wants to forward the
1354 beginning of the frame to its output as soon as possible and not delay transmission (and
1355 use buffering) waiting until the CRC. In the rare case that the CRC is bad, it wants to
1356 abort the frame and start scanning for a new frame. If the downstream worker is
1357 already buffering the frame (e.g. when it is waiting to DMA a whole frame over a bus),
1358 then it can easily implement the "abort" feature with no extra buffering, since it is
1359 buffering already.

1360 If the downstream worker cannot handle aborting a message, then a buffer can easily
1361 be inserted between the two workers.

6.7   WSI Infrastructure Coping

1363 Infrastructure IP supporting WSI interfaces (attached to application IP) falls into three
1364 categories:

1365 • Adapter IP for interposing between workers that do not have directly interoperable
1366   data plane interfaces, but are otherwise logically, locally, connected.

1367 • Communications IP when workers are not in the same device and need the data
1368   messages conveyed across some inter-chip interconnection or fabric (e.g. PCI,
1369   RapidIO, LVDS parallel, Aurora serial, XAUI serial, etc.

1370 • I/O IP when workers are receiving or sending IP to an I/O device (sink or source of
1371   data) directly attached to the device (e.g. ADC/DAC).

1372 The first category (adapters) does the most "coping" as that is their job: to enable
1373 interoperability when worker developers make different attribute choices. Essentially
1374 adapters convert between choices like:

1375 • Early Request or not

1376 • Fixed or Variable size messages

1377 • Precise or Imprecise Bursts

1378 • Different width data paths

1379 • Continuous data or not.

1380 • Abortable or not.

1381 • Different Endian

1382 Due to the design of the WIPs, interoperability is well defined in nearly all cases, and
1383 trivial to accomplish (by the infrastructure) in many cases. Burst conversions

1384 sometimes are simple logic, sometimes require counting, and sometimes require
1385 buffering.

1386 **Table 16 – WSI Slave Infrastructure Handling of Optional WSI Signals**

OCP Signal	When Signal would be absent at the <u>master</u>	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]
MFlag	Abortable = FALSE	OCP Tieoff rules apply: Abort never happens, MFlag[0] = 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).

1387 **Table 17 – WSI Master Infrastructure Handling of Optional WSI Signals**

OCP Signal	When Signal would be absent at the <u>slave</u>	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
MFlag	Abortable = FALSE	OCP Tieoff tules apply: Abort never is asserted, MFlag[0] = 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).

1388 These tables above assume an infrastructure that supports all features (byte enables,
1389 early request/data handshake, and opcodes). Conversion between precise and
1390 imprecise burst lengths is slightly more complicated, but well defined.

1391 Note that when infrastructure is conveying messages between application workers,
1392 interoperability can be determined even when the infrastructure supports features that
1393 the application worker does not. A good example is if an infrastructure master/producer
1394 generically supports 256 opcodes, but the application slave/consumer does not. Here
1395 interoperability checking is based on knowing that the upstream application worker
1396 (wherever it is), also doesn't produce opcodes, so the infrastructure master will never in
1397 fact drive the opcode signals to anything other than the tieoff. If the infrastructure was a
1398 data source rather than an interconnect between workers, then interoperability would be
1399 based on the infrastructure itself.

1400 ### 6.7.1 Infrastructure Best Practice for Master/Producers

1401 Infrastructure IP acting as WSI producer/master should:

1402 • use PreciseBurst = TRUE unless this adds buffering or latency.

1403 • use EarlyRequest = FALSE unless this adds latency (i.e. when the infrastructure IP
1404   actually does know that a request should start before it has the data).

1405 • use VariableMessageLength = FALSE when it knows that all messages it produces
1406   will be the same size.

1407 • Use Continuous = TRUE when it knows that bursts it produces will never have idle
1408   cycles.

1409 • Use Abortable = TRUE if the data source supports aborting (defer buffering until
1410   needed)

1411 ### 6.7.2 Infrastructure Best Practice for Slave/Consumers

1412 • Support both precise and imprecise bursts, use MReqLast rather than
1413   MBurstLength, unless burst length is in fact required to be known early (in which
1414   case an adapter should add the imprecise-to-precise conversion when needed).
1415   Using both precise and imprecise bursts is not legal for an application worker
1416   producer in any case, but an infrastructure consumer may indeed support both.

1417 • Support EarlyRequest = TRUE.

1418 • Support VariableMessageLength = TRUE unless its function in fact requires fixed
1419   message sizes.

1420 • Use Continuous = FALSE, allowing intra-burst idle cycles on input.

1421 • Use Abortable = TRUE if the data sink does, or if buffering is already taking place

1422

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.

1462   When a worker is a message-producer, it is always an OCP master and must write, or
1463   "push", data to a slave. The infrastructure supplies this slave message-consumer
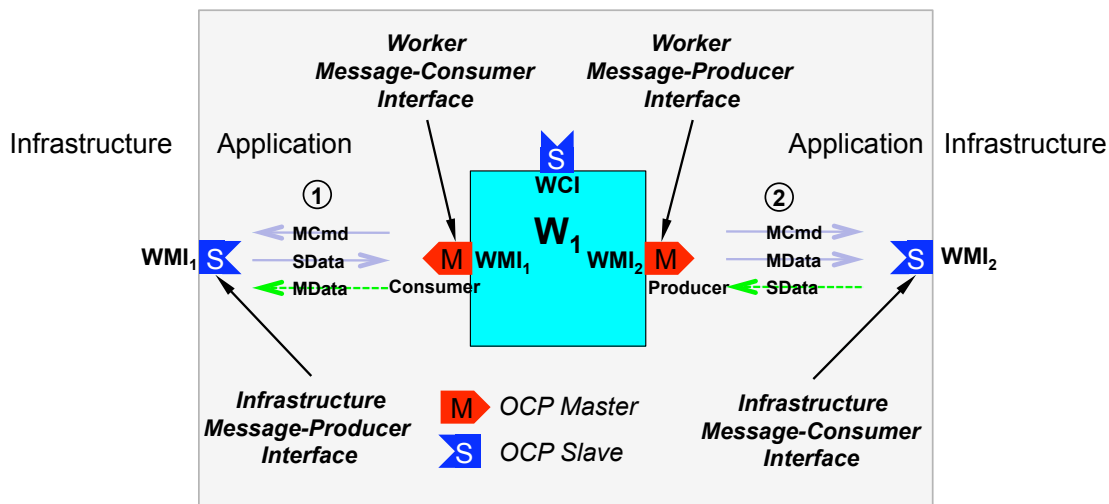1464   interface as shown below on the right side of the link marked 2.

1465

1466   **Figure 6 - Signals on Worker with two WMI interfaces**

1467   For both worker-consuming and worker-producing, WMI provides the symmetric
1468   capabilities of slave buffer availability (to process on input, to fill on output) and a "done
1469   with message" indication.

1470   For the worker-consumer:

1471   •   Slave/infrastructure indicates source buffer availability (source buffer ready to
1472   consume)

1473   •   Master/worker indicates "Done with Message" (discard/recycle the consumed
1474   source buffer)

1475   For the worker-producer:

1476   •   Slave/infrastructure indicates target buffer availability (target buffer ready to fill
1477   with message)

1478   •   Master/worker indicates "Done with Message" (transmit the finalized contents of
1479   the output buffer)

1480   Although the attributes are identical for consumer and producer, the OCP configuration
1481   parameters and signals differ owing to the fact that the worker is in an OCP Master role
1482   in both cases.

1483   7.2.2   Indication of message length

1484   Since in WMI messages are *not* bursts, the length of a message must be indicated
1485   independent of burst length (unless VariableMessageLength = FALSE). For a WMI
1486   consumer, both the opcode and the message length are supplied separately from data
1487   using the OCP SFlag signals. For the WMI producer, the opcode and message length
1488   are indicated on the OCP MFlag signals (again, unless VariableMessageLength =
1489   FALSE). The exact semantics are discussed below.

1490 7.2.3 Indication of done-with-message

1491 Since the consumer can randomly access the contents of a message, there is no
1492 implicit way for the slave to know when the consumer is done processing the message.
1493 Thus, during any request to access the current message (input or output), the OCP
1494 MReqInfo[0] signal is used to indicate whether that access represents the last access
1495 required to complete the processing (input or output) of the message. This indicates,
1496 for input, that the message can be discarded and the buffer reused, and for output, that
1497 the buffer can be made available to transmit to the directly (locally) or indirectly (remote)
1498 consuming worker or output device ("downstream").

1499 Since the worker may not know that it is done processing a message until after it has
1500 performed all required data accesses for the contents of a message, there is a provision
1501 for making a request that accesses no data at all, but simply allows the done-with-
1502 message indication with no overhead for accessing memory. The exact semantics are
1503 discussed below. This indication is the same whether producing or consuming.

1504 This scheme enables two optimizations: no *extra access* (OCP transaction) is required
1505 to complete the message, and no *data delay* is required to wait until it is known that an
1506 access will be the last one. The mechanism is identical for consuming and producing.

1507 7.2.4 Indication of buffer available

1508 When the worker is "between" messages (before the first one, or after indicating "done"
1509 with the previously processed one), the OCP SThreadBusy signal acts as the indication
1510 that a buffer is available, for both consuming and producing. This indication is the same
1511 whether producing or consuming. It uses OCP semantics directly.

1512 7.2.5 Message content addressing

1513 WMI supports the worker, as OCP master, accessing message contents via random
1514 access reads and writes. Address 0 is always the first word of the "current" message.
1515 There is an additional "talkback" option, which allows the worker to write to input buffers
1516 or read from output buffers. This option enables read/write access to buffers (rather
1517 than read-only input buffers and write-only output buffers), with the associated hardware
1518 cost, but can reduce overall buffer requirements.

1519 *7.3 WMI Worker Attributes*

1520 Worker authors using a WMI specify their implementation choices via the following
1521 attributes (as well as the data protocol attributes above), from which all OCP
1522 configuration parameters and signals are derived. Light green shading of the description
1523 indicates where the attribute has the same meaning as in WSI.

1524 The Clock and ClockSignal attributes, described in section 2.6, are applicable to WMI.

1525 Since WMI is a data plane interface, the worker data interface protocol attributes are
1526 applicable, as described in section 4.2. These are: DataValueWidth,
1527 DataValueGranularity, DiverseDataSizes, MaxMessageValues, NumberOfOpcodes,
1528 Producer, VariableMessageLength, and ZeroLengthMessages.

1529   7.3.1 WMI Attribute Summary

1530   **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	8	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

1531   7.3.2 WMI Worker Attribute Details

1532   This subsection contains the per-attribute details of choices available for a WMI on a
1533   worker.

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

1534   See WSI Attribute Details.

DataWidth	unsigned Int	8	Physical width in bit/wires of data path (same as WSI)

1535   See WSI Attribute Details.

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

1536   ByteWidth expresses the size in bits of the smallest data value that the interface needs
1537   to handle, and is used to enable byte enable functionality in WSI, WMI, and WMemI.

1538   If the WMI, when it is writing to buffers, requires byte enable support, set ByteWidth to a
1539   value that divides into DataWidth. Normally this would be DataValueWidth, but that is
1540   not required. Note that byte enables in WMI are never needed to express message
1541   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

1542   See WSI Attribute Details.

1543   For WMI:

1544 • There is no relationship between bursts and messages.

1545 • One, but not both, of these attributes must be TRUE (same as WSI).

1546 • Maximum message length determines maximum burst length, but bursts can be
1547   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*

1548 When TRUE, this attribute configures a particular WMI interface to have both read and
1549 write data capability. For a consumer, this attribute enables write-back to the input
1550 buffer, in addition to the default message read capability. For a producer, this attribute
1551 enables read-back from the output buffer, in addition to the default message write
1552 capability. When FALSE, input buffers can only be read, and output buffers can only be
1553 written.

1554 This capability can obviate the need for workers to have extra internal buffers or use
1555 external memories.

1556 *7.4   WMI OCP Profile/Configuration Parameters*

1557 The WMI OCP profile is the aspect of the WMI that has to do with OCP configuration
1558 parameters and signal configuration. This section describes the concrete, complete,
1559 and unambiguous transformation from WMI WIP attributes described above, to the OCP
1560 configuration parameters inferred from those attributes. This transformation is
1561 implemented in automation tools.

1562 The table below lists only those parameters that are different from the OCP
1563 configuration parameter defaults (per the OCP specification).

1564 ## 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

1565 ## 7.5 WMI OCP Signals

1566   Signal configuration is determined by the interface configuration parameters and as
1567   called out in the "OCP Signal Configuration Parameter" table in the OCP specification.
1568   The tables below show the signal from master-to-slave and slave-to-master. The
1569   positional order of signals is alphabetical.

1570   Light yellow shading of the description indicates where an OCP signal has had WMI
1571   semantics added. Details described in following section.

1572

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)

1573

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

1574

1575 This section describes specific semantic signal meanings in WMI that are layered on top
1576 of the fundamental OCP signals. They are organized by function and then the signals
1577 affected.

1578 7.6.1 "Done with Message" (DWM) Indicated on MReqInfo[0]

1579 MReqInfo[0] adds information to the current OCP command request (to read or write
1580 message contents) to indicate that the data access associated with this command is the
1581 final access of the associated message. It is sent in this fashion to eliminate the need
1582 for an extra OCP request to indicate this.

1583 This mechanism is used for both consuming and producing workers. The associated
1584 data access can be suppressed (possibly reducing latency) by using the no-data
1585 indication via MAddrSpace[0] described next.

1586 7.6.2 No-Data Indication on MAddrSpace[0]

1587 WMI transactions initiated by a worker that include the DWM indication, either consumer
1588 or producer, may also indicate that the OCP command, read or write, has "No Data" by
1589 setting the MAddrSpace[0] signal to '1'. This allows a master to send to the slave the
1590 Done-with-Message (DWM) on MReqInfo[0] without any read or write side effects or
1591 overheads. Thus when the worker decides it is done with the current message, but
1592 requires no additional data accesses, it can issue the DWM request with no data access
1593 required by the infrastructure.

1594 Normal transactions to read and write "Data" require that MAddrSpace be set to '0'.
1595 Setting MAddrSpace[0] = 1 when MReqInfo[0] = 0 should not be done.

1596 7.6.3 Buffer Availability Indicated on SThreadBusy

1597 When between messages (either after reset or after the master indicates "done with
1598 message"), SThreadBusy is used to indicate the availability of a source data buffer to
1599 read (worker consumer) or an output data buffer to write (worker producer). An asserted
1600 SThreadBusy[0] signal blocks the worker from issuing an OCP request. When not
1601 between messages, SThreadBusy is simply the backpressure for reading or writing
1602 message data. For a consumer, the infrastructure side (slave) may allow access as data
1603 arrives (for improved latency), and thus latency could depend on the address of that first
1604 access to a (new) message.

1605 As an additional constraint on OCP, the master *cannot* assert a request in the cycle
1606 after the DWM is asserted, in order to allow the slave to assert the (pipelined)

1607   SThreadBusy for at least one cycle between messages. Thus the worker must not
1608   issue a request based on the normal pipelined SThreadBusy from the slave _or_ the
1609   MReqInfo[0] in the previous cycle. For the purpose of this non-blocking flow control,
1610   both signals are sampled at the same clock: at the end of the cycle before the request is
1611   to be asserted.

1612   7.6.4 Opcode and Message Size Move Downstream on {M|S}Flag

1613   Opcodes and message size flow downstream. They are driven together on MFlag and
1614   SFlag as shown in the following diagram. A worker message-consumer _observes_ these
1615   on SFlag and a worker message-producer _drives_ them on MFlag. They are only
1616   configured when NumberOfOpcodes > 1 or VariableMessageLength = TRUE.
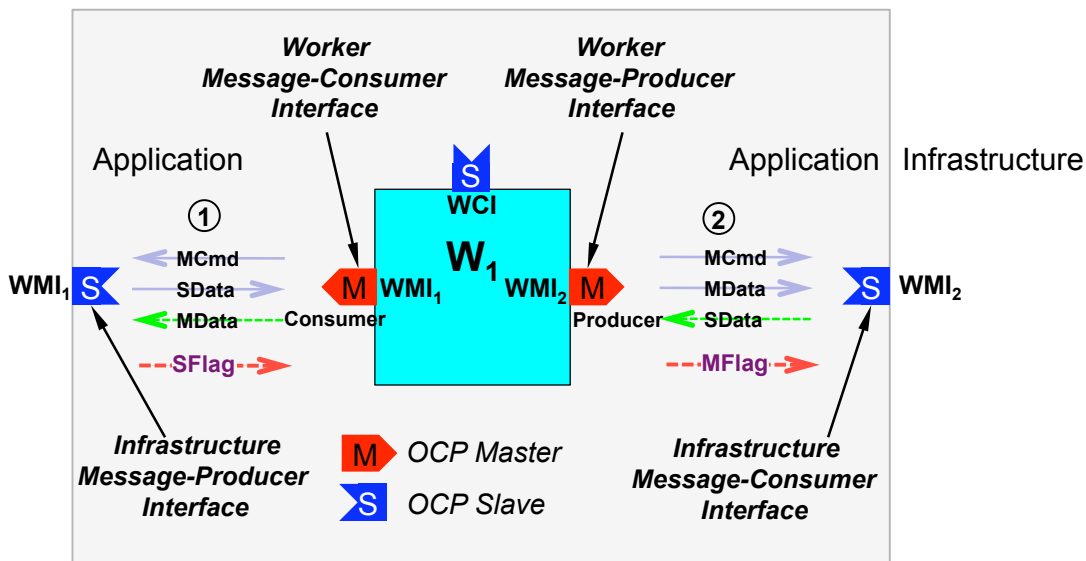
Encoding of Opcode and Message Length on both SFlag and Mflag:

P	..	8	7	..	0

Message Length   **Opcode**

P = Number of Message Length bits + 7
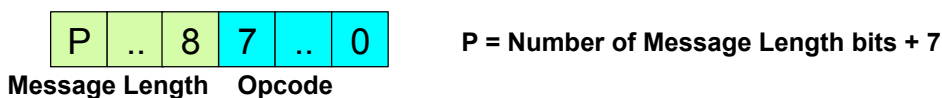
1617

1618   **Figure 7 WMI Use of SFlag/MFlag**

1619   When the field is configured, the opcode is driven on {S|M}Flag[7:0]. When
1620   VariableMessageLength = TRUE the message length is driven on {S|M}Flag[P:8],
1621   where $P = ceil(log2(MaxMessageValues+1)) + 7$.

1622   For a worker _consumer_, this information is driven on SFlag by the infrastructure and the
1623   values will be valid for a given message starting in the first cycle when SThreadBusy is
1624   not asserted (after reset or the previous DWM), through the cycle when the worker
1625   asserts DWM (MReqInfo[0] = 1). Thus the worker may not need to capture them.

1626   For a worker _producer_, this information is driven on MFlag by the worker and the values
1627   must be valid in the (single) same cycle as the request with DWM asserted.

1628 **_7.7 WMI Infrastructure Coping_**

1629 The key options chosen by the worker for WMI are data width, burst capabilities, and,
1630 like WSI, the continuous attribute. Some of these attributes allow for infrastructure
1631 optimization, such as when the TalkBack option is FALSE, more buffer sharing is
1632 possible (multiple consumers could share the same input buffer).

1633 **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]

1634 7.7.1 Infrastructure Best Practice for Slave Producers (feeding Worker Consumers)

1635 • Support CONTINUOUS when the buffer memory can support it.

1636 • Support both precise and imprecise bursts, use MReqLast rather than
1637 MBurstLength.

1638 • Support Byte Enables on Writes.

1639 • Support VariableMessageLength = TRUE.

1640 • Support 8 bits of opcode.

1641 • Have a variant IP to support TalkBack (since it might be costly)

1642 7.7.2 Infrastructure Best Practice for Slave Consumers (fed by Worker Producers)

1643 • Support CONTINUOUS when the buffer memory can support it.

1644 • Support both precise and imprecise bursts, use MReqLast rather than
1645 MBurstLength.

1646 • Support Byte Enables on Writes.

1647 • Support VariableMessageLength = TRUE.

1648 • Support 8 bits of opcode.

1649 • Have a variant IP to support TalkBack (since it might be costly)

1650 [Issue: packing non-power-of-two for DMA: map each byte to round up to 2^N?]

1651 8   Worker Memory Interface (WMemI)

1652 8.1   WMemI Motivation

1653 WMemI is an interface that allows workers to access memory in a technology
1654 independent fashion, with enough choices to make workers simple and also allow for
1655 high performance use of local memories. It exists to enable workers to express their
1656 needs for FPGA-attached (or FPGA-internal) memory based on their own requirements
1657 and preferences. Since explicitly buffered data plane communication is handled by
1658 WMI, WMemI is focused on memory usage for non-communication purposes such as
1659 internal history buffers and LUTs.

1660 Since memory usage ranges from latency-optimized non-burst LUT-style random word
1661 access, to throughput optimized burst-style sequential access, the WMemI attribute
1662 choices enable this range of behavior.

1663 The intended purpose of WMemI is *not* for a worker to communicate via memory to any
1664 other worker, since that is the job of WSI or WMI, which may have underlying memory
1665 used for buffering. An example of this is shown earlier in the WMI section describing
1666 master-to-master adaptation.

1667 8.2   WMemI Overview

1668 Typical FPGA-attached memory scenarios include both SRAM and DRAM[4]. WMemI
1669 balances between abstracting enough away so that workers may be portable across
1670 heterogeneous memory architectures; vs. exposing enough of the micro-architectural
1671 details when "ultimate performance" is required.

1672 Workers may have zero or more WMemIs, each with their own attributes.

1673 The figure below shows in isolation a (WMemI) interface (1) between Worker1 and
1674 MemController1. The worker specifies the WMemI attributes it desires at master
1675 interface (1M), such as data width and memory capacity. The infrastructure satisfies
1676 those requests on the other side on slave interface (1S). The worker side of a WMemI is
1677 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.
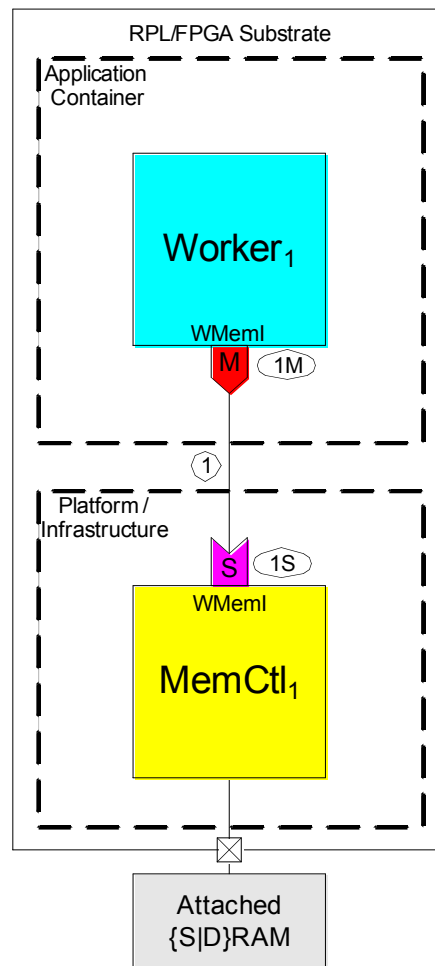
Figure 8 - Master/Slave Roles for WMemI

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:
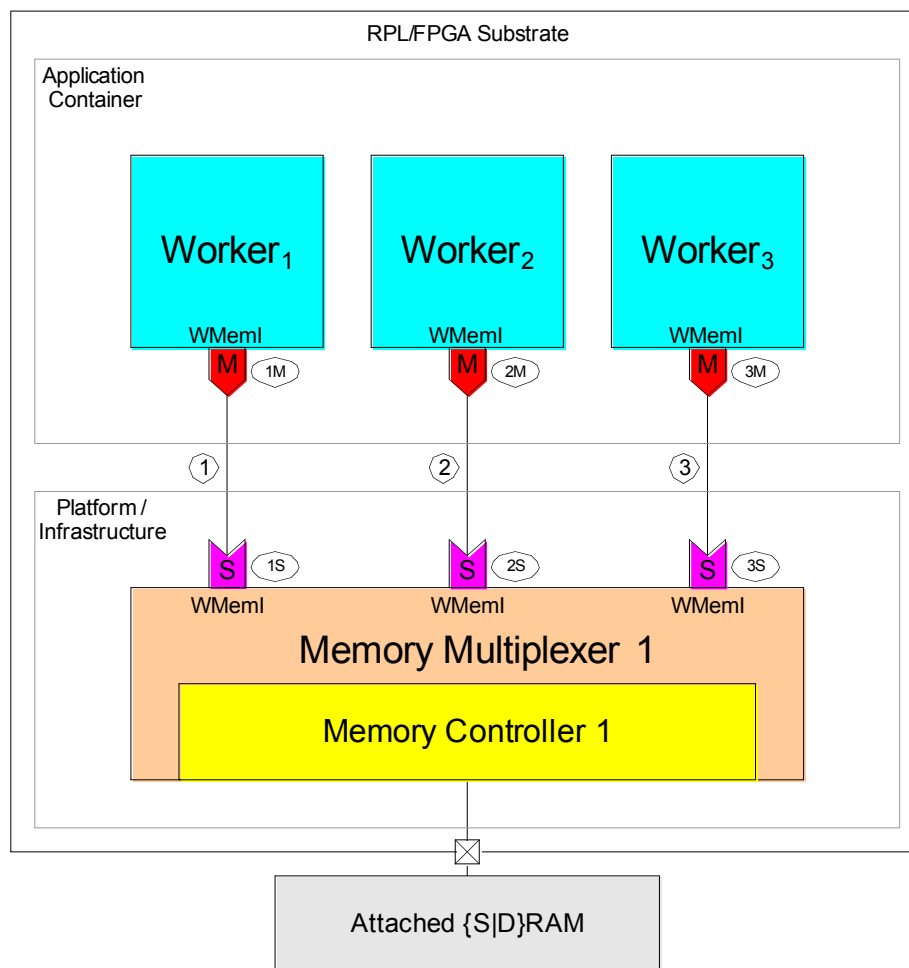
Figure 9 - Memory Multiplexer with WMemI

1685 Each of the workers shown above may have different FPGA-attached memory
1686 requirements, and would express those as unique WMemI attributes associated with
1687 each WMemI. Because of these differences the three WMemI interfaces may have
1688 different OCP configuration parameters. Thus, in the general case, the worker masters
1689 (1M, 2M, 3M) are loosely constrained for data-width and capacity; and the infrastructure
1690 slaves (1S, 2S, 3S) must conform or be adapted to them as the worker attributes are
1691 generating the OCP configuration parameters. Furthermore, the memory multiplexer
1692 must adapt these potentially disparate requests to a common access method for the
1693 memory controller.

1694 The most significant choice made in specifying a WMemI is the use of bursts, which
1695 adds complexity to the interface (e.g. pipelined requests) and determines several
1696 significant differences in signal configuration (e.g. how byte enables are specified).

1697 The key choices made when a worker specifies a WMemI are:

1698 • **Burst usage**: does the worker want burst-style (throughput optimized) access or
1699   SRAM-style latency-optimized single word access? How long will bursts be?

1700 • **Burst style**: will the worker issue precise or imprecise bursts? Or *both*?

1701 • **Intra-burst flow control**: will the worker handle flow control within a burst (after it is
1702   started)? Will it use (issue) flow control against the read data in a burst read?

1703 • **Memory width and size**: how much memory is required, and how wide should it
1704   be? Should it support bytes of some width (byte enables on writes)?

1705 WMemI is specified using blocking request flow control (where the master issues the
1706 request and waits for it to be accepted) rather than the non-blocking flow control used in
1707 the other profiles (where the master waits for the not-busy indication before it can issue
1708 the request). This is because memory behavior (and the delay in processing requests)
1709 frequently depends on the address associated with the request (i.e. same bank/row or
1710 not).

1711 ### 8.3   WMemI Worker Attributes

1712 Worker authors using a WMemI specify their implementation choices via the following
1713 attributes, from which all OCP configuration parameters and signals are derived. Light
1714 green shading of the description indicates where the attribute has the same meaning as
1715 in WMI.

1716 The Clock and ClockSignal attributes, described in section 2.6, are applicable to
1717 WMemI.

1718   8.3.1   WMemI Attribute Summary

1719   **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*

1720   8.3.2   WMemI Attribute Details

1721   This subsection contains the per-attribute details of choices available for a WMemI on a
1722   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.*

1723   These attributes are as described under the WMI profile, except that it is allowed to
1724   configure a WMemI such that the master (worker) may issue _both_ precise and imprecise
1725   bursts, or neither. MaxBurstLength (only for WMemI) is the maximum number of words
1726   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.*

1727   DataWidth and ByteWidth are as in WSI and WMI (data path width in bits). ByteWidth
1728   expresses the byte enable capabilities for writes. There are no byte enable capabilities
1729   in WSI, WMI or WMemI for reads. Memory is sized in words, with MemoryWords
1730   expressing the amount of memory available (or required), starting at address zero.
1731   MemoryWords implies the width of the address path. When ByteWidth equals
1732   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

1733 The WriteDataFlowControl attribute indicates if the worker will support and accept intra-
1734 burst flow control on memory writes. The default (FALSE) indicates that the worker
1735 cannot support any intra-burst backpressure, and that once a burst requests is
1736 accepted, all the data in the burst will be accepted. Only valid when bursts are used.

1737 The ReadDataFlowControl attribute indicates if the worker will assert intra-burst flow
1738 control on burst reads. The default (FALSE) indicates that the worker will not assert any
1739 intra-burst backpressure, and that it is prepared to accept all the data it requests. Only
1740 valid when bursts are used.

8.4   WMemI OCP Profile/Configuration Parameters

1742 The WMemI OCP profile is the aspect of the WMemI that has to do with OCP
1743 configuration parameters and signal configuration. This section describes the concrete,
1744 complete, and unambiguous transformation from WMemI WIP attributes described
1745 above, to the OCP configuration parameters inferred from those attributes. This
1746 transformation is implemented in automation tools.

1747 The table below lists only those parameters that are different from the OCP
1748 configuration parameter defaults (per the OCP specification).

1749   **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:
		    $N = floor(log_2(max(2,MaxBurstLength))) + 1$
		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	{P}	If ByteWidth != DataWidth and ByteWidth != 8: Q = DataWidth – (8 * DataWidth/ByteWidth)
mdatainfobyte_wdth	{P}	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

1750   ### 8.5   *WMemI OCP Signals*

1751   Signal configuration is determined by the interface configuration parameters and as
1752   called out in the "OCP Signal Configuration Parameter" table in the OCP specification.
1753   The tables below show the signals from master-to-slave and slave-to-master. The
1754   positional order of signals is alphabetical.

1755   **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(log_2(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.

1756   **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

1757

1758 This section describes specific semantic signal meanings in WMemI that are layered on
1759 top of the fundamental OCP signals. They are organized by function and then the
1760 signals affected.

1761 There are no special semantics with WMemI, but there are additional restrictions.

1762 8.6.1 Two basic modes of WMemI: bursts or not

1763 When neither type of burst is enabled by the WMemI attributes, the interface is
1764 simplified to have no data handshake phase, and thus no pipelining of requests before
1765 write data. The worker always presents data with the request, doesn't deal with bursts,
1766 and uses the MByteEn signals for write byte enables.

1767 When either type of burst is enabled by the WMemI attributes, the interface is
1768 configured with pipelined requests, bursts, and the (data handshake phase)
1769 MDataByteEn signals are used for write byte enables.

1770 8.6.2 Use only Single-Request Multiple-Data for Precise Bursts

1771 When the interface supports precise bursts, they must be issued with a single request,
1772 which is the semantic when the MBurstSingleReq signal is true (or tied true). Thus
1773 WMemI masters must not issue precise bursts with multiple requests. When the
1774 interface supports imprecise bursts, OCP requires using multiple-request-multiple-data
1775 bursts. When both precise bursts and imprecise bursts are supported, the
1776 MBurstSingleReq signal is configured (as well as the MPreciseBurst signal) and they
1777 both always have the same value: if the burst is precise, a single request is used, and if
1778 the burst is imprecise, multiple requests are used.

1779 8.6.3 Intra-burst write data flow control

1780 With WMemI, the SCmdAccept signal is always enabled and used to accept requests.
1781 The WriteDataFlowControl attribute, which can only be TRUE when bursts are used,
1782 enables the dataaccept OCP configuration parameter and thus the SDataAccept signal.
1783 In this case every data word asserted with MDataValid, must be held until accepted via
1784 the SDataAccept signal being asserted by the slave.

1785 When WriteDataFlowControl is FALSE, the dataaccept parameter is zero, and the
1786 MDataAccept signal is not in the interface. In this case, precise and imprecise bursts
1787 may work differently since imprecise may throttle write data via SCmdAccept within the
1788 burst.

1789 Regardless of the setting of WriteDataFlowControl, if the first write data word in the
1790 burst (first data handshake phase in the burst) is presented (via MDataValid) *before*
1791 SCmdAccept is asserted (which indicates the end of the first request phase), that data
1792 (and the associated MDataValid) must be held constant until SCmdAccept is asserted.

1793   *8.6.3.1 Precise Bursts with no Write Flow Control.*

1794   After SCmdAccept is asserted for the single request for the precise burst, any data
1795   words are simply presented in a single cycle with MDataValid asserted, and the slave
1796   must/will capture the data. Masters issuing precise bursts when WriteDataFlowControl
1797   is FALSE are assured that all data asserted after the cycle in which SCmdAccept is
1798   asserted will be accepted by the slave immediately.

1799   *8.6.3.2 Imprecise Bursts with no Write Flow Control.*

1800   In WMemI imprecise bursts use multiple requests for the burst. Thus, per OCP, the
1801   data associated with a request may be asserted starting in the same cycle as the
1802   request, but can also be arbitrarily delayed. As mentioned above, data asserted (with
1803   MDataValid) before the associated SCmdAccept must be held until SCmdAccept is
1804   asserted. Data asserted during or after the associated SCmdAccept assertion must/will
1805   be captured by the slave in that cycle.

1806   Masters issuing imprecise bursts must obey the SCmdAccept behavior for every
1807   request in the burst. Thus even when WriteDataFlowControl is FALSE, the master must
1808   support request flow control throughout the burst.

1809   8.6.4   Intra-burst read data flow control

1810   The WMemI ReadDataFlowControl attribute configures the interface to provide this
1811   feature by way of the MRespAccept signal, which causes the slave to hold read data
1812   until accepted by the master. This option is only used when bursts are enabled, and is
1813   typically only used for precise bursts, since intra-burst read flow control can be
1814   accomplished by the worker/master via throttling its own issuance of read requests.

1815   **8.7   WMemI Infrastructure Issues**

1816   WMemI infrastructure IP acts as a slave to worker interfaces specified with the WMemI
1817   attributes above. There are three significant issues in supporting worker choices: burst
1818   support (implying datahandshake), intra-burst flow control (for precise bursts), and
1819   data/byte widths.

1820   8.7.1   Burst support

1821   If memory IP supports both precise bursts and imprecise bursts then it will be able to
1822   satisfy any worker burst behavior. The major challenge/cost is in adding the dynamic
1823   behavior to optimize the imprecise case where the controller does not know how much
1824   data will be in the burst. In both cases, the memory IP does not know how many intra-
1825   burst idle cycles there will be (either in requests or data), and is usually optimized for
1826   the continuous case.

1827   When the worker has no burst support, and thus no data handshake phase, tie-offs on
1828   the burst-related signals (imprecise burst with burst length of 1) should suffice.

1829   If the memory IP has no burst support, then a slightly more complex adapter is required,
1830   to align request and data phases, and, for precise bursts, an address counter/generator
1831   is required.

1832   8.7.2   Intra-burst flow control

1833   When a worker does not support write data flow control, all data in the burst must be
1834   captured in the cycle it is asserted (via MDataValid), up to the maximum burst size.
1835   This may require adding buffering to the memory IP (or adding an adapter with such
1836   memory). If the worker implements read data flow control, the memory IP may require
1837   extra buffering when it cannot apply backpressure to the underlying memory
1838   mechanisms.

1839   When there are burst size mismatches, without flow control, extra buffering is also
1840   required (i.e. when the worker writes a burst without flow control, larger than the
1841   buffering available in the memory IP).

1842   8.7.3   Data/byte widths

1843   There are many cases of width mismatches, but a common one is when the memory
1844   itself has more bits in width than the worker requires or wants to use. The simple case
1845   is to simply ignore the extra bits (sometimes the extra bits-per-byte when memory bytes
1846   are actually > 8 bits wide). Most Memory IP should work when the worker wants simple
1847   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 port 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

1887 its data input and/or output interfaces, but if it is specified to be its own clock domain, it
1888 will ikely be in a clock domain of the timekeeping infrastructure itself.

1889 The WTI is configured (per OCP) such that the time is delivered via (qualified by) an
1890 OCP "write" command on every clock cycle when the time value is valid and the worker
1891 allows, via flow control (SThreadBusy), the time value to be presented. This flow
1892 control aspect of WTI allows the worker to indicate to the infrastructure when it actually
1893 needs to know the time, enabling the infrastructure the possibility to make power
1894 optimizations when the worker does not need the time value. The presence of the OCP
1895 "write" command indicates valid time. The absence of the OCP "write" command
1896 indicates that time is not available (not valid). [Should we allow a worker to say
1897 something like "I need valid time to run at all"?]

1898 So to know time, the worker author specifies a WTI, and the number of data bits on both
1899 sides of the radix point.

1900 ### *9.3   WTI Worker Attributes*

1901 Worker authors using a WTI specify their implementation choices via the following
1902 attributes, from which all OCP configuration parameters and signals are derived.

1903 The Clock and ClockSignal attributes, described in section 2.6, are applicable to WTI.

1904 The default attribute settings provide GPS time in integer seconds.

1905 9.3.1   WTI Attribute Summary

1906 **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.*

1907 9.3.2   WTI Attribute Details

1908 This subsection contains the per-attribute details of choices available for a WMemI on a
1909 worker.

1910 The two attributes, SecondsWidth, and FractionWidth, specify the number of bits to the
1911 left and right, respectively, of the radix point, of the fixed point binary value made
1912 available to the worker, on the OCP data signals (MData). The data path width is the
1913 sum of these two attributes.

1914 ### *9.4   WTI OCP Profile/Configuration Parameters*

1915 The WTI OCP profile is the aspect of the WTI that has to do with OCP configuration
1916 parameters and signal configuration. This section describes the concrete, complete,
1917 and unambiguous transformation from WTI WIP attributes described above, to the OCP

1918 configuration parameters inferred from those attributes. This transformation is
1919 implemented in automation tools.

1920 The table below lists only those parameters that are different from the OCP
1921 configuration parameter defaults (per the OCP specification).

1922 **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.*

1923 ### 9.5   WTI OCP Signals

1924 Signal configuration is determined by the interface configuration parameters and as
1925 called out in the "OCP Signal Configuration Parameter" table in the OCP specification.
1926 The tables below show the signals from master-to-slave and slave-to-master. The
1927 positional order of signals is alphabetical.

1928 **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*

1929 **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.*

1930 ### 9.6   WTI Semantics

1931 This section describes specific semantic signal meanings in WTI that are layered on top
1932 of the fundamental OCP signals. They are organized by function and then the signals
1933 affected.

1934 9.6.1 Indicating to the worker when time is valid and available

1935 The MCmd signal field having the value "WRITE", indicates that the data is valid (on the
1936 MData signals), which in the case of WTI indicates when the time value is available and
1937 valid. A worker that has set the "AllowUnavailable" attribute to "false" (the default),
1938 indicates that it expects time to always be available before becoming operational
1939 (before the "start" control operation is issued to the worker via its WCI), and thus can
1940 assume this value is always "WRITE" before it receives the "start" control operation.

1941 9.6.2 Indicating to the infrastructure when time is not required

1942 If the worker would like to indicate to the infrastructure when a valid time value is *not*
1943 needed, it can assert the SThreadBusy signal. According to the OCP specification, this
1944 disallows a WRITE command. In the case of WTI, the infrastructure can use this
1945 indication to disable certain timekeeping activities to save power. The OCP tieoff for
1946 this signal is deasserted, so it can be ignored for workers that choose not to have such
1947 an indication.

1948 9.6.3 The simplest case

1949 With all attributes set to default values, and the worker not caring to indicate when it
1950 does not need the timevalue, the worker simple samples the MData signals with the
1951 clock indicated for the WTI. SThreadBusy will tie off to deasserted, and MCmd will
1952 always have the "WRITE" value. Other than sampling the MData signals, the worker
1953 must assert the SReset_n signal (in the WTI's clock domain), when the MReset_n
1954 signal is asserted on the worker's WCI.

1955 **9.7 WTI Infrastructure Issues**

1956 WTI infrastructure IP acts as a master to worker interfaces specified with the WTI
1957 attributes above. There are two significant issues in supporting worker behavior: time
1958 being unavailable, and the worker indicating that time is not needed.

1959 9.7.1 Time not needed

1960 The WTI infrastructure (master) must respect the SThreadBusy signal (with the exact,
1961 pipelined behavior per OCP) by not asserting the WRITE value on the MCmd signals
1962 unless the SThreadBusy signal from the worker is deasserted in the previous cycle.
1963 The infrastructure *may* take advantage of this indication of SThreadBusy to reduce
1964 activity/power in the infrastructure since the worker is explicitly indicating that it does not
1965 need a valid time value in the following cycle.

1966 9.7.2 Time not available

1967 When a worker cannot tolerate time being unavailable (when its AllowUnavailable
1968 attribute is false), then the control system will ensure that time is indeed available before
1969 issuing the "start" control operation to the worker's WCI interface. Furthermore, the
1970 infrastructure must be made aware (through implementation-defined mechanisms), that
1971 the worker cannot tolerate time becoming unavailable, and thus must inform the control
1972 system (again through implementation-defined mechanisms), when time becomes
1973 *unavailable*.

1974

10 WIP XML Schema

10.1 Introduction

1976

1977 This section describes the WIP schema for users creating XML documents that
1978 describe WIP-based IP modules. It is the format of the WIP metadata.

10.2 Quick XML Introduction

1979

1980 XML documents are files that contain information formatted according to XML
1981 (Extensible Markup Language) syntax and structure according to a particular
1982 application-specific "XML schema". XML information itself is formatted into Elements,
1983 Attributes, and Textual Content. The WIP XML Schema does not allow Textual Content
1984 at this time. XML Elements take two forms. The simpler one is when an element has
1985 no child (embedded) elements and no Textual Content. It looks like this (for element of
1986 type , "xyz", with attribute "abc"):

```
1987        <xyz abc="123"/>
```

1988 Thus the element begins with the "<" character and the element type, and terminated
1989 with the "/>" characters. Attributes have values in double-quotes. Any white space,
1990 indentation, or new lines can be inserted for readability between the element name and
1991 attributes. Thus the above example could also be:

```
1992        <xyz
1993           abc="123"
1994           />
```

1995 When the element has child elements (in this case element of type "ccc" with attribute
1996 "cat"), it looks like:

```
1997        <xyz abc="123">
1998           <ccc cat="345"/>
1999        </xyz>
```

2000 So, an XML Schema defines which Elements, Attributes, and child Elements the
2001 document may contain. Every XML document (in this context) has a single top level
2002 element that must be structured (attributes and sub-elements) according to the WIP
2003 XML schema.

2004 An element can be entered directly (as above) or entered by referring to a separate file
2005 that contains that element. So the example above might have a file "ccc1.xml"
2006 containing:

```
2007        <ccc cat="345"/>
```

2008 And then a top level file called "xyz1.xml" containing:

```
2009        <xyz abc="123">
2010           <xi:include href="ccc1.xml"/>
2011        </xyz>
```

2012 However, the schema specifies which elements are allowed to be top-level elements in
2013 any file.

2014 There are various built-in attributes that are part of the XML system itself and elements
2015 (such as the xi:include element above), and are required in certain cases. In particular,
2016 the top level element of a top level file (not an included file) must have these built-in, or
2017 "boilerplate" attributes:

2018 To specify the schema according to which this document is structured:

```
2019         xmlns:x="http://www.w3.org/2001/XMLSchema-instance"
2020         x:schemaLocation="http://www.mc.com/CPI WIP-schema1.xsd"
2021         xmlns="http://www.mc.com/CPI"
```

2022 To allow the use of the "xi:include" feature above, the top level element that will have
2023 inclusions inside it must contain the attribute:

```
2024         xmlns:xi="http://www.w3.org/2001/XInclude"
```

2025 If all these boilerplate attributes were included in the example above, it would look like:

```
2026         <xyz
2027             xmlns:x="http://www.w3.org/2001/XMLSchema-instance"
2028             x:schemaLocation="http://www.omg.org/CPI WIP-schema1.xsd"
2029             xmlns="http://www.omg.org/CPI"
2030             xmlns:xi="http://www.w3.org/2001/XInclude"
2031           abc="123">
2032           <xi:include href="ccc1.xml"/>
2033         </xyz>
```

2034 The schema reference attributes allow the files to be processed with very generic tools,
2035 since the files contain the reference to the schema they are following. If they are not
2036 included, then they must be supplied as command-line arguments on the tools that
2037 process these files. With these attributes included any XML processor can process the
2038 files without being told about the schema.

2039 Note that the default value of all boolean attributes is false unless otherwise noted.

10.3 Top Level Element: Component Specification

2040
2041 A component specification is the XML element describing a component at a high level
2042 *without implementation choices*. It describes enough about how the component can be
2043 integrated into an application to ensure possible WIP-based interoperability and
2044 interchangeability among different implementations. Many such implementations are
2045 possible given this *component specification*. In particular, a component specification
2046 can usually be completely derived from a higher level component model such as that
2047 defined in the various software-defined radio component standards such as SCA and
2048 the OMG's "PIM/PSM for software-based communication", as well as the generic
2049 component standards such as the ones in UML v2 or CCM v4.

2050 The component specification contains component-global attributes, control plane
2051 aspects and data plane aspects. A component specification is contained in the XML
2052 element whose type is "ComponentSpec", which should be a top level element in a file,
2053 structured as:

```
2054        <ComponentSpec
2055            xmlns:x="http://www.w3.org/2001/XMLSchema-instance"
2056            x:schemaLocation="http://www.mc.com/CPI WIP-schema1.xsd"
2057            xmlns="http://www.mc.com/CPI"
2058        ---attributes---
2059            >
2060        ---child elements---
2061        </ComponentSpec>
```

2062 In the various examples below the boilerplate attributes will be omitted.

2063 10.3.1 Attributes of a Component Specification

2064 *10.3.1.1   Name*

2065 The "Name" attribute of the component specification is constrained to be acceptable as
2066 an identifier in several contexts, including VHDL. It identifies the component
2067 specification as a whole. It is case *insensitive*: when in a library or application, two
2068 different component specifications cannot have the same name, when compared in a
2069 case-insensitive way.

2070 10.3.2 Control Plane Aspects of a Component Specification

2071 *10.3.2.1   Property Summary Element*

2072 The property summary contains information about the configuration properties of the
2073 component that are relevant to the control interface, based on the WCI profile. It is
2074 called a "summary" since there is normally more information about configuration
2075 properties (the "details") that are neither relevant to nor required by the component's
2076 control interface. This property summary is expressed as an XML element whose type
2077 is "PropertySummary", and is a child element of a ComponentSpec. [A later version of
2078 this specification will include a PropertyDetails element that can be used in place of this
2079 when appropriate. The property detail will enumerate each configuration property value
2080 and its attributes to aid in generating more convenience items in the VHDL skeletons.]

2081 The property summary includes the following attributes, as documented in the WIP WCI
2082 profile:

2083 *10.3.2.1.1   SizeOfConfigSpace attribute*

2084 The type of this attribute is of type unsignedInt <= 2^20 with a default value of zero.

2085 *10.3.2.1.2   WritableConfigProperties attribute of type boolean*

2086 The type of this attribute is of type boolean.

2087 *10.3.2.1.3   ReadableConfigProperties attribute*

2088 The type of this attribute is of type boolean.

2089 *10.3.2.1.4   Sub32BitConfigProperties attribute*

2090 The type of this attribute is of type boolean.

2091 Thus an example is:

```
2092        <ComponentSpec
2093          ---attributes---
2094            >
2095          <PropertySummary SizeOofConfigSpace="20"/>
2096          ---other child elements---
2097        </ComponentSpec>
```

2098 If there is no PropertySummary, then all the related attributes take on default values for
2099 the ComponentSpec, as if "<PropertySummary/>" was included. This implies that the
2100 specified component has no configuration properties as all.

2101 10.3.3 Data Plane Aspects of a Component Specification

10.3.3.1   Data Interface Specification Element

2103 The component specification defines data plane interfaces through the use of the
2104 "DataInterfaceSpec" element. It specifies the direction of the interface (producer or
2105 consumer) and the message-level protocol used at that interface. In particular it does
2106 *not* specify the WIP data profile (WSI, WMI) for that interface as that is an
2107 implementation choice.

2108 The data interface specification element has several attributes and one child element:
2109 the Protocol Summary

10.3.3.1.1   Name attribute

2111 This attribute specifies the name of the interface for this component. The value of the
2112 name attribute is a string that is constrained to be valid in various languages like OCP
2113 configuration files and VHDL. It must be unique (case insensitive) within the component
2114 specification.

10.3.3.1.2   Producer attribute

2116 This boolean attribute is as specified in the WIP data plane profiles, and indicates
2117 whether data is produced at this worker interface or consumed at this interface. A value
2118 of "true", indicates a producer interface.

10.3.3.2   Protocol Summary element

2120 The protocol summary element, which is a child element of the data interface
2121 specification element, specifies attributes of the message protocol used at this interface.
2122 We use the term "summary" since this element only contains the interface-related
2123 summary aspects of the message protocol, not the details (analogous to the
2124 PropertySummary above). [In a future version of this specification we will define a
2125 ProtocolDetails element that can be used in place of this]. The ProtocolSummary is a
2126 separate element since it will likely be reused across a variety of components and
2127 interfaces and thus may be referred to in a separate common file via the "xi:include"
2128 mechanism mentioned above.

2129 The attributes of the ProtocolSummary element are attributes defined for the data
2130 interfaces in the WIP profiles (WSI and WMI). They may be omitted if they have default

2131 values as defined in the WIP profiles. The attributes are described in the worker data
2132 interface section above.

2133 *10.3.3.2.1   DataValueWidth*

2134 The type of this attribute is of type unsignedInt.

2135 *10.3.3.2.2   DataValueGranularity*

2136 The type of this attribute is of type unsignedInt.

2137 *10.3.3.2.3   DiverseDataSizes*

2138 The type of this attribute is of type boolean.

2139 *10.3.3.2.4   MaxMessageValues attribute*

2140 The type of this attribute is of type unsignedInt.

2141 *10.3.3.2.5   NumberOfOpcodes*

2142 The type of this attribute is of type unsignedInt.

2143 *10.3.3.2.6   VariableMessageLength*

2144 The type of this attribute is of type boolean.

2145 *10.3.3.2.7   ZeroLengthMessages*

2146 The type of this attribute is of type boolean.

2147 10.3.4 Component Specification Examples

```
2148        <ComponentSpec Name="K1spec"
2149           <PropertySummary
2150              SizeofConfigSpace="8" WritableConfigProperties="true"/>
2151           <DataInterfaceSpec Name="lvds_tx" Producer="true">
2152              <ProtocolSummary
2153                 MaxMessageValues="1024" VariableMessageLength="true"/>
2154           </DataInterfaceSpec>
2155           <DataInterfaceSpec Name="lvds_rx" Producer="false">
2156              <ProtocolSummary
2157                 MaxMessageValues="1024" VariableMessageLength="true"/>
2158           </DataInterfaceSpec>
2159        </ComponentSpec>
```

```
2160            <ComponentSpec Name="S1spec"
2161              <PropertySummary
2162                 SizeofConfigSpace="256"
2163                 WritableConfigProperties="true"
2164                 ReadableConfigProperties="true"/>
2165              <DataInterfaceSpec Name="sfpdp_prod" Producer="true">
2166                 <ProtocolSummary
2167                    MaxMessageValues="1024" VariableMessageLength="true"/>
2168              </DataInterfaceSpec>
2169              <DataInterfaceSpec Name="sfpdp_cons" Producer="false">
2170                 <ProtocolSummary
2171                    MaxMessageValues="1024" VariableMessageLength="true"/>
2172              </DataInterfaceSpec>
2173            </ComponentSpec>
```

2174 *10.4 Top Level Element: Component Implementation Description*

2175 A component implementation description contains information provided by someone
2176 creating an implementation based on a component specification. It includes or
2177 references a component specification, and then describes implementation information
2178 about a particular implementation of that specification. Thus, after the attributes, it must
2179 either include as a child element a complete ComponentSpec, or include one by
2180 reference, for example, if the "fastcore" implementation of the "corespec1" specification
2181 referenced the component specification found in the "corespec1.xml" file:

```
2182            <ComponentImplementation
2183                xmlns:x="http://www.w3.org/2001/XMLSchema-instance"
2184                x:schemaLocation="http://www.omg.org/CPI WIP-schema1.xsd"
2185                xmlns="http://www.omg.org/CPI"
2186                xmlns:xi="http://www.w3.org/2001/XInclude"
2187              Name="fastcore"
2188              ---other attributes---
2189              >
2190              <xi:include href="corespec1.xml"/>
2191              ---other child elements---
2192            </ComponentImplementation>
```

2193 10.4.1 Attributes of a Component Implementation

2194 *10.4.1.1   Name*

2195 The "Name" attribute of the component implementation is constrained to be acceptable
2196 as an identifier in several contexts, including VHDL. It identifies the component
2197 specification as a whole. It is also used at the "core" name in OCP.

2198 10.4.2 Control Plane Aspects of a Component Implementation

2199 *10.4.2.1   Control Interface element*

2200 The control interface child element (ControlInterface) of the component implementation
2201 description (ComponentImplementation) specifies implementation aspects of the

2202 worker's control interface (its interface based on the WCI profile). If this element is not
2203 present, then the PropertySummary must also not be present in the referenced
2204 Component Specification, and indicates that this worker has no control interface at all.
2205 This is only allowed for infrastructure cores, not application cores. There can be zero or
2206 one control interface elements per component implementation. The attributes of the
2207 control interface are:

2208 *10.4.2.1.1   Name attribute*
2209 This attribute specifies the name used for this WCI-based interface, which defaults to
2210 "control" if not specified. The string value of this attribute is constrained to be a valid
2211 identifier in both OCP and VHDL.

2212 *10.4.2.1.2   ControlOperations attribute*
2213 This attribute is defined as a WCI profile attribute, and contains a comma-separated list
2214 of strings identifying the implemented control operations (the start operation must be
2215 included, and the default value is "start"). o

2216 *10.4.2.1.3   ResetWhileSuspended attribute*
2217 This boolean attribute is defined as a WCI profile attribute.

2218 *10.4.2.1.4   Clock*
2219 This attribute specifies the name of the clock domain used at this interface. It must
2220 match the name of a clock child element of the component implementation. It may also
2221 refer to the name of another interface, in which case the clock for this interface will use
2222 the clock specified at that other interface. If not specified, the WCI has its own clock
2223 input signal, named according to the prefix rules, associated with the other signals at
2224 this interface.

2225 *10.4.2.2   Clock element*
2226 The clock child element (Clock) of the component implementation description
2227 (ComponentImplementation) specifies a clock domain (clock input) to the
2228 implementation. It may be used by any interfaces, or be independent of any of those
2229 interfaces.   There can be zero or more clock elements in the component
2230 implementation. If there are none, then each interface (control or data) is assumed to
2231 have the same clock as the WCI, behaving per OCP. The attributes of the clock element
2232 are:

2233 *10.4.2.2.1   Name attribute*
2234 This attribute specifies the name used for this clock, which defaults to "clock" if not
2235 specified.

2236 *10.4.2.2.2   Signal attribute*
2237 This attribute specifies the name of the signal used inside the implementation for this
2238 clock. If the clock is "owned" by an interface (i.e. the "MyClock" attribute of that

2239 interface element is true), then this attribute must not be specified. The string value of
2240 this attribute is constrained to be a valid identifier in both OCP and VHDL.

2241 [future versions of this specification are expected to include constraints for the clocks
2242 required by an implementation]

2243 10.4.3 Data Plane Aspects of a Component Implementation

2244 *10.4.3.1   Stream Interface element*

2245 The stream interface child element (StreamInterface) of the component implementation
2246 description (ComponentImplementation) specifies a data interface that uses the
2247 streaming WIP profile (WSI). It references a DataInterfaceSpec by its Name attribute.
2248 Thus the Name attribute of the StreamInterface element must match the Name attribute
2249 of a DataInterfaceSpec element of the ComponentSpec. The Stream Interface element
2250 adds implementation-specific information about the interface initially defined in that
2251 DataInterfaceSpec.

2252 *10.4.3.1.1   Name attribute*

2253 This attribute specifies the name used to reference the DataInterfaceSpec in the
2254 ComponentSpec.

2255 *10.4.3.1.2   Clock attribute*

2256 This attribute specifies the name of the clock domain used at this interface. It must
2257 match the name of a clock child element of the component implementation or refer to
2258 the name of another interface, in which case the clock for this interface will use the
2259 clock specified at that other interface. If not specified (and the MyClock attribute is not
2260 true), then this interface uses the same clock as the WCI.

2261 *10.4.3.1.3   MyClock attribute*

2262 This Boolean attribute when true specifies that the clock used at this interface will be
2263 associated with the other OCP signals at this interface, and be named with the prefix
2264 rules for signals at this interface. It specifies that the clock is "owned" by this interface.
2265 If this attribute is true, and the "Clock" attribute is not specified, then this interface has
2266 its own clock, named with the other signals of this interface.

2267 *10.4.3.1.4   DataWidth attribute*

2268 As specified for the WSI.

2269 *10.4.3.1.5   PreciseBurst attribute*

2270 As specified for the WSI.

2271 *10.4.3.1.6   ImpreciseBurst attribute*

2272 As specified for the WSI.

2273 *10.4.3.1.7   Continuous attribute*

2274 As specified for the WSI.

2275 *10.4.3.1.8   Abortable attribute*

2276 As specified for the WSI.

2277 *10.4.3.1.9   EarlyRequest attribute*

2278 As specified for the WSI.

2279 *10.4.3.2   Message Interface element*

2280 The message interface child element (MessageInterface) of the component
2281 implementation description (ComponentImplementation) specifies a data interface that
2282 uses the messaging WIP profile (WMI). It references a DataInterfaceSpec by its Name
2283 attribute. Thus the Name attribute of the MessageInterface element must match the
2284 Name attribute of a DataInterfaceSpec element of the ComponentSpec. The Message
2285 Interface element adds implementation-specific information about the interface initially
2286 defined in that DataInterfaceSpec.

2287 *10.4.3.2.1   Name attribute*

2288 This attribute specifies the name used to reference the DataInterfaceSpec in the
2289 ComponentSpec.

2290 *10.4.3.2.2   Clock attribute*

2291 See StreamInterface Clock attribute above.

2292 *10.4.3.2.3   MyClock attribute*

2293 See StreamInterface MyClock attribute above.

2294 *10.4.3.2.4   DataWidth attribute*

2295 As specified in the WIP data profiles.

2296 *10.4.3.2.5   ByteWidth attribute*

2297 As specified in the WIP data profiles.

2298 *10.4.3.2.6   PreciseBurst attribute*

2299 As specified in the WIP data profiles.

2300 *10.4.3.2.7   ImpreciseBurst attribute*

2301 As specified in the WIP data profiles.

2302 *10.4.3.2.8   Continuous attribute*

2303 As specified in the WIP data profiles.

2304 *10.4.3.2.9   TalkBack attribute*

2305 As specified in the WIP data profiles.

2306   10.4.4 Memory Aspects of a Component Implementation

2307   *10.4.4.1   Memory Interface element*

2308   The memory interface child element (MemoryInterface) of the component
2309   implementation description (ComponentImplementation) specifies a memory interface
2310   that uses the memory WIP profile (WMemI).

2311   *10.4.4.1.1   Name attribute*

2312   This attribute specifies the name used for the memory interface (default is "memory").

2313   *10.4.4.1.2   Clock attribute*

2314   See StreamInterface Clock attribute above.

2315   *10.4.4.1.3   MyClock attribute*

2316   See StreamInterface MyClock attribute above.

2317   *10.4.4.1.4   DataWidth attribute*

2318   As specified in the WIP memory profiles.

2319   *10.4.4.1.5   ByteWidth attribute*

2320   As specified in the WIP memory profiles.

2321   *10.4.4.1.6   PreciseBurst attribute*

2322   As specified in the WIP data profiles.

2323   *10.4.4.1.7   ImpreciseBurst attribute*

2324   As specified in the WIP data profiles.

2325   *10.4.4.1.8   MemoryWords*

2326   As specified in the WIP data profiles.

2327   *10.4.4.1.9   MaxBurstLength attribute*

2328   As specified in the WIP data profiles.

2329   *10.4.4.1.10   WriteDataFlowControl attribute*

2330   As specified in the WIP data profiles.

2331   *10.4.4.1.11   ReadDataFlowControl attribute*

2332   As specified in the WIP data profiles.

2333   10.4.5 Time Aspects of a Component Implementation

2334   *10.4.5.1   Time Interface element*

2335   The time interface child element (InterfaceInterface) of the component implementation
2336   description (ComponentImplementation) specifies a time interface that uses the time
2337   WIP profile (WTI).

2338   *10.4.5.1.1   Name attribute*

2339   This attribute specifies the name used for the time interface (default is "time").

2340   *10.4.5.1.2   Clock attribute*

2341   See StreamInterface Clock attribute above.

2342   *10.4.5.1.3   MyClock attribute*

2343   See StreamInterface MyClock attribute above.

2344   *10.4.5.1.4   SecondsWidth attribute*

2345   As specified in the WTI profile.

2346   *10.4.5.1.5   FractionWidth attribute*

2347   As specified in the WTI profile.

2348   *10.4.5.1.6   AllowUnavailable attribute*

2349   As specified in the WTI profile.

2350