AMBA® CXS

Protocol Specification

Document number	ARM IHI 0079
Document quality	Released
Document version	Issue D
Document confidentiality	Non-Confidential
Date of issue	20 Sep 2025

Release information

The following releases of this document have been made.

Date	Version	Changes
2025/Sep/20	D	• Support for multi-node networks, CXS-Lite interface, and options for CXS flit width and packet positioning.
2023/Nov/21	C	• Support for a packetless CXS interface.
2020/Dec/18	B	• Support for multiple protocol streams. • Extension to interface protection signaling to support new signals. • Clarification of continuous delivery guarantees.
2018/Mar/12	A	• Non-Confidential First release.

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the use or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm Limited ("Arm"). **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether the subject matter of this document infringes any third party patents.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

This document may include technical inaccuracies or typographical errors. THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, any patents, copyrights, trade secrets, trademarks, or other rights.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted use, duplication, or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of this document shall prevail.

The validity, construction and performance of this notice shall be governed by English Law.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. Please follow Arm's trademark usage guidelines at http://www.arm.com/company/policies/trademarks. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-21451 version 3

AMBA SPECIFICATION LICENCE

THIS END USER LICENCE AGREEMENT (“LICENCE”) IS A LEGAL AGREEMENT BETWEEN YOU (EITHER A SINGLE INDIVIDUAL, OR SINGLE LEGAL ENTITY) AND ARM LIMITED (“ARM”) FOR THE USE OF ARM’S INTELLECTUAL PROPERTY (INCLUDING, WITHOUT LIMITATION, ANY COPYRIGHT) IN THE RELEVANT AMBA SPECIFICATION ACCOMPANYING THIS LICENCE. ARM LICENSES THE RELEVANT AMBA SPECIFICATION TO YOU ON CONDITION THAT YOU ACCEPT ALL OF THE TERMS IN THIS LICENCE. BY CLICKING “I AGREE” OR OTHERWISE USING OR COPYING THE RELEVANT AMBA SPECIFICATION YOU INDICATE THAT YOU AGREE TO BE BOUND BY ALL THE TERMS OF THIS LICENCE.

“LICENSEE” means You and your Subsidiaries. “Subsidiary” means, if You are a single entity, any company the majority of whose voting shares is now or hereafter owned or controlled, directly or indirectly, by You. A company shall be a Subsidiary only for the period during which such control exists.

1. Subject to the provisions of Clauses 2, 3 and 4, Arm hereby grants to LICENSEE a perpetual, non-exclusive, non-transferable, royalty free, worldwide licence to:
   (i) use and copy the relevant AMBA Specification for the purpose of developing and having developed products that comply with the relevant AMBA Specification;
   (ii) manufacture and have manufactured products which either: (a) have been created by or for LICENSEE under the licence granted in Clause 1(i); or (b) incorporate a product(s) which has been created by a third party(s) under a licence granted by Arm in Clause 1(i) of such third party’s AMBA Specification Licence; and
   (iii) offer to sell, sell, supply or otherwise distribute products which have either been (a) created by or for LICENSEE under the licence granted in Clause 1(i); or (b) manufactured by or for LICENSEE under the licence granted in Clause 1(ii).
2. LICENSEE hereby agrees that the licence granted in Clause 1 is subject to the following restrictions:
   (i) where a product created under Clause 1(i) is an integrated circuit which includes a CPU then either: (a) such CPU shall only be manufactured under licence from Arm; or (b) such CPU is neither substantially compliant with nor marketed as being compliant with the Arm instruction sets licensed by Arm from time to time;
   (ii) the licences granted in Clause 1(iii) shall not extend to any portion or function of a product that is not itself compliant with part of the relevant AMBA Specification; and
   (iii) no right is granted to LICENSEE to sublicense the rights granted to LICENSEE under this Agreement.
3. Except as specifically licensed in accordance with Clause 1, LICENSEE acquires no right, title or interest in any Arm technology or any intellectual property embodied therein. In no event shall the licences granted in accordance with Clause 1 be construed as granting LICENSEE, expressly or by implication, estoppel or otherwise, a licence to use any Arm technology except the relevant AMBA Specification.
4. THE RELEVANT AMBA SPECIFICATION IS PROVIDED “AS IS” WITH NO REPRESENTATION OR WARRANTIES EXPRESS, IMPLIED OR STATUTORY, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF SATISFACTORY QUALITY, MERCHANTABILITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE, OR THAT ANY USE OR IMPLEMENTATION OF SUCH ARM TECHNOLOGY WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADE SECRETS OR OTHER INTELLECTUAL PROPERTY RIGHTS.
5. NOTWITHSTANDING ANYTHING TO THE CONTRARY CONTAINED IN THIS AGREEMENT, TO THE FULLEST EXTENT PETMITTED BY LAW, THE MAXIMUM LIABILITY OF ARM IN AGGREGATE FOR ALL CLAIMS MADE AGAINST ARM, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT MATTER OF THIS AGREEMENT (INCLUDING WITHOUT LIMITATION (I) LICENSEE’S USE OF THE ARM TECHNOLOGY; AND (II) THE IMPLEMENTATION OF THE ARM TECHNOLOGY IN ANY PRODUCT CREATED BY LICENSEE UNDER THIS AGREEMENT) SHALL NOT EXCEED THE FEES PAID (IF ANY) BY LICENSEE TO ARM UNDER THIS AGREEMENT. THE EXISTENCE OF MORE THAN ONE CLAIM OR SUIT WILL NOT ENLARGE OR EXTEND THE LIMIT. LICENSEE RELEASES ARM FROM ALL OBLIGATIONS, LIABILITY, CLAIMS OR DEMANDS IN EXCESS OF THIS LIMITATION.
6. No licence, express, implied or otherwise, is granted to LICENSEE, under the provisions of Clause 1, to use the Arm tradename, or AMBA trademark in connection with the relevant AMBA Specification or any products based thereon. Nothing in Clause 1 shall be construed as authority for LICENSEE to make any representations on behalf of Arm in respect of the relevant AMBA Specification.
7. This Licence shall remain in force until terminated by you or by Arm. Without prejudice to any of its other rights if LICENSEE is in breach of any of the terms and conditions of this Licence then Arm may terminate this Licence immediately upon giving written notice to You. You may terminate this Licence at any time. Upon expiry or termination of this Licence by You or by Arm LICENSEE shall stop using the relevant AMBA Specification and destroy all copies of the relevant AMBA Specification in your possession together with all documentation and related materials. Upon expiry or termination of this Licence, the provisions of clauses 6 and 7 shall survive.

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

https://www.arm.com

Contents

AMBA CXS Protocol Specification

Preface

Preface

This preface introduces the *AMBA Credited eXtensible Stream (CXS) protocol Specification.*

It contains the following sections:

- About this specification
- Using this specification
- Additional reading
- Feedback on this specification

About this specification

This specification describes the CXS streaming interface protocol. The protocol is designed for point-to-point packetized communication that can support sharing a common CXS link between different protocols.

Intended audience

This specification is written for hardware and software engineers who want to design or debug systems and modules that are compatible with the CXS protocol.

Using this specification

This specification is organized into the following chapters:

Introduction

Introduces the AMBA Credited eXtensible Stream (CXS) protocol.

CXS operation

Provides an overview of the CXS protocol operations and the properties that describe the configuration of a CXS interface.

Signal descriptions

Describes the signal requirements of the CXS interface.

CXS packets

Describes the CXS packet control field structure, packet control signals, and packets examples.

CXS interface activation and deactivation

Describes the activation and deactivation mechanisms for a CXS interface.

CXS packet continuous delivery guarantees

Provides information on CXS packet continuous delivery guarantee.

CXS-Lite

Describes CXS-Lite, a subset of the CXS specification.

Revisions

Information about the technical changes between released issues of this specification.

Conventions

The following sections describe conventions that this specification can use:

- Typographic conventions
- Signals
- Numbers
- Timing diagrams

Typographic conventions

The typographical conventions are:

italic Highlights important notes, introduces special terminology, and indicates internal cross-references and citations.

bold Denotes signal names, and is used for terms in descriptive lists, where appropriate.

`monospace` Used for assembler syntax descriptions, pseudocode, and source code examples.

Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.

SMALL CAPITALS

Used for a few terms that have specific technical meanings.

Signals

The signal conventions are:

Signal level The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means:

- HIGH for active-HIGH signals
- LOW for active-LOW signals

Lowercase n At the start or end of a signal name denotes an active-LOW signal.

Lowercase x At the second letter of a signal name denotes a collective term for both Read and Write.

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`.

In both cases, the prefix and the associated value are written in a `monospace` font, for example, `0xFFFF0000`.

Timing diagrams

The components used in timing diagrams are explained in Figure 0-1. Variations have clear labels, when they occur. Do not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.

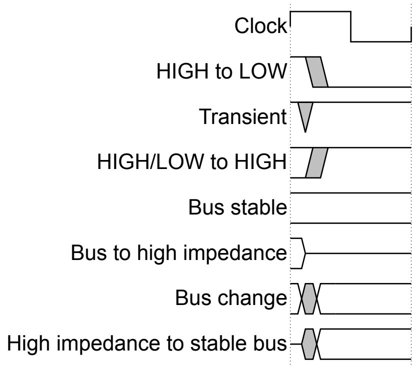

Figure 0-1 Key to timing diagram conventions

Timing diagrams sometimes show single-bit signals as HIGH and LOW at the same time and they look similar to the bus change shown in Figure 0-1 diagram conventions. If a timing diagram shows a single-bit signal in this way, then its value does not affect the accompanying description.

Additional reading

This section lists relevant publications from Arm. See Arm Developer https://developer.arm.com/documentation for access to Arm documentation.

Arm publications

- *AMBA® CXS Protocol Specification, Issue C*

Other publications

This section lists relevant documents published by third parties:

- *PCI Express Base Specification* http://www.pcisig.com
- *CCIX specification* https://www.ccixconsortium.com
- *CXL Specification* https://www.computeexpresslink.org

Feedback on this specification

If you have any comments or suggestions for additions and improvements, create a ticket at https://support.developer.arm.com. As part of the ticket, please include:

- The title, AMBA CXS Protocol Specification
- The number, ARM IHI 0079D
- The page number(s) that your comments apply
- A concise explanation of your comments

Note

Arm tests PDFs only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the appearance or behavior of any document when viewed with any other PDF reader.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used terms that can be offensive.

Arm strives to lead the industry and create change. Previous issues of this document included terms that can be offensive. We have replaced these terms.

If you find offensive terms in this document, please contact `terms@arm.com`.

Chapter 1
Introduction

This chapter introduces the CXS protocol. It contains the following sections:

1.1 About the CXS streaming interface protocol

This specification describes the Credited eXtensible Stream (CXS) streaming interface protocol. The CXS protocol can be used for any point-to-point packetized communication, specifically optimized for wide interfaces.

Wide interface optimization means that the protocol can be used to pass packets to a high data rate external interface. The availability of a wide interface permits merging of multiple packets into a single transfer.

1.2 Use case

The primary use case for a CXS interface is to transport packets between an on-chip interconnect and PCIe controller. Data transfer in CXS is unidirectional, so it is typical to have a pair of CXS interfaces between communicating blocks. Figure 1-1 shows a typical implementation of CXS that transports both CCIX and CXL packets on both of the CXS interfaces.

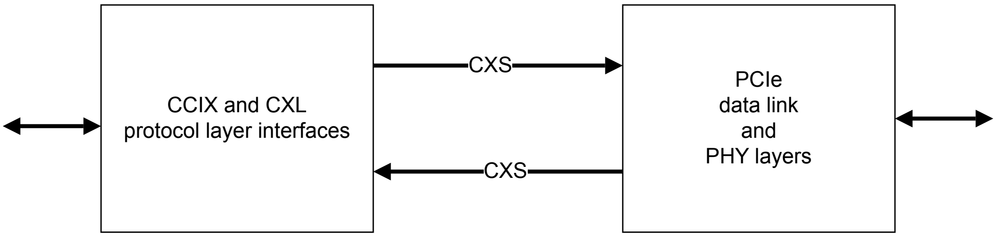

Figure 1-1 Typical implementation of CXS

Chapter 2
CXS operation

This chapter gives an overview of the operation of the CXS protocol and the properties that describe the configuration of a CXS interface.

It contains the following sections:

2.1 Protocol operation

A single instance of the interface has one Transmitter (TX) connected to one Receiver (RX). The data is sent in one direction. The data that is transferred in one cycle is known as a flit. A packet can occupy one or more flits.

Table 2-1 shows the mandatory signals of the interface.

Table 2-1 CXS mandatory signals

Name	Direction	Description
CLK	External	External Clock signal.
RESETn	External	External Reset signal.
CXSVALID	Transmitter to Receiver	Indicates that valid information is being passed this cycle.
CXSDATA	Transmitter to Receiver	The flit data containing the packet bytes being transmitted. Not applicable and recommended to be zero when **CXSVALID** is deasserted.
CXSCNTL	Transmitter to Receiver	Control information for identifying the start and end of packets within the data field. Not applicable when **CXSVALID** is deasserted.
CXSCRDGNT	Receiver to Transmitter	Flow control information indicating that the Receiver can accept one flit of data.

2.1.1 Clock and reset

The following rules apply to the clock and reset:

- All input signals are sampled on the rising edge of **CLK**.
- All output signal changes can only occur after the rising edge of **CLK**.
- There must be no combinatorial paths between input and output signals on an interface.
- The reset signal **RESETn** is active-LOW and can be asserted asynchronously, but deassertion can only be synchronous with a rising edge of **CLK**.
- During reset the following control signals, if present, must be deasserted:
  - **CXSVALID**
  - **CXSCRDGNT**
  - **CXSCRDRTN**
  - **CXSACTIVEREQ**
  - **CXSACTIVEACK**
  - **CXSDEACTHINT**
- During reset all credits are assumed to be at the receiver end of the link.
- The earliest point after reset that a receiver is permitted assert any control signals is after a rising **CLK** edge when **RESETn** is HIGH.

2.1.2 Credit exchange mechanism

The Transmitter transfers data by driving **CXSDATA**, placing packet control information on **CXSCNTL**, and asserting the **CXSVALID** signal. See Packet examples for more details.

Flow control on the interface is implemented through a credit exchange mechanism. The rules of the credit mechanism are:

- Data can only be sent when the Transmitter has at least one credit from the Receiver.
- When the interface is reset or first activated, the Transmitter has no credits and therefore cannot send data across the interface.
- Credits are transferred to the Transmitter using the **CXSCRDGNT** signal.

- When the **CXSCRDGNT** signal is asserted, one credit is transferred to the Transmitter every cycle. Each credit permits one flit of data transfer.
- The Receiver must guarantee that it can receive one flit of data for each credit that it grants.
- The Transmitter sends one flit of data for each cycle when the **CXSVALID** signal is asserted, which consumes one credit.
- The maximum number of credits that a Receiver can grant a Transmitter is specified using the CXS_MAX_CREDIT property.
  - — If the interface is configurable, the Transmitter must be able to track up to CXS_MAX_CREDIT number of credits.
  - — If the interface is not configurable, the Transmitter must be able to track 15 credits.
- A Transmitter cannot use a credit to send a flit until the cycle after the **CXSCRDGNT** signal is asserted. This specification recommends against a combinational path between the **CXSCRDGNT** and **CXSVALID** signals.
- Optionally, credits can be returned to the Receiver without a flit transfer, using the **CXSCRDRTN** signal. When the **CXSCRDRTN** signal is asserted, one credit is returned to the Receiver every cycle. The **CXSCRDRTN** and **CXSVALID** signals must not be asserted in the same cycle. See Interface control with explicit credit return for more details.
- A Receiver cannot reuse a consumed or returned credit until the cycle after the **CXSVALID** signal or the **CXSCRDRTN** signal is asserted. This specification recommends against having a combinational path between the **CXSVALID** and **CXSCRDGNT** signals, or between the **CXSCRDRTN** and **CXSCRDGNT** signals.
- If the Transmitter receives a credit in the same cycle that it returns or uses a credit, the number of available credits does not change.

This specification expects that most Receivers have sufficient storage to issue multiple credits to the Transmitter. The number of credits that are required to keep the interface flowing at full bandwidth depends on the credit latency. Credit latency is the number of cycles between the Receiver issuing a credit and that credit being reissued after being returned by the Transmitter. If the number of credits the Receiver can issue is greater than or equal to the credit latency, then the interface can sustain one flit per cycle.

This specification defines signal names for a CXS connection. Port names can be differentiated by adding TX or RX into the name after the CXS designation. Figure 2-1 shows an example of a pair of CXS links between two components.

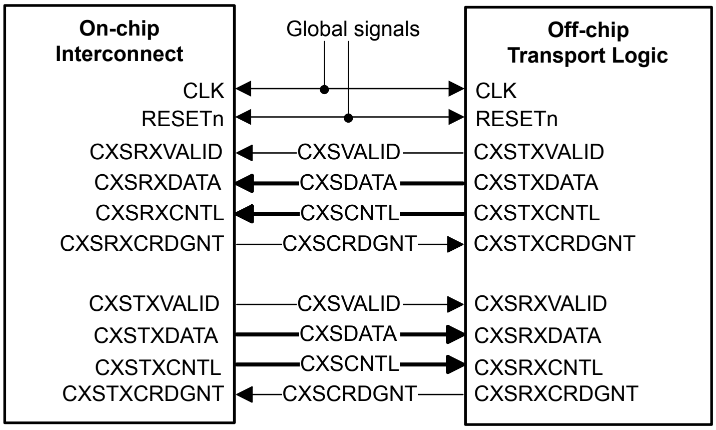

Figure 2-1 CXS connection example

An example of flit transfers on a channel is shown in Figure 2-2. In this example, the receiver has two credits available.

- At reset, the transmitter has no credits.
- One credit is issued by the receiver at b and a second one in the next cycle.
- One credit is used by the transmitter at c to transfer flit f1 and one is used to transfer flit f2 in the next cycle. A credit is reissued by the receiver at d.

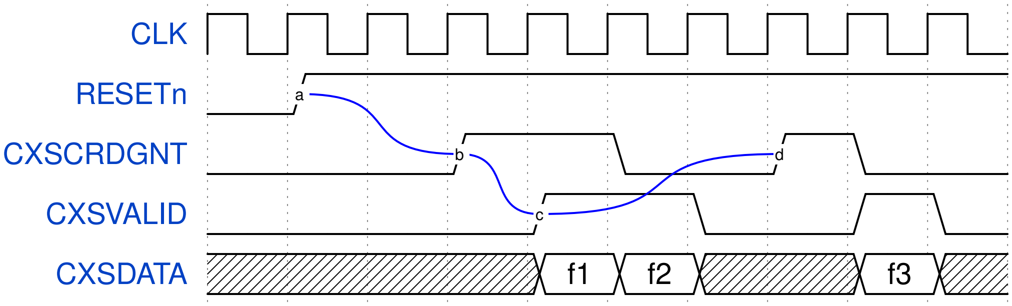

Figure 2-2 An example of flit transfers on a channel

2.2 CXS interface properties

A CXS interface is configured for a particular application by setting properties. Table 2-2 describes the properties and their options.

Table 2-2 Interface properties

Property	Options	Default	Description and rules
CXS_LAST	True, False	False	Used to indicate support for the flit insertion indicator. **True** The link interface includes the **CXSLAST** signal. **False** The link interface does not include the **CXSLAST** signal.
CXS_MAX_CREDIT	1-63	15	Specifies the maximum number of credits that the Receiver can issue.
CXS_MAX_CREDIT_LATENCY	1-32	-	**Receiver:** Specifies the maximum number of cycles before a credit is re-used, from the **CXSVALID** signal HIGH to the **CXSCRDGNT** signal HIGH. **Transmitter:** Specifies the maximum number of cycles before a credit is used when it has flits to send, from the **CXSCRDGNT** signal to the **CXSVALID** signal. This property must be defined if CXSCONTINUOUSDATA is True.
CXS_PROTOCOL_TYPE	True, False	False	Used to indicate support for the protocol type indicator. **True** The link interface includes the **CXSPRCLTYPE** signal. **False** The link interface does not include the **CXSPRCLTYPE** signal. CXS_PROTOCOL_TYPE must be False when CXSMAXPKTPERFLIT = 1. See CXS-Lite for more information.
CXSCHECKTYPE	None, Odd_Byte_Parity	None	Integrity checking on the CXS interface. **None:** No signals for integrity checking. **Odd_Byte_Parity:** Odd parity error detection signals included with a nominal granularity of one byte. See CXS interface checking signals.

Property	Options	Default	Description and rules
CXSCONTINUOUSDATA	True, False	False	**Receiver:** If set to True, the Receiver requires that after a packet is started, it is completed in consecutive cycles if enough credits are available. **Transmitter:** If set to True, the Transmitter must not begin a packet until it can deliver the complete packet in consecutive cycles while credits are available.
CXSDATAFLITWIDTH	8..2048	256	Width of the **CXSDATA** signal in bits. CXSDATAFLITWIDTH must be a multiple of 8.
CXSERRORFULLPKT[a]	True, False	False	**Receiver:** If set to True, the Receiver requires that the length of every packet matches the packet length that is specified in the packet header. This includes packets that end with EndError. **Transmitter:** If set to True, the Transmitter sends the number of bytes specified in the packet header, even if this packet ends with an EndError indication.
CXSLINKCONTROL	None, Explicit_Credit_Return	None	**None:** Interface has no link control signals. **Explicit_Credit_Return:** The interface includes the following signals: • **CXSACTIVEREQ** • **CXSACTIVEACK** • **CXSDEACTHINT** • **CXSCRDRTN** See Interface control with explicit credit return for more details.
CXSMAXPKTPERFLIT	1, 2, 3, 4	2	The maximum number of packets that can be present in a single flit of data. • When CXSMAXPKTPERFLIT is greater than 1, CXSDATAFLITWIDTH must be 256, 512, 1024, or 2048. CXSMAXPKTPERFLIT must be 1 or 2 if CXSDATAFLITWIDTH is 256. • When CXSMAXPKTPERFLIT is 1: — CXSCNTLWIDTH must be 0. — CXS_LAST must be False. — CXS_PROTOCOL_TYPE must be False. **Note** When CXSMAXPKTPERFLIT = 1, for example, in CXS-Lite, then every flit has exactly one packet which occupies all byte lanes. It is useful for the CXS applications that do not use packets or have a fixed packet size.

Property	Options	Default	Description and rules
CXS_START_ALIGNMENT	4, 16	16	Alignment of packet start pointers in bytes. CXS_START_ALIGNMENT is applicable only when CXSMAXPKTPERFLIT is > 1.
CXS_USER_WIDTH	0..128	0	Number of user bits added to each CXS flit.
CXS_SRCID_WIDTH	0..8	0	Width of the **CXSSRCID** signal.
CXS_TGTID_WIDTH	0..8	0	Width of the **CXSTGTID** signal.

[a] The encoding of the packet length within the packet is outside of the scope of the CXS document. For use of this interface for CCIX packet transmission, see the CCIX specification for packet length encoding.

Parameters can be set independently for the Transmitter and the Receiver. When assembling a system, the parameters for connected Transmitter and Receiver interfaces must be compatible. The compatibility requirements for each of the defined properties are shown in Table 2-3.

Table 2-3 Property compatibility requirements

Parameter	Compatibility requirement
CXSCHECKTYPE	Transmitter and Receiver must match.
CXS_LAST	If Receiver CXS_LAST = True, then transmitter CXS_LAST must be True.
CXS_MAX_CREDIT	The value of the TX must be the same or greater than the value of the RX.
CXS_MAX_CREDIT_LATENCY	TX and RX can be different.
CXS_PROTOCOL_TYPE	TX and RX can be different. Any undriven input can be tied LOW.
CXSCONTINUOUSDATA	If Receiver CXSCONTINUOUSDATA = True, then Transmitter CXSCONTINUOUSDATA must be True.
CXSDATAFLITWIDTH	Transmitter and Receiver must match.
CXSERRORFULLPKT	If Receiver CXSERRORFULLPKT = True, then Transmitter CXSERRORFULLPKT must be True.
CXSLINKCONTROL	Transmitter and Receiver must match.
CXSMAXPKTPERFLIT	If TX or RX is 1, then they must match. If RX is > 1, TX must be same or lower than RX. In this case, CXSCNTL bits must be remapped.
CXS_SRCID_WIDTH	TX and RX can be different, any undriven inputs can be tied LOW. If TX > RX, there might be a loss of functionality if the TX uses high order IDs.
CXS_TGTID_WIDTH	TX and RX can be different, any undriven inputs can be tied LOW. If TX > RX, there might be a loss of functionality if the TX uses high order IDs.
CXS_START_ALIGNMENT	If TX is 16, RX can be 16 or 4 but CXSCNTL must be remapped. If TX is 4, RX must be 4.
CXS_USER_WIDTH	TX and RX can be different, but there might be a loss of functionality.

2.3 Support for contiguous packets

The CXS protocol supports the transport of packets, which must remain together after passing through different layers of the communication stack.

The **CXSLAST** signal indicates when flits, such as data link layer information, can be inserted between the protocol layer packets. When the **CXSLAST** signal is deasserted, the Receiver must expect additional packets from the Transmitter and must not insert packets from another source.

The **CXSLAST** signal is defined in Table 2-4.

Table 2-4 CXSLAST signal definition

Signal	Width	Direction	Description
CXSLAST	1	Transmitter to Receiver	Indicates that flits can be inserted after this cycle.

CXSLAST must be deasserted when:

- The following packet must remain after the current packet.
- When there is an incomplete packet in the current cycle. In other words, a packet has already started, but not ended in the current cycle.

If **CXSLAST** is not present, it is assumed to be asserted, unless a packet has been started, but not ended, in the current flit.

The CXS_LAST property is used to indicate that a link interface supports contiguous packets. CXS_LAST can have the following values:

True The interface includes the **CXSLAST** signal.

False The interface does not include the **CXSLAST** signal.

If the CXS_LAST property is not declared, it is considered False.

2.4 Support for multiple protocol streams

This specification enables sharing a CXS link, where packets carrying different protocols can be transmitted on the same link.

CXSPRCLTYPE is an optional signal that indicates the protocol type of a flit.

The **CXSPRCLTYPE** signal is defined in Table 2-5.

Table 2-5 CXSPRCLTYPE signal definition

Signal	Width	Direction	Description	
CXSPRCLTYPE	3	Transmitter to Receiver	Indicates the protocol type of a flit, encoded as:	
			`0b000`	Protocol Type 0
			`0b001`	Protocol Type 1
			Other	Reserved

The following rules apply when interleaving packets from different protocols on a CXS link:

- A CXS packet that is transported using multiple flits must use the same protocol type for each flit.
- Where a flit contains more than one packet, each must have the same protocol type.
- If CXSCONTINUOUSDATA is False, flits with different protocol types can be interleaved on any cycle.
- If CXSCONTINUOUSDATA is True, the protocol type can only change after the **CXSLAST** signal is asserted.

The CXS_PROTOCOL_TYPE property is used to indicate that a link interface supports the protocol type indicator, it can have the following values:

True The interface includes the **CXSPRCLTYPE** signal.

False The interface does not include the **CXSPRCLTYPE** signal.

If the CXS_PROTOCOL_TYPE property is not declared, it is considered False. CXS_PROTOCOL_TYPE must be False when CXSMAXPKTPERFLIT = 1.

2.5 Support for multi-node networks using source and target IDs

Source and target IDs can be included as an option for CXS interfaces to support the use of multi-node networks using CXS connections.

2.5.1 Source and target ID signaling

Table 2-6 shows the signals that can be added to a CXS interface.

Table 2-6 Source and target ID signals

Signal	Width	Source	Default	Description
CXSSRCID	CXS_SRCID_WIDTH	TX	0	Source ID of a CXS flit. It can be used to identify the node from which the flit originated.
CXSTGTID	CXS_TGTID_WIDTH	TX	0	Target ID of a CXS flit. It can be used to identify the node to which the flit is directed.

Table 2-7 shows the properties used to define the width of the source and target IDs.

Table 2-7 Source and target ID width properties

Name	Values	Default	Description
CXS_SRCID_WIDTH	0-8	0	Width of the **CXSSRCID** signal.
CXS_TGTID_WIDTH	0-8	0	Width of the **CXSTGTID** signal.

The following rules apply to the source and target ID signals:

- If CXS_SRCID_WIDTH is 0, there is no **CXSSRCID** signal on the interface.
- If CXS_TGTID_WIDTH is 0, there is no **CXSTGTID** signal on the interface.
- Source and target IDs apply to all packets in the flit.
- Connected interfaces can have different widths for **CXSSRCID** and **CXSTGTID**. Any undriven input bits are tied LOW.
- **CXSSRCID** and **CXSTGTID** are valid if **CXSVALID** is asserted.

2.5.2 Flit streams and interleaving

A CXS stream is a number of flits with the same protocol type, source ID and target ID, as indicated using the **CXSPRCLTYPE**, **CXSSRCID**, and **CXSTGTID** signals respectively.

- Flits from the same stream must remain in order.
- Flits from different streams can be inserted into traffic according to the CXSCONTINUOUSDATA property and **CXSLAST** signal, if present.

Table 2-8 shows a summary of when a flit from a different stream can be inserted into CXS traffic.

Table 2-8 Flit insertion

CXSCONTINUOUSDATA	CXSLAST	Flit can be inserted
False	LOW	Yes
	HIGH	Yes
True	LOW	No
	HIGH	Yes

See Continuous delivery guarantees for CXS packets for more information.

2.6 User bits

This feature enables an implementor to add a configurable number of bits to each CXS flit, where there is a requirement not covered by the existing CXS specification. The use of User bits is not defined, so an implementation must design each transmitter and receiver pair with a consistent understanding of their meaning.

The User bits feature enables up to 128 bits of payload to be added to a flit using the CXS_USER_WIDTH property. When CXS_USER_WIDTH is greater than zero, **CXSUSER** user signal is added, as shown in Table 2-10:

Table 2-10 CXS User signal

Signal	Width	Presence	Default	Description
CXSUSER	CXS_USER_WIDTH	CXS_USER_WIDTH > 0	0	User extension for a flit.

Table 2-11 describes the CXS_USER_WIDTH property:

Table 2-11 CXS_USER_WIDTH property

Name	Values	Default	Description
CXS_USER_WIDTH	0..128	0	Number of user bits added to each CXS flit.

The applicable rule for User bits is:

- User bits are valid if **CXSVALID** is asserted.

Chapter 3
Signal descriptions

This chapter describes the signal requirements of the CXS interface. It contains the following sections:

3.1 Mandatory and optional CXS signals

Table 3-1 shows the mandatory and optional signals on a CXS interface.

Table 3-1 CXS interface signals

Signal	Width	Optional	Property	Description
CLK	1	-	-	External Clock signal.
RESETn	1	-	-	External Reset signal.
CXSVALID	1	-	-	Current cycle has a valid data flit.
CXSDATA	8..2048	-	CXSDATAFLITWIDTH	Data bytes being transmitted.
CXSCNTL	0, 14, 18, 22, 27, 33, 36, or 44. Depends on CXSMAXPKTPERFLIT and CXSDATAFLITWIDTH. See Table 4-5.	Y	CXSCNTLWIDTH	Information on packet start, end, and errors. If CXSCNTLWIDTH is 0, CXSCNTL is not present on the interface.
CXSLAST	1	Y	CXS_LAST	Indicates that flits can be inserted into the stream after this cycle.
CXSPRCLTYPE	3	Y	CXS_PROTOCOL_TYPE	Indicates the protocol type of the flit.
CXSSRCID	0..8	Y	CXS_SRCID_WIDTH	Source ID of a CXS flit. Can be used to identify the node from which the flit originated.
CXSTGTID	0..8	Y	CXS_TGTID_WIDTH	Target ID of a CXS flit. Can be used to identify the node to which the flit is directed.
CXSUSER	0..128	Y	CXS_USER_WIDTH	User bits for a flit.
CXSCRDGNT	1	-	-	Grants a single credit to Transmitter.
CXSCRDRTN	1	Y	CXSLINKCONTROL	Returns a single credit to Receiver.
CXSACTIVEREQ	1	Y	CXSLINKCONTROL	Link activation or deactivation request.
CXSACTIVEACK	1	Y	CXSLINKCONTROL	Link activation or deactivation acknowledge.
CXSDEACTHINT	1	Y	CXSLINKCONTROL	Indicates Receiver wants the link deactivated.

3.2 CXS interface checking signals

If the CXSCHECKTYPE property is set to Odd_Byte_Parity, the interface has additional signals, which can be used to improve the integrity of the interface.

Odd_Byte_Parity describes an error detection scheme where check bits are added such that the total count of 1s across the signal and check bits is an odd number. In this scheme, signals wider than 8 bits have one bit added per byte. If the signal width is not divisible by 8, then the most significant parity bit covers less than 8 bits. For example, when **CXSCNTL** is 27 bits wide, **CXSCNTLCHK**[3] covers **CXSCNTL**[26:24].

Check signals are synchronous to **CLK** and must be driven correctly in every cycle that the signal in the Check enable column is HIGH, see Table 3-2.

Single bit control signals have one odd parity bit, so they are effectively duplicated with an inverted signal.

When the CXSCHECKTYPE property is set to Odd_Byte_Parity, then the included check signals are listed in Table 3-2. If the corresponding signal is not present on the interface, then the check signal is not present either.

Table 3-2 Check signal widths (bits)

Signal	Check signal	Check signal width	Check enable
CXSVALID	**CXSVALIDCHK**	1	**CXSRESETn**
CXSDATA	**CXSDATACHK**	CXSDATAFLITWIDTH/8	**CXSVALID**
CXSCNTL	**CXSCNTLCHK**	ceil(CXSCNTLWIDTH/8)	**CXSVALID**
CXSLAST	**CXSLASTCHK**	1	**CXSVALID**
CXSPRCLTYPE	**CXSPRCLTYPECHK**	1	**CXSVALID**
CXSSRCID	**CXSSRCIDCHK**	1	**CXSVALID**
CXSTGTID	**CXSTGTIDCHK**	1	**CXSVALID**
CXSUSER	**CXSUSERCHK**	ceil(CXS_USER_WIDTH/8)	**CXSVALID**
CXSCRDGNT	**CXSCRDGNTCHK**	1	**CXSRESETn**
CXSCRDRTN	**CXSCRDRTNCHK**	1	**CXSRESETn**
CXSACTIVEREQ	**CXSACTIVEREQCHK**	1	**CXSRESETn**
CXSACTIVEACK	**CXSACTIVEACKCHK**	1	**CXSRESETn**
CXSDEACTHINT	N/A	-	-

Chapter 4
CXS packets

The data that is transmitted on the CXS interface is organized into packets. When the property CXSMAXPKTPERFLIT is greater than 1, there can be multiple packets per flit. In CCIX or PCIe terminology, this is a Transaction Layer Packet (TLP). This chapter describes how multiple packets are packed into a CXS flit.

Note

For CXS-Lite, a packetless CXS interface, the property CXSMAXPKTPERFLIT is set to 1.

The CXS packets are described in the following sections:

4.1 Packet position constraints

CXS places restrictions on the placement of packets within each flit of data to simplify data path implementation. The rules for packet positioning are dependent on the CXSMAXPKTPERFLIT and CXS_START_ALIGNMENT properties.

Note that the CXS_START_ALIGNMENT property as described in Table 4-1, is applicable only when CXSMAXPKTPERFLIT is > 1.

Table 4-1 CXS_START_ALIGNMENT property

Name	Values	Default	Description
CXS_START_ALIGNMENT	4, 16	16	The alignment of packet start pointers in bytes.

4.1.1 Packet positioning rules

When CXSMAXPKTPERFLIT is 1:

- Packets start on the first byte of CXSDATA.
- Packets end on the last byte of CXSDATA.
- Packets do not span across multiple flits.

When CXSMAXPKTPERFLIT is greater than 1:

- A packet that starts in a flit occupies every subsequent byte in that flit until the packet ends or the flit ends.
- If a packet starts but does not end in a flit, it continues at byte[0] of the next flit.
- When CXS_START_ALIGNMENT is 16:
  - Packets start on a 16-byte aligned boundary and end on a 4-byte aligned boundary.
  - Packets must start at the first available 16-byte boundary relative to the start of the flit or the ending of a previous packet.
  - When a packet ends within a flit, the remaining bytes in the flit can be unused.
- When CXS_START_ALIGNMENT is 4:
  - Packets start and end on a 4-byte aligned boundary.
  - It is permitted to have any number of 4-byte gaps before the start or after the end of a packet.
  - It is permitted to have a flit that does not contain any packets.

4.2 Flit width and packet configuration options

Table 4-2 shows the permitted combinations of flit width and maximum packets per flit:

- 'Y' indicates a permitted combination.
- 'C' indicates a combination that can be permitted by configuration.
- '-' indicates an illegal combination.

Table 4-2 CXSDATAFLITWIDTH and CXSMAXPKTPERFLIT

CXSDATAFLITWIDTH	CXSMAXPKTPERFLIT			
	1	**2**	**3**	**4**
256	Y	Y	C[a]	C[a]
512	Y	Y	Y	Y
1024	Y	Y	Y	Y
2048	Y	Y	Y	Y
Other values	Y	-	-	-

[a] If CXS_START_ALIGNMENT is 4.

4.3 Packet control signal

The control fields of a CXS packet are signaled using **CXSCNTL**. This section describes the fields and positioning within the **CXSCNTL** signal.

4.3.1 Packet control fields

The **CXSCNTL** signal contains five fields. The widths of each field, and therefore the bit position of each field, vary with the properties of the interface. The fields are described in Table 4-4.

Table 4-4 Packet control fields

Field	Description
START	Each bit in **START** indicates that a packet is starting in this flit. The number of bits in **START** is CXSMAXPKTPERFLIT, which is the number of packets that can be present in a flit of data. For example, when CXSMAXPKTPERFLIT = 4, then: • **START**[0] = 1, at least one packet is starting in this flit. • **START**[1] = 1, at least two packets are starting in this flit. • **START**[2] = 1, at least three packets are starting in this flit. • **START**[3] = 1, at least four packets are starting in this flit. If any bit of **START** is 1, all lower bits of **START** must be 1.
START[N:0]PTR	This field is an array of pointers to the starting location of each of the packets in this flit. • There is one pointer for each bit in the **START** field, valid if that bit of **START** is set. • If the corresponding **START** bit is 0, the pointer can have any value and should be ignored. • When 16-byte start alignment is used: — The width of each pointer is $\log_2$(CXSDATAFLITWIDTH/128) bits. — The first byte of the N^{th} starting packet is (**START**[N]**PTR** << 4). • When 4-byte start alignment is used: — The width of each pointer is $\log_2$(CXSDATAFLITWIDTH/32) bits. — The first byte of the N^{th} starting packet is (**START**[N]**PTR** << 2). • Start pointers are defined to be monotonically increasing, for example **START1PTR** must be greater than **START0PTR**.
END	Each bit in **END** indicates that a packet is ending in this flit. The number of bits in **END** is CXSMAXPKTPERFLIT, which is the number of packets that can be present in a flit of data. For example, when CXSMAXPKTPERFLIT = 4, then: • **END**[0] = 1, at least one packet is ending in this flit. • **END**[1] = 1, at least two packets are ending in this flit. • **END**[2] = 1, at least three packets are ending in this flit. • **END**[3] = 1, at least four packets are ending in this flit. If any bit of **END** is 1, all lower bits of **END** must be 1.

Field	Description
ENDERROR	Each bit in **ENDERROR** indicates that a packet is ending with an error condition in this flit, and the associated packet might be malformed. • **ENDERROR**[0] = 1, the first packet ending this cycle has an error. • **ENDERROR**[1] = 1, the second packet ending this cycle has an error. • **ENDERROR**[2] = 1, the third packet ending this cycle has an error. • **ENDERROR**[3] = 1, the fourth packet ending this cycle has an error. The number of bits in **ENDERROR** is the number of bits in **END**. If **ENDERROR**[N] is asserted, **END**[N] must be asserted.
END[N:0]PTR	This field is an array of pointers to the last 4 bytes of packets ending in this flit. • There is one pointer for each **END** bit, valid only if that **END** bit is set. • If the corresponding **END** bit is 0, the pointer can have any value and should be ignored. • All packet ends are 4-byte aligned. • The width of each pointer is $\log_2$(CXSDATAFLITWIDTH/32) bits. • Each end pointer points to the first byte of the last aligned 4 bytes of the packet. • The last byte of the N^{th} ending packet is therefore ((**END**[N]**PTR** << 2)+3). • Valid end pointers are defined to be monotonically increasing, for example if two packets end in this flit then **END1PTR** must be greater than **END0PTR**. **Note** **START**[N]**PTR** and **END**[N]**PTR** might not point to the same packet, for example if a packet started in a previous flit.

4.3.2 Control field mapping with 16-byte start alignment

Table 4-5 shows packet control field widths and placement information for all combinations of CXSMAXPKTPERFLIT, CXSDATAFLITWIDTH and CXSCNTLWIDTH, with 16-byte start alignment.

See Packet examples for illustrations of how these structures are used.

Table 4-5 Control field widths and placement with 16-byte start alignment

CXSMAXPKTPERFLIT	CXSDATAFLITWIDTH	CXSCNTLWIDTH	Field	Bit positions in CXSCNTL
1	Any	0	-	-
2	256	14	**START**[1:0]	**CXSCNTL**[1:0]
			START0PTR[0]	**CXSCNTL**[2]
			START1PTR[0]	**CXSCNTL**[3]
			END[1:0]	**CXSCNTL**[5:4]
			ENDERROR[1:0]	**CXSCNTL**[7:6]
			END0PTR[2:0]	**CXSCNTL**[10:8]
			END1PTR[2:0]	**CXSCNTL**[13:11]
2	512	18	**START**[1:0]	**CXSCNTL**[1:0]
			START0PTR[1:0]	**CXSCNTL**[3:2]
			START1PTR[1:0]	**CXSCNTL**[5:4]
			END[1:0]	**CXSCNTL**[7:6]
			ENDERROR[1:0]	**CXSCNTL**[9:8]
			END0PTR[3:0]	**CXSCNTL**[13:10]
			END1PTR[3:0]	**CXSCNTL**[17:14]
2	1024	22	**START**[1:0]	**CXSCNTL**[1:0]
			START0PTR[2:0]	**CXSCNTL**[4:2]
			START1PTR[2:0]	**CXSCNTL**[7:5]
			END[1:0]	**CXSCNTL**[9:8]
			ENDERROR[1:0]	**CXSCNTL**[11:10]
			END0PTR[4:0]	**CXSCNTL**[16:12]
			END1PTR[4:0]	**CXSCNTL**[21:17]
2	2048	26	**START**[1:0]	**CXSCNTL**[1:0]
			START0PTR[3:0]	**CXSCNTL**[5:2]
			START1PTR[3:0]	**CXSCNTL**[9:6]
			END[1:0]	**CXSCNTL**[11:10]
			ENDERROR[1:0]	**CXSCNTL**[13:12]
			END0PTR[5:0]	**CXSCNTL**[19:14]
			END1PTR[5:0]	**CXSCNTL**[25:20]
3	256	-	Not legal: 256-bit interface has maximum of 2 packets per flit.	

Continued on next page

(Continued)

CXSMAXPKTPERFLIT	CXSDATAFLITWIDTH	CXSCNTLWIDTH	Field	Bit positions in CXSCNTL
3	512	27	**START**[2:0]	**CXSCNTL**[2:0]
			START0PTR[1:0]	**CXSCNTL**[4:3]
			START1PTR[1:0]	**CXSCNTL**[6:5]
			START2PTR[1:0]	**CXSCNTL**[8:7]
			END[2:0]	**CXSCNTL**[11:9]
			ENDERROR[2:0]	**CXSCNTL**[14:12]
			END0PTR[3:0]	**CXSCNTL**[18:15]
			END1PTR[3:0]	**CXSCNTL**[22:19]
			END2PTR[3:0]	**CXSCNTL**[26:23]
3	1024	33	**START**[2:0]	**CXSCNTL**[2:0]
			START0PTR[2:0]	**CXSCNTL**[5:3]
			START1PTR[2:0]	**CXSCNTL**[8:6]
			START2PTR[2:0]	**CXSCNTL**[11:9]
			END[2:0]	**CXSCNTL**[14:12]
			ENDERROR[2:0]	**CXSCNTL**[17:15]
			END0PTR[4:0]	**CXSCNTL**[22:18]
			END1PTR[4:0]	**CXSCNTL**[27:23]
			END2PTR[4:0]	**CXSCNTL**[32:28]
3	2048	39	**START**[2:0]	**CXSCNTL**[2:0]
			START0PTR[3:0]	**CXSCNTL**[6:3]
			START1PTR[3:0]	**CXSCNTL**[10:7]
			START2PTR[3:0]	**CXSCNTL**[14:11]
			END[2:0]	**CXSCNTL**[17:15]
			ENDERROR[2:0]	**CXSCNTL**[20:18]
			END0PTR[5:0]	**CXSCNTL**[26:21]
			END1PTR[5:0]	**CXSCNTL**[32:27]
			END2PTR[5:0]	**CXSCNTL**[38:33]
4	256	-	Not legal: 256-bit interface has maximum of 2 packets per flit.	
4	512	36	**START**[3:0]	**CXSCNTL**[3:0]
			START0PTR[1:0]	**CXSCNTL**[5:4]
			START1PTR[1:0]	**CXSCNTL**[7:6]
			START2PTR[1:0]	**CXSCNTL**[9:8]
			START3PTR[1:0]	**CXSCNTL**[11:10]
			END[3:0]	**CXSCNTL**[15:12]
			ENDERROR[3:0]	**CXSCNTL**[19:16]
			END0PTR[3:0]	**CXSCNTL**[23:20]
			END1PTR[3:0]	**CXSCNTL**[27:24]
			END2PTR[3:0]	**CXSCNTL**[31:28]
			END3PTR[3:0]	**CXSCNTL**[35:32]

Continued on next page

(Continued)

CXSMAXPKTPERFLIT	CXSDATAFLITWIDTH	CXSCNTLWIDTH	Field	Bit positions in CXSCNTL
4	1024	44	**START**[3:0]	**CXSCNTL**[3:0]
			START0PTR[2:0]	**CXSCNTL**[6:4]
			START1PTR[2:0]	**CXSCNTL**[9:7]
			START2PTR[2:0]	**CXSCNTL**[12:10]
			START3PTR[2:0]	**CXSCNTL**[15:13]
			END[3:0]	**CXSCNTL**[19:16]
			ENDERROR[3:0]	**CXSCNTL**[23:20]
			END0PTR[4:0]	**CXSCNTL**[28:24]
			END1PTR[4:0]	**CXSCNTL**[33:29]
			END2PTR[4:0]	**CXSCNTL**[38:34]
			END3PTR[4:0]	**CXSCNTL**[43:39]
4	2048	52	**START**[3:0]	**CXSCNTL**[3:0]
			START0PTR[3:0]	**CXSCNTL**[7:4]
			START1PTR[3:0]	**CXSCNTL**[11:8]
			START2PTR[3:0]	**CXSCNTL**[15:12]
			START3PTR[3:0]	**CXSCNTL**[19:16]
			END[3:0]	**CXSCNTL**[23:20]
			ENDERROR[3:0]	**CXSCNTL**[27:24]
			END0PTR[5:0]	**CXSCNTL**[33:28]
			END1PTR[5:0]	**CXSCNTL**[39:34]
			END2PTR[5:0]	**CXSCNTL**[45:40]
			END3PTR[5:0]	**CXSCNTL**[51:46]

4.3.3 Control field mapping with 4-byte start alignment

Table 4-7 shows packet control field widths and placement information for all combinations of CXSMAXPKTPERFLIT, CXSDATAFLITWIDTH, and CXSCNTLWIDTH with 4-byte start alignment, set by the CXS_START_ALIGNMENT property.

Table 4-7 Control field widths and placement with 4-byte start alignment

CXSMAXPKTPERFLIT	CXSDATAFLITWIDTH	CXSCNTLWIDTH	Field	Bit positions in CXSCNTL
2	256	18	**START**[1:0]	**CXSCNTL**[1:0]
			START0PTR[2:0]	**CXSCNTL**[4:2]
			START1PTR[2:0]	**CXSCNTL**[7:5]
			END[1:0]	**CXSCNTL**[9:8]
			ENDERROR[1:0]	**CXSCNTL**[11:10]
			END0PTR[2:0]	**CXSCNTL**[14:12]
			END1PTR[2:0]	**CXSCNTL**[17:15]
2	512	22	**START**[1:0]	**CXSCNTL**[1:0]
			START0PTR[3:0]	**CXSCNTL**[5:2]
			START1PTR[3:0]	**CXSCNTL**[9:6]
			END[1:0]	**CXSCNTL**[11:10]
			ENDERROR[1:0]	**CXSCNTL**[13:12]
			END0PTR[3:0]	**CXSCNTL**[17:14]
			END1PTR[3:0]	**CXSCNTL**[21:18]
2	1024	26	**START**[1:0]	**CXSCNTL**[1:0]
			START0PTR[4:0]	**CXSCNTL**[6:2]
			START1PTR[4:0]	**CXSCNTL**[11:7]
			END[1:0]	**CXSCNTL**[13:12]
			ENDERROR[1:0]	**CXSCNTL**[15:14]
			END0PTR[4:0]	**CXSCNTL**[20:16]
			END1PTR[4:0]	**CXSCNTL**[25:21]
2	2048	30	**START**[1:0]	**CXSCNTL**[1:0]
			START0PTR[5:0]	**CXSCNTL**[7:2]
			START1PTR[5:0]	**CXSCNTL**[13:8]
			END[1:0]	**CXSCNTL**[15:14]
			ENDERROR[1:0]	**CXSCNTL**[17:16]
			END0PTR[5:0]	**CXSCNTL**[23:18]
			END1PTR[5:0]	**CXSCNTL**[29:24]
3	256	27	**START**[2:0]	**CXSCNTL**[2:0]
			START0PTR[2:0]	**CXSCNTL**[5:3]
			START1PTR[2:0]	**CXSCNTL**[8:6]
			START2PTR[2:0]	**CXSCNTL**[11:9]
			END[2:0]	**CXSCNTL**[14:12]
			ENDERROR[2:0]	**CXSCNTL**[17:15]
			END0PTR[2:0]	**CXSCNTL**[20:18]
			END1PTR[2:0]	**CXSCNTL**[23:21]
			END2PTR[2:0]	**CXSCNTL**[26:24]

Continued on next page

(Continued)

CXSMAXPKTPERFLIT	CXSDATAFLITWIDTH	CXSCNTLWIDTH	Field	Bit positions in CXSCNTL
3	512	33	**START**[2:0]	**CXSCNTL**[2:0]
			START0PTR[3:0]	**CXSCNTL**[6:3]
			START1PTR[3:0]	**CXSCNTL**[10:7]
			START2PTR[3:0]	**CXSCNTL**[14:11]
			END[2:0]	**CXSCNTL**[17:15]
			ENDERROR[2:0]	**CXSCNTL**[20:18]
			END0PTR[3:0]	**CXSCNTL**[24:21]
			END1PTR[3:0]	**CXSCNTL**[28:25]
			END2PTR[3:0]	**CXSCNTL**[32:29]
3	1024	39	**START**[2:0]	**CXSCNTL**[2:0]
			START0PTR[4:0]	**CXSCNTL**[7:3]
			START1PTR[4:0]	**CXSCNTL**[12:8]
			START2PTR[4:0]	**CXSCNTL**[17:13]
			END[2:0]	**CXSCNTL**[20:18]
			ENDERROR[2:0]	**CXSCNTL**[23:21]
			END0PTR[4:0]	**CXSCNTL**[28:24]
			END1PTR[4:0]	**CXSCNTL**[33:29]
			END2PTR[4:0]	**CXSCNTL**[38:34]
3	2048	45	**START**[2:0]	**CXSCNTL**[2:0]
			START0PTR[5:0]	**CXSCNTL**[8:3]
			START1PTR[5:0]	**CXSCNTL**[14:9]
			START2PTR[5:0]	**CXSCNTL**[20:15]
			END[2:0]	**CXSCNTL**[23:21]
			ENDERROR[2:0]	**CXSCNTL**[26:24]
			END0PTR[5:0]	**CXSCNTL**[32:27]
			END1PTR[5:0]	**CXSCNTL**[38:33]
			END2PTR[5:0]	**CXSCNTL**[44:39]
4	256	36	**START**[3:0]	**CXSCNTL**[3:0]
			START0PTR[2:0]	**CXSCNTL**[6:4]
			START1PTR[2:0]	**CXSCNTL**[9:7]
			START2PTR[2:0]	**CXSCNTL**[12:10]
			START3PTR[2:0]	**CXSCNTL**[15:13]
			END[3:0]	**CXSCNTL**[19:16]
			ENDERROR[3:0]	**CXSCNTL**[23:20]
			END0PTR[2:0]	**CXSCNTL**[26:24]
			END1PTR[2:0]	**CXSCNTL**[29:27]
			END2PTR[2:0]	**CXSCNTL**[32:30]
			END3PTR[2:0]	**CXSCNTL**[35:33]

Continued on next page

(Continued)

CXSMAXPKTPERFLIT	CXSDATAFLITWIDTH	CXSCNTLWIDTH	Field	Bit positions in CXSCNTL
4	512	44	**START**[3:0]	**CXSCNTL**[3:0]
			START0PTR[3:0]	**CXSCNTL**[7:4]
			START1PTR[3:0]	**CXSCNTL**[11:8]
			START2PTR[3:0]	**CXSCNTL**[15:12]
			START3PTR[3:0]	**CXSCNTL**[19:16]
			END[3:0]	**CXSCNTL**[23:20]
			ENDERROR[3:0]	**CXSCNTL**[27:24]
			END0PTR[3:0]	**CXSCNTL**[31:28]
			END1PTR[3:0]	**CXSCNTL**[35:32]
			END2PTR[3:0]	**CXSCNTL**[39:36]
			END3PTR[3:0]	**CXSCNTL**[43:40]
4	1024	52	**START**[3:0]	**CXSCNTL**[3:0]
			START0PTR[4:0]	**CXSCNTL**[8:4]
			START1PTR[4:0]	**CXSCNTL**[13:9]
			START2PTR[4:0]	**CXSCNTL**[18:14]
			START3PTR[4:0]	**CXSCNTL**[23:19]
			END[3:0]	**CXSCNTL**[27:24]
			ENDERROR[3:0]	**CXSCNTL**[31:28]
			END0PTR[4:0]	**CXSCNTL**[36:32]
			END1PTR[4:0]	**CXSCNTL**[41:37]
			END2PTR[4:0]	**CXSCNTL**[46:42]
			END3PTR[4:0]	**CXSCNTL**[51:47]
4	2048	60	**START**[3:0]	**CXSCNTL**[3:0]
			START0PTR[5:0]	**CXSCNTL**[9:4]
			START1PTR[5:0]	**CXSCNTL**[15:10]
			START2PTR[5:0]	**CXSCNTL**[21:16]
			START3PTR[5:0]	**CXSCNTL**[27:22]
			END[3:0]	**CXSCNTL**[31:28]
			ENDERROR[3:0]	**CXSCNTL**[35:32]
			END0PTR[5:0]	**CXSCNTL**[41:36]
			END1PTR[5:0]	**CXSCNTL**[47:42]
			END2PTR[5:0]	**CXSCNTL**[53:48]
			END3PTR[5:0]	**CXSCNTL**[59:54]

4.4 Packet size constraints

A CXS interface transmits packets of data that meet the following requirements:

- At least 4 bytes in size
- A multiple of 4 bytes in size
- No upper limit on packet size

When used to transmit CCIX packets, there may be further constraints on packet size. Refer to the CCIX specification for more details.

4.5 Packet examples

The following examples illustrate packet placement rules and the **CXSCNTL** field usage. Examples that are shown in Table 4-9 and Table 4-10 both have CXSCONTINUOUSDATA = True and CXSDATACHECK = None. Each data packet in the figures has a unique identifier. Unused packet slots have dashes instead of identifiers.

Table 4-9 shows an example with 256-bit data, up to two packets per flit and 16-byte start alignment.

Table 4-9 Example 256-bit wide interface with maximum of two packets per flit

		Cycle											
Signal	**Field**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
CXSVALID		0	1	1	0	1	1	1	1	1	1	1	1
CXSDATA[31:0]		-	TLPA	TLPB	-	TLPD	TLPD	TLPE	TLPE	TLPF	TLPH	TLPI	TLPK
CXSDATA[63:32]		-	TLPA	TLPB	-	TLPD	-	TLPE	TLPE	-	TLPH	TLPI	TLPK
CXSDATA[95:64]		-	TLPA	TLPB	-	TLPD	-	TLPE	TLPE	-	TLPH	TLPI	TLPK
CXSDATA[127:96]		-	TLPA	-	-	TLPD	-	TLPE	TLPE	-	TLPH	TLPI	TLPK
CXSDATA[159:128]		-	TLPA	TLPC	-	TLPD	TLPE	TLPE	TLPE	TLPG	TLPI	TLPJ	TLPL
CXSDATA[191:160]		-	TLPA	TLPC	-	TLPD	TLPE	TLPE	-	TLPG	TLPI	TLPJ	TLPL
CXSDATA[223:192]		-	TLPA	TLPC	-	TLPD	TLPE	TLPE	-	TLPG	TLPI	TLPJ	TLPL
CXSDATA[255:224]		-	-	TLPC	-	TLPD	TLPE	TLPE	-	TLPG	TLPI	TLPJ	TLPL
CXSCNTL[1:0]	**START**[1:0]	-	0x1	0x3	-	0x1	0x1	0x0	0x0	0x3	0x3	0x1	0x3
CXSCNTL[2]	**START0PTR**[0]	-	0x0	0x0	-	0x0	0x1	-	-	0x0	0x0	0x1	0x0
CXSCNTL[3]	**START1PTR**[0]	-	-	0x1	-	-	-	-	-	0x1	0x1	-	0x1
CXSCNTL[5:4]	**END**[1:0]	-	0x1	0x3	-	0x0	0x1	0x0	0x1	0x3	0x1	0x3	0x3
CXSCNTL[7:6]	**ENDERROR**[1:0]	-	0x0	0x0	-	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0
CXSCNTL[10:8]	**END0PTR**[2:0]	-	0x6	0x2	-	-	0x0	-	0x4	0x0	0x3	0x3	0x3
CXSCNTL[13:11]	**END1PTR**[2:0]	-	-	0x7	-	-	-	-	-	0x7	-	0x7	0x7

Table 4-10 shows an example with 512-bit data, CXSDATAFLITWIDTH = 512. It has up to four packets per flit, CXSMAXPKTPERFLIT = 4.

Table 4-10 Example 512-bit wide interface with maximum of four packets per flit

		Cycle											
Signal	**Field**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
CXSVALID		0	1	1	0	1	1	1	1	1	1	1	0
CXSDATA[31:0]		-	TLPA	TLPB	-	TLPD	TLPD	TLPE	TLPE	TLPF	TLPI	TLPM	-
CXSDATA[63:32]		-	TLPA	TLPB	-	TLPD	-	TLPE	TLPE	-	TLPI	TLPM	-
CXSDATA[95:64]		-	TLPA	TLPB	-	TLPD	-	TLPE	TLPE	-	TLPI	TLPM	-
CXSDATA[127:96]		-	TLPA	TLPB	-	TLPD	-	TLPE	TLPE	-	TLPI	TLPM	-
CXSDATA[159:128]		-	TLPA	TLPB	-	TLPD	TLPE	TLPE	TLPE	TLPG	TLPJ	TLPN	-
CXSDATA[191:160]		-	TLPA	TLPB	-	TLPD	TLPE	TLPE	TLPE	TLPG	TLPJ	TLPN	-
CXSDATA[223:192]		-	TLPA	-	-	TLPD	TLPE	TLPE	TLPE	TLPG	TLPJ	TLPN	-
CXSDATA[255:224]		-	TLPA	-	-	TLPD	TLPE	TLPE	TLPE	TLPG	TLPJ	TLPN	-
CXSDATA[287:256]		-	TLPA	TLPC	-	TLPD	TLPE	TLPE	TLPE	TLPH	TLPK	TLPO	-
CXSDATA[319:288]		-	-	TLPC	-	TLPD	TLPE	TLPE	TLPE	TLPH	TLPK	TLPO	-
CXSDATA[351:320]		-	-	TLPC	-	TLPD	TLPE	TLPE	TLPE	TLPH	TLPK	TLPO	-
CXSDATA[383:352]		-	-	TLPC	-	TLPD	TLPE	TLPE	TLPE	TLPH	TLPK	TLPO	-
CXSDATA[415:384]		-	-	TLPC	-	TLPD	TLPE	TLPE	TLPE	TLPI	TLPL	TLPP	-
CXSDATA[447:416]		-	-	TLPC	-	TLPD	TLPE	TLPE	-	TLPI	TLPL	TLPP	-
CXSDATA[479:448]		-	-	TLPC	-	TLPD	TLPE	TLPE	-	TLPI	TLPL	TLPP	-
CXSDATA[511:480]		-	-	TLPC	-	TLPD	TLPE	TLPE	-	TLPI	TLPL	TLPP	-
CXSCNTL[3:0]	**START**[3:0]	-	`0x1`	`0x3`	-	`0x1`	`0x1`	`0x0`	`0x0`	`0xF`	`0x7`	`0xF`	-
CXSCNTL[5:4]	**START0PTR**[1:0]	-	`0x0`	`0x0`	-	`0x0`	`0x1`	-	-	`0x0`	`0x1`	`0x0`	-
CXSCNTL[7:6]	**START1PTR**[1:0]	-	-	`0x2`	-	-	-	-	-	`0x1`	`0x2`	`0x1`	-
CXSCNTL[8:9]	**START2PTR**[1:0]	-	-	-	-	-	-	-	-	`0x2`	`0x3`	`0x2`	-
CXSCNTL[11:10]	**START3PTR**[1:0]	-	-	-	-	-	-	-	-	`0x3`	-	`0x3`	-
CXSCNTL[15:12]	**END**[3:0]	-	`0x1`	`0x3`	-	`0x0`	`0x1`	`0x0`	`0x1`	`0x7`	`0xF`	`0xF`	-
CXSCNTL[19:16]	**ENDERROR**[3:0]	-	`0x0`	`0x0`	-	`0x0`	`0x0`	`0x0`	`0x0`	`0x0`	`0x0`	`0x0`	-
CXSCNTL[23:20]	**END0PTR**[3:0]	-	`0x8`	`0x5`	-	-	`0x0`	-	`0xC`	`0x0`	`0x3`	`0x3`	-
CXSCNTL[27:24]	**END1PTR**[3:0]	-	-	`0xF`	-	-	-	-	-	`0x7`	`0x7`	`0x7`	-
CXSCNTL[31:28]	**END2PTR**[3:0]	-	-	-	-	-	-	-	-	`0xB`	`0xB`	`0xB`	-
CXSCNTL[35:32]	**END3PTR**[3:0]	-	-	-	-	-	-	-	-	-	`0xF`	`0xF`	-

Table 4-11 shows an example sequence of flits using 4-byte start alignment.

Table 4-11 Example sequence of flits using 4-byte start alignment

	Cycle								
Signal	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
CXSVALID	0	1	1	1	1	1	1	1	1
CXSDATA [31:0]	-	-	10A	00B	-	01A	01A	-	01B
CXSDATA [63:32]	-	00A	10A	00B	-	01A	01A	00C	01B
CXSDATA [95:64]	-	00A	10A	00B	-	01A	01A	00C	-
CXSDATA [127:96]	-	00A	10A	-	-	01A	-	00C	-
CXSDATA [159:128]	-	00B	10A	-	-	01A	-	00C	-
CXSDATA [191:160]	-	00B	10A	-	-	01A	-	00C	-
CXSDATA [223:192]	-	00B	-	-	-	01A	-	-	-
CXSDATA [255:244]	-	00B	-	-	-	01A	01B	-	-
CXSLAST	-	0	1	1	1	0	0	1	1
CXSSRCID [3:0]	-	`0x0`	`0x1`	`0x0`	`0x0`	`0x0`	`0x0`	`0x0`	`0x0`
CXSTGTID [1:0]	-	`0x0`	`0x0`	`0x0`	`0x1`	`0x1`	`0x1`	`0x0`	`0x1`
START [2:0]	-	`0x3`	`0x1`	`0x0`	`0x0`	`0x1`	`0x1`	`0x1`	`0x0`
START0PTR [2:0]	-	`0x1`	`0x0`	-	-	`0x0`	`0x7`	`0x1`	-
START1PTR [2:0]	-	`0x4`	-	-	-	-	-	-	-
START2PTR [2:0]		-	-	-	-	-	-	-	-
END [2:0]	-	`0x1`	`0x1`	`0x1`	`0x0`	`0x0`	`0x1`	`0x1`	`0x1`
ENDERROR [2:0]	-	`0x0`	`0x0`	`0x1`	-	-	`0x0`	`0x0`	`0x0`
END0PTR [2:0]	-	`0x3`	`0x5`	`0x2`	-	-	`0x2`	`0x5`	`0x1`
END1PTR [2:0]	-	-	-	-	-	-	-	-	-
END2PTR [2:0]	-	-	-	-	-	-	-	-	-

The link has the following properties:

- CXSDATAFLITWIDTH = 256
- CXSMAXPKTPERFLIT = 3
- CXS_START_ALIGNMENT = 4
- CXS_SRCID_WIDTH = 4
- CXS_TGTID_WIDTH = 2
- CXSCONTINOUSDATA = False

Cycle 1 One flit is sent from source 0 to target 0. Packet 00A is complete and packet 00B starts but does not complete.

Cycle 2 A flit from source 1 is inserted, which contains a packet for target 0.

Cycle 3 Packet 00B completes but has an error. This packet might not include valid data.

Cycle 4 This flit has no packets.

Cycle 5 Packet 01A starts but does not complete in this flit.

Cycle 6 Packet 01A completes and 01B starts. There is a 16B gap between these packets.

Cycle 7 A flit to a different target is inserted in the sequence.

Table 4-13 shows an example with 512-bit data, CXSDATAFLITWIDTH = 512. It has up to two packets per flit, CXSMAXPKTPERFLIT = 2 and CXSCONTINUOUSDATA is True. In this example, Protocol 0 uses packets that are variable length and Protocol 1 uses packets that are always 64B. In the Protocol 0 stream the following groups of packets must remain together:

- P0D and P0E

In the Protocol 1 stream the following groups of packets must remain together:

- P1B and P1C
- P1E, P1F, and P1G

Table 4-13 Example 512-bit interface with multiple protocols and CXSCONTINUOUSDATA is True

	Cycle															
Signal	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
CXSVALID	-	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
CXSDATA [31:0]	-	P1A	P0A	P1B	P1C	P0B	P1D	P0D	P0D	P0E	P0E	P1E	P1F	-	P1G	P1H
CXSDATA [63:32]	-	P1A	P0A	P1B	P1C	P0B	P1D	P0D	-	P0E	P0E	P1E	P1F	-	P1G	P1H
CXSDATA [95:64]	-	P1A	P0A	P1B	P1C	P0B	P1D	P0D	-	P0E	P0E	P1E	P1F	-	P1G	P1H
CXSDATA [127:96]	-	P1A	P0A	P1B	P1C	P0B	P1D	P0D	-	P0E	P0E	P1E	P1F	-	P1G	P1H
CXSDATA [159:128]	-	P1A	P0A	P1B	P1C	P0B	P1D	P0D	P0E	P0E	P0E	P1E	P1F	-	P1G	P1H
CXSDATA [191:160]	-	P1A	P0A	P1B	P1C	P0B	P1D	P0D	P0E	P0E	P0E	P1E	P1F	-	P1G	P1H
CXSDATA [223:192]	-	P1A	P0A	P1B	P1C	-	P1D	P0D	P0E	P0E	P0E	P1E	P1F	-	P1G	P1H
CXSDATA [255:224]	-	P1A	P0A	P1B	P1C	-	P1D	P0D	P0E	P0E	P0E	P1E	P1F	-	P1G	P1H
CXSDATA [287:256]	-	P1A	P0A	P1B	P1C	P0C	P1D	P0D	P0E	P0E	P0E	P1E	P1F	-	P1G	P1H
CXSDATA [319:288]	-	P1A	-	P1B	P1C	P0C	P1D	P0D	P0E	P0E	P0E	P1E	P1F	-	P1G	P1H
CXSDATA [351:320]	-	P1A	-	P1B	P1C	P0C	P1D	P0D	P0E	P0E	P0E	P1E	P1F	-	P1G	P1H
CXSDATA [383:352]	-	P1A	-	P1B	P1C	P0C	P1D	P0D	P0E	P0E	P0E	P1E	P1F	-	P1G	P1H
CXSDATA [415:384]	-	P1A	-	P1B	P1C	P0C	P1D	P0D	P0E	P0E	P0E	P1E	P1F	-	P1G	P1H
CXSDATA [447:416]	-	P1A	-	P1B	P1C	P0C	P1D	P0D	P0E	P0E	-	P1E	P1F	-	P1G	P1H
CXSDATA [479:448]	-	P1A	-	P1B	P1C	P0C	P1D	P0D	P0E	P0E	-	P1E	P1F	-	P1G	P1H
CXSDATA [511:480]	-	P1A	-	P1B	P1C	P0C	P1D	P0D	P0E	P0E	-	P1E	P1F	-	P1G	P1H
CXSLAST	-	1	1	0	1	1	1	0	0	0	1	0	0	-	1	1
CXSPRCLTYPE [2:0]	-	0x1	0x0	0x1	0x1	0x0	0x1	0x0	0x0	0x0	0x0	0x1	0x1	-	0x1	0x1
START [1:0]	-	0x1	0x1	0x1	0x1	0x3	0x1	0x1	0x1	0x0	0x0	0x1	0x1	-	0x1	0x1
START0PTR [1:0]	-	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x1	-	-	0x0	0x0	-	0x0	0x0
START1PTR [1:0]	-	-	-	-	-	0x2	-	-	-	-	-	-	-	-	-	-
END [1:0]	-	0x1	0x1	0x1	0x1	0x3	0x1	0x0	0x1	0x0	0x1	0x1	0x1	-	0x1	0x1
ENDERROR [1:0]	-	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	-	0x0	0x0
END0PTR [3:0]	-	0xF	0x8	0xF	0xF	0x5	0xF	-	0x0	-	0xC	0xF	0xF	-	0xF	0xF
END1PTR [3:0]	-	-	-	-	-	0xF	-	-		-	-	-	-	-	-	-

Table 4-14 shows an example with 512-bit data, CXSDATAFLITWIDTH = 512. It has up to two packets per flit, CXSMAXPKTPERFLIT = 2 and CXSCONTINUOUSDATA is False.

In this example, Protocol 0 uses packets that have variable length and Protocol 1 uses packets that are always 64B.

With CXSCONTINUOUSDATA set False, packets with a different protocol can be interleaved, even if the **CXSLAST** signal is deasserted.

In the example, **CXSLAST** is used to indicate that Protocol 1 packets must not be inserted after cycles 3, 8, or 10.

Table 4-14 Example 512-bit interface with multiple protocols and CXSCONTINUOUSDATA is False

	Cycle															
Signal	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
CXSVALID	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
CXSDATA [31:0]	-	P1A	P0A	P1B	P0B	P1C	P1D	P0D	P1E	P0D	P1F	-	P0E	P1G	P0E	P1H
CXSDATA [63:32]	-	P1A	P0A	P1B	P0B	P1C	P1D	P0D	P1E	-	P1F	-	P0E	P1G	P0E	P1H
CXSDATA [95:64]	-	P1A	P0A	P1B	P0B	P1C	P1D	P0D	P1E	-	P1F	-	P0E	P1G	P0E	P1H
CXSDATA [127:96]	-	P1A	P0A	P1B	P0B	P1C	P1D	P0D	P1E	-	P1F	-	P0E	P1G	P0E	P1H
CXSDATA [159:128]	-	P1A	P0A	P1B	P0B	P1C	P1D	P0D	P1E	P0E	P1F	-	P0E	P1G	P0E	P1H
CXSDATA [191:160]	-	P1A	P0A	P1B	P0B	P1C	P1D	P0D	P1E	P0E	P1F	-	P0E	P1G	P0E	P1H
CXSDATA [223:192]	-	P1A	P0A	P1B	-	P1C	P1D	P0D	P1E	P0E	P1F	-	P0E	P1G	P0E	P1H
CXSDATA [255:224]	-	P1A	P0A	P1B	-	P1C	P1D	P0D	P1E	P0E	P1F	-	P0E	P1G	P0E	P1H
CXSDATA [287:256]	-	P1A	P0A	P1B	P0C	P1C	P1D	P0D	P1E	P0E	P1F	-	P0E	P1G	P0E	P1H
CXSDATA [319:288]	-	P1A	-	P1B	P0C	P1C	P1D	P0D	P1E	P0E	P1F	-	P0E	P1G	P0E	P1H
CXSDATA [351:320]	-	P1A	-	P1B	P0C	P1C	P1D	P0D	P1E	P0E	P1F	-	P0E	P1G	P0E	P1H
CXSDATA [383:352]	-	P1A	-	P1B	P0C	P1C	P1D	P0D	P1E	P0E	P1F	-	P0E	P1G	P0E	P1H
CXSDATA [415:384]	-	P1A	-	P1B	P0C	P1C	P1D	P0D	P1E	P0E	P1F	-	P0E	P1G	P0E	P1H
CXSDATA [447:416]	-	P1A	-	P1B	P0C	P1C	P1D	P0D	P1E	P0E	P1F	-	P0E	P1G	-	P1H
CXSDATA [479:448]	-	P1A	-	P1B	P0C	P1C	P1D	P0D	P1E	P0E	P1F	-	P0E	P1G	-	P1H
CXSDATA [511:480]	-	P1A	-	P1B	P0C	P1C	P1D	P0D	P1E	P0E	P1F	-	P0E	P1G	-	P1H
CXSLAST	-	1	1	0	1	1	1	0	0	0	0	-	0	1	1	1
CXSPRCLTYPE [2:0]	-	`0x1`	`0x0`	`0x1`	`0x0`	`0x1`	`0x1`	`0x0`	`0x1`	`0x0`	`0x1`	-	`0x0`	`0x1`	`0x0`	`0x1`
START [1:0]	-	`0x1`	`0x1`	`0x1`	`0x3`	`0x1`	`0x1`	`0x1`	`0x1`	`0x1`	`0x1`	-	`0x0`	`0x1`	`0x0`	`0x1`
START0PTR [1:0]	-	`0x0`	`0x0`	`0x0`	`0x0`	`0x0`	`0x0`	`0x0`	`0x0`	`0x1`	`0x0`	-	-	`0x0`	-	`0x0`
START1PTR [1:0]	-	-	-	-	`0x2`	-	-	-	-	-	-	-	-	-	-	-
END [1:0]	-	`0x1`	`0x1`	`0x1`	`0x3`	`0x1`	`0x1`	`0x0`	`0x1`	`0x1`	`0x1`	-	`0x0`	`0x1`	`0x1`	`0x1`
ENDERROR [1:0]	-	`0x0`	`0x0`	`0x0`	`0x0`	`0x0`	`0x0`	`0x0`	`0x0`	`0x0`	`0x0`	-	`0x0`	`0x0`	`0x0`	`0x0`
END0PTR [3:0]	-	`0xF`	`0x8`	`0xF`	`0x5`	`0xF`	`0xF`	-	`0xF`	`0x0`	`0xF`	-	-	`0xF`	`0xC`	`0xF`
END1PTR [3:0]	-	-	-	-	`0xF`	-	-	-	-	-	-	-	-	-	-	-

Chapter 5
CXS interface activation and deactivation

This chapter describes the activation and deactivation mechanisms when the CXSLINKCONTROL property is set to Explicit_Credit_Return. It contains the following sections:

5.1 Interface control with explicit credit return

A CXS interface can be optionally configured to include signaling for activation and deactivation, using the CXSLINKCONTROL property. The property can be set to *None* or *Explicit_Credit_Return*.

When the CXSLINKCONTROL property is set to *None*, there are no signals for interface activation and deactivation. This setting requires that the Receiver must always send credits when they are available, and the Transmitter must always be able to receive them.

When the CXSLINKCONTROL property is set to *Explicit_Credit_Return*, the following specification applies, and signals are added to the interface as shown in Table 5-1.

Table 5-1 Signals for link control using explicit credit return

Name	Direction	Description
CXSCRDRTN	Transmitter to Receiver	Flow control information indicating that the Transmitter is returning a previously granted credit without using it. Can only be asserted if the **CXSVALID** signal is not asserted.
CXSACTIVEREQ	Transmitter to Receiver	Link activation or deactivation request.
CXSACTIVEACK	Receiver to Transmitter	Link activation or deactivation acknowledge.
CXSDEACTHINT	Receiver to Transmitter	Hint that Receiver would like the link to be deactivated.

The interface starts in an idle state either on exit from reset or when moving to a full operational state. Transfer of flits can commence when credits have been granted by the Receiver side. Credits can be granted when the Transmitter side indicates that it is ready to receive them.

A two-signal, four-phase, handshake mechanism is used. This mechanism synchronizes the state of the link between the Transmitter and Receiver and is initiated by the Transmitter. In addition, a signal is available for the Receiver to request that the link be deactivated.

Figure 5-1 shows a typical connection, with one outbound and one inbound CXS interface, each of which has an instance of the credit control signals.

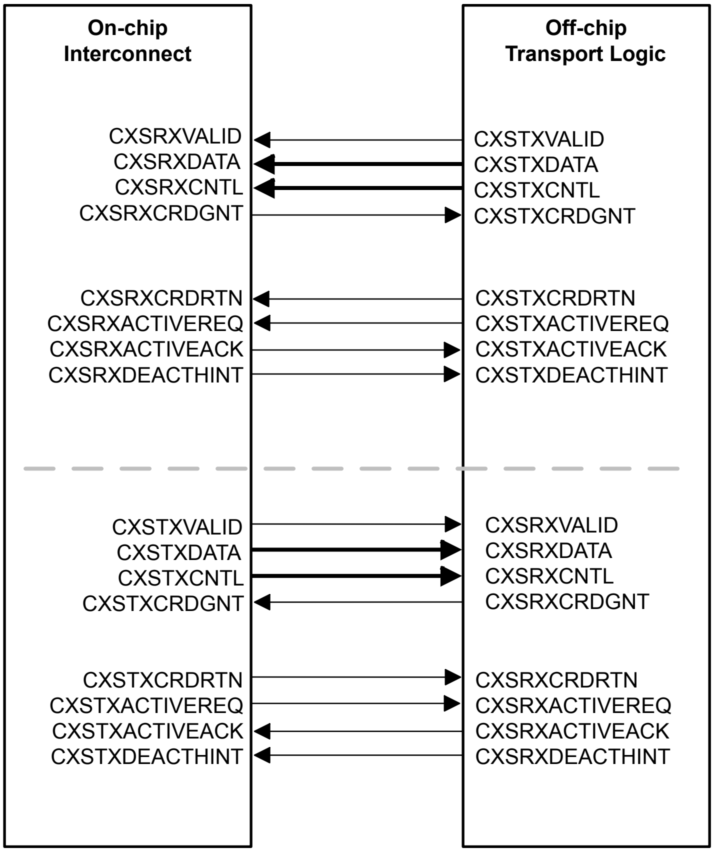

Figure 5-1 Example with two CXS links

5.2 Request and acknowledgment handshaking

Request and acknowledge handshaking uses **CXSACTIVEREQ** and **CXSACTIVEACK** as primary signals.

The Transmitter requires a credit before it can send a flit. A credit is passed from the Receiver when it has resources available to accept a flit.

- On exit from reset, all credits are held by the Receiver and at least one must be passed to the Transmitter before flit transfer can begin.
- During normal operation, there is an ongoing exchange of flits and credits between the two sides of the interface.
- Before entering a low-power state, the sending of payload flits must be stopped, and all credits must be returned to the Receiver. This action returns the interface to the same state that it was at immediately after reset.

Four states are defined for the interface operation:

STOP

- The interface does not operate in the **STOP** state. All credits are held by the Receiver.
- **STOP** is a stable state. When this state is entered, a channel can remain in it for an indefinite time.
- The Receiver is guaranteed not to receive flits or credit returns. It must not send credit.
- The Transmitter is guaranteed not to receive credit. It must not send flits or credit returns.
- The Transmitter can move from the **STOP** state to the **ACTIVATE** state when it has flits waiting to be sent.

ACTIVATE

- This state is used when transitioning from the **STOP** state to the **RUN** state.
- It is expected that when this state is entered, a channel moves to the next stable state in a relatively short time.
- The Transmitter must accept credit, but it cannot send flits until it observes the move to the **RUN** state.
- The Receiver is guaranteed not to receive flits or credit returns.
- The Receiver is not permitted to send credit in the **ACTIVATE** state. It is permitted to send credit in the same cycle that it moves to the **RUN** state. Because of a potential race condition, it is therefore possible for the Transmitter to receive credit while in the **ACTIVATE** state.
- The Receiver can move from the **ACTIVATE** state to the **RUN** state when it is prepared to receive flits.

RUN

- This state has an ongoing exchange of flits and credits between the two components.
- **RUN** is a stable state. A channel can remain in it for an indefinite time when this state is entered.
- The Receiver can send credit and receive flits or credit returns.
- The Transmitter can send flits and receive credits.
- The Transmitter is permitted to send credit returns but this is not expected.
- The Transmitter can move from the **RUN** state to the **DEACTIVATE** state for a number of reasons, such as when it has no flits to send. See Response to a new state for more information.

DEACTIVATE

- This state is used when transitioning from **RUN** state to the **STOP** state.
- **DEACTIVATE** is a transient state. It is expected that when this state is entered, a channel moves to the next stable state in a relatively short time.
- The Transmitter must stop sending flits before entering this state. Because of a potential race condition, it is possible for the Receiver to receive flits in this state.
- The Receiver can send credit when entering this state. In a timely manner, it must stop sending credit to allow all credit to be returned to the Receiver.
- The Receiver can receive credit returns.

- The Transmitter must send credit returns to allow all credits to be returned to the Receiver.
- The Receiver must only exit this state and move to the **STOP** state when all credits have been returned.

This specification does not define a maximum time in a transient state, but it is expected that for any given implementation that it is deterministic.

The state transitions are triggered by the **CXSACTIVEREQ** and **CXSACTIVEACK** signals. Figure 5-2 shows the relationship between the four states, with values of the signals **CXSACTIVEREQ** and **CXSACTIVEACK** respectively, on each transition.

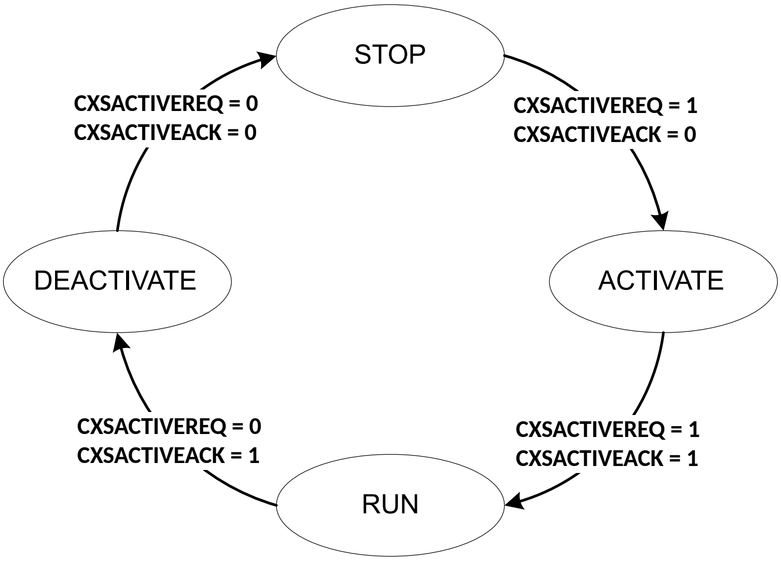

Figure 5-2 Request and acknowledge handshake states

5.3 Response to a new state

If the state change has been initiated by the other side of the interface, a component might be required to change its behavior when moving to a new state.

If the state change requires a component to stop sending flits or credits, then the component is permitted to take some time to respond.

The Transmitter is always responsible for initiating the state change from RUN to STOP, or from STOP to RUN. This state change requirement can be detected through several mechanisms. The following examples are not exhaustive:

- The Transmitter can determine that it has flits to send, so must move from STOP to RUN.
- The Transmitter can determine that it has no activity to perform for a significant period, so can move from RUN to STOP.
- The Transmitter can observe an independent sideband signal that indicates it should move either from RUN to STOP, or from STOP to RUN.
- The Transmitter can observe the **CXSDEACTHINT** signal from the Receiver and decide to move from RUN to STOP.

5.4 Race conditions

A race condition exists when one side of the interface performs two actions at, or around, the same time. The CXS specification permits different delays between the data flow and link control groups of signals. Therefore, the order of the actions at arrival might not be the same as the order of issue.

The following race conditions can occur:

- The Receiver asserts the **CXSACTIVEACK** signal, to move from ACTIVATE to RUN, and starts sending credits:
  - The Receiver is permitted to assert **CXSCRDGNT** in the same cycle that the **CXSACTIVEACK** signal is asserted.
  - The credit might be received at the Transmitter before its local **CXSACTIVEACK** signal is asserted.
  - Therefore, the Transmitter must accept credits while in the ACTIVATE or RUN state.
- The Transmitter stops sending flits and then deasserts the **CXSACTIVEREQ** signal, to move from RUN to DEACTIVATE:
  - The Transmitter must not send flits when the **CXSACTIVEREQ** signal is deasserted.
  - An in-flight flit might be received at the Receiver after its local **CXSACTIVEREQ** signal is deasserted.
  - Therefore, the Receiver must accept flits while in the DEACTIVATE state and it can only move to the STOP state when all credits are returned.

These race conditions are possible because the **CXSACTIVEREQ** and **CXSACTIVEACK** signals need not have the same delay between Transmitter and Receiver as the other signals.

5.5 Timing relationships between data and link control signals

Permitted timing relationships between the CXS signals depend on signal type.

The following signals must be synchronous with identical delay:

- **CXSVALID**
- **CXSDATA**
- **CXSCNTL**
- **CXSCRDRTN**

The following signals must be synchronous but can have any delay:

- **CXSCRDGNT**
- **CXSACTIVEACK**
- **CXSDEACTHINT**

The following signal must be driven synchronously but can be captured asynchronously with any delay:

- **CXSACTIVEREQ**

CHK signals must be clocked and pipelined identically to their corresponding signals.

Usually, the physical distance between the Transmitter and Receiver will determine the number of flip-flop stages that are required to achieve the necessary frequency. The required number of flip-flop stages will most likely be applied to all the signals on the interface.

The exception is the **CXSACTIVEREQ** signal. It is common for the clock in the Receiver to stop during the STOP state due to clock gating. The assertion of the **CXSACTIVEREQ** signal might be used to restart that clock. It is possible that the flip-flops between Transmitter and Receiver are in the Receiver clock domain and are also clock gated during STOP state. The **CXSACTIVEREQ** signal might need to have a combinational path between Transmitter and Receiver. This path might be a multicycle path due to distance and required frequency of the interface. This multicycle character is acceptable because the **CXSACTIVEREQ** and **CXSACTIVEACK** signals participate in a four-phase handshake and can run asynchronously.

The **CXSACTIVEREQ** signal must therefore be treated by the Receiver as an asynchronous signal and run through appropriate synchronization logic to avoid metastability before use.

It is permitted for the **CXSACTIVEACK** and **CXSDEACTHINT** signals to be treated as synchronous input signals by the Transmitter. These signals must not be multicycle path between Receiver and Transmitter, although they can have as many flip-flops as is needed.

5.6 Interface activation and deactivation examples

This section provides interface activation and deactivation examples. An activation of an interface is shown in Figure 5-3.

1. In cycle 0, both the Transmitter and Receiver are in the STOP state. Both sides could be clock gated or powered down.
2. The Transmitter asserts **CXSACTIVEREQ** and moves into the ACTIVATE state in cycle 2.
3. The Receiver wakes up and asserts **CXSACTIVEACK** in cycle 5.
4. In this case, **CXSCRDGNT** is asserted the same cycle as **CXSACTIVEACK**.
5. Having received a credit, the Transmitter sends a flit in cycle 6.
6. Transmitter continues to send while it is receiving credits.

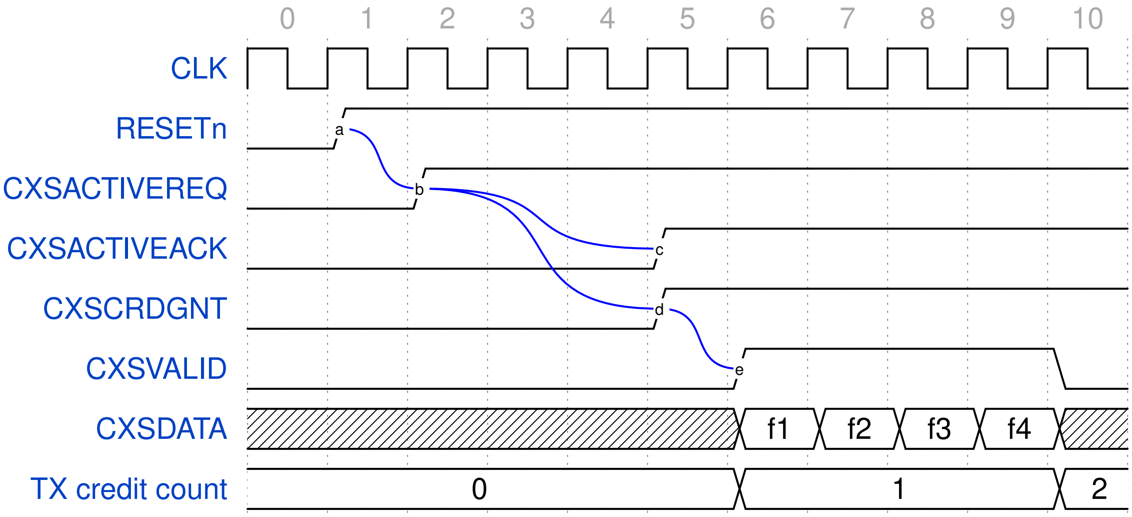

Figure 5-3 Interface activation example

Figure 5-4 shows the same example with more delay between the Receiver and Transmitter on the **CXSACTIVEACK** signal path than there is on the **CXSCRDGNT** path. Because of the additional delay, the Transmitter receives a credit while in the ACTIVATE state. However, the Transmitter cannot send a flit until **CXSACTIVEACK** signal goes HIGH in cycle 5 and it moves into the RUN state.

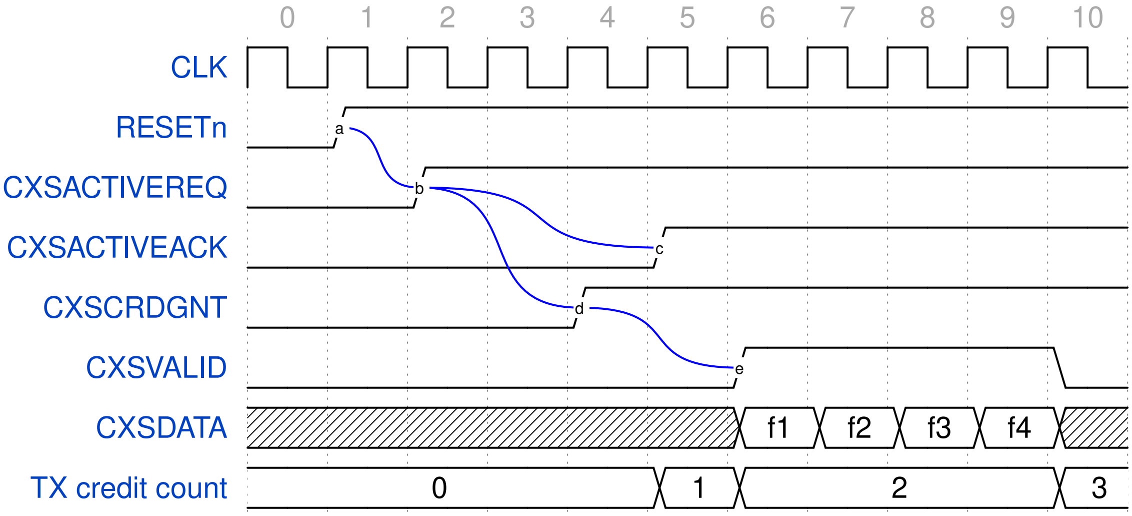

Figure 5-4 Interface activation example with race

Figure 5-5 shows an interface deactivation example. Both sides of the link start in RUN state. The Transmitter has no more flits to send and decides to deactivate the interface. The Transmitter deasserts the **CXSACTIVEREQ** signal, taking the interface into DEACTIVATE state. The Transmitter has a nonzero credit count, so it returns credits by asserting **CXSCRDRTN**.

The Receiver continues to grant credits for several cycles until it recognizes that the link is being deactivated. The Transmitter must return the additional credits as well, asserting **CXSCRDRTN** until its credit count is zero. The Receiver must not deassert the **CXSACTIVEACK** signal until it has all the credits and there are no credit grants in flight. The Transmitter will never see that the **CXSACTIVEACK** signal is deasserted while it still has credits.

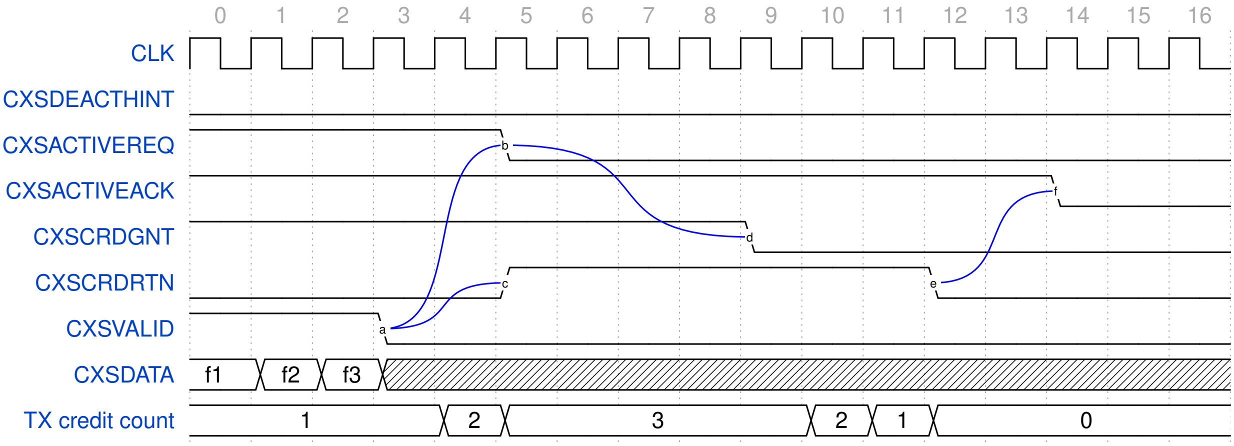

Figure 5-5 Interface deactivation example

Chapter 6
CXS packet continuous delivery guarantees

This chapter describes CXS packet continuous delivery guarantees. It contains the following section:

- Continuous delivery guarantees for CXS packets.

6.1 Continuous delivery guarantees for CXS packets

Some CXS implementations have a downstream interface that cannot tolerate interruptions in the data flow. PCIe is an example. An uninterrupted flow can be achieved using a CXS interface if either the Receiver or Transmitter has a store-and-forward buffer.

If the Receiver is built with the store-and-forward buffer, a packet must be able to be received in full before it is transmitted on the downstream interface. The Receiver must have enough buffering to store the largest packet that can be sent by the Transmitter. In this case, the Receiver and Transmitter can set the CXSCONTINUOUSDATA property False and the Transmitter is not required to buffer the packet.

If the Receiver does not have a buffer and requires continuous data, then it sets the CXSCONTINUOUSDATA property True and the attached Transmitter must also have CXSCONTINUOUSDATA as True. The Transmitter must then be able to issue all flits within a packet without dependence on another interface.

The Transmitter must also not attempt to deactivate the link if that deactivation could occur at a time when some, but not all, data of a packet has been issued.

If a continuous flow is required, the integrator must ensure that the Receiver has enough credits to cover the worst-case round-trip credit latency. This includes:

- The delay on **CXSCRDGNT** between Receiver and Transmitter.
- The maximum internal delay between **CXSTXCRDGNT** and **CXSTXVALID** when the Transmitter has a packet that is stalled waiting for credits. This is described by the CXS_MAX_CREDIT_LATENCY property of the Transmitter.
- The delay on the **CXSVALID** signal between the Transmitter and Receiver.
- The maximum internal Receiver delay between **CXSRXVALID** and **CXSRXCRDGNT**. This is described by the CXS_MAX_CREDIT_LATENCY property of the Receiver.

The maximum number of credits that a Receiver can issue is dependent on the size of its buffer, it can be described by its CXS_MAX_CREDIT property.

If the downstream interface is clocked slower than the CXS link, then it might not be necessary to transmit one flit every cycle. In this case, the number of credits required for the Receiver to maintain a constant flow might be fewer than the round-trip latency.

Chapter 7
CXS-Lite

This chapter decribes CXS-Lite, a subset of the CXS specification.

It contains the following sections:

7.1 CXS-Lite overview

CXS-Lite uses a subset of the CXS specification. It can be used for simple streaming applications, where packets are not needed and it is not required to identify multiple protocols within a stream.

CXS-Lite interfaces follow the rules of the CXS specification with the following property constraints:

- CXSMAXPKTPERFLIT is 1 which implies that:
  - — Packets start on the first byte of **CXSDATA**.
  - — Packets end on the last byte of **CXSDATA**.
  - — Packets do not span across multiple flits.
- CXSCNTLWIDTH is 0, all bytes of a flit are valid and there are no error indicators.
- CXS_LAST is False, there is no **CXSLAST** signal.
- CXS_PROTOCOL_TYPE is False, there is no **CXSPRCLTYPE** signal.

The following properties are inapplicable to CXS-Lite and can be undefined or take any value:

- CXSCONTINUOUSDATA
- CXSERRORFULLPKT
- CXS_START_ALIGNMENT

Table 7-1 shows the property constraints for CXS-Lite.

A dash means that the property can take any legal value, as shown for the core CXS interface.

Table 7-1 Property constraints for CXS-Lite

Name	Values	CXS	CXS-Lite
CXSMAXPKTPERFLIT	1..4	-	1
CXSDATAFLITWIDTH	8..2048 (multiples of 8)	-	-
CXSCNTLWIDTH	0..60 (selected values)	-	0
CXSCONTINUOUSDATA	True, False	-	N/A
CXS_LAST	True, False	-	False
CXS_PROTOCOL_TYPE	True, False	-	False
CXS_MAX_CREDIT	1..63	-	-
CXS_MAX_CREDIT_LATENCY	1..16	-	-
CXSCHECKTYPE	None, Odd_Byte_Parity	-	-
CXSERRORFULLPKT	True, False	-	N/A
CXSLINKCONTROL	None, Explicit_Credit_Return	-	-
CXS_SRCID_WIDTH	0..8	-	-
CXS_TGTID_WIDTH	0..8	-	-
CXS_START_ALIGNMENT	4, 16	-	N/A
CXS_USER_WIDTH	0..128	-	-

7.2 CXS-Lite signals

The CXS-Lite interface signals are shown in Table 7-2.

Table 7-2 CXS-Lite signals

Name	Width	Source	Property	Description
CLK	1	External	-	Clock signal.
RESETn	1	External	-	Reset signal.
CXSVALID	1	TX	-	Current cycle has a valid data flit.
CXSDATA	CXSDATAFLITWIDTH	TX	-	Data bytes being transmitted.
CXSSRCID	CXS_SRCID_WIDTH	TX	CXS_SRCID_WIDTH	Source ID of a CXS flit.
CXSTGTID	CXS_TGTID_WIDTH	TX	CXS_TGTID_WIDTH	Target ID of a CXS flit.
CXSUSER	CXS_USER_WIDTH	TX	CXS_USER_WIDTH	User bits for a flit.
CXSCRDGNT	1	RX	-	Grants a single credit to Transmitter.
CXSCRDRTN	1	TX	CXSLINKCONTROL	Returns a single credit to Receiver.
CXSACTIVEREQ	1	TX	CXSLINKCONTROL	Link activation or deactivation request.
CXSACTIVEACK	1	RX	CXSLINKCONTROL	Link activation or deactivation acknowledge.
CXSDEACTHINT	1	RX	CXSLINKCONTROL	Indicates Receiver wants the link deactivated.

Part A

Appendices

Chapter A1
Revisions

This appendix describes the technical changes between released issues of this specification.

Table A1-1 Issue A

Change	Location
First release of Version A	-

Table A1-2 Issue B

Change	Location
Support for multiple protocol streams.	CXS operation
Extension to interface protection signaling to support new signals.	CXS operation
Clarification of continuous delivery guarantees.	CXS packet continuous delivery guarantees

Table A1-3 Issue C

Change	Location
Support for simplified interfaces which send 1 packet per flit. Extended data width options when there is 1 packet per flit.	CXS interface properties
New rules added for CXSMAXPKTPERFLIT.	Table 2-2
Added definitions of clock and reset signals.	Table 2-1
Increased range of CXS_MAX_CREDIT to 1-63.	CXS interface properties

Table A1-4 Issue D

Change	Location
Added support for multi-node networks using source and target ID signaling.	• Support for multi-node networks using source and target IDs • Flit streams and interleaving
Added support to permit multiple packets per flit with a 2048 bit flit width.	CXSDATAFLITWIDTH and CXSMAXPKTPERFLIT properties in Table 2-2
A new property, CXS_START_ALIGNMENT, is added to allow packets to start on a 4-byte boundary rather than 16-byte.	• Packet position constraints • CXS interface properties • Flit width and packet configuration options
Control field widths and placement with 4-byte start alignment.	Table 4-7
Introduced CXS-Lite, as a packetless CXS interface.	CXS-Lite
Added support for User bits feature that allows to add a configurable number of bits to each CXS flit.	User bits
Other clarifications and corrections.	Table 3-2 in CXS interface checking signals
Increased the permitted maximum value of CXS_MAX_CREDIT_LATENCY to 32.	CXS interface properties