AMBA® CXS Protocol Specification

arm

ARM IHI 0079 (ID031218)

AMBA CXS Protocol Specification

Release Information

The following changes have been made to this specification.

Change history

Date	Issue	Confidentiality	Change
12 March 2018	A	Non-Confidential	First release

Proprietary Notice

This document is NON-CONFIDENTIAL and any use by you is subject to the terms of this notice and the Arm AMBA Specification Licence set about below.

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications.

Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

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 the Agreement shall prevail.

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

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

In this document, where the term Arm is used to refer to the company it means "Arm or any of its subsidiaries as appropriate"

ARM AMBA SPECIFICATION LICENCE

THIS END USER LICENCE AGREEMENT (“LICENCE”) IS A LEGAL AGREEMENT BETWEEN YOU (EITHER ASINGLE INDIVIDUAL, OR SINGLE LEGAL ENTITY) AND ARM LIMITED (“ARM”) FOR THE USE OF THE RELEVANT AMBA SPECIFICATION ACCOMPANYING THIS LICENCE. ARM IS ONLY WILLING TO LICENSE 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. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENCE, ARM IS UNWILLING TO LICENSE THE RELEVANT AMBA SPECIFICATION TO YOU AND YOU MAY NOT USE OR COPY THE RELEVANT AMBA SPECIFICATION AND YOU SHOULD PROMPTLY RETURN THE RELEVANT AMBA SPECIFICATION TO ARM.

“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 Arm 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.

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

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

`http://www.arm.com`

Contents
AMBA CXS Protocol Specification

Preface

This specification describes the CXS streaming interface protocol. The protocol can be used for any point-to-point packetized communication, but is optimized for the transport of CCIX packets.

This chapter contains the following sections:

About this document

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.

Typographic conventions

Convention	Meaning
italic	Introduces special terminology, denotes cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`underline`	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace italic`	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the ARM® Glossary. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Additional reading

This section lists relevant documents published by third parties:

PCI Express Base Specification `http://www.pcisig.com`

CCIX specification `https://www.ccixconsortium.com`

Feedback on this specification

If you have comments on the content of this specification, send e-mail to `errata@arm.com`. Give:

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

Feedback

Arm welcomes feedback on its documentation.

Chapter 1
Introduction

This chapter introduces the CXS protocol:

1.1 About the CXS streaming interface protocol

This specification describes the CXS streaming interface protocol. The protocol can be used for any point-to-point packetized communication. CXS protocol is optimized for wide interfaces, which allows the protocol to 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 CCIX 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.

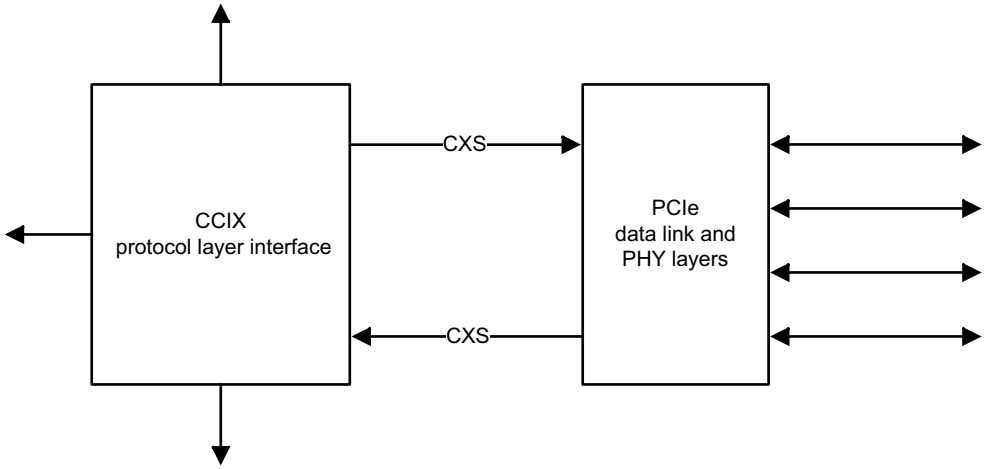

Figure 1-1 Typical implementation of CXS

Chapter 2
CXS operation

This chapter gives an overview of the operation of the CXS protocol and the properties which 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) and data is sent in one direction.

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

Table 2-1 CXS mandatory signals

Name	Direction	Description
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. Ignore if **CXSVALID** is not asserted.
CXSCNTL	Transmitter to receiver	Control information for identifying the start and end of packets within the data field. Ignore if **CXSVALID** is not asserted.
CXSCRDGNT	Receiver to Transmitter	Flow control information indicating that the receiver can accept one flit of data.

The transmitter transfers data by driving **CXSDATA**, placing packet control information on **CXSCNTL**, and asserting the **CXSVALID** signal. The data that is transferred in one cycle is known as a flit. A packet can occupy one or more flits. See *Packet examples* on page 4-30 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 **CXSCRDGNT** 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.
- Each cycle in which **CXSVALID** is asserted, the transmitter sends one flit of data, which consumes one credit.
- The maximum number of credits that a receiver grants a transmitter is IMPLEMENTATION DEFINED but must be no more than 15 credits. The transmitter must be able to track up to 15 credits at a time.
- A transmitter cannot use a credit to send a flit until the cycle after the **CXSCRDGNT** signal is asserted. A combinational path between **CXSCRDGNT** and **CXSVALID** is not recommended.
- Optionally, credits can be returned to the receiver without a flit transfer, using the **CXSCRDRTN** signal. When **CXSCRDRTN** is asserted, one credit is returned to the receiver every cycle. **CXSCRDRTN** and **CXSVALID** must not be asserted in the same cycle. See *Interface control with explicit credit return* on page 5-34 for more details.
- A receiver cannot reuse a consumed or returned credit until the cycle after **CXSVALID** or **CXSCRDRTN** is asserted. . A combinational path between **CXSVALID** and **CXSCRDGNT**, and between **CXSCRDRTN** and **CXSCRDGNT** is not recommended.
- 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. Figure 2-1 shows an example of a pair of CXS links between two components.

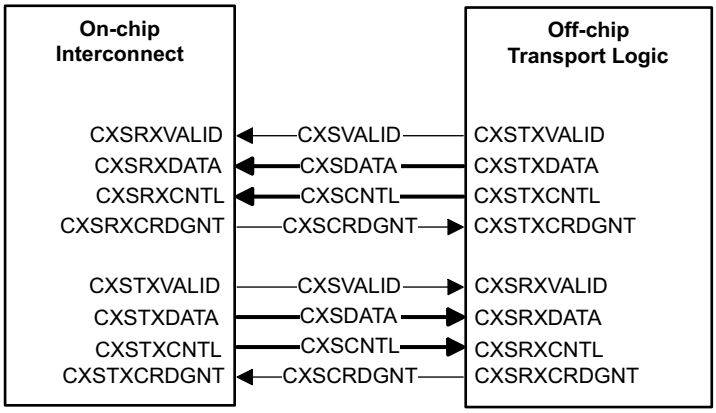

Figure 2-1 CXS connection example

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
CXSDATAFLITWIDTH	256, 512, 1024	256	Width of the **CXSDATA** signal in bits.
CXSMAXPKTPERFLIT[a]	2, 3, 4	2	Maximum number of packets that can be present in a single flit of data.
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 will not begin a packet until it can deliver the complete packet in consecutive cycles while credits are available.
CXSERRORFULLPKT[b]	TRUE, FALSE	FALSE	**Receiver:** If set to TRUE, this receiver requires that the length of every packet matches the packet length that is specified in the packet header. This includes packets which 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.
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 section *CXS interface checking signals* on page 3-21.
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* on page 5-34 for more details

a. If CXSDATAFLITWIDTH is 256, then CXSMAXPKTPERFLIT must be 2.

b. 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
CXSDATAFLITWIDTH	Transmitter and receiver must match.
CXSMAXPKTPERFLIT	Transmitter must be less than or equal to receiver.
CXSCONTINUOUSDATA	If receiver CXSCONTINUOUSDATA = TRUE, then transmitter CXSCONTINUOUSDATA must be TRUE.
CXSERRORFULLPKT	If receiver CXSERRORFULLPKT = TRUE, then transmitter CXSERRORFULLPKT must be TRUE.
CXSCHECKTYPE	Transmitter and receiver must match.
CXSLINKCONTROL	Transmitter and receiver must match.

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	Optional	Property	Description
CXSDATA	-	-	Data bytes being transmitted.
CXSCNTL	-	-	Information on packet start, end, and errors.
CXSVALID	-	-	Current cycle has a valid data flit.
CXSCRDGNT	-	-	Grants a single credit to transmitter.
CXSCRDRTN	Y	CXSLINKCONTROL	Returns a single credit to receiver.
CXSACTIVEREQ	Y	CXSLINKCONTROL	Link activation or deactivation request.
CXSACTIVEACK	Y	CXSLINKCONTROL	Link activation or deactivation acknowledge.
CXSDEACTHINT	Y	CXSLINKCONTROL	Indicates receiver wants the link deactivated.

Table 3-2 shows the width of the CXS signals, when present on the interface.

Table 3-2 Signal widths

Signal	Width
CXSDATA	CXSDATAFLITWIDTH (256, 512, or 1024 bits)
CXSCNTL	14, 18, 22, 27, 33, 36 or 44 bits. Depends on CXSMAXPKTPERFLIT and CXSDATAFLITWIDTH, See Table 4-2 on page 4-26.
CXSVALID	1 bit
CXSCRDGNT	1 bit
CXSCRDRTN	1 bit
CXSACTIVEREQ	1 bit
CXSACTIVEACK	1 bit
CXSDEACTHINT	1 bit

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].

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

Table 3-3 shows the check signals which are included if the CXSCHECKTYPE property is set to Odd_Byte_Parity. If the corresponding signal is not present on the interface, then the check signal is not present either.

Table 3-3 Check signal widths (bits)

Signal	Check signal	Signal width	Check signal width
CXSVALID	**CXSVALIDCHK**	1	1
CXSDATA	**CXSDATACHK**	256	32
		512	64
		1024	128
CXSCNTL	**CXSCNTLCHK**	14	2
		18	3
		22	3
		27	4
		33	5
		36	5
		44	6
CXSCRDRTN	**CXSCRDRTNCHK**	1	1
CSXACTIVEREQ	**CXSACTIVEREQCHK**	1	1
CXSCRDGNT	**CXSCRDGNTCHK**	1	1
CXSACTIVEACK	**CXSACTIVEACKCHK**	1	1
CXSDEACTHINT	N/A	1	-

Chapter 4
CXS packets

The data that is transmitted on the CXS interface is organized into packets. In CCIX or PCIe terminology, this is a Transaction Layer Packet (TLP). This chapter describes CXS packets:

4.1 Packet position constraints

CXS places restrictions on the placement of packets within each flit of data to simplify data path implementation.

- The first byte of a packet must be placed on an aligned 16-byte boundary.
- Subsequent bytes of the packet occupy subsequent bytes of the flit.
- Packets can end on any four-byte aligned boundary.
- When a packet has been started at a byte position in the current flit, that packet will occupy every subsequent byte in that flit until the packet ends or the flit ends.
- If there are remaining bytes in the packet when the flit ends, that packet will start at byte[0] of the next flit and occupy every subsequent byte position until the packet ends or the flit ends.
- When a packet ends within a flit, the remaining bytes in the flit can be unused.
- Any packet in a flit must begin at the first available 16-byte boundary relative to the start of the flit or the ending of a previous packet.

CXSMAXPKTPERFLIT specifies the maximum number of packets that can have bytes in a flit. There can be up to CXSMAXPKTPERFLIT new packets starting in a flit, and up to CXSMAXPKTPERFLIT packets ending in a flit.

If a packet started on a previous flit and is continuing in the current flit, that continuing packet will count towards the packet limit. That packet could therefore only have one less than CXSMAXPKTPERFLIT new packets starting.

4.2 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.

4.2.1 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 4-1.

Table 4-1 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. • All packet starts are 16-byte aligned. • The width of each pointer is $\log_2$(CXSDATAFLITWIDTH/128) bits. • The first byte of the X^{th} starting packet is (**START**[X]**PTR** << 4). • Start pointers are defined to be monotonically increasing, for example **START1PTR** must be greater than **START0PTR**.

Table 4-1 Packet control fields (continued)

Field	Description
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.
ENDERROR	Each bit in **ENDERROR** indicates that a packet is ending with an error condition in this flit • **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 X^{th} ending packet is therefore ((**END**[X]**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**[X]**PTR** and **END**[X]**PTR** might not point to the same packet, for example if a packet started in a previous flit.

4.2.2 Packet control field structure

Table 4-2 shows Packet control field widths and placement information for all combinations of CXSMAXPKTPERFLIT and CXSDATAFLITWIDTH. See *Packet examples* on page 4-30 for illustrations of how these structures are used.

Table 4-2 Packet control field widths and placement

CXSMAXPKTPERFLIT	CXSDATAFLITWIDTH	Width of CXSCNTL	Field	Bit positions in CXSCNTL
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]

Table 4-2 Packet control field widths and placement (continued)

CXSMAXPKTPERFLIT	CXSDATAFLITWIDTH	Width of CXSCNTL	Field	Bit positions in CXSCNTL
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]
3	256	-	Not legal: 256-bit interface has maximum of 2 packets per flit.	
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]
4	256	-	Not legal: 256-bit interface has maximum of 2 packets per flit.	

Table 4-2 Packet control field widths and placement (continued)

CXSMAXPKTPERFLIT	CXSDATAFLITWIDTH	Width of CXSCNTL	Field	Bit positions in CXSCNTL
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]
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]

4.3 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.

4.4 Packet examples

The following examples illustrate packet placement rules and the CXSCNTL field usage. Examples that are shown in Table 4-3 and Table 4-4 on page 4-31 both have CXSCONTINUOUSDATA = TRUE and CXSDATACHECK = None. Each data packet in the figures is shaded and has a unique identifier. Unused packet slots have dashes instead of identifiers.

Table 4-3 shows an example with 256-bit data, CXSDATAFLITWIDTH = 256. It has up to two packets per flit, CXSMAXPKTPERFLIT = 2.

Table 4-3 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 4-4 shows an example with 512-bit data, CXSDATAFLITWIDTH = 512. It has up to four packets per flit, CXSMAXPKTPERFLIT = 4.

Table 4-4 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[159128]		-	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	-

Chapter 5
CXS interface activation and deactivation

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 set to *None*, there are no signals for interface activation and deactivation, which requires that the receiver must always send credits when they are available and the transmitter must always be able to receive them. This chapter describes the activation and deactivation mechanisms when the property is set to Explicit_Credit_Return, and has the following sections:

5.1 Interface control with explicit credit return

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 sender is returning a previously granted credit without using it. Can only be asserted if **CXSVALID** is not asserted.
CXSACTIVEREQ	Transmitter to receiver	Link activation or deactivation request.
CXSACTIVEACK	Reciever to transmitter	Link activation or deactivation acknowledge.
CXSDEACTHINT	Reciever to transmitter	Hint that receiver would like 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.

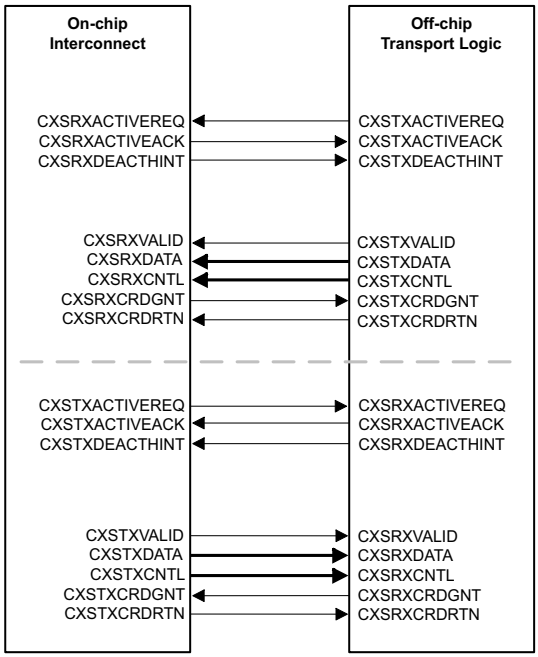

Figure 5-1 Example with two CXS links

5.2 Request and acknowledgement 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:

RUN There is an ongoing exchange of flits and credits between the two components.

STOP The interface is in an idle state and is not operating. All credits are held by the receiver and the transmitter is not permitted to send any flits.

ACTIVATE This state is used when transitioning from the STOP state to the RUN state.

DEACTIVATE
This state is used when transitioning from the RUN state to the STOP state.

RUN and STOP are stable states and when one of these states is entered, a channel can remain in this state for an indefinite time.

DEACTIVATE and ACTIVATE are transient states. It is expected that when one of these states is entered a channel will move to the next stable state in a relatively short time.

Note

The 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 **CXSACTIVEREQ** and **CXSACTIVEACK** respectively, on each transition.

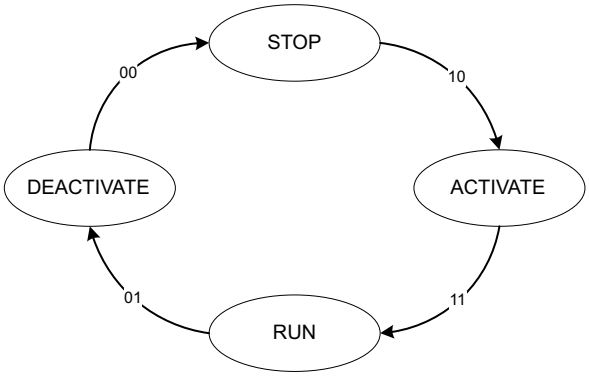

Figure 5-2 Request and acknowledge handshake states

5.3 Response to a new state

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

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 state change from RUN to DEACTIVATE is the point that flits, credits, and credit returns stop being sent.

The protocol requires that the receiver has stopped sending credits and has had all credits returned before it signals the change from DEACTIVATE to STOP by deasserting **CXSACTIVEACK**.

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 **CXSACTIVEACK**, to move from ACTIVATE to RUN, and starts sending credits:
  - The receiver is permitted to assert **CXSCRDGNT** in the same cycle that **CXSACTIVEACK** is asserted.
  - The credit might be received at the transmitter before its local **CXSACTIVEACK** is asserted.
  - Therefore, the transmitter must accept credits while in the ACTIVATE or RUN state.
- The transmitter stops sending flits and then deasserts **CXSACTIVEREQ**, to move from RUN to DEACTIVATE:
  - The transmitter must not send flits when **CXSACTIVEREQ** is deasserted.
  - An in-flight flit might be received at the receiver after its local **CXSACTIVEREQ** 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** 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 are dependent 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**
- **CXSACTIVEHINT**

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. That number of flip-flop stages will most likely be applied to all the signals on the interface.

The exception is **CXSACTIVEREQ**. It is common for the clock in the receiver to stop during the STOP state due to clock gating. The assertion of **CXSACTIVEREQ** 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. **CXSACTIVEREQ** 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 **CXSACTIVEREQ** and **CXSACTIVEACK** participate in a four-phase handshake and can run asynchronously.

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

CXSACTIVEACK and **CXSDEACTHINT** are asserted only when both transmitter and receiver clocks are running and therefore must be treated as synchronous signals. This restriction means that **CXSACTIVEACK** and **CXSDEACTHINT** must not be multicycle path between receiver and transmitter, although they can have multiple flip-flops as 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. At Time 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 at Time 1.
3. The transmitter then waits for the receiver to wake up and assert **CXSACTIVEACK** at Time 6.
4. In this case, **CXSCRDGNT** is asserted the same cycle as **CXSACTIVEACK**.
5. Having received a credit, the transmitter sends a flit at Time 7.
6. Transmitter continues to send while it is receiving credits.

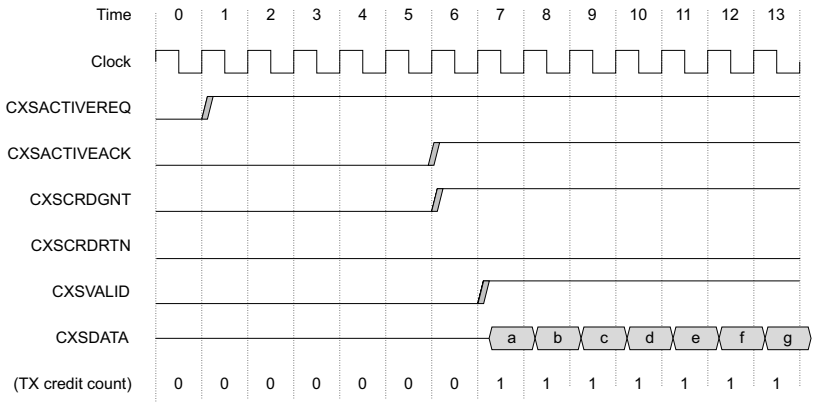

Figure 5-3 Interface activation example

Figure 5-4 shows the same example with more delay between the receiver and transmitter on the **CXSACTIVEACK** 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** goes high at Time 6 and it moves into the RUN state.

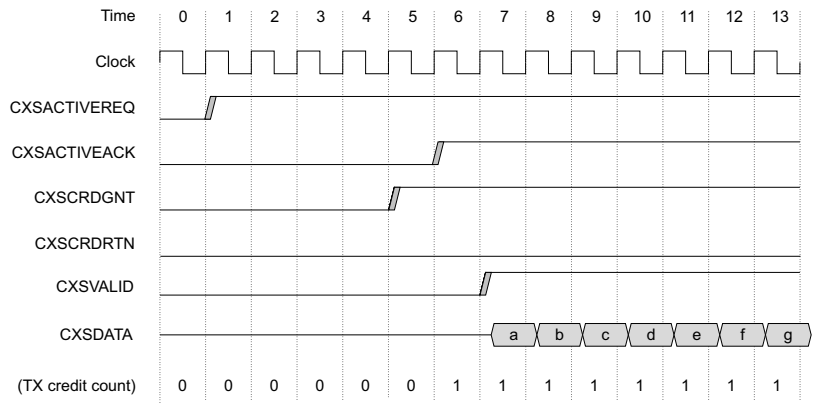

Figure 5-4 Interface activation example with race

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 **CXSACTIVEREQ**, 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 **CXSACTIVEACK** until it has all the credits and there are no credit grants in flight. The transmitter will never see that **CXSACTIVEACK** is deasserted while it still has credits.

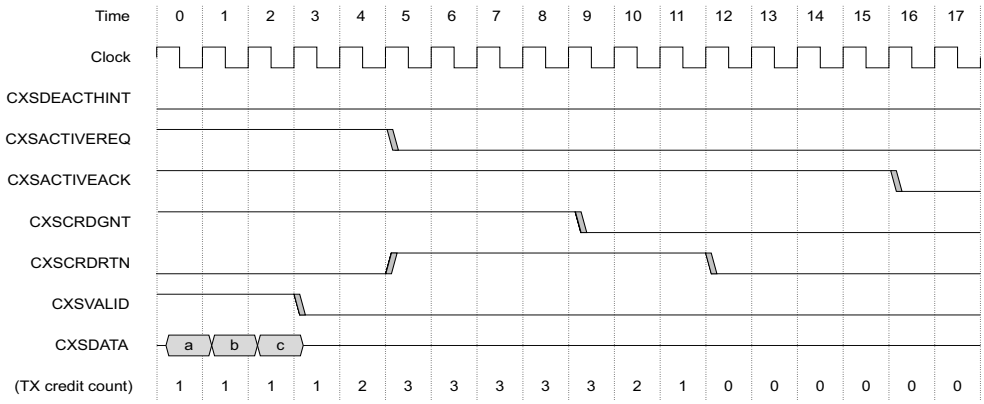

Figure 5-5 Interface deactivation example

Chapter 6
CXS packet continuous delivery guarantees

This chapter describes CXS packet continuous delivery guarantees:

6.1 Continuous delivery guarantees

A receiver on a CXS interface can be built with a *store and forward* approach. In this approach, the packet is fully received before it is used or sent out on another interface. Any receiver that is built in this way will not need to assert the CXSCONTINUOUSDATA property.

If lowest latency is wanted, a receiver can begin transmission of a packet on another interface before the full packet has been received on the CXS interface. This design will result in lower latency. However, if the downstream interface cannot tolerate interruptions in the data flow (for example, PCIe), then there is an additional requirement on the CXS interface. This requirement is to deliver data at a rate high enough to ensure uninterrupted transmission of the full packet on the other interface.

The CXSCONTINUOUSDATA property is set if the transmitter can guarantee that a packet will not be started on the CXS interface until the transmitter can deliver all the data of that packet in subsequent cycles with no interruptions. If a transmitter does not assert the CXSCONTINUOUSDATA property, the receiver cannot rely on the continuous delivery of data.

A transmitter that guarantees continuous delivery will generally be designed such that a packet is not started on the interface until all the following conditions are true:

- The full packet is available in the transmitter.
- The packet is stored in the same clock domain or a higher frequency clock domain than the CXS interface.
- The data path, clocking, and arbitration logic can guarantee delivery of the packet at full interface bandwidth.

One further implication is that the transmitter must 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.

For continuous delivery to be effective, the receiver must grant enough credits, and therefore have sufficient buffering, to keep the packet flowing at the required bandwidth. Specifying the buffering that is required is outside the scope of this specification. In general, the number of credits that are needed will be a function of:

- The delay on **CXSCRDGNT** between receiver and transmitter.
- The maximum internal transmitter delay between **CXSTXCRDGNT** and **CXSTXVALID** when the transmitter has a packet that is stalled waiting for credits.
- The delay on **CXSVALID** between the transmitter and receiver.
- The microarchitecture of the receiver.
- The bandwidth of the downstream interface.

Appendix A
Revisions

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

Table A-1 Issue A

Change	Location
First release of Version A	–