AMBA® AXI-Stream

Protocol Specification

arm

ARM IHI 0051B (ID040921)

AMBA AXI-Stream
Protocol Specification

Release Information

The following changes have been made to this specification.

Change history

Date	Issue	Confidentiality	Change
03 March 2010	A	Non-Confidential	First release, version 1.0
09 April 2021	B	Non-Confidential	Two protocols described, AXI4-Stream and AXI5-Stream. Regularized terminology to be Transmitter and Receiver.

Proprietary Notice

Arm Limited. Company 02557590 registered in England.
110 Fulbourn Road, Cambridge, England CB1 9NJ.
LES-PRE-21451 version 2.2

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.

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 AXI-Stream Protocol Specification

Preface

This preface introduces the *AMBA AXI-Stream Protocol Specification*. It contains the following sections:

About this specification

This specification defines the AMBA AXI-Stream protocols:

- AXI4-Stream
- AXI5-Stream

The collective term AXI-Stream is used in instances that describes common features.

Intended audience

This specification is written for hardware and software engineers who want to become familiar with the *Advanced Microcontroller Bus Architecture* (AMBA) and engineers who design systems and modules that are compatible with the AMBA AXI-Stream protocol.

Using this specification

This specification is organized into the following chapters:

Chapter 1 *Introduction*

Read this for an introduction to the AXI-Stream protocol and some examples of stream types.

Chapter 2 *Interface Signals*

Read this for a description of the AXI-Stream signals and the baseline rules governing signal use.

Chapter 3 *Default Signaling Requirements*

Read this for a description of the default signaling requirements.

Chapter 4 *Transfer Interleaving and Ordering*

Read this for a description of the stream interleaving and ordering restrictions.

Chapter 5 *Interface parity protection*

Read this for a description of parity protection in the AXI5-Stream interface.

Appendix A *Comparison with the AXI Write Data Channel*

Read this for a description of the key differences between the AXI-Stream interface and the AXI write data channel.

Appendix B *Interface Signals*

Read this for a summary of the interface signals used in AXI4-Stream and AXI5-Stream.

Appendix C *Revisions*

Read this for a description of the revisions of this specification.

Conventions

Conventions that this book can use are described in:

- *Typographical* on page ix
- *Timing diagrams* on page ix
- *Signals* on page ix

Typographical

The typographical conventions are:

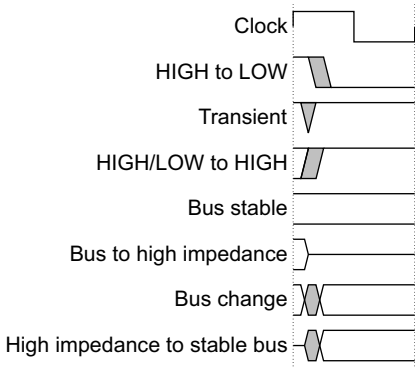
italic Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.

bold Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

Timing diagrams

The components used in timing diagrams are explained in the figure *Key to timing diagram conventions*. 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.



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 *Key to timing diagram conventions*. If a timing diagram shows a single-bit signal in this way, then its value does not affect the accompanying description.

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 The start or end of a signal name denotes an active-LOW signal.

Additional reading

This section lists publications by Arm and by third parties.

See Arm Developer `https://developer.arm.com/documentation` for access to Arm documentation.

Arm publications

This book contains information that is specific to this product. See the following documents for other relevant information:

- *AMBA AXI Protocol Specification* (ARM IHI 0022)

Feedback

Arm welcomes feedback on this product and its documentation.

Feedback on content

If you have comments on content then send an email to `errata@arm.com`. Give:

- The title, AMBA AXI-Stream Protocol Specification
- The number, ARM IHI 0051B
- The page numbers to which your comments apply
- A concise explanation of your comments

Arm also welcomes general suggestions for additions and improvements.

Progressive terminology commitment

Arm values inclusive communities. Arm recognizes that we and our industry have 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 describes the AXI-Stream protocol and gives some examples of stream types. It contains the following sections:

1.1 About the AXI-Stream protocol

The AXI-Stream protocol is used as a standard interface to exchange data between connected components. AXI-Stream is a point-to-point protocol, connecting a single Transmitter and a single Receiver.

An interconnect can be employed to provide connectivity between multiple AXI-Stream components. The interconnect can perform upsizing, see *Upsizing considerations* on page 2-23, downsizing, see *Downsizing considerations* on page 2-22, and clock domain crossing, see *Clock* on page 2-28.

A variety of higher-level protocols can use AXI-stream for transportation, even on a single AXI-Stream channel.

This specification describes how data is transferred but does not describe the meaning of the data.

Issue A of this specification defines the AXI4-Stream protocol.

Issue B introduces the AXI5-Stream protocol, extending the AXI4-Stream protocol with additional features:

- Wake-up signaling
- Interface protection using parity

1.1.1 Byte definitions

The following byte definitions are used in this specification:

Data byte

A byte of data containing valid information. The data byte is transmitted between the source and destination.

Position byte

A byte that indicates the relative positions of data bytes within the stream. A position byte is a placeholder that does not contain any relevant data values that are transmitted between the source and destination.

Null byte

A byte that does not contain any data information or any information about the relative position of data bytes within the stream.

1.1.2 Stream terms

The following stream terms are used in this specification:

Transfer

A single transfer of data across an AXI-Stream interface. A single transfer is defined by a single **TVALID** and **TREADY** signal handshake. See *Handshake signaling* on page 2-18.

Packet

A group of bytes that are transported together across an AXI-Stream interface. A packet can consist of a single transfer or multiple transfers. Interconnect components can use packets to deal more efficiently with a stream in packet-sized groups. A packet is similar to an AXI burst.

Frame

A frame contains an integer number of packets. A frame can have a very large number of bytes, for example an entire video frame buffer. A frame is the highest level of byte grouping in an AXI-Stream interface.

Data stream

The transport of data from one source to one destination.

A data stream can be:

- A series of individual byte transfers.
- A series of byte transfers grouped together in packets.

1.2 Data streams

AXI-Stream enables many forms of data streams. This section provides some examples of different data stream styles.

The examples shown in this section contain hexadecimal values across a 4-byte wide bus, with columns ordered in time from left to right.

1.2.1 Byte stream

A byte stream is the transmission of data bytes and null bytes. A byte stream transfer can transmit any number of data bytes, dependent upon the data width. Null bytes can be inserted or removed from the stream at any point between the Transmitter and Receiver.

Figure 1-1 provides two examples of a byte stream, with each vertical column representing the bytes in a single transfer.

The two examples given in Figure 1-1 transfers identical information because a null byte conveys no information in a stream.

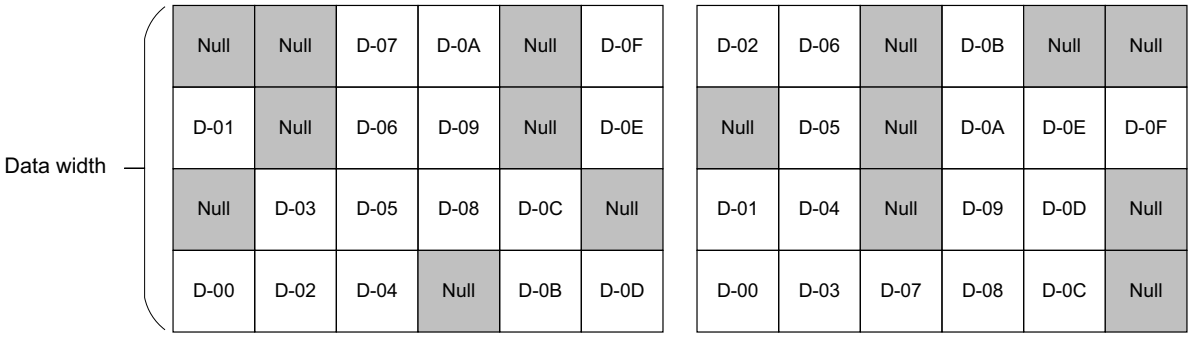


Figure 1-1 Example of byte streams

1.2.2 Continuous aligned stream

A continuous aligned stream is the transmission of a number of data bytes where every packet has no position bytes or null bytes.

Figure 1-2 shows an example of a continuous aligned stream.

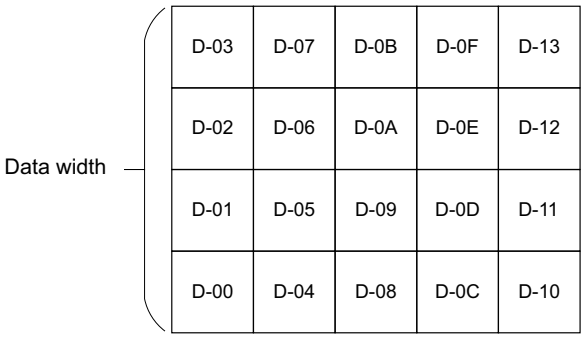


Figure 1-2 Example of a continuous aligned stream

1.2.3 Continuous unaligned stream

A continuous unaligned stream is the transmission of a number of data bytes with no position bytes between the first data byte and the last data byte of each packet.

A continuous unaligned stream can have any number of contiguous position bytes at the start, at the end, or both at the start and end of a packet.

Figure 1-3 shows two examples of a continuous unaligned stream.

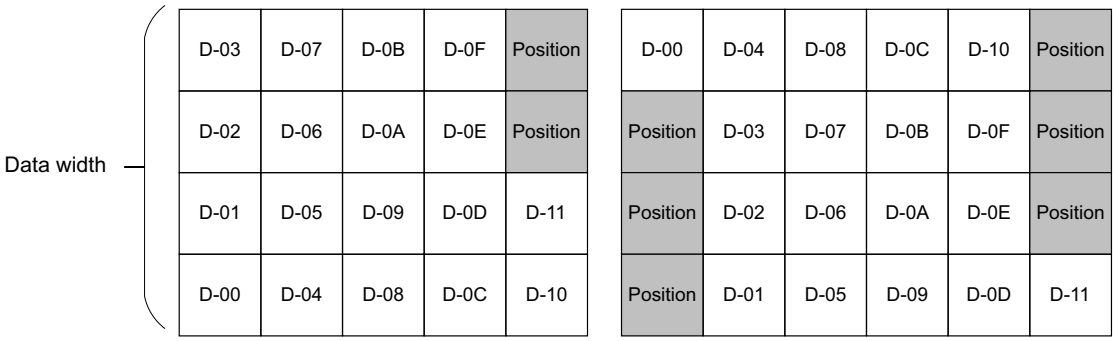


Figure 1-3 Example of continuous unaligned streams

1.2.4 Sparse stream

A sparse stream is the transmission of a number of data bytes and position bytes. All data bytes must be transmitted from source to destination. The relative position of all data byes and position bytes must not change between source and destination. Typically, the majority of the bytes are data bytes.

Figure 1-4 shows an example of a sparse stream.

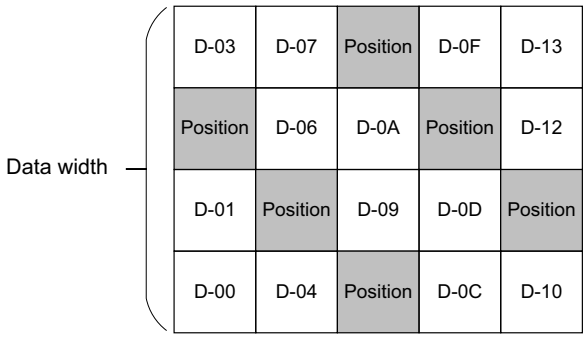


Figure 1-4 Example of a sparse stream

Chapter 2
Interface Signals

This chapter describes the signal requirements of the AXI-Stream interface. It contains the following sections:

2.1 Signal list

This section describes the interface signals for the AXI-Stream interface.

Some signals have a fixed width and some can take a variety of widths. Where the width is not fixed, it is described using a property. If the property value is zero, then the signal is not present on the interface.

The interface signals are listed in Table 2-1.

Table 2-1 Interface signals list

Signal	Source	Width	Description
ACLK	Clock	1	**ACLK** is a global clock signal. All signals are sampled on the rising edge of **ACLK**. See *Clock* on page 2-28 for more information.
ARESETn	Reset	1	**ARESETn** is a global reset signal. See *Reset* on page 2-28 for more information.
TVALID	Transmitter	1	**TVALID** indicates the Transmitter is driving a valid transfer. A transfer takes place when both **TVALID** and **TREADY** are asserted. See *Handshake signaling* on page 2-18.
TREADY	Receiver	1	**TREADY** indicates that a Receiver can accept a transfer. See *Handshake signaling* on page 2-18.
TDATA	Transmitter	TDATA_WIDTH	**TDATA** is the primary payload used to provide the data that is passing across the interface. TDATA_WIDTH must be an integer number of bytes and is recommended to be 8, 16, 32, 64, 128, 256, 512 or 1024-bits. See *Data signaling* on page 2-21.
TSTRB	Transmitter	TDATA_WIDTH/8	**TSTRB** is the byte qualifier that indicates whether the content of the associated byte of **TDATA** is processed as a data byte or a position byte. See *Byte qualifiers* on page 2-24.
TKEEP	Transmitter	TDATA_WIDTH/8	**TKEEP** is the byte qualifier that indicates whether content of the associated byte of **TDATA** is processed as part of the data stream. See *Byte qualifiers* on page 2-24.
TLAST	Transmitter	1	**TLAST** indicates the boundary of a packet. See *Packet boundaries* on page 2-26.
TID	Transmitter	TID_WIDTH	**TID** is a data stream identifier. TID_WIDTH is recommended to be no more than 8. See *Source and destination signaling* on page 2-27.

Table 2-1 Interface signals list (continued)

Signal	Source	Width	Description
TDEST	Transmitter	TDEST_WIDTH	**TDEST** provides routing information for the data stream. TDEST_WIDTH is recommended to be no more than 8. See *Source and destination signaling* on page 2-27.
TUSER	Transmitter	TUSER_WIDTH	**TUSER** is a user-defined sideband information that can be transmitted along the data stream. TUSER_WIDTH is recommended to be an integer multiple of TDATA_WIDTH/8. See *User signaling* on page 2-29.
TWAKEUP	Transmitter	1	**TWAKEUP** identifies any activity associated with AXI-Stream interface. See *Wake-up signaling* on page 2-20.

2.2 Handshake signaling

This section gives details of the handshake signaling and defines the relationship of the **TVALID** and **TREADY** signals.

The **TVALID** and **TREADY** handshake determines when information is passed across the interface. A two-way flow control mechanism enables both the Transmitter and Receiver to control the rate at which the data and control information is transmitted across the interface.

For a transfer to occur, both **TVALID** and **TREADY** must be asserted. Either **TVALID** or **TREADY** can be asserted first, or both can be asserted in the same **ACLK** cycle.

A Transmitter is not permitted to wait until **TREADY** is asserted before asserting **TVALID**. Once **TVALID** is asserted, it must remain asserted until the handshake occurs.

A Receiver is permitted to wait for **TVALID** to be asserted before asserting **TREADY**. It is permitted that a Receiver asserts and deasserts **TREADY** without **TVALID** being asserted.

The following sections give examples of the handshake sequence.

2.2.1 Handshake with TVALID asserted before TREADY

In Figure 2-1, the Transmitter presents the data and control information and asserts **TVALID** as HIGH. The valid data bytes and control information from the Transmitter must remain unchanged once **TVALID** has been asserted by the Transmitter. The Receiver can accept the data and control information once **TREADY** is HIGH. The transfer takes place once the Receiver asserts **TREADY** HIGH. Figure 2-1 shows the transfer occurs at T3.

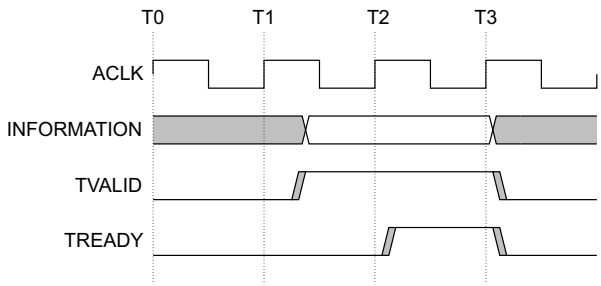


Figure 2-1 Handshake with TVALID asserted before TREADY

2.2.2 Handshake with TREADY asserted before TVALID

In Figure 2-2, the Receiver drives **TREADY** HIGH before the data and control information is valid. This indicates the Receiver can accept the data and control information in a single **ACLK** cycle. In this case, the transfer occurs once the Transmitter drives **TVALID** HIGH. Figure 2-2 shows the transfer occurring at T3.

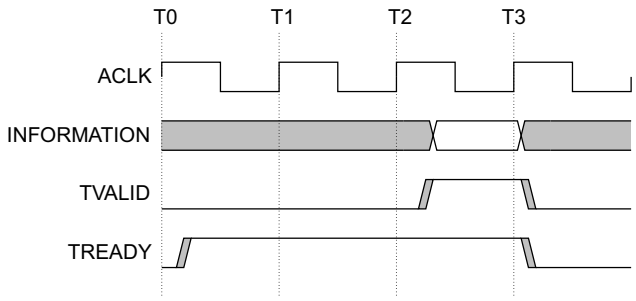


Figure 2-2 Handshake with TREADY asserted before TVALID

2.2.3 Handshake with TVALID and TREADY asserted simultaneously

In Figure 2-3, the Transmitter asserts **TVALID** HIGH and the Receiver asserts **TREADY** HIGH in the same **ACLK** cycle. In this case, transfer takes place in the same cycle, as shown in T2 of Figure 2-3.

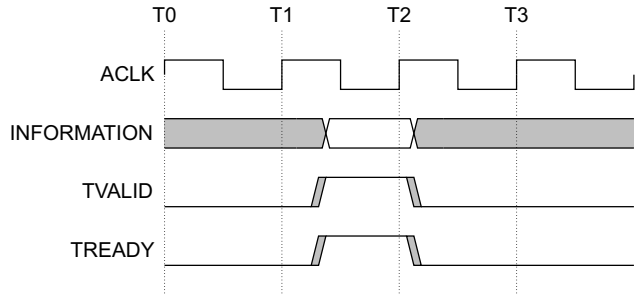


Figure 2-3 Handshake with TVALID and TREADY asserted simultaneously

2.3 Wake-up signaling

The wake-up signal, **TWAKEUP**, is used to indicate any activity associated with an AXI5-Stream interface. **TWAKEUP** can be routed to a clock controller, or similar component, to enable power and clocks to connected components.

Table 2-2 shows the **TWAKEUP** description.

Table 2-2 TWAKEUP signal

Signal	Source	Width	Description
TWAKEUP	Transmitter	1	**TWAKEUP** indicates if there is activity associated with an AXI5-Stream interface.

The Wakeup_Signal property is used to indicate whether a component includes wake-up signaling:

True **TWAKEUP** is present.

False **TWAKEUP** is not present. If the Wakeup_Signal property is not declared, it is considered False.

The Wakeup_Signal property can only be True for AXI5-Stream interfaces.

The rules for **TWAKEUP** are:

- **TWAKEUP** is synchronous to **ACLK** and must be suitable for sampling asynchronously in a different clock domain. This requires **TWAKEUP** to be glitch-free. This can be achieved, for example, by being generated directly from a register, or from a glitch-free OR tree.
- **TWAKEUP** is permitted to be asserted at any point before or after **TVALID** has been asserted.
- A Receiver is permitted to wait for **TWAKEUP** to be asserted before asserting **TREADY**. This means that the interface could deadlock if **TWAKEUP** is present but never asserted.
- If **TWAKEUP** and **TVALID** are HIGH in the same cycle, **TWAKEUP** must remain asserted until **TREADY** is asserted.

It is recommended:

- **TWAKEUP** is asserted at least one cycle prior to the assertion of **TVALID**.
- **TWAKEUP** is deasserted when no further transfers are required.

It is permitted, but not recommended, for **TWAKEUP** to assert then deassert without a transfer occurring.

2.4 Data signaling

This section describes the data signaling requirements of the AXI-Stream interface.

TDATA is the primary payload of the AXI-Stream interface and is used to transport data from a source to a destination.

2.4.1 Byte locations

The low order bytes of the data bus are the earlier bytes in a data stream.

For a fully packed stream, with no null bytes, the location of a given byte in the stream can be determined using the following:

Within a data stream:

- The sequence of bytes `n` are numbered from 0 upwards
- Each transfer, `t`, in a sequence is numbered from 0 upwards
- The width of the data bus is `w` bytes
- `INT(x)` is the rounded-down integer value of `x`

Byte `n` is contained within transfer `t` where:

```
t = INT(n/w)
```

at byte position `b` where:

```
b = n - (t * w)
```

which is contained within:

```
TDATA[(8b+7):8b]
```

2.4.2 Byte types

The AXI-Stream protocol defines three byte types:

Data byte

The following attributes of data bytes must not change between source and destination:

- The number of data bytes
- The associated **TDATA** signal data values
- The relative position of the data byte in the stream, with respect to other data bytes and position bytes

Position byte

The following attributes of position bytes must not change between source and destination:

- The number of position bytes.
- The relative position of the position byte in the stream, with respect to other data bytes and position bytes.

Null byte

A null byte contains no information.

Null bytes may be inserted and removed from the stream. They are not required to be transmitted between source and destination.

An interconnect must not modify the number or relative position of data bytes or position bytes within a stream.

An interconnect is permitted to insert or remove null bytes from a stream. For example, when upsizing to a wider data bus, null bytes can be inserted to form a complete data width transfer.

Transmitter and Receiver components are not required to support null bytes. Any interconnect with the capacity of inserting null bytes into a stream should also be capable of removing them before the stream arrives at a destination that is not capable of handling null bytes.

2.4.3 Data merging, packing, and width conversion

It is recommended to build AXI-Stream components with an adaptable data width. An adaptable data width can be configured to match the system in which a component is integrated. If an adaptable bus width is not possible, it might be necessary to merge, pack, upsize, or downsize streams.

It is expected that in most applications, the data width is a power-of-two bytes wide. However, this is not a requirement of the protocol. The data width must be an integer number of bytes wide.

All upsizing and downsizing operations are required to preserve any data bytes and position bytes. If an upsizing or downsizing operation does not have sufficient bytes of data to compile a full width output, it can add null bytes.

Merging considerations

Merging is the process of combining bytes from two different transfers into one transfer.

Merging can take place when a transfer has null bytes that can be removed, allowing later data or position bytes to be included. Merging can occur in association with packing. See *Packing considerations*.

The rules associated with merging are:

- Only transfers with matching **TID** and **TDEST** identifiers can be merged.
- If the current transfer is signaled with **TLAST**, it must not be merged with a following transfer. **TLAST** also indicates that there is no following transfer that can be merged, so the transmission of the current transfer should not be delayed.
- The correct order of data bytes and position bytes must be maintained.
- The correct associations of **TLAST**, **TSTRB**, and **TUSER** must be maintained.

Partial merging is allowed. For example, if the current 4 byte transfer only has three data bytes, it is permissible to merge a single data byte from the following transfer. Partial merging is still acceptable even if the following transfer has more than one data byte.

Packing considerations

Packing is the process of removing null bytes from a stream.

Packing generally takes place in association with some other activity such as upsizing, downsizing, or merging.

A data stream that uses **TKEEP** associations can be packed by the removal of null bytes, which provides a more compressed data stream. Fully packed data is not required, so null bytes might be present both before and after packing.

Downsizing considerations

Downsizing is the conversion from a given data bus width to a narrower data bus width.

This process typically involves the generation of multiple output transfers for a single input transfer. Typically, downsizing is by a factor of 2:1, but other downsizing ratios can be performed.

The rules associated with downsizing are:

- The order of the bytes in the input and output streams must match.
- **TSTRB** must be downsized in a similar manner to the data, ensuring the correct relationship between data bytes and position bytes is maintained.
- **TLAST** must only be associated with the final transfer of a downsizing operation.

- **TID** and **TDEST** of all output transfers must match the value of the input transfer.
- **TUSER** information must remain associated with the same byte.
- A transfer that only contains null bytes can be suppressed, if **TKEEP** and **TLAST** are deasserted.

Upsizing considerations

Upsizing is the conversion from a given data bus width to a wider data bus width.

This process can be combined with merging so that a single output transfer can be generated for multiple input transfers.

Upsizing can be a more complex process if upsizing is combined with merging. This means when receiving a transfer, an upsizer will not always be able to determine if the following transfer is in the same packet.

If there are insufficient transfers to construct a full width upsized stream, null bytes can be added to the transfer.

The rules associated with upsizing are:

- The order of the bytes in the input and output streams must match.
- **TSTRB** must be upsized in a similar manner to the data, ensuring the correct relationship between data bytes and position bytes is maintained.
- **TLAST** must be preserved.
- **TID** and **TDEST** of an output transfer must match the value of the input transfers.
- **TUSER** information must remain associated with the same byte.

2.5 Byte qualifiers

This section defines the two byte qualifiers supported by the AXI-Stream protocol:

TKEEP A byte qualifier signal used to indicate whether the content of the associated byte must be transported to the destination.

TSTRB A byte qualifier signal used to indicate whether the content of the associated byte is a data byte or a position byte.

Each bit of the **TKEEP** and **TSTRB** is associated with a byte of payload:

- **TKEEP[x]** is associated with **TDATA[(8x+7):8x]**
- **TSTRB[x]** is associated with **TDATA[(8x+7):8x]**

2.5.1 TKEEP qualification

When **TKEEP** is asserted, it indicates that the associated byte must be transmitted to the destination.

When **TKEEP** is deasserted, it indicates a null byte that can be removed from the stream. A transfer is permitted with all **TKEEP** bits deasserted.

A transfer with all **TKEEP** bits deasserted is permitted to be suppressed if **TLAST** is deasserted. See *Packet boundaries* on page 2-26 for the usage model.

Transmitters and Receivers are not required to handle null bytes. Any interconnect with the capacity of inserting null bytes in a stream should also be capable of removing them before the stream arrives at a destination that is not capable of handling null bytes. An interface that does not support null bytes does not include the **TKEEP** signal.

2.5.2 TSTRB qualification

When **TKEEP** is asserted, **TSTRB** is used to indicate whether the associated byte is a data byte or a position byte.

When **TSTRB** is asserted, it indicates that the associated byte contains valid information and is a data byte. When **TSTRB** is deasserted, it indicates that the associated byte does not contain valid information and is a position byte.

A position byte is used to indicate the correct relative position of the data bytes within the stream. Position bytes are typically used when the data stream is performing a partial update of information at the destination.

Since the data associated with a position byte is not valid, an interconnect need not transmit the **TDATA** associated with a byte for which **TSTRB** is deasserted.

2.5.3 TKEEP and TSTRB combinations

Table 2-3 lists the valid combinations for **TKEEP** and **TSTRB** byte qualifications. Not all components require **TKEEP** and **TSTRB** byte qualifiers. See *Optional TKEEP and TSTRB* on page 3-34.

Table 2-3 TKEEP and TSTRB byte qualifications

TKEEP	TSTRB	Data Type	Description
HIGH	HIGH	Data byte	The associated byte contains valid information that must be transmitted between source and destination.
HIGH	LOW	Position byte	The associated byte indicates the relative position of the data bytes in a stream, but does not contain any relevant data values.
LOW	LOW	Null byte	The associated byte does not contain information and can be removed from the stream.
LOW	HIGH	Reserved	Must not be used.

2.6 Packet boundaries

A packet is a grouping of bytes that are transmitted together across the interface. Interconnect components can be made more efficient by handling transfers that are grouped in packets. An AXI-Stream packet is similar to an AXI burst.

The signals to be considered during a packet transfer are **TID**, **TDEST**, and **TLAST**. For more information on **TID** and **TDEST**, see *Source and destination signaling* on page 2-27.

The uses of **TLAST** are:

- When deasserted, **TLAST** indicates that another transfer can follow. This means it is acceptable to delay the current transfer for the purpose of upsizing, downsizing, or merging.
- When asserted, **TLAST** can be used by a destination to indicate a packet boundary.
- When asserted, **TLAST** indicates an efficient point to make an arbitration change on a shared link.

Note

It is not required that arbitration only occurs on a **TLAST** boundary. **TLAST** can be used to potentially improve efficiency by keeping transfers in the same packet together.

TLAST can be used to transmit information between the source and destination. The number of packets and the number of assertions of **TLAST** must be preserved between the Transmitter and Receiver.

No explicit signaling of the start of a packet boundary is given in the protocol. The start of a packet is determined as:

- The first occurrence of a **TID** and **TDEST** pair after reset
- The first transfer after the end of the preceding packet for any unique set of **TID** and **TDEST** values

All bytes within a packet are from the same source and for the same destination and have the same **TID** and **TDEST** values.

The merging of transfers that belong to different packets is not permitted. This requires that two transfers with the same **TID** and **TDEST** values must not merge if the earlier transfer has the **TLAST** asserted. See *Merging considerations* on page 2-22 for more information.

The merging of transfers with different **TID** or **TDEST** values is not permitted.

2.6.1 Transfer with zero data or position bytes

A transfer can have **TLAST** asserted but contain no data or position bytes. This can be used to:

- Indicate the end of a packet when there are no more data or position bytes to transmit.
- Push through any data that is held in intermediate buffers.
- Complete an operation at an end-point that is expecting a **TLAST** at the end of a packet.

A transfer that has **TLAST** asserted, but does not have any data or position bytes, can be merged with an earlier transfer, if:

- Both transfers have the same **TID** and **TDEST** values.
- The earlier transfer has **TLAST** deasserted.

See *Merging considerations* on page 2-22 for more information.

A transfer being sent with zero data bytes pushes through all transfers between a Transmitter and Receiver because reordering is not supported. See *Transfer ordering* on page 4-41 for more information.

2.7 Source and destination signaling

The signals associated with source and destination signaling are:

TID Provides a stream identifier that can be used to differentiate between multiple streams of data that are being transferred across the same interface.

TDEST Provides coarse routing information for the data stream.

Transfers that have the same **TID** and **TDEST** values are from the same stream. Merging of transfers belonging to different streams is not permitted.

Interleaving of transfers in different steams is permitted on a per transfer basis and is not limited to **TLAST** boundaries. See *Transfer interleaving* on page 4-40 for more information.

Interconnect components can manipulate the **TID** and **TDEST** signals:

- Additional **TID** signals can be generated by an interconnect. For example, additional **TID** signals can be used to distinguish between two streams that have converged.
- An interconnect can generate or manipulate **TDEST** to provide routing information for a stream.
- Any manipulation of **TID** or **TDEST** must ensure that two different streams remain unique.

A common usage model is for an interconnect to generate **TDEST** information for an outgoing stream based on **TID** information of the incoming stream.

2.8 Clock and Reset

This section describes the requirements of the clock and reset signals.

2.8.1 Clock

A single clock signal, **ACLK**, is used by each component. All input signals are sampled on the rising edge of **ACLK**. All output signal changes must occur after the rising edge of **ACLK**.

2.8.2 Reset

ARESETn is a single, active-LOW reset signal. The reset signal can be asserted asynchronously, but deassertion must be synchronous after the rising edge of **ACLK**.

During reset, **TVALID** must be driven LOW. All other signals can be driven to any value.

A Transmitter interface must only begin driving **TVALID** at a rising **ACLK** edge following a rising edge at which **ARESETn** is asserted HIGH. Figure 2-4 shows the first point after reset that **TVALID** can be driven HIGH at T2.

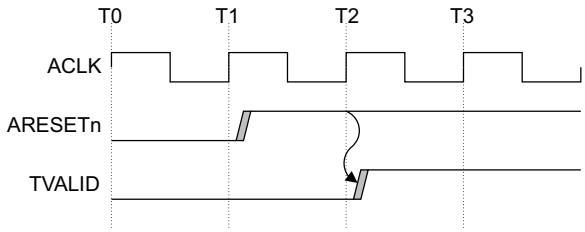


Figure 2-4 Exit from reset

2.9 User signaling

Typical uses of a streaming interface require some user-defined sideband signaling. Sideband signaling can be used for data byte-based, transfer-based, packet-based, or frame-based information.

Uses for User signaling include:

- Marking the location or type of special data items.
- Providing ancillary information that must accompany the data, such as control signals, and flags.
- Identifying segments of a packet.

The AXI-Stream interface defines that User signaling is transferred on a byte basis to ensure a consistent method of transporting User information.

It is recommended, but does not require, that the number of **TUSER** bits is an integer multiple of the width of the interface in bytes. The User signals for each byte must be packed together in adjacent bits within **TUSER**.

The location of the User signals for byte `x` are located at:

```
TUSER[((x*m)+(m-1)):(x*m)]
```

Where:

- Each data byte has `m` User signals associated with it
- The total width of the interface is `w` bytes,
- The total number of User bits is `u`, where `u = m * w`
- `x` = 0 to `w`-1

The transfer of **TUSER** bits is not required or guaranteed when the associated **TKEEP** is deasserted LOW.

User bits associated with a null byte must be removed from the data stream if the null byte is removed from the stream.

If a null byte is inserted in the data stream, the appropriate number of User bits must also be inserted. When inserting additional bits, they must be fixed LOW.

2.9.1 User signals for transfer-based information

TUSER can be used to convey information that is relevant to an entire transfer rather than to individual bytes. For example, when the same information applies to every byte in a transfer, it is more efficient to indicate the additional information once only for the entire transfer rather than replicating it for each byte within the transfer.

TUSER can be used to convey transfer based information but the transport mechanism will divide **TUSER** information between the data bytes being transported. Reliable transport of transfer-based **TUSER** information can only be guaranteed under the following constraints:

- The data bus width at the input and output of the interconnect must match.
- Any data width conversion that occurs in the interconnect must not modify the packing of the data between the input to the interconnect and the output of the interconnect.

2.9.2 User signal width matching

As a transfer passes through a complex interconnect, it might pass across sections of the interconnect that support a different **TUSER** width than is supported at either the Transmitter or the Receiver. This section describes the manipulation that is required for **TUSER** to ensure reliable transmission across such an interconnect.

Padding and trimming of TUSER information

This section describes how **TUSER** information must be padded or trimmed at an interface where the number of **TUSER** bits per byte does not match. All the examples in this section require that the data width conversion has already been performed to ensure the number of data bytes on both sides of the interface match.

Padding or trimming of **TUSER** is done by adding or removing the upper bits per byte, rather than the upper bits of the entire **TUSER** signal data. When padding, any additional bits must be fixed LOW.

Figure 2-5 shows the padding from 1 bit per byte to 2 bits per byte for an 8-byte data interface.

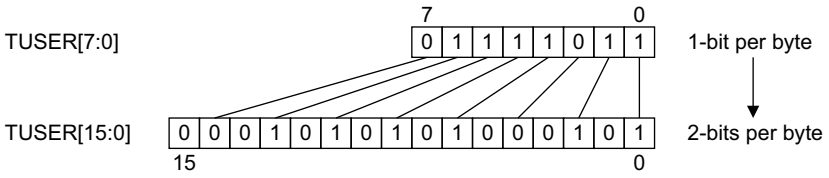


Figure 2-5 Example 1-bit to 2-bits padding for an 8 byte data interface

Figure 2-6 shows the trimming from 2 bits per byte to 1 bit per byte for an 8-byte data interface.

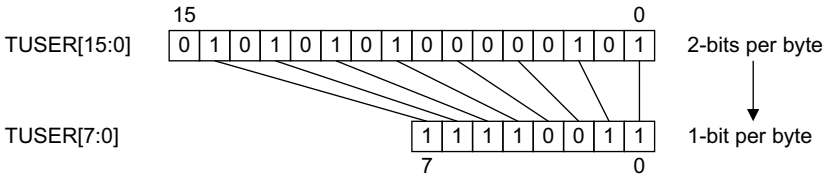


Figure 2-6 Example 2-bits to 1-bit trimming for an 8 byte data interface

Figure 2-7 shows the padding from 2 bits per byte to 4 bits per byte for a 4-byte data interface.

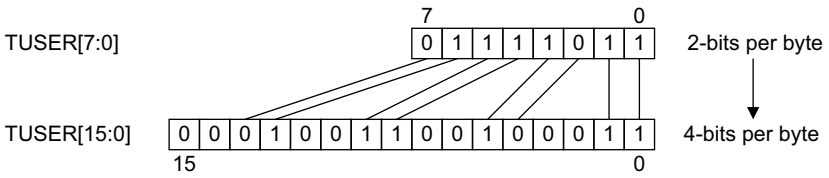


Figure 2-7 Example 2-bits to 4-bits padding for a 4 byte data interface

Figure 2-8 shows the padding from 2 bits per byte to 3 bits per byte for a 4-byte data interface.

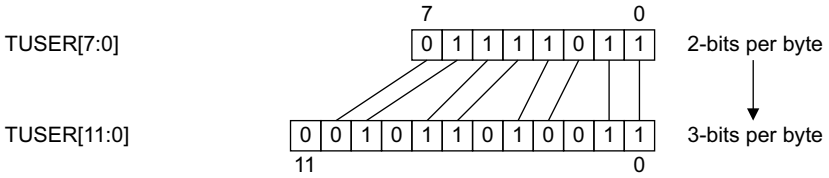


Figure 2-8 Example 2-bits to 3-bits padding for a 4 byte data interface

Determining the width of TUSER

For an interconnect with various Transmitter interfaces and Receiver interfaces, the number of **TUSER** bits per byte that must be supported by the interconnect is defined as:

```
MIN(MAX[TUSER bits per byte of Transmitters], MAX[TUSER bits per byte of Receivers])
```

In practice:

1. Find which of the Transmitters have the largest number of **TUSER** bits per byte of the **TDATA**.
2. Find which of the Receivers have the largest number of **TUSER** bits per byte of the **TDATA**.
3. Use the smaller of these two values to define the number of **TUSER** bits per byte of the **TDATA** that must be supported by the interconnect.

If it is the Transmitters that have the smaller maximum, then one or more Receivers can have a wider **TUSER** per byte of **TDATA** than can be generated by any Transmitter.

If it is the Receivers that have the smaller maximum, then one or more Transmitters can have a wider **TUSER** per byte of **TDATA** than can be accepted by any Receiver.

The guidelines for the adaptation of **TUSER** are:

- Transmitters that have a narrower **TUSER** than the interconnect must zero-pad **TUSER** to match the interconnect port.
- Transmitters that have a wider **TUSER** than the interconnect must trim **TUSER** to match the interconnect port. This must only apply to Transmitters that have a **TUSER** wider than the widest Receiver.
- **TUSER** must be zero-padded if at any point in the interconnect **TUSER** is narrower than the downstream **TUSER**.
- **TUSER** must be trimmed if at any point in the interconnect, **TUSER** is wider than the downstream **TUSER**. This is only permitted to apply when all possible downstream Receivers have a narrower or equal **TUSER**.
- **TUSER** must be trimmed if the Receiver **TUSER** is narrower than the **TUSER** provided by the interconnect upon reaching a Receiver.
- **TUSER** must be zero-padded if the Receiver **TUSER** is wider than the **TUSER** provided by the interconnect upon reaching a Receiver. This must only apply to Receivers that have a wider **TUSER** than the widest Transmitter.

During interconnect construction, information regarding which Transmitters communicate with which Receivers can be used to optimize necessary **TUSER** width support.

Chapter 3
Default Signaling Requirements

This chapter describes the interface signaling requirements of the AXI-Stream interface. It contains the following sections:

3.1 Default value signaling

This section describes the default signaling values for the AXI-Stream interface.

3.1.1 Optional TREADY

TREADY can be omitted in certain circumstances. This specification recommends **TREADY** is present.

The default value for **TREADY** is HIGH.

Receiver interface considerations

TREADY output can be omitted from the Receiver interface if the Receiver is able to always accept a transfer.

It is recommended that all usage models are considered before omitting the **TREADY**. For example, a Receiver might be able to always accept transfers when clocked above a particular frequency, but might require the use of **TREADY** when clocked below a particular threshold.

Transmitter interface considerations

For a Transmitter interface, omitting the **TREADY** input indicates that the Transmitter interface is unable to accept back-pressure and assumes that **TREADY** is always HIGH.

It is recommended that a Transmitter interface includes **TREADY**, even if it is unable to accept back-pressure. A Transmitter interface that is unable to accept back-pressure can use **TREADY** to flag an error condition if **TREADY** is driven LOW during a transfer. By including **TREADY** in the interface, it enables the source of the error to be correctly identified.

3.1.2 Optional TKEEP and TSTRB

TKEEP and **TSTRB** are optional signals and are not required by all types of data stream.

When upsizing a stream, **TKEEP** can be used to insert null bytes if there are insufficient data bytes to construct a full width transfer on the wider interface.

Default value rules

The default value rules are:

- If **TKEEP** is absent, **TKEEP** defaults to all bits HIGH.
- If **TSTRB** is absent, **TSTRB** = **TKEEP**.
- If **TSTRB** and **TKEEP** are absent, **TSTRB** and **TKEEP** default to all bits HIGH.

3.1.3 Optional TLAST

The default value of **TLAST** is indeterminate for data streams that have no concept of packets or frames. In this instance, the following options are available:

- Set **TLAST** LOW. This indicates that all transfers are within the same packet. This option provides maximum opportunity for merging and upsizing but means that transfers could be delayed in a stream with intermittent bursts. A permanently LOW **TLAST** might also affect the interleaving of streams across a shared channel because the interconnect can use **TLAST** to influence the arbitration process.
- Set **TLAST** HIGH. This indicates that all transfers are individual packets. This option ensures that transfers do not get delayed in the interconnect. It also ensures that the stream does not unnecessarily block the use of a shared channel by influencing the arbitration scheme. This option prevents merging on streams from Transmitters that have this default setting and prevents efficient upsizing.
- Automatically generate a pulsed **TLAST** value. This option asserts **TLAST** after a fixed number of transfers, for example after two or sixteen transfers. This option might prove a good compromise, allowing efficient operation without excessively blocking the sharing of a channel.

The recommended approach is:

- Any component that has the concept of packet boundaries must include **TLAST**. When included on a component interface, **TLAST** must be preserved through the interconnect.
- When a component does not support **TLAST** signaling and the topology or functionality within the interconnect is unknown, then **TLAST** must default to HIGH. This ensures that transfers do not get delayed indefinitely in the interconnect by components that use **TLAST** signaling to force a buffer draining operation.
- **TLAST** can be fixed LOW when a component does not support **TLAST** signaling and it can be guaranteed that no interconnect component requires use of **TLAST** to prevent transfers being delayed in the interconnect because of interconnect construction.

3.1.4 Optional TID, TDEST, and TUSER signals

TID, **TDEST**, and **TUSER** are all optional signals on the interface:

- A Transmitter is not required to support these output signals.
- A Receiver with additional **TID**, **TDEST**, and **TUSER** inputs must have all undriven bits fixed LOW.

3.1.5 Optional TDATA

Most applications of the interface will transfer a data payload. However, it is allowable to implement an interface that does not have a **TDATA** data payload.

If **TDATA** is not present, it is required that **TSTRB** is not present.

In the absence of **TDATA**, if **TKEEP** is present then the bit width of **TKEEP** is used to determine the correct manipulation for all upsizing and downsizing operations.

In the absence of **TDATA** and **TKEEP**, all upsizing and downsizing operations must operate in the same manner as they would for a single data byte.

3.2 Compatibility considerations

This section describes the interface compatibility considerations for the interface components.

Interface compatibility does not provide a guarantee that two components will function together. Higher-level considerations, such as the format of shared data structures, also must be considered.

Because the AXI-Stream interface has several optional features, it is possible for a Transmitter and a Receiver component to implement different sets of features.

Full compatibility for any individual component is ensured by supporting all input signals. Output signals can be optionally supported and using the default values described in *Default value signaling* on page 3-34 ensures compatible operation.

There are two aspects to consider regarding compatibility:

- Direct connection compatibility. This considers the direct connection of a Transmitter component to a Receiver.
- Interconnect compatibility. This considers the effect that an interconnect implementation may have on the compatibility of two components.

3.2.1 Transmitter compatibility

The data width of the interfaces must be the same for a Transmitter and Receiver interface to be compatible. If the data widths are not the same, then an interconnect component providing data width conversion is required to match the data widths.

The unidirectional nature of the AXI-Stream interface means that any Transmitter component that supports **TREADY** can be made interface compatible with any Receiver component that supports the full feature set. This is because any output signal not provided by a Transmitter component can be given a fixed default value. See *Default value signaling* on page 3-34.

3.2.2 Receiver compatibility

Compatibility issues primarily derive from the ability of a Receiver component to support the output signals generated by any Transmitter that is connected to it.

Data width

The first requirement for compatibility is that the data width of the two interfaces must be the same. If the data widths are not the same, an interconnect component providing data width conversion is required to match the data widths of the interfaces.

Source and destination signaling

If a Receiver component is required to differentiate between multiple different streams, it must support sufficient source and destination signaling using the **TID** and **TDEST** inputs respectively. Typically, a Receiver component is able to deal with any level of stream interleaving. If a Receiver component is required to differentiate between multiple streams, then it must include appropriate **TID** and **TDEST** inputs. In this case, the Receiver will have an upper-limit of interleaving that must not be exceeded. See *Transfer interleaving* on page 4-40.

Null and position bytes

Receivers are not required to support null or position bytes. The following is the recommended approach for Receivers that do not support either:

- If a Receiver does not support position bytes, any position bytes must be converted to data bytes. This approach does not allow the partial update of a data structure, and it might cause corruption of the data bytes that are converted. However, it does ensure that all data bytes remain correctly located within the stream.
- If a Receiver does not support null bytes, then a component that performs packing is used to remove null bytes from the stream.

3.2.3 Interconnect compatibility

An interconnect is required to pass a data stream from a Transmitter to a Receiver. The interconnect is required to reliably transport all data bytes and position bytes from the Transmitter to the Receiver. Arbitrary upsizing and downsizing operations might introduce null bytes.

There are two techniques that can be used to ensure that null bytes are not presented to a Receiver that does not support them:

- The interconnect topology can be constrained to ensure that null bytes are only introduced in quantities equal to the data width of the destination Receiver. This ensures that entire transfers can be suppressed.
- A component that performs packing can be used to remove null bytes from the stream.

3.3 Continuous packets

Some applications use a subset of the AXI-Stream interface as a means of transporting messages in a continuous manner. This means that each packet can only contain data bytes and never interleave with other packets. Transmitters, Receivers, and interconnects can be simpler and more efficient when designed to this subset.

The property, Continuous_Packets, is used to describe an interface that only supports continuous packets:

True Only continuous packets are supported.

False Packets are not constrained to be continuous. If Continuous_Packets is not declared, it is considered to be False.

Continuous_Packets can be true for AXI4-Stream and AXI5-Stream interfaces.

When Continuous_Packets is True, the following rules apply:

- Transfer interleaving for different packets must not occur. This means that when **TLAST** is LOW, **TID** and **TDEST** must not change in the next transfer.
- **TSTRB** is not present. Position bytes are not supported.
- There must not be null bytes within a packet. This means:
  - If **TLAST** is LOW, all bits of **TKEEP** must be HIGH.
  - If **TLAST** is HIGH, null bytes are only permitted in the bytes above all the data bytes.

An interconnect supporting continuous packets must only arbitrate when **TLAST** is HIGH. This means **TLAST** cannot be tied LOW and there might be low utilization if there are significant gaps between transfers within a packet.

Chapter 4
Transfer Interleaving and Ordering

This chapter describes the transfer interleaving and ordering processes allowed by the AXI-Stream protocol. It contains the following sections:

4.1 Transfer interleaving

Transfer interleaving is the process of interleaving transfers from different streams on a transfer-by-transfer basis.

A Transmitter can interleave streams at source and an interconnect can interleave streams from multiple Transmitters at a point of convergence.

Receivers can be designed to accept any number of interleaved streams or a limited number of interleaved streams.

A Receiver, with the Continuous_Packets property set to True, cannot accept interleaved streams.

If a Receiver has a limited interleaving capability, one of the following techniques must be used to ensure a Receiver capabilities are not exceeded:

- The Receiver can be accessed by just one Transmitter, which must not exceed the interleaving capability of the Receiver.
- The Receiver is accessed by multiple Transmitters, each of which does not interleave packets. The system design, or some higher-level control mechansim, ensures that the number of Transmitters accessing the Receiver at one time does not exceed the interleaving capabilities of the Receiver.
- The Receiver is accessed by multiple Transmitters and a higher-level control mechanism ensures that the overall interleaving capability of the Receiver is not exceeded.

In some systems, it may be beneficial to limit the number of interleaved streams to improve the efficiency of a Receiver or interconnect function, such as upsizing.

4.2 Transfer ordering

The AXI-Stream protocol requires that all transfers remain ordered. The reordering of transfers is not permitted. The advantages of not permitting reordering are:

- The stream interleaving observed by a Receiver is not increased by reordering.
- The overall predictability of the system is improved.
- A given transfer can be determined, independent of the **TID** of the transfers, to reach a destination by observing that a later transfer from the same Transmitter has reached the same destination.
- The complexity of a system is reduced.

Chapter 5
Interface parity protection

This chapter describes a parity scheme for detecting single-bit errors on the AXI-Stream interface between components. It contains the following sections:

5.1 Protection using parity

For safety-critical applications, it is necessary to detect and correct transient and functional errors on individual wires within an SoC.

An error in a system component can propagate and cause numerous errors across connected components. Error Detection and Correction (EDC) is required to operate end-to-end, covering all logic and wires from source to destination.

One way to implement end-to-end protection is to employ customized Error Detection and Correction schemes in components and implement a simple error detection scheme between components. Between these components, there is no logic and single-bit errors do not propagate to multi-bit errors. This section describes a parity scheme for detecting single-bit errors on the AXI-Stream interface between components. Multi-bit errors can be detected if they occur in different parity signal groups.

Figure 5-1 shows the locations where parity can be used in AXI-Stream.

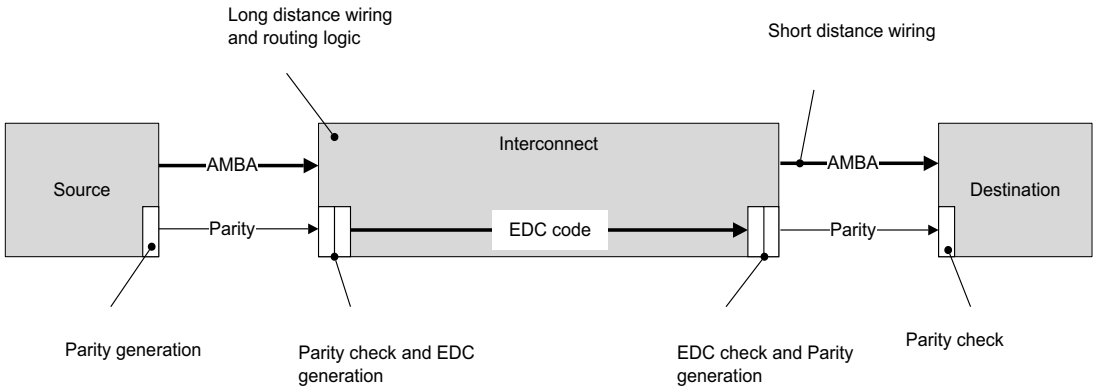


Figure 5-1 Example use of parity protection

5.2 Configuration of interface protection

The Error Detection and Correction scheme of the AXI-Stream interface is defined by the Check_Type property. The following Check_Type values are defined:

False

There are no checking signals on the AXI-Stream interface.

Odd_Parity_Byte_All

Odd parity checking is included for all signals. Each bit of the parity signal covers up to 8 bits.

If Check_Type is not declared, it is considered to be False.

Check signaling can be added to AXI5-Stream interfaces only.

5.3 Parity check

The following attributes are common to the check signals added for byte parity interface protection:

- Odd parity is used. Odd parity means that there is always an odd number of bits asserted across the interface signal and check signal.
- Each parity check bit covers no more than 8 bits of payload. This limitation assumes that there is a maximum of three logic levels available in the timing allowance for generating each parity bit.
- Parity signals that cover critical control signals are defined with a single parity bit. The single odd parity bit is the inversion of the original critical control signal. Critical control signals are likely to have a smaller timing allowance available.
- For a check signal that is wider than 1 bit:
  - Check bit `[n]` corresponds to `[(8n+7):8n]` in the associated signal.
  - If the associated signal is not an integer number of bytes, the most significant bit of the check signal covers fewer than 8-bits in the most significant portion of the associated signal.
- Check signals must be driven correctly in every cycle that the Check Enable term is True. See *Parity check signals* on page 5-48.
- Parity signals must be driven appropriately to all the bits in the associated signal, whether or not the bits are actively used in the transfer. For example, all bits of **TDATACHK** must be driven correctly, even if some bytes are not data bytes.
- If the signal covered by a check signal is not present on an interface, then the check signal is omitted from the interface.

5.4 Error detection behavior

This specification is not prescriptive regarding component or system behavior when a parity error is detected. Depending on the system and affected signals, a flipped bit can have a wide range of effects. It might be harmless, cause performance issues, cause data corruption, cause security violations, or deadlock.

When an error is detected, the Receiver can:

- Terminate or propagate the transfer.
- Correct the parity check signal or propagate the error.

5.5 Parity check signals

Table 5-1 shows the check signals. The check signals are synchronous to **ACLK** and must be driven correctly in every cycle in which the signal in Check Enable column in Table 5-1 is True.

Table 5-1 Parity check signals

Check Signal	Signals Covered	Width	Granularity	Check Enable
TVALIDCHK	**TVALID**	1	1	**ARESETn**
TREADYCHK	**TREADY**	1	1	**ARESETn**
TDATACHK	**TDATA**	TDATA_WIDTH/8	8	**TVALID**
TSTRBCHK	**TSTRB**	`ceil(TDATA_WIDTH/64)`	1-8	**TVALID**
TKEEPCHK	**TKEEP**	`ceil(TDATA_WIDTH/64)`	1-8	**TVALID**
TLASTCHK	**TLAST**	1	1	**TVALID**
TIDCHK	**TID**	`ceil(TID_WIDTH/8)`	1-8	**TVALID**
TDESTCHK	**TDEST**	`ceil(TDEST_WIDTH/8)`	1-8	**TVALID**
TUSERCHK	**TUSER**	`ceil(TUSER_WIDTH/8)`	1-8	**TVALID**
TWAKEUPCHK	**TWAKEUP**	1	1	**ARESETn**

Appendix A
Comparison with the AXI Write Data Channel

This appendix lists the key differences between the AXI-Stream interface and the AXI write data channel. It contains the following sections:

A.1 Differences to the AXI write data channel

The AXI-Stream interface has many similarities to an AXI write data channel. However, there are some key differences. These differences are summarized as:

- The AXI write data channel does not permit interleaving.
- The AXI-Stream interface does not have a defined or maximum burst or packet length.
- The AXI-Stream interface allows the data width to be any integer number of data bytes.
- The AXI-Stream interface includes **TID** and **TDEST** signals to indicate the source and destination respectively.
- The AXI-Stream interface defines more precisely the manipulation of the **TUSER** sideband signals.
- The AXI-Stream interface includes **TKEEP** signals to allow the insertion and removal of null bytes.

Appendix B
Interface Signals

This appendix describes a summary of the interface signals used in AXI4-Stream and AXI5-Stream interfaces.

B.1 AXI-Stream signals

Table B-1shows the interface signals. Table B-2 on page B-52 shows the check signals.

The following codes are used across both tables:

Y Mandatory
N Must not be present
O Optional for inputs and outputs
C Conditional, must be present if the property is True or non-zero
OC Optional conditional, optional but can only be present if the property is True or non-zero.

Table B-1 Interface signals

Signal	Width	Default	Property	AXI5-Stream	AXI4-Stream
ACLK	1	-	-	Y	Y
ARESETn	1	-	-	Y	Y
TVALID	1	-	-	Y	Y
TREADY	1	1	-	O	O
TDATA	TDATA_WIDTH	LOW	TDATA_WIDTH	C	O
TSTRB	TDATA_WIDTH/8	**TKEEP** or HIGH	-	O	O
TKEEP	TDATA_WIDTH/8	HIGH	-	O	O
TLAST	1	1	-	O	O
TID	TID_WIDTH	LOW	TID_WIDTH	C	O
TDEST	TDEST_WIDTH	LOW	TDEST_WIDTH	C	O
TUSER	TUSER_WIDTH	LOW	TUSER_WIDTH	C	O
TWAKEUP	1	-	Wakeup_Signal	C	N

Table B-2 Check signals

Signal	Width	Property	AXI5-Stream	AXI4-Stream
TVALIDCHK	1	Check_Type	C	N
TREADYCHK	1	Check_Type	C	N
TDATACHK	TDATA_WIDTH/8	Check_Type & TDATA_WIDTH	C	N
TSTRBCHK	`ceil(TDATA_WIDTH/64)`	Check_Type	OC	N
TKEEPCHK	`ceil(TDATA_WIDTH/64)`	Check_Type	OC	N
TLASTCHK	1	Check_Type	OC	N
TIDCHK	`ceil(TID_WIDTH/8)`	Check_Type & TID_WIDTH	C	N

Table B-2 Check signals (continued)

Signal	Width	Property	AXI5-Stream	AXI4-Stream
TDESTCHK	`ceil(TDEST_WIDTH/8)`	Check_Type & TDEST_WIDTH	C	N
TUSERCHK	`ceil(TUSER_WIDTH/8)`	Check_Type & TUSER_WIDTH	C	N
TWAKEUPCHK	1	Check_Type & Wakeup_Signal	C	N

Table B-3 shows the list of interface properties.

If an entry is not Y, the property must be False or undeclared for that interface type.

Table B-3 Interface properties

Property	Issue introduced	AXI5-Stream	AXI4-Stream
Check_Type	B	Y	-
Continuous_Packets	B	Y	Y
TADDR_WIDTH	B	Y	Y
TDEST_WIDTH	B	Y	Y
TID_WIDTH	B	Y	Y
TUSER_WIDTH	B	Y	Y
Wakeup_Signal	B	Y	-

Appendix C
Revisions

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

Table C-1 Issue A

Change	Location	Affects
First release	-	-

Table C-2 Differences between Issue A and Issue B

Change	Location
Added support for interface parity	Chapter 5 *Interface parity protection*
Added wake-up signaling	*Wake-up signaling* on page 2-20
Included regularized terminology to use *Transmitter* to indicate the agent that initiates transactions and *Receiver* to indicate the agent that receives and responds to requests.	Throughout the specification