AMBA® 4 AXI4-Stream Protocol

Version: 1.0

Specification

ARM®

ARM IHI 0051A (ID030510)

AMBA 4 AXI4-Stream Protocol
Specification

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
03 March 2010	A	Non-Confidential	First release

Proprietary Notice

ARM 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 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 WARRANTIES EXPRESS, IMPLIED OR STATUTORY, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF SATISFACTORY QUALITY, MERCHANTABILITY, NONINFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE.

5. 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.

6. 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.

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

ARM contract references: LEC-PRE-00490-V4.0 ARM AMBA Specification Licence

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.

Unrestricted Access is an ARM internal classification.

Product Status

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

Web Address

`http://www.arm.com`

Contents
AMBA 4 AXI4-Stream Protocol Specification

Appendix B Revisions

List of Tables
AMBA 4 AXI4-Stream Protocol Specification

List of Figures
AMBA 4 AXI4-Stream Protocol Specification

Preface

This preface introduces the *AMBA 4 AXI4-Stream Protocol Specification*. It contains the following sections:

About this book

This book is for AMBA 4 AXI4-Stream Protocol Specification.

Intended audience

This book 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 4 AXI4-Stream protocol.

Using this book

This book is organized into the following chapters:

Chapter 1 ***Introduction***

Read this for an introduction to the AXI4-Stream protocol and some examples of stream types.

Chapter 2 ***Interface Signals***

Read this for a description of the AXI4-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.

Appendix A ***Comparison with the AXI4 Write Data Channel***

Read this for a description of the key differences between the AXI4-Stream interface and the AXI4 write data channel.

Appendix B ***Revisions***

Read this for a description of the technical changes between released issues of this book.

Conventions

Conventions that this book can use are described in:

- *Typographical* on page x
- *Timing diagrams* on page x
- *Signals* on page xi.

Typographical

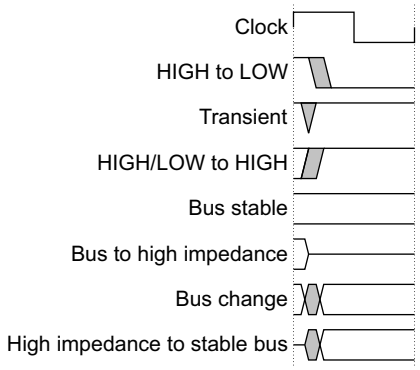
The typographical conventions are:

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.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`mono`space	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 >	Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`

Timing diagrams

The figure named *Key to timing diagram conventions* explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must 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.

Lower-case n At 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 Infocenter, http://infocenter.arm.com, 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 Version 2.0* (ARM IHI 0022)

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

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

- the title, AMBA 4 AXI4-Stream Protocol Specification
- the number, ARM IHI 0051A
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter describes the AXI4-Stream protocol and gives some examples of stream types. It contains the following sections:

1.1 About the AXI4-Stream protocol

The AXI4-Stream protocol is used as a standard interface to connect components that wish to exchange data. The interface can be used to connect a single master, that generates data, to a single slave, that receives data. The protocol can also be used when connecting larger numbers of master and slave components. The protocol supports multiple data streams using the same set of shared wires, allowing a generic interconnect to be constructed that can perform upsizing, downsizing and routing operations.

The AXI4-Stream interface also supports a wide variety of different stream types. The stream protocol defines the association between Transfers and Packets.

1.1.1 Byte definitions

The following byte definitions are used in this specification:

Data byte A byte of data that contains valid information that is transmitted between the source and destination.

Position byte A byte that indicates the relative positions of data bytes within the stream. This 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 AXI4-Stream interface. A single transfer is defined by a single **TVALID**, **TREADY** handshake. See *Handshake process* on page 2-3.

Packet A group of bytes that are transported together across an AXI4-Stream interface. A packet is similar to an AXI4 burst. A packet may consist of a single transfer or multiple transfers. Infrastructure components can use packets to deal more efficiently with a stream in packet-sized groups.

Frame The highest level of byte grouping in an AXI4-Stream. A frame contains an integer number of packets. A frame can be a very large number of bytes, for example an entire video frame buffer.

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

Data Streams take many forms. This section provides some examples of different data stream styles that might use the defined AXI4-Stream byte types.

1.2.1 Byte stream

A byte stream is the transmission of a number of data and null bytes. On each **TVALID**, **TREADY** handshake, any number of data bytes can be transferred. Null bytes have no meaning and can be inserted or removed from the stream. Figure 1-1 shows two examples of a byte stream. In the diagram, each vertical column represents the bytes in a single transfer and for this example a 4-byte wide data bus is used. Columns are time-ordered left to right

Note

Since a null byte conveys no information and nor does its presence in a stream, the two examples given in Figure 1-1 transfer identical information.

Null	Null	D-07	D-0A	Null	D-0F
D-01	Null	D-06	D-09	Null	D-0E
Null	D-03	D-05	D-08	D-0C	Null
D-00	D-02	D-04	Null	D-0B	D-0D

D-02	D-06	Null	D-0B	Null	Null
Null	D-05	Null	D-0A	D-0E	D-0F
D-01	D-04	Null	D-09	D-0D	Null
D-00	D-03	D-07	D-08	D-0C	Null

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

D-03	D-07	D-0B	D-0F	D-13
D-02	D-06	D-0A	D-0E	D-12
D-01	D-05	D-09	D-0D	D-11
D-00	D-04	D-08	D-0C	D-10

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 where there are no position bytes between the first data byte and the last data byte of each packet. Figure 1-3 shows two examples of a continuous unaligned stream.

Note

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.

D-03	D-07	D-0B	D-0F	Position
D-02	D-06	D-0A	D-0E	Position
D-01	D-05	D-09	D-0D	D-11
D-00	D-04	D-08	D-0C	D-10

D-00	D-04	D-08	D-0C	D-10	Position
Position	D-03	D-07	D-0B	D-0F	Position
Position	D-02	D-06	D-0A	D-0E	Position
Position	D-01	D-05	D-09	D-0D	D-11

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 and position bytes must be maintained and transmitted from source to destination. Figure 1-4 shows an example of a sparse stream.

Note

A sparse stream can contain any mix of data bytes and position bytes, but typically the majority of the bytes are data bytes. A sparse stream does not imply that only a minority of the bytes are data bytes.

D-03	D-07	Position	D-0F	D-13
Position	D-06	D-0A	Position	D-12
D-01	Position	D-09	D-0D	Position
D-00	D-04	Position	D-0C	D-10

Figure 1-4 Example of a sparse stream

Chapter 2
Interface Signals

This chapter describes the signal requirements of the AXI4-Stream interface. It contains the following sections:

2.1 Signal list

The interface signals are listed in Table 2-1. For additional information on these signals see further sections of this chapter.

Table 2-1 uses the following parameters to define the signal widths:

n Data bus width in bytes.

i **TID** width. Recommended maximum is 8-bits.

d **TDEST** width. Recommended maximum is 4-bits.

u **TUSER** width. Recommended number of bits is an integer multiple of the width of the interface in bytes.

Table 2-1 Interface signals list

Signal	Source	Description
ACLK	Clock source	The global clock signal. All signals are sampled on the rising edge of **ACLK**.
ARESETn	Reset source	The global reset signal. **ARESETn** is active-LOW.
TVALID	Master	**TVALID** indicates that the master is driving a valid transfer. A transfer takes place when both **TVALID** and **TREADY** are asserted.
TREADY	Slave	**TREADY** indicates that the slave can accept a transfer in the current cycle.
TDATA[(8n-1):0]	Master	**TDATA** is the primary payload that is used to provide the data that is passing across the interface. The width of the data payload is an integer number of bytes.
TSTRB[(n-1):0]	Master	**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.
TKEEP[(n-1):0]	Master	**TKEEP** is the byte qualifier that indicates whether the content of the associated byte of **TDATA** is processed as part of the data stream. Associated bytes that have the **TKEEP** byte qualifier deasserted are null bytes and can be removed from the data stream.
TLAST	Master	**TLAST** indicates the boundary of a packet.
TID[(i-1):0]	Master	**TID** is the data stream identifier that indicates different streams of data.
TDEST[(d-1):0]	Master	**TDEST** provides routing information for the data stream.
TUSER[(u-1):0]	Master	**TUSER** is user defined sideband information that can be transmitted alongside the data stream.

2.2 Transfer signaling

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

2.2.1 Handshake process

The **TVALID** and **TREADY** handshake determines when information is passed across the interface. A two-way flow control mechanism enables both the master and slave to control the rate at which the data and control information is transmitted across the interface. For a transfer to occur both the **TVALID** and **TREADY** signals must be asserted. Either **TVALID** or **TREADY** can be asserted first or both can be asserted in the same **ACLK** cycle.

A master 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 slave is permitted to wait for **TVALID** to be asserted before asserting the corresponding **TREADY**.

If a slave asserts **TREADY**, it is permitted to deassert **TREADY** before **TVALID** is asserted.

The following sections give examples of the handshake sequence.

TVALID before TREADY handshake

In Figure 2-1 the master presents the data and control information and asserts the **TVALID** signal HIGH. Once the master has asserted **TVALID**, the data or control information from the master must remain unchanged until the slave drives the **TREADY** signal HIGH, indicating that it can accept the data and control information. In this case, transfer takes place once the slave asserts **TREADY** HIGH. The arrow shows when the transfer occurs.

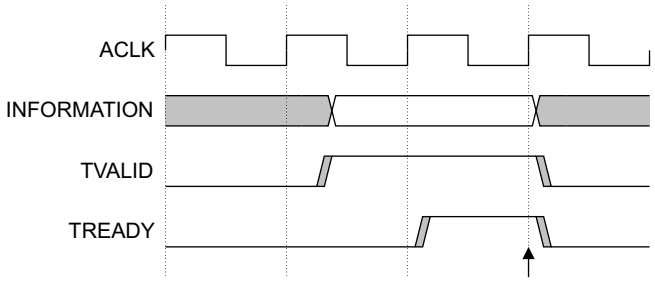


Figure 2-1 TVALID before TREADY handshake

TREADY before TVALID handshake

In Figure 2-2 the slave drives **TREADY** HIGH before the data and control information is valid. This indicates that the destination can accept the data and control information in a single cycle of **ACLK**. In this case, transfer takes place once the master asserts **TVALID** HIGH. The arrow shows when the transfer occurs.

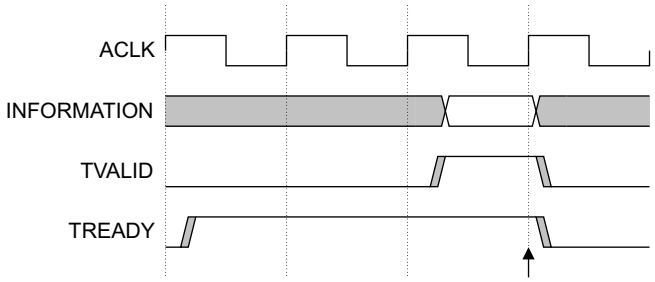


Figure 2-2 TREADY before TVALID handshake

TVALID with TREADY handshake

In Figure 2-3 the master asserts **TVALID** HIGH and the slave asserts **TREADY** HIGH in the same cycle of **ACLK**. In this case, transfer takes place in the same cycle as shown by the arrow.

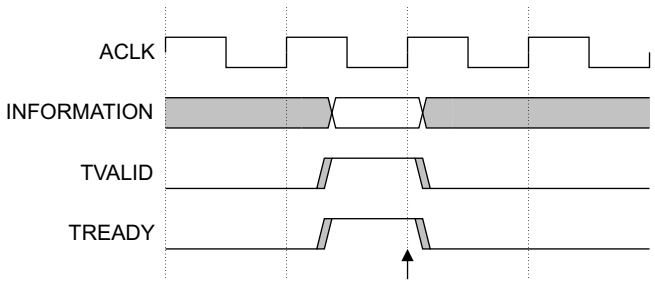


Figure 2-3 TVALID with TREADY handshake

2.3 Data signaling

This section describes the data signaling requirements of the AXI4-Stream interface.

TDATA is the primary payload of the AXI4-Stream interface and is used to transport data from a source to a destination.

2.3.1 Byte locations

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

therefore 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]
```

2.3.2 Byte types

The AXI4-Stream protocol defines three byte types:

Data byte A data byte must be transmitted from source to destination.

The number of data bytes, the associated **TDATA** data values, and the relative position of the data byte in the stream (with respect to other data bytes and position bytes), must be maintained between source and destination.

Position byte A position byte must be transmitted from source to destination.

The number of position bytes and the relative position of the position byte in the stream (with respect to other data bytes and position bytes), must be maintained between source and destination.

The **TDATA** data value associated with a position byte is not required to be transmitted between source and destination.

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. The insertion of null bytes may be required for certain operations, such as upsizing to a wider data bus, where there are insufficient data and position bytes to make a complete data width transfer.

Master and slave components are not required to support null bytes, therefore any interconnect that is capable 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.

2.3.3 Data merging, packing, and width conversion

It is desirable to be able to adapt the data width of a streaming interface. This allows a component to be built with a fixed width of interface, and that component to then be reused in system designs that use a different width of interface. An adaptable interface width also enables components with different native widths to be combined and to share a common link.

It is expected that in the majority of applications, the interface width is a power-of-two bytes wide. However, this is not a requirement of the protocol, so the interface can be an integer number of bytes wide.

All upsizing and downsizing operations are required to preserve any bytes which have the associated **TKEEP** data qualifier asserted. If an upsizing or downsizing operation does not have sufficient bytes of data to compile a full width output, additional bytes can be added with the associated **TKEEP** signal deasserted.

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 take place 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 four byte transfer only has three data bytes, it is allowable to merge a single data byte from the following transfer 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, to provide 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 converting 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 output stream must match the order of the bytes in the input stream
- **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.

Note

A transfer that contains only null bytes and has **TKEEP** deasserted, and does not have **TLAST** asserted, can be suppressed.

Upsizing considerations

Upsizing is converting 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 because, on receiving a transfer, an upsizer will not always be able to determine if the following transfer is in the same packet.

The signaling rules associated with upsizing are:

- the order of the bytes in the output stream must match the order of the bytes in the input stream
- **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 all output transactions must match the value of the input transactions
- **TUSER** information must remain associated with the same byte.

Note

If there are insufficient transfers to construct a full width upsized stream, **TKEEP** signals are required to indicate the null bytes.

2.4 Byte qualifiers

This section defines the two byte qualifiers supported by the AXI4-Stream protocol:

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

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

Each bit of **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]**

2.4.1 TKEEP qualification

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

When **TKEEP** is deasserted LOW, it indicates a null byte that can be removed from the stream.

It is legal to have a transfer that has all **TKEEP** bits deasserted LOW.

It is permissible for a transfer that has all **TKEEP** bits deasserted LOW to be suppressed unless it has **TLAST** asserted HIGH. See *Packet boundaries* on page 2-9 for the usage model.

It is not mandatory for masters and slaves to handle null bytes, therefore any interconnect that is capable 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.

2.4.2 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 HIGH it indicates that the associated byte contains valid information, and is a data byte. When **TSTRB** is deasserted LOW 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 LOW.

2.4.3 TKEEP and TSTRB combinations

Table 2-2 lists the valid combinations for **TKEEP** and **TSTRB** byte qualifications.

Table 2-2 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.

Not all components require the **TKEEP** and **TSTRB** data qualifiers. See *Optional TKEEP and TSTRB* on page 3-2.

2.5 Packet boundaries

A packet is a grouping of bytes that are transmitted together across the interface. Infrastructure components can typically be made more efficient by dealing with transfers that are grouped together in packets. An AXI4-Stream packet is similar to an AXI4 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 2-10.

The uses of **TLAST** are:

- when deasserted, **TLAST** indicates that another transfer can follow and therefore 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, but **TLAST** signaling 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 master and slave.

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 be merged if the earlier transfer has **TLAST** asserted. See *Merging considerations* on page 2-6.

The merging of transfers with different **TID** or **TDEST** values is never permitted.

2.5.1 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 with matching **TID** and **TDEST** values that does not also have **TLAST** asserted. See *Merging considerations* on page 2-6.

Note

Because reordering is not supported, sending a transfer with zero data bytes will effectively push through all transfers between a master and slave. See *Transfer ordering* on page 4-3.

2.6 Source and destination signaling

The signals associated with source and destination signaling are:

TID Provides a stream identifier and 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 4-2.

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

- Additional **TID** signals can be generated by an interconnect. This allows two otherwise identical streams to be distinguished.
- An interconnect can generate or manipulate the **TDEST** signals to provide routing information for a stream.

Any manipulation of **TID** or **TDEST** must ensure that two different streams remain unique.

Note

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

2.7 Clock and Reset

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

2.7.1 Clock

Each component uses a single clock signal, **ACLK**. All input signals are sampled on the rising edge of **ACLK**. All output signal changes must occur after the rising edge of **ACLK**.

2.7.2 Reset

The protocol includes a single active-LOW reset signal, **ARESETn**. 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 master 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.

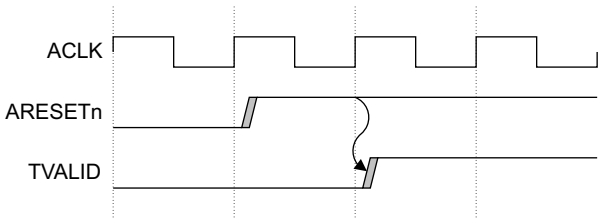


Figure 2-4 Exit from reset

2.8 User signaling

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

There are several uses of User signaling. For example:

- marking the location or type of special data items
- providing ancillary information that must accompany the data, such as parity, control signals, and flags
- identifying segments of a packet.

To ensure a consistent method of transporting User information the protocol defines that User signaling is transferred on a byte basis.

It is recommended, but not required, 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 bits is defined as:

- each data byte has `m` User signals associated with it
- the total width of the interface is `n` bytes,
- the total number of User bits is `u`, where `u = m * n`

The user signals for byte `x`, where `x = 0 ... (n-1)`, are located at:

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

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

Note

User bits associated with a null byte, as indicated by the associated **TKEEP** bit, 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.

2.8.1 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. An example of this is where the same information applies to every byte in a transfer and 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 the **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 to the interconnect must match the data bus width at the output of the interconnect
- 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

2.8.2 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 master or the slave. 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** data. When padding, any additional bits must be fixed LOW.

Figure 2-5 shows the padding from one bit per byte to two bits per byte for an eight byte data interface.

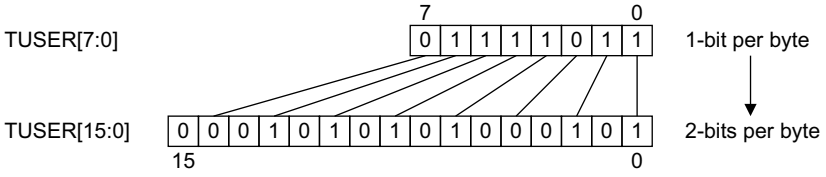


Figure 2-5 Example 1-bit to 2-bits padding for an eight byte data interface

Figure 2-6 shows the trimming from two bits per byte to one bit per byte for an eight byte data interface.

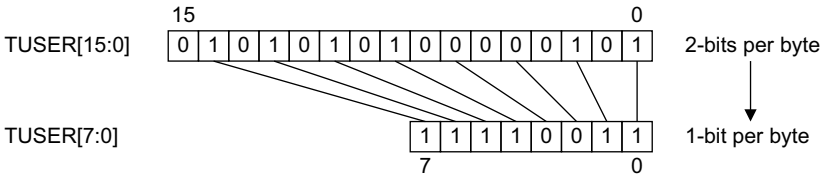


Figure 2-6 Example 2-bits to 1-bit trimming for an eight byte data interface

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

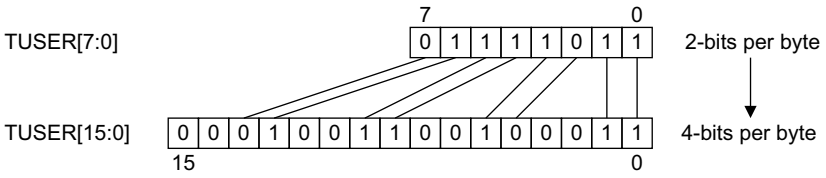


Figure 2-7 Example 2-bits to 4-bits padding for a four byte data interface

Figure 2-8 shows the padding from two bits per byte to three bits per byte for a four byte data interface.

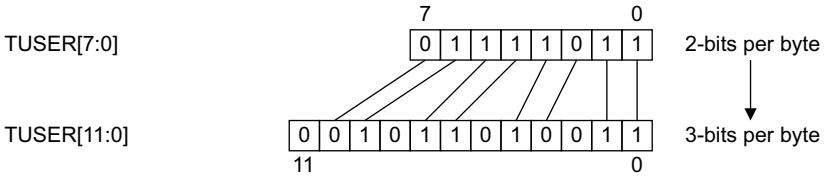


Figure 2-8 Example 2-bits to 3-bits padding for a four byte data interface

Determining the width of TUSER

For an interconnect with a number of master interfaces and slave 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 masters], MAX[TUSER bits per byte of slaves])
```

In practice:

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

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

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

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

- Masters that have a narrower **TUSER** than the interconnect must zero-pad **TUSER** to match the interconnect port.
- Masters that have a wider **TUSER** than the interconnect must trim **TUSER** to match the interconnect port. This must only apply to masters that have a **TUSER** wider than the widest slave.
- At any point in the interconnect where **TUSER** is narrower than the downstream **TUSER**, **TUSER** must be zero-padded.
- At any point in the interconnect where **TUSER** is wider than the downstream **TUSER**, **TUSER** must be trimmed. This must only apply where all possible downstream slaves have an equal or narrower **TUSER**.
- On reaching a slave, if the slave **TUSER** is narrower than that provided by the interconnect, then the **TUSER** must be trimmed.
- On reaching a slave, if the slave **TUSER** is wider than that provided by the interconnect, then the **TUSER** must be zero-padded. This must only apply to slaves that have a **TUSER** wider than the widest master.

If during construction of the interconnect additional information is known about which masters communicate with which slaves, further optimizations on the width of **TUSER** that must be supported are allowable.

Chapter 3
Default Signaling Requirements

This chapter describes the interface signaling requirements of the AXI4-Stream interface. It contains the following sections:

3.1 Default value signaling

This section describes the default signaling values for the streaming interface.

3.1.1 Optional TREADY

The **TREADY** signal can be omitted in certain circumstances. However, it is recommended that **TREADY** is always implemented.

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

Slave interface considerations

For a slave interface, the **TREADY** output can be omitted if the slave is able to always accept a transfer.

Note

It is recommended that all usage models are considered before omitting the **TREADY** signal. For example, a slave 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.

Master interface considerations

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

Note

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

3.1.2 Optional TKEEP and TSTRB

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

Note

If a data stream is subject to upsizing, and it cannot be ensured that there are always sufficient transfers to construct a full width upsized stream, **TKEEP** is required and must indicate the null bytes.

Default value rules

The default value rules are:

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

3.1.3 Optional TLAST

For data streams that have no concept of packets or frames the default value of **TLAST** is indeterminate. 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** signal might also affect the interleaving of streams across a shared channel because the interconnect can use the **TLAST** signal 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 infrastructure. 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 masters 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 a **TLAST** signal. When included on a component interface, the **TLAST** signal 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 the **TLAST** signal 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.
- When a component does not support **TLAST** signaling and it can be guaranteed, because of interconnect construction, that no interconnect component requires use of the **TLAST** signal to prevent transfers being delayed in the interconnect, then the **TLAST** signal can be fixed LOW.

3.1.4 Optional TID, TDEST, and TUSER

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

- a master is not required to support these output signals
- a slave with additional **TID**, **TDEST**, and **TUSER** inputs must have all bits fixed LOW.

3.1.5 Optional TDATA

Most applications of the AXI4-Stream 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 discusses the interface compatibility considerations for AXI4-Stream components.

Note

Interface compatibility does not provide a guarantee that two components will function together because higher level considerations, such as the format of shared data structures, also need to be taken into consideration.

Because AXI4-Stream has a number of optional features it is possible for a master and a slave 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 3-2 ensures compatible operation.

There are two aspects to consider with regard to compatibility:

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

3.2.1 Master compatibility

For a master and slave interface to be compatible, the data width of the interfaces must be the same. If this is not the case, then an interconnect component providing data width conversion is required to match the data widths.

The uni-directional nature of the AXI4-Stream interface means that any master component that supports **TREADY** can be made interface compatible with any slave component that supports the full feature set. This is because any output signal not provided by a master component can be given a fixed default value. See *Default value signaling* on page 3-2.

3.2.2 Slave compatibility

Compatibility issues primarily derive from the ability of a slave component to support the output signals generated by any masters that are connected to it.

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

Source and destination signaling

If a slave component is required to differentiate between multiple different streams then it must support sufficient source and destination signaling using the**TID** and **TDEST** inputs respectively. Typically a slave component will either be able to deal with any level of stream interleaving. If a slave component is required to differentiate between multiple streams then it must include appropriate **TID** and **TDEST** inputs. In this case, the slave will have an upper-limit of interleaving that must not be exceeded. See *Transfer interleaving* on page 4-2.

Null and Position bytes

Slaves are not required to support both null and position bytes. The following is the recommended approach for slaves that do not support both:

- If a slave does not support position bytes, all 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 overridden. However, it does ensure that all data bytes remain correctly located within the stream.
- If a slave does not support null bytes, then a component that performs packing is used to remove null bytes from the stream.

  Note

  A generic component that performs packing might need to support multiple concurrent streams.

3.2.3 Interconnect compatibility

An interconnect is required to pass a data stream from a master to a slave. The interconnect is required to reliably transport all data bytes and position bytes from the master to the slave. 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 slave 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 slave. This ensures that entire transfers can be suppressed.
- A component that performs packing can be used to remove null bytes from the stream.

Chapter 4
Transfer Interleaving and Ordering

This chapter describes the transfer interleaving and ordering processes allowed by the AXI4-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.

The interconnect is not required to constrain the interleaving of streams so that the capabilities of a slave are not exceeded.

Note

In some interconnect topologies transfer interleaving might be limited to packet boundaries, to increase the potential for improved efficiency through transfer merging.

4.1.1 Slave considerations

Slaves can be designed to have no restriction on the number of interleaved streams that can be handled. It is possible that the component operates less efficiently when presented with interleaved streams, but functionally the slave will operate correctly.

In system designs where a slave operates more efficiently when handling an entire packet, the arbitration can be designed to operate on **TLAST** boundaries.

Note

Limiting any arbitration mechanism to only operate on **TLAST** boundaries is not compatible with masters that permanently fix **TLAST** LOW. It is recommended that any arbitration that makes use of **TLAST** signaling to influence the selection process also includes an override mechanism.

Slaves can also be designed to support limited interleaving. If this is the case, then one of the following techniques must be used to ensure slave capabilities are not exceeded:

- The slave can be accessed by just one master and all the interleaving capability is available to that master.
- The slave is accessed by multiple masters, each of which does not interleave packets. The system design, or some higher level control mechanism, ensures that the number of masters accessing the slave at one time does not exceed the interleaving capabilities of the slave.
- The slave is accessed by multiple masters and a higher level control mechanism ensures that the overall interleaving capability of the slave is not exceeded.

4.2 Transfer ordering

The AXI4-Stream protocol requires that all transfers remain ordered. It does not permit the reordering of transfers. The advantages of not permitting reordering are:

- it ensures that reordering cannot increase the stream interleaving observed by a slave
- the overall predictability of the system is improved
- it can be determined, independent of the **TID** of the transfers, that a given transfer has reached a destination by observing that a later transfer from the same master has reached the same destination
- the complexity of a system is reduced.

Appendix A
Comparison with the AXI4 Write Data Channel

This appendix lists the key differences between the AXI4-Stream interface and the AXI4 write data channel. It contains the following sections:

A.1 Differences to the AXI4 write data channel

The AXI4-Stream interface has many similarities to an AXI4 write data channel. However, there are some key differences. These differences are summarized as:

- the AXI4 write data channel does not permit interleaving
- the AXI4-Stream interface does not have a defined or maximum burst or packet length
- the AXI4-Stream interface allows the data width to be any integer number of data bytes
- the AXI4-Stream interface includes **TID** and **TDEST** signals to indicate the source and destination respectively
- the AXI4-Stream interface defines more precisely the manipulation of the **TUSER** sideband signals
- the AXI4-Stream interface includes **TKEEP** signals to allow the insertion and removal of null bytes.

Appendix B
Revisions

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

Table B-1 Issue A

Change	Location	Affects
First release	-	-