Avalon Streaming Interface

Specification

FS-0100-1.3

I.S. EN ISO 9001

This specification provides comprehensive specification for the Avalon® Streaming Interface, version 1.3.

How to Contact Altera

For the most up-to-date information about Altera products, refer to the following table.

Contact *(1)*	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Altera literature services	Email	literature@altera.com
Non-technical support (General)	Email	nacomp@altera.com
(Software Licensing)	Email	authorization@altera.com

Note to Table:
(1)   You can also contact your local Altera sales office or sales representative.

Typographic Conventions

This document uses the typographic conventions shown below.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: **Save As** dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: f_{MAX}, **\qdesigns** directory, **d:** drive, **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: *AN 75: High-Speed Board Design.*
Italic type	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA}, $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: *<file name>*, *<project name>***.pof** file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
"Subheading Title"	References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: "Typographic Conventions."

Visual Cue	Meaning
`Courier type`	Signal and port names are shown in lowercase Courier type. Examples: `data1`, `tdi`, `input`. Active-low signals are denoted by suffix n, e.g., `resetn`. Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\qdesigns\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the VHDL keyword `BEGIN`), as well as logic function names (e.g., `TRI`) are shown in Courier.
1., 2., 3., and a., b., c., etc.	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ● •	Bullets are used in a list of items when the sequence of the items is not important.
✓	The checkmark indicates a procedure that consists of one step only.
☞	The hand points to information that requires special attention.
⚠ CAUTION	The caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.
⚠ WARNING	The warning indicates information that should be read prior to starting or continuing the procedure or processes
↵	The angled arrow indicates you should press the Enter key.
👣	The feet direct you to more information on a particular topic.

Altera Corporation

Table of Contents

1. Introduction

The Avalon® Streaming interface protocol is designed to accommodate the development of high bandwidth low latency components for the system-on-a-programmable-chip (SOPC) environment. The *Avalon Streaming (Avalon-ST) Interface Specification* provides component designers with a framework to create interfaces that support the unidirectional flow of data, including multiplexed streams, packets, and DSP data.

The purpose of the Avalon-ST interface is to define a standard, flexible, and modular protocol for data transfers from a source interface to a sink interface. This specification outlines a protocol that lets component designers develop components quickly and easily while ensuring interoperability. It also allows designers to connect components together using SOPC builder or a custom HDL.

The Avalon-ST interface signals can describe traditional streaming interfaces supporting a single stream of data without knowledge of channels or packet boundaries. The interface can also support more complex protocols capable of burst and packet transfers with packets interleaved across multiple channels.

All Avalon-ST source and sink interfaces are not necessarily interoperable. However, if two interfaces provide compatible functions for the same application space, adapter logic is available to allow them to interoperate.

1.1. Features

Some of the prominent features of the Avalon-ST interface are:

- Low latency, high throughput data transfer
- Multiple channel support with flexible packet interleaving
- Sideband signaling of channel, error, and start and en of packet delineation
- Support for data bursting
- Automatic interface adaptation

1.2. History

The Avalon Streaming interface protocol evolved from the Atlantic I interface. However, it is more flexible than the earlier protocol, allowing the interface to be defined to fit the specific requirements of the components.

The Atlantic I protocol uses the `ena`, `dat` and `dav` signals to control the flow of data from a master source to a slave sink or from a slave source to a master sink. You can convert components that used the Atlantic I interface to the Avalon Streaming by substituting the `ready/valid` signals for `ena/val` signals.

1.3. Terms and Concepts

This section defines terms and concepts used in the *Avalon Streaming Interface Specification*.

■ Avalon Streaming System – An Avalon Streaming System is a system of one or more Avalon-ST connections that transmit data from a source interface to a sink interface. The system shown in Figure 1 consists of Avalon-ST data interfaces to transfer data from the system input to output and Avalon control/status register interfaces to allow software control.

■ Avalon Streaming Components – A typical system using Avalon Streaming interfaces combines multiple functional modules, called *components*. The system designer configures the components and connects them together to implement a system.

Figure 1. - Example System on a Programmable Chip
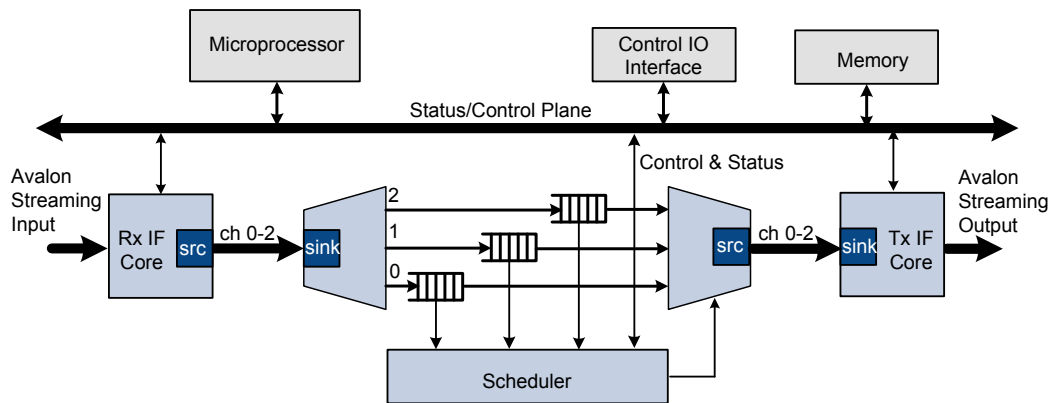
- Source and Sink Interfaces and Connections – Whenever two components are connected, the data flows from the *source interface* to the *sink interface*. The combination of a source interface connected to a sink interface is referred to as a *connection*.

- Backpressure - *Backpressure* is a mechanism by which a sink can signal to a source to stop sending data. The sink typically uses backpressure to stop the flow of data when its FIFOs are full or when there is congestion on its output port.

- Parameters – The *Avalon Streaming Interface Specification* defines *parameters* to provide a full description of its operation. Each parameter either defines an attribute of a signal, or an attribute of the entire interface.

- Signal Types – The *Avalon Streaming Interface Specification* does not dictate a naming convention for the signals on a component, allowing component designers to name each signal according to its function. Each signal's type is identified by a type parameter for each signal.

- Cycle – A *cycle* is the basic unit of one clock period, which is defined from rising-edge to rising-edge of the clock associated with the particular interface. The shortest duration of a transfer is one cycle.

■ Transfers, Ready Cycles, and Beats – A *transfer* is an operation that results in data and control being propagated from a source interface to a sink interface. For data interfaces, a *ready cycle* is a cycle during which the sink can accept a transfer. A *beat* is the transfer of one unit of data from a source to a sink.

■ Symbol – A *symbol* is the smallest atomic unit of data. For most packet interfaces, a symbol is a byte.

■ Channel –A *channel* is a physical or logical path or link through which information passes between two ports.

■ Packet – A *packet* is an aggregation of data and control signals that is transmitted and switched as a composite whole. A packet typically contains a header to help routers and other network devices direct the packet to the correct destination. The packet format is defined by the application, not this specification. Avalon-ST packets can be variable in length and can be interleaved across a connection.

■ Bursts – A *burst* guarantees that the source will send a minimum amount of data without interleaving data for other channels. Once a source begins a burst, it does not send any data to any other channel until the burst is complete. A source may insert idle cycles in the middle of a burst.

■ Blocks – A *block* executes multiple transfers as a unit, rather than treating each cycle as an independent transfer, maximizing throughput for interfaces that are more efficient when handling multiple units of data. Once a source begins a block, it does not insert any idle cycles or send data to any other channel until the block is complete.

2. Data Interface Signals

The *Avalon Streaming Interface Specification* defines the behavior for the Avalon-ST signals of several types. Each signal in an Avalon-ST source or sink interface corresponds to one Avalon-ST signal type; an Avalon-ST interface uses only one instance of each. All Avalon-ST signal types apply to both sources and sinks and have the same meaning for both.

Table 1 lists the signal types that comprise an Avalon-ST data interface.

Table 1. Data Interface Signals			
Signal Type	**Width**	**Direction**	**Description**
Fundamental Signals			
clk	1	In	Synchronization clock for the Avalon-ST interface. All signals are synchronous to `clk`.
ready	1	Sink → Source	Indicates that the sink can accept data. On interfaces supporting flow control, `ready` is asserted by the sink on cycle N to mark cycle N+READY_LATENCY as a ready cycle, during which the source may assert `valid` and transfer data.
valid	1	Source → Sink	Asserted by the source to qualify all other source to sink signals. On ready cycles where `valid` is asserted, the data bus and other source to sink signals are sampled by the sink, and on other cycles are ignored. `Valid` is required on every Avalon-ST interface.
data	1.. 256	Source → Sink	The data signal from the source to the sink, typically carries the bulk of the information being transferred. The contents and format of the data signal is further defined by parameters.
channel	0 .. 31	Source → Sink	The `channel` number for data being transferred on the current cycle. If an interface supports the channel signal, it must also define the MAX_CHANNEL parameter.
error	0 .. 255	Source → Sink	A bit mask used to mark errors affecting the data being transferred in the current cycle. A single bit in `error` is used for each of the errors recognized by the component.
Packet Transfer Signals			
startofpacket	1	Source → Sink	Asserted by the source to mark the beginning of a packet.
endofpacket	1	Source → Sink	Asserted by the source to mark the end of a packet.

Table 1. Data Interface Signals

Signal Type	Width	Direction	Description
Fundamental Signals			
empty	$\log_2$(`SYMBOLS_PER_CYCLE`)	Source → Sink	Indicates the number of symbols that are empty during cycles that contain the end of a packet. The `empty` signal is not used on interfaces where the `SYMBOLS_PER_BEAT` is 1.
Other Signals			
reset_n	1	In	Active-low component reset signal. When asserted, the entire component, or the portion of the component specific to this interface, must enter a deterministic reset state.

Table 2 lists the parameters types that comprise an Avalon-ST data interface.

Table 2. Data Interface Parameters

Parameter Name	Description
Fundamental Parameters	
BITS_PER_SYMBOL	These parameters define how the data signal is divided into symbols. The width of the data signal must be `BITS_PER_SYMBOL` × `SYMBOLS_PER_BEAT`.
SYMBOLS_PER_BEAT	
READY_LATENCY	Defines the relationship between assertion/deassertion of the `ready` signal, and cycles which are considered to be ready for data transfer separately for each interface. For more information, "Data Transfer with Backpressure" on page 1–16.
MAX_CHANNEL	The maximum number of channels that a data interface can support.
Burst and Block Parameters	
CYCLES_PER_BURST	Set 1 parameter to a non-zero value to define the size of a burst, during which the source may not change to a different channel. An interface that does not support bursts is equivalent to `CYCLES_PER_BURST=1`.
PACKETS_PER_BURST	If `PACKETS_PER_BURST` is used, the only value it can assume is 1, indicating that an entire packet is transferred before the channel changes.

Table 2. Data Interface Parameters	
Parameter Name	**Description**
Fundamental Parameters	
`CYCLES_PER` `_BLOCK`	Set 1 parameter to a non-zero value to define the size of a block, during which the source may not pause data transfer. An interface that does not support blocks is equivalent to `CYCLES_PER_BLOCK=1`.
`PACKETS_PER_` `BLOCK`	If `PACKETS_PER_BLOCK` is used, the only value it can assume is 1, indicating that an entire packet must be transferred before the source can insert an idle cycle.

2.1. Signal Polarity

All signal types listed in Table 1 are active high.

2.2. Signal Naming Conventions

The *Avalon Streaming Interface Specification* does not dictate a naming convention for the signals that appear on Avalon-ST components. A signal name chosen on an Avalon-ST interface can be the same as its signal type or it can be named differently to comply with a system-wide naming convention. For example, an Avalon-ST component may have a sink interface with an input signal named `my_enable_signal`, of type `valid`.

2.3. Signal Sequencing and Timing

This section describes issues related to timing and sequencing of Avalon-ST signals.

2.3.1. Synchronous Interface

All transfers of an Avalon-ST connection occur synchronous to the rising edge of the associated clock signal. All outputs from a source interface to a sink interface, including the data, channel, and error signals, must be registered on the rising edge of clock. The Specification does not require inputs to a sink interface to be registered. Registering signals at the source provides for high frequency operation while eliminating back-to-back registers with no intervening logic. Registers are not required for signals in the sink to source direction.

Avalon-ST interfaces must not be edge sensitive to signals other than the clock, because the signals may transition multiple times before they stabilize. The exact timing of signals toggling and stabilizing between clock edges varies depending upon the characteristics of the Altera device selected to implement the system.

2.3.2. Clock Enables

Avalon-ST components typically do not include a clock enable input, because the Avalon-ST signaling itself is sufficient to determine the cycles that a component should and should not be enabled. Avalon-ST compliant components may have a clock enable input for their internal logic, but they must take care to ensure that the timing of the interface control signals still adheres to the specification.

2.3.3. Performance

There is no fixed or maximum performance of an Avalon-ST interface. The maximum performance is dependent upon component design and system implementation. After synthesis and place-and-route of the system for a specific device, standard timing analysis determines the maximum speed at which transfers can be performed.

2.3.4. Electrical Characteristics

The *Avalon Streaming Interface Specification* defines the interface between IP blocks on a programmable logic device. It does not specify any electrical or physical characteristics required by traditional inter-chip interfaces.

2.4. Resets

Each Avalon-ST Streaming interface has an associated active-low `reset` signal. This `reset` can be for the interface alone, for the entire component, or for a number of interfaces on the component. The `reset` used by each interface may be asserted at any time, but must only be deasserted synchronously to the clock used by the interface: Avalon-ST components are not required to do any metastable hardening for `reset` signals. Avalon-ST components can reset synchronously to the clock signal when the `reset` signal is asserted, or can be reset synchronously to the falling edge of the `reset` signal itself. The Avalon-ST specification provides requirements for some control signals while the interface is in reset. The reset requirement for each signal is discussed with the details of the signal. The reset requirements are intentionally kept to a minimum to allow for flexibility in component design.

2.5. Transfer Properties

Different Avalon-ST interfaces have different transfer capabilities, because not all Avalon-ST source and sink interfaces transfer data in the same way. The *Avalon Streaming Interface Specification* defines a set of properties that transfers can exhibit. An Avalon-ST source or sink interface can support one or more of these properties, depending on the component design. The transfer properties supported by an Avalon-ST component are determined at design time and do not change from transfer to transfer.

The *Avalon Streaming Interface Specification* defines the following optional transfer properties that source and sink interfaces can implement:

■ Packet Support
■ Component-Specific Signals

The fundamental Avalon-ST transfer does not implement any of these optional transfer properties. It provides a reference point for describing how each transfer property modifies the interface and the behavior of the Avalon-ST signals. Specific properties may change the behavior of certain signal types or require the use of one or more new signal types or properties.

Avalon-ST interfaces can support multiple properties simultaneously. If the source and sink in a connection support different transfer properties or parameters, it might be necessary for the system designer to insert logic between the interfaces for them to interoperate.

3. Fundamental Data Transfers

This section defines the transfer of data from a source interface to a sink interface. In all cases, the data source and the data sink must comply with the specification. It is not the responsibility of the data sink to detect source protocol errors.

3.1. Signal Details

This section describes the basic Avalon-ST protocol that all data transfers must follow. It also highlights the flexibility a designer has in choosing Avalon-ST signals to meet the needs of a particular component and makes recommendations concerning the signals that should be used.

Figure 2. Fundamental Avalon-ST Interface Signals
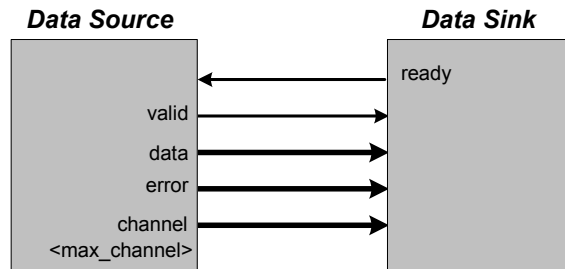
■ ready – On interfaces supporting backpressure, `ready` is asserted by the sink to mark *ready cycles*, cycles where transfers may take place. Data interfaces that support backpressure must define the `READY_LATENCY` parameter so that if `ready` is asserted on cycle *N*, cycle (*N+READY_LATENCY*) is considered a ready cycle.

■ valid – The `valid` signal qualifies valid data on any cycle where data is being transferred from the source to the sink. Such active cycles are called *beats*. The `valid` signal is required by all interfaces. On each active cycle the `data` signal and other source to sink signals are sampled by the sink.

■ data – The `data` signal typically carries the bulk of the information being transferred from the source to the sink, and consists of one or more symbols being transferred on every clock cycle. The `BITS_PER_SYMBOL` parameter defines how the data signal is divided into symbols.

■ error – Errors are signaled with the `error` signal, where each bit in `error` corresponds to a possible error condition. A value of zero on any beat indicates the data on that beat is error-free. The action that a component takes when an error is detected is up to the component and is not covered by this specification.

■ channel – The optional `channel` signal is driven by the source on every beat to indicate the channel to which the data belongs. The meaning of `channel` for a given interface depends on the application: some applications use `channel` as a port number indication, while other applications use `channel` as a page number or timeslot indication. When the `channel` signal is used, all of the

data transferred in each active cycle belongs to the same channel. The source may change to a different channel on successive active cycles.

An interface that uses the channel signal must define the MAX_CHANNEL parameter to indicate the maximum channel number. If the number of channels that the interface supports varies while the component is operating, MAX_CHANNEL is the maximum channel number that the interface can support.
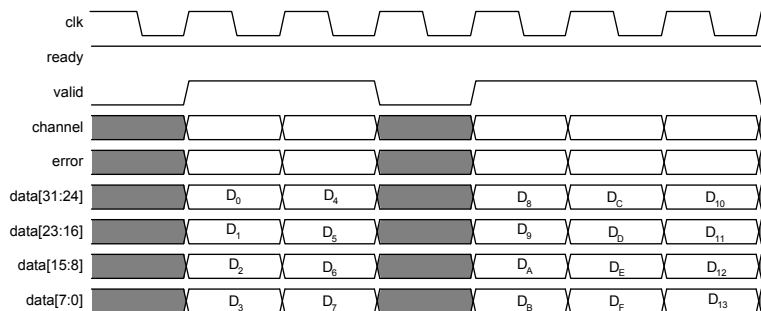
3.2. Data Layout

Symbol ordering is big endian, such that the first symbol is composed of the most significant bits. Figure 4 shows a data signal with SYMBOLS_PER_BEAT=4 and BITS_PER_SYMBOL=16.

Figure 3. Data Symbols
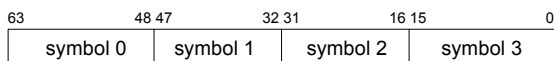
The timing diagram in Figure 4, provides an example where SYMBOLS_PER_BEAT=4 and BITS_PER_SYMBOL=8.

Figure 4. Big Endian Layout of Data

4. Data Transfer without Backpressure

The data transfer without backpressure is the most basic of Avalon-ST data transfers. On any given clock cycle, the source interface drives the `data` and the optional `channel` and `error` signals, and asserts `valid`. The sink interface samples these signals on the rising edge of the reference clock if `valid` is asserted. Figure 5 shows an example of data transfer without backpressure.

Figure 5. Data Transfer without Backpressure
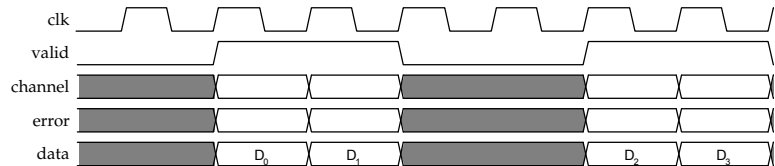
5. Data Transfer with Backpressure

The sink indicates to the source that it is ready for an active cycle by asserting the `ready` signal for a single clock cycle. Cycles during which the sink is ready for data are called *ready cycles*. During a `ready` cycle, the source may assert `valid` and provide data to the sink. If it has no data to send, it deasserts `valid` and can drive `data` to any value.

Each interface that supports backpressure defines the READY_LATENCY parameter to indicate the number of cycles from the time that `ready` is asserted until valid data can be driven. If an interface defines READY_LATENCY to be zero, then the cycle during which ready is asserted is the `ready` cycle. If READY_LATENCY has a positive value for an interface, the interface considers cycle (N + READY_LATENCY) to be a `ready` cycle if `ready` is asserted on cycle N. Any interface that includes the `ready` signal and defines the READY_LATENCY parameter supports backpressure.

When READY_LATENCY=0, data is transferred only when `ready` and `valid` are asserted on the same cycle. In this mode of operation, the source does not receive the sink's `ready` signal before it begins sending valid data. The source provides the data and asserts `valid` whenever it can and waits for the sink to capture the data and assert `ready`. The sink only captures input data from the source when `ready` and `valid` are both asserted.

Figure 6 illustrates a transfer with backpressure and READY_LATENCY=0. The source provides data and asserts `valid` on cycle one, even though the sink isn't ready. The source waits until cycle two, when the sink does

assert ready, before moving onto the next data cycle. In cycle three, the source drives data on the same cycle and the sink is ready to receive it; the transfer happens immediately. In cycle four, the sink asserts ready, but the source does not drive valid data.
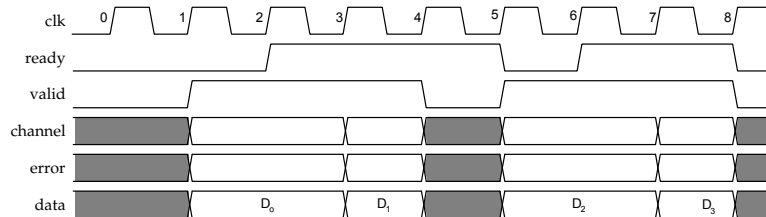
Figure 6. Transfer with Backpressure, Ready_Latency=0

Figure 7 and Figure 8 show data transfers with READY_LATENCY=1 and READY_LATENCY=2, respectively. In both these cases, ready is asserted before the ready cycle, and the source responds one or two cycles later by providing data and asserting valid. When READY_LATENCY is not zero, the source must deassert valid on non-ready cycles. The sink captures data on any cycle where valid is asserted, regardless of the value of ready on that cycle.
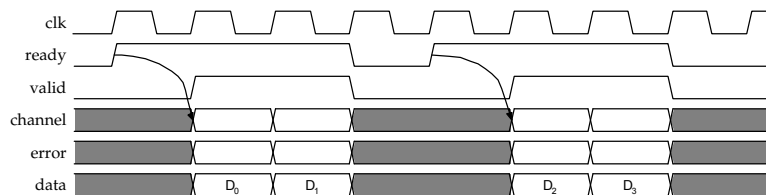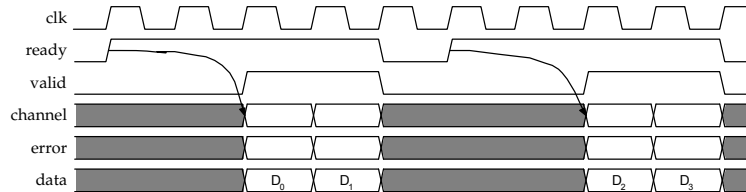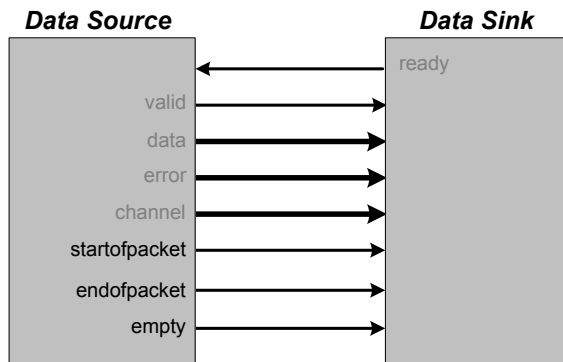
Figure 7. Transfer with Backpressure, Ready_Latency=1

Figure 8. Transfer with Backpressure, Ready_Latency=2

6. Packet Data Transfers

The packet transfer property adds support for transferring packets from a source interface to a sink interface. Three additional signals are defined to implement the packet transfer.

Figure 9. Avalon-ST Packet Interface Signals

6.1. Signal Details

■ startofpacket – The `startofpacket` signal is required by all interfaces supporting packet transfers and marks the active cycle containing the start of the packet.

■ endofpacket – The `endofpacket` signal is required by all interfaces supporting packet transfer and marks the active cycle containing the end of the packet.

■ empty – The optional empty signal indicates the number of symbols that are empty during the cycles that mark the end of a packet. The sink only checks the value of the empty signal during active cycles that have endofpacket asserted. The empty symbols are always the last symbols in data, those carried by the low-order bits. The empty signal is required on all packet interfaces whose data signal carries more than one symbol of data and have a variable length packet format. The size of the empty signal is SYMBOLS_PER_BEAT.

6.2. Protocol Details

Packet data transfer follows the same protocol as the fundamental data transfer described on page 16, with the addition of the startofpacket, endofpacket, and empty.

Figure 10 illustrates the transfer of a 17-byte packet from a source interface to a sink interface, where READY_LATENCY=0 and SYMBOLS_PER_BEAT=4. Data transfer occurs on cycles one, two, four, five, and six, when both ready and valid are asserted. During cycle one, startofpacket is asserted, and the first four bytes of packet are transferred. During cycle six, endofpacket is asserted, and empty has a value of three, indicating that this is the end of the packet and that three of the four symbols are empty.

Figure 10. Packet Transfer
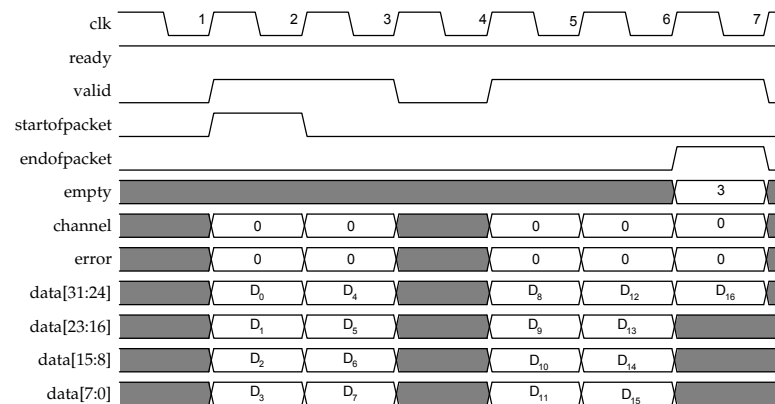
7. Document Revision History

Table 3 shows the revision history for this document.

Table 3. Document Revision History		
Date and Document Version	**Changes Made**	**Summary of Changes**
June 2007, v1.3	Removed display of features that will be implemented in a future version.	—
May 2007, v1.2	Removed display of features that will be implemented in future version.	—
May 2007 v1.1	Completed definition for empty signal in Table 1. Added revision history, contact information and typographical conventions sections.	Document now includes standard front matter.
November 2006 v1.0	Initial release	—