Avalon Tri-State Conduit Components

User Guide

Subscribe

Contents

The Avalon® Tri-State Conduit components available in the Qsys component library allow you to create on-chip controllers that connect to off-chip devices: The Generic Tri-State Conduit Controller includes parameters that you can specify to control the connected off-chip device, frequently a memory device. The Tri-State Conduit Pin Sharer arbitrates between multiple connected tri-state controllers. It drives signals from the selected controller to the Tri-State Conduit Bridge. The Tri-State Conduit Bridge converts an on-chip encoding of tri-state signals into true bidirectional signals on the PCB. Figure 1–1 illustrates the use of these three Qsys components in an Altera® FPGA.

Figure 1–1. Qsys System Using the Generic Tri-State Controller, Tri-State Conduit Pin Sharer and Bridge
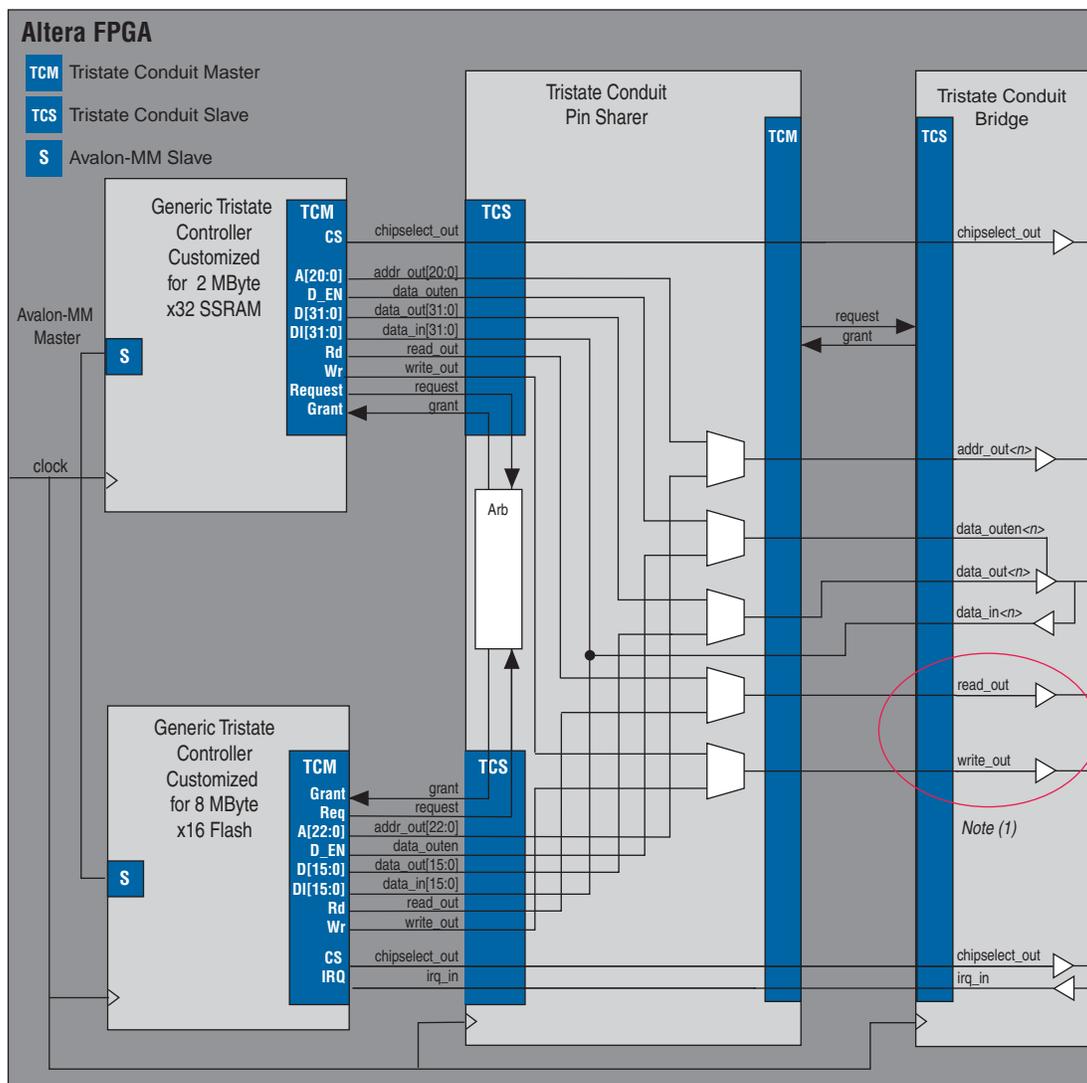
Note to Figure 1–1:

(1)   Refer to Figure 3–3 on page 3–2 for details of the logic that controls `read_out` and `write_out`.

In Figure 1–1 two instances of the Generic Tri-State Controller are customized to control off-chip SSRAM and flash memories. The Avalon Tri-state Conduit (Avalon-TC) master interfaces of these components connect to separate Avalon-TC slave interfaces of the Tri-State Conduit Pin Sharer. The Tri-State Conduit Pin Sharer arbitrates between the connected masters and drives signals from the selected master on its Avalon-TC interface which connects to the Avalon-TC slave interface of the Tri-State Conduit Bridge. Finally, the Tri-State Conduit Bridge converts the on-chip representation of the signals to bidirectional signals. It drives the bidirectional signals over its Avalon Conduit Interface to SSRAM and flash devices on the PCB. Figure 1–2 shows this system in Qsys with the addition of a Nios II processor that drives the Avalon-MM slave interfaces of the customized controllers.

Figure 1–2. Qsys Tri-State Conduit System

This user guide explains how to use the Generic Tri-State Controller and Tri-State Conduit Pin Sharer to create systems that interface to off-chip devices. It does not include a separate chapter for the Tri-State Conduit Bridge because the sole purpose of this device is to convert between the on-chip and off-chip representation of connected signals. After reading this user guide, you should be able to define controllers that interface with off-chip devices and identify signals that can be shared between interfaces to reduce the total pin count of your FPGA. This document includes the following chapters:

■ Generic Tri-State Controller

■ Tri-State Conduit Pin Sharer

The Generic Tri-State Controller provides a template for a controller that you can parameterize to reflect the behavior of an off-chip device. This component includes the following four interfaces:

■ Avalon Memory-Mapped (Avalon-MM) slave—This is the interface that connects to an Avalon-MM master, typically an embedded processor which sends read and write requests to the Generic Tri-State Controller.

■ Avalon-TC master—This is an interface that connects to the Tri-State Conduit Pin Sharer or Tri-State Conduit Bridge if pin multiplexing is not required. You easily parameterize the core to utilize enable any subset of the available signals as required by your off-chip device.

■ Avalon Clock sink—This is a clock sink interface. All Generic Tri-State Controllers connected to a single Tri-State Conduit Pin Sharer must operate in the same clock domain.

■ Avalon Reset sink—This a reset sink interface. All Generic Tri-State Controllers connected to a single Tri-State Conduit Pin Sharer must operate in the same reset domain.

Figure 2–1 illustrates the Generic Tri-State Controller interfaces and signals. This figure shows a typical set of signals for the Avalon-MM slave interface. It shows all of the possible signals for the Avalon-TC interface. Only the `request` and `grant` signals of the Avalon-TC interface are required.

Figure 2–1. Available Signals for the Generic Tri-state Controller
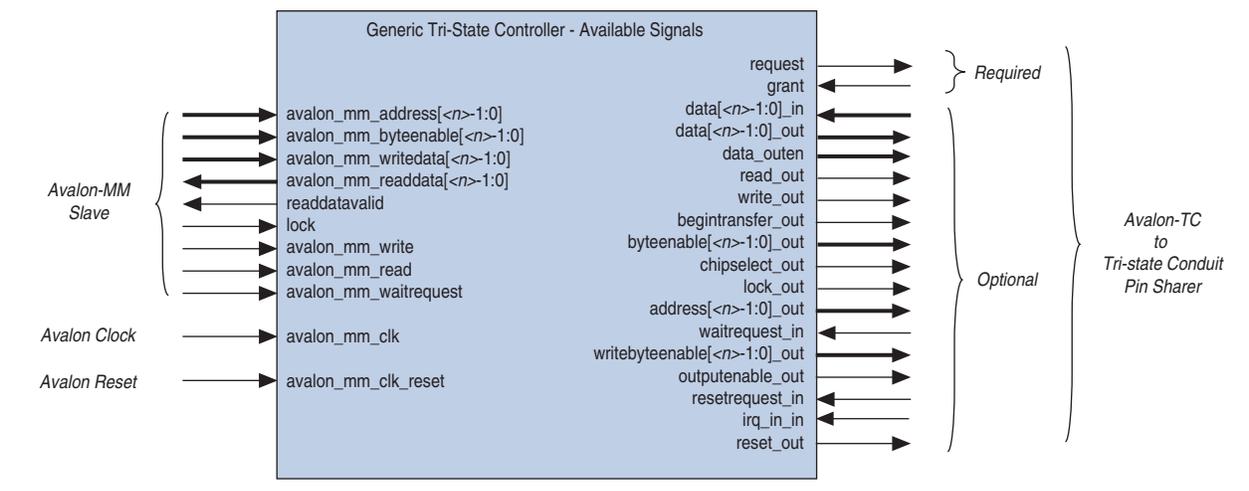
Parameters

The Generic Tri-State Controller provides preset configurations for many commonly used external devices. If you select one of the preset configurations, all of the parameters are automatically assigned the correct values. Many preset configurations are available, including presets for all of the following devices:

■ Legacy AMD 29LV065D Flash

■ AMD 29LV128M Flash with Legacy SDK support

■ Intel 128P30 Flash

■ Intel 256P30 Flash

■ SST39VF20090 Flash

■ Flash Memory Interface (CFI)

■ ISSI IS61LPS25636A-200TQL1 SSRAM

■ Cypress CY7C1380C SSRAM

■ IDT71V416 SRAM

■ LAN91C111 Interface

■ C8900 Interface (Ethernet)

You can use the parameter editor to specify the required settings for other external devices. The appropriate values for these parameters are typically listed in the vendor's data sheet for the device.

Table 2–1 describes the parameters available on the **Signal Selection** tab of the parameter editor for the Generic Tri-State Controller.

Table 2–1. Signal Selection Parameters

Parameter	Value	Description
Address Width	1–30	Specifies the width of the `address` signal.
Data Width	8,16,32,64,128, 256,512,1024	Specifies the width of the `data` signal.
Byteenable Width	1,2,4,8,16, 32,64,128	Specifies the width of the `byteenable` signal.
Bytes per word	1–128	Specifies the number of bytes per word.
readdata	**On/Off**	Enables or disables the `readdata` signal which is driven from the external device to the Generic Tri-State Controller in response to a read transfer.
writedata	**On/Off**	Enables or disables the `writedata` signal which is driven from the Generic Tri-state Controller to the external device during writes transfers.
read	**On/Off**	Enables or disables the `read` signal. If present `readdata` is required.
write	**On/Off**	Enables or disables the `write` signal. If present `writedata` is required.
begintransfer	**On/Off**	Enables or disables the `begintransfer` signal, which is asserted during the first cycle of any transfer regardless of the value of the `waitrequest` signal.

Table 2–1. Signal Selection Parameters

Parameter	Value	Description
byteenable	**On/Off**	Enables or disables the `byteenable` signal, which enables specific byte lanes during transfers on ports with widths greater than 8 bits. Each bit in the `byteenable` signal corresponds to a byte in `writedata` and `readdata`. Bit *<n>* of `byteenable` indicates whether byte *<n>* is being written to. During writes, the `byteenable` signal specifies which bytes are being written to; other bytes should be ignored by the external device. During reads, the `byteenable` signal indicates which bytes the Generic Tri-State Controller is reading.
chipselect	**On/Off**	Enables or disables the `chipselect` signal, which is always used in conjunction with the `read` or `write` `signal`.
lock	**On/Off**	Enables or disables the `lock` signal, which ensures that once a master wins arbitration, it maintains access to the slave for multiple transactions. It is asserted coincident with the first `read` or `write` of a locked sequence of transactions, and is deasserted on the final transaction of a locked sequence of transactions.
address	**On/Off**	Enables or disables the `address` signal, which represents a byte address regardless of the word size of the external device.
waitrequest	**On/Off**	Enables or disables the `waitrequest` signal, which is asserted by the external device when it is unable to respond to a `read` or `write` request.
writebyteenable	**On/Off**	Enables or disables the `writebyteenable` signal, which is equivalent to the logical AND of the `byteenable` and `write` signals. When the `writebyteenable` signal is used, the `write` and `byteenable` signals are not used.
resetrequest	**On/Off**	Enables or disables the `resetrequest` signal, which is an input from the off-chip device requesting a system reset.
irq	**On/Off**	Enables or disables the `irq` signal, which allows the external device to interrupt the Generic Tri-state Controller.
resetoutput	**On/Off**	Enables or disables the a `resetoutput` signal, from the Generic Tri-state Controller to the external device which requests that the external device be reset.
Other Parameters		
`embeddedsw.configuration.isNonVolatileStorage`		These are configuration names that you can use to identify your components to downstream embedded software tools. A value of 1 identifies the parameter as `true`, a value of 0 identifies it as `false`. For more information about these configuration names, refer to the *Publishing Component Information to Embedded Software* chapter in the *Nios II Software Developer's Handbook*.
`embeddedsw.configuration.isPrintableDevice`		
`embeddedsw.configuration.isMemoryDevice`		
`embeddedsw.configuration.isFlash`		
`embeddedsw.configuration.isEthernetMacDevice`		

Table 2–2 lists parameters that you can use to define the signal timing of an external memory. The appropriate values for these parameters are typically listed in the vendor's data sheet for the memory device.

Refer to "Example Read and Write Using Setup, Hold and Wait Times" on page 2–5 for an example that illustrates the use of the parameters defined in Table 2–2.

Table 2–2. Signal Timing

Parameter	Value	Description
Read wait time	0–10000 `cycles`	Specifies additional time in units of **timing units** read to be asserted to indicate a read transfer.
Write wait time	0–10000 `cycles`	Specifies additional time in units of **timing units** for write to be asserted to indicate a write transfer.
Setup wait time	0–10000 `cycles`	Specifies time in **timing units** between the assertion of `chipselect`, `address`, and `data` and the assertion of `read` or `write`.
Data hold time	0–1000 `cycles`	Specifies time in **timing units** between the deassertion of `write` and the deassertion of `chipselect`, `address`, and `data`. (Only applies to write transactions.)
Maximum pending read transactions	1–64	Specifies the maximum number of read transactions that can be in progress at one time. This value controls the amount of buffering in the interconnect fabric.
Turn around time	>=0	Specifies the number of clock cycles required to change access from a read to a write.
Timing units	`cycles, nanoseconds`	Specifies the units for **setup wait time**, **data hold time**, **write wait time**, and **read wait time**. Use cycles for synchronous devices and nanoseconds for asynchronous devices.
Read latency	`cycles`	Specifies the read latency for fixed-latency slaves.
Chip select through read latency	**On/Off**	When on, `chipselect` remains asserted until all pending read transfers have completed; otherwise, `chipselect` is asserted for one cycle.

☞ Because the Tri-State Conduit Pin Bridge registers incoming and out-going signals you must add two cycles latency to the **read latency** numbers in the vendor's data sheet. In calculating delays, the Generic Tri-State Controller chooses the larger of the **turn around time** and (**read latency** + two cycles). **Turn around time** is measured from the time that a command is accepted, not from the time that the previous read returned data.

Table 2–3 allows you to specify whether a signal is asserted high or low.

Table 2–3. Signal Polarities (Part 1 of 2)

Parameter	Value	Description
read	**On/Off**	When **On**, `read` is asserted low and is called `read_n`.
arbiterlock	**On/Off**	When **On**, `arbiterlock` is asserted low and is called `arbiterlock_n`.
write	**On/Off**	When **On**, `write` is asserted low and is called `write_n`.
chipselect	**On/Off**	When **On**, `chipselect` is asserted low and is called `chipselect_n`.
byteenable	**On/Off**	When **On**, `byteenable` is asserted low and is called `byteenable_n`.
outputenable	**On/Off**	When **On**, `outputenable` is asserted low and is called `outputenable_n`.
writebyteenable	**On/Off**	When **On**, `writebyteenable` is asserted low and is called `writebyteenable_n`.
waitrequest	**On/Off**	When **On**, `waitrequest` is asserted low and is called `waitrequest_n`.
begintransfer	**On/Off**	When **On**, `begintransfer` is asserted low and is called `begintransfer_n`.

Table 2–3. Signal Polarities   (Part 2 of 2)

Parameter	Value	Description
resetrequest	On/Off	When **On**, `resetrequest` is asserted low and is called `resetrequest_n`.
irq	On/Off	When **On**, `irq` is asserted low and is called `irq_n`.
reset output	On/Off	When **On**, `resetoutput` is asserted low and is called `resetoutput_n`.

Example Read and Write Using Setup, Hold and Wait Times

Figure 2–2 on page 2–6 illustrates the timing for a memory device that has asynchronous read and write transfers, assuming a 50 MHz clock. Table 2–4 lists the parameter values set in the Generic Tri-State Controller to access this device.

Table 2–4. Wait Times Expressed in Nanoseconds - 50 MHz Clock

| Timing Parameter	Time in Nanoseconds	Read or Write Time
Setup wait time	50 ns	60 ns (3 clock periods)
Data hold time	10 ns	20 ns (1 clock period)
Write wait time	30 ns	40 ns (2 clock periods)
Read wait time	30 ns	40 ns (2 clock periods)

When the wait time is expressed in nanoseconds, the read or write period, as seen on the FPGA pins, is the duration of the specified wait time, rounded up to the next clock period as Example 2–1 illustrates.

Example 2–1. Cycles When Wait Time Equals 21 ns

```
clock cycles = ceil(wait time in ns /clock period in ns)
clock cycles = ceil(21/20) = ceil (1.05) = 2
```

Figure 2–2 illustrates the timing of reads and writes given parameter settings specified in Table 2–4.

Figure 2–2. Read and Write Transfers with Setup Time and Wait States
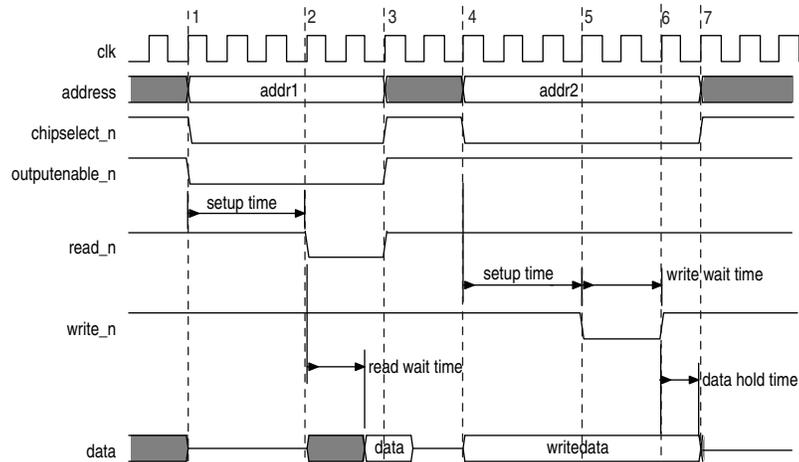
Notes to Figure 2–2:

(1) The master drives `address` and asserts `chipselect_n`.

(2) After 3 cycles **setup wait time**, the interconnect asserts `read_n`.

(3) The slave port deasserts `read_n` after 2 cycles (from 30 ns) of **read wait time**. Data is sampled at the rising clock edge.

(4) `address` and `writedata` are driven.

(5) `write_n` is driven after 3 cycles (from 50 ns) **setup wait time**.

(6) `write_n` is deasserted after two cycles (from 30 ns) of **write wait time**.

(7) `address`, `chipselect`, and the data bus stop being driven after 1 cycle (from 10 ns) of **hold time**.

The Tri-State Conduit Pin Sharer multiplexes between the signals of the connected tri-state controllers. You can connect controllers created by customizing the Generic Tri-State Controller or your own custom controllers. When you instantiate the Tri-State Conduit Pin Sharer, you specify the number of connected interfaces and identify the signals that share pins. The pin sharer arbitrates between connected masters using a round-robin algorithm. It drives the signals of the granted Avalon-TC master to the Tri-State Conduit Bridge.

The following sections explain how to use the Tri-State Conduit Pin Sharer in more detail.

Signal Naming

The *Avalon Interface Specifications* for Avalon-TC interfaces requires that signal names have the following two parts:

■ A role—The role defines the signal to Qsys and typically represents the function of the signal. Signals with identical roles can be shared. Typical roles include: address, data, read, and write.

■ A pin type—The pin type must be specified using a suffix appended to a signal's role. The Tri-State Conduit Pin Sharer recognizes three pin type suffixes: `_out`, `_outen`, and `_in`. Theses three suffixes define the following four pin types:

  ■ Bidirectional—Bidirectional pins define three signals: `<role>_outen`, `<role>_out`, and `<role>_in`.

  ■ Tri-State output—Tri-State output pins define two signals: `<role>_outen` and `<role>_out`. Use this pin type for outputs that should be driven to a high impedance state during system reset.

  ■ Output—Output pins define a single signal: `<role>_out`.

  ■ Input—Input pins define a single signal: `<role>_in`.

Figure 3–1 illustrates the naming conventions for Avalon-TC shared pins.

Figure 3–1. Shared Pin Types
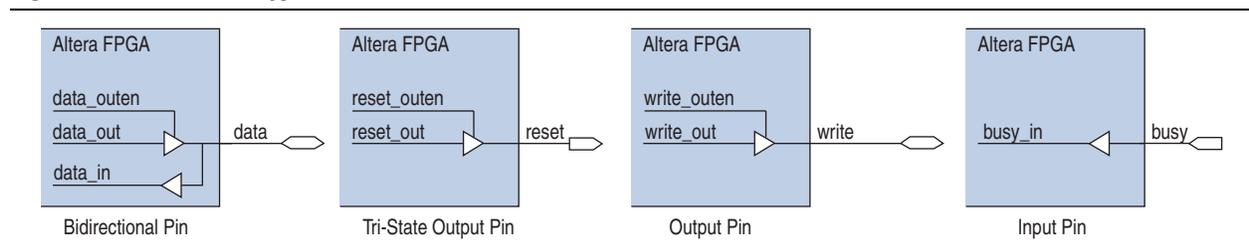
If the widths of shared signals differ, the Tri-State Conduit Pin Sharer aligns them on their 0th bit and drives the higher-order pins to 0 whenever the narrower signal has control of the bus. Signals that are not shared propagate directly through the Tri-State Conduit Pin Sharer. In Figure 3–1, `chipselect_out` and `irq_in` are not shared.

Parameters

The parameter editor for the Tri-State Conduit Pin Sharer allows you to specify a single parameter, the **Number of Interfaces**. This parameter specifies the number of controllers that connect to the Tri-State Conduit Pin Sharer.

Complete the following steps to specify shared pins among connected controllers:

1. Add the Tri-State Conduit Pin Sharer to your Qsys design, specifying the number of interfaces that connect to it.

2. In the Qsys **Connections** column, connect the tri-state conduit controllers to the Tri-State Conduit Pin Sharer.

3. The Tri-State Conduit Pin Sharer parameter editor has a table with the following columns: **Interface**, **Signal Role**, **Signal Type**, **Signal Width**, and **Shared Signal Name**. To share a signal, type values in the **Interface**, **Signal Role**, and **Shared Signal Name** columns for all controllers that share that signal.

4. You can use the **Update Interface Table** button to automatically populate these values. Figure 3–2 shows that the CFI_Flash memory and IDT_SRAM memory share the `address` signal.

Figure 3–2. Specifying Shared Signals Using the Tri-State Conduit Pin Sharer
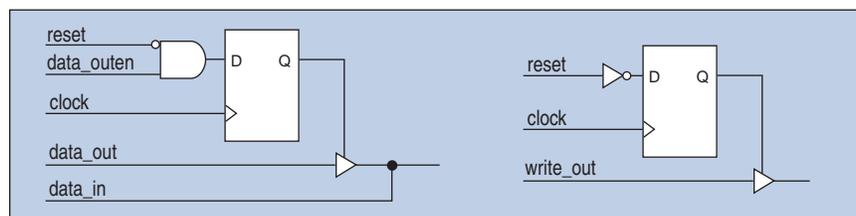

Signal Behavior During Reset

At power-on reset, the enables for all bidirectional and tri-state output signals are disabled. Figure 3–3 shows the logic that controls these signals.

Figure 3–3. Control of Bidirectional and Tri-State Output Signals

Arbitration

Each Avalon-TC master and slave pair includes separate `request` and `grant` signals. Arbitration logic in the Tri-State Conduit Pin Sharer grants requesting masters in round-robin order. The meaning of the `request` signal depends on the state of the `grant` signal. The `request`/`grant` algorithm has the following dependency on the current state:

1. When `request` is asserted and `grant` is deasserted, `request` is requesting access for the *current* cycle.

2. When `request` is asserted and `grant` is asserted, `request` is requesting access for the *next* cycle; consequently, `request` should be deasserted on the final cycle of an access.

Because `request` is deasserted in the final cycle of a bus access, it can be reasserted immediately following the final cycle of a transfer, making both rearbitration and continuous bus access possible if no other masters are requesting access. After it is asserted, request must remain asserted until granted; consequently, the shortest bus access is two cycles.

The `grant` signal is asserted in response to the `request` signal and remains asserted until one cycle following the deassertion of `request`. The design of the Avalon-TC interface does not allow a default Avalon-TC master to be granted bus access when no masters are requesting.

Figure 3–4 illustrates arbitration timing for the Tri-State Conduit Pin Sharer. As this figure illustrates, a device can drive or receive valid data in the granted cycle. Figure 3–4 shows the following sequence of events:

1. In cycle 1, the tri-state conduit master asserts grant. The granted slave drives valid data in cycles 1 and 2.

2. In cycle 4, the tri-state conduit master asserts grant. The granted slave drives valid data in cycles 4–7.

3. In cycle 8, the tri-state conduit master asserts grant. The granted slave drives valid data in cycles 8–16.

4. Cycle 3 is the only cycle that does not contain valid data.

Figure 3–4. Arbitration Timing
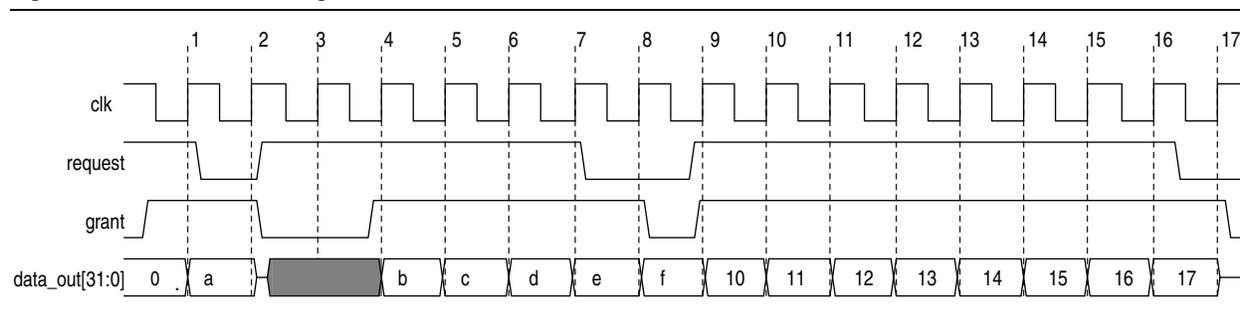
Hierarchical Pin Sharing

Figure 1–1 on page 1–1 provides the simplest use of the Tri-State Conduit Pin Sharer; it multiplexes between two controllers. You can also create a design that includes hierarchy of Tri-State Conduit Pin Sharers that access the same pins on the FPGA. In Figure 3–5, Sub-systems A and B each include two controllers that access off-chip devices. The pin sharer in the Top Level system connects to a single Tri-State Conduit Bridge that drives signals on the PCB.

Figure 3–5. Using The Tri-State Conduit Pin Sharer in Hierarchical Designs
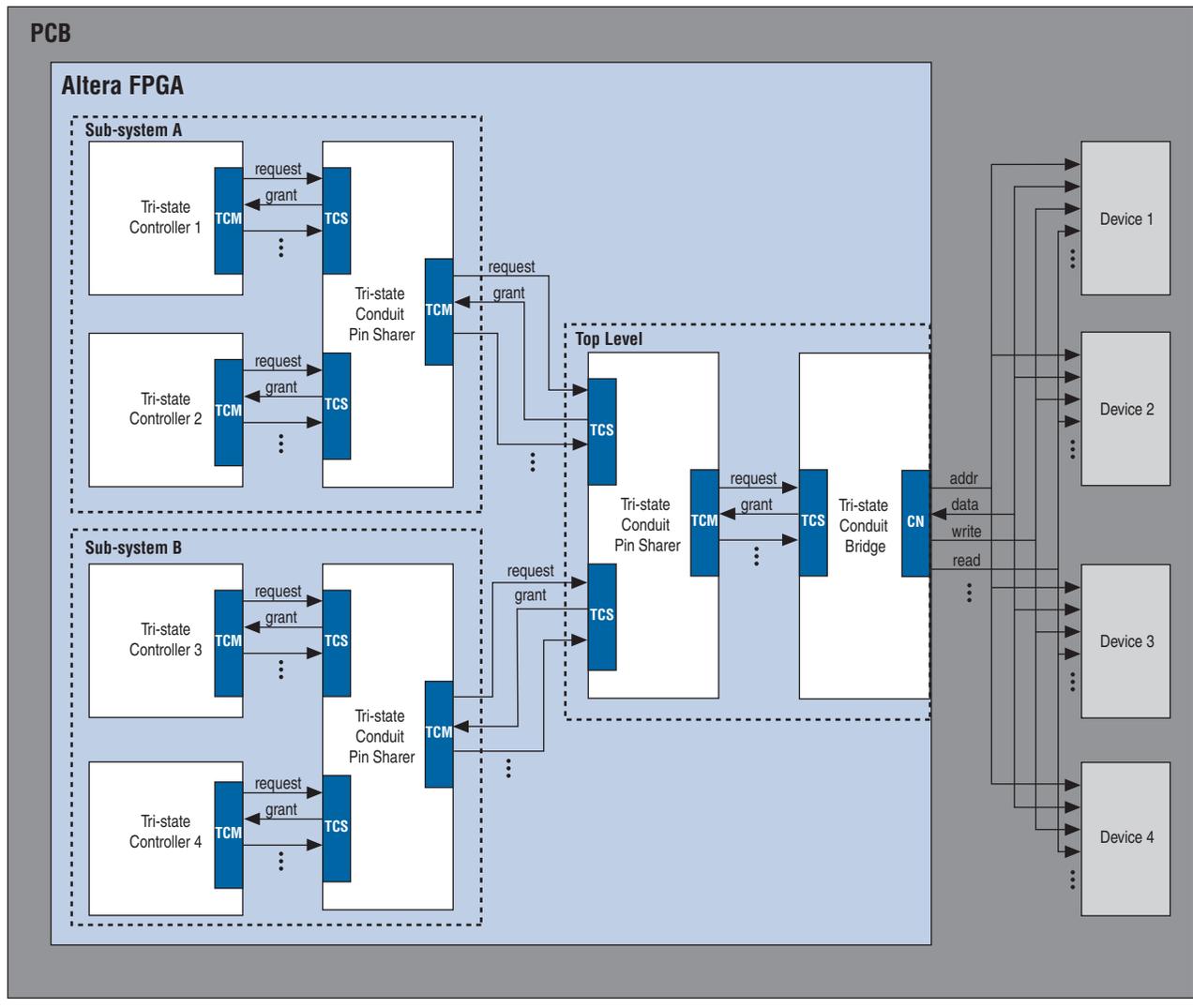
This chapter provides additional information about the document and Altera.

Document Revision History

The following table shows the revision history for this document.

Date	Version	Changes
May 2011	1.0	Initial release.

How to Contact Altera

To locate 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
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

The following table shows the typographic conventions this document uses.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Indicate document titles. For example, *Stratix IV Design Guidelines*.
italic type	Indicates variables. For example, $n + 1$. Variable names are enclosed in angle brackets (< >). For example, *<file name>* and *<project name>*.**pof** file.
Initial Capital Letters	Indicate keyboard keys and menu names. For example, the Delete key and the Options menu.

Visual Cue	Meaning
"Subheading Title"	Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, "Typographic Conventions."
Courier type	Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix `n` denotes an active-low signal. For example, `resetn`.
	Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`.
	Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`).
↵	An angled arrow instructs you to press the Enter key.
1., 2., 3., and a., b., c., and so on	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ■ ■	Bullets indicate a list of items when the sequence of the items is not important.
☞	The hand points to information that requires special attention.
ⓘ	A question mark directs you to a software help system with related information.
👣	The feet direct you to another document or website with related information.
⚠ CAUTION	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
⚠ WARNING	A warning calls attention to a condition or possible situation that can cause you injury.
✉	The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents.