


The Qsys interconnect is a high-bandwidth structure for connecting components that use Avalon® interfaces. This chapter describes the Qsys interconnect. The interconnect uses algorithmic transformations to insert interconnect components in implementing the Qsys system. This chapter also provides brief descriptions of the Qsys interconnect components that implement the interconnect. All Qsys interconnect components are available to be used in your own designs. The Qsys interconnect connects the following Avalon interface types:

- Avalon-ST—Connects Avalon-ST sources and sinks that stream unidirectional data.
- Avalon-MM—Connects Avalon-MM master and slaves that communicate using read and write commands.
- Avalon Tri-state Conduits (Avalon-TC)— Connects tri-state conduit controllers in the FPGA to tri-state devices on the PCB using a three-signal encoding of tri-state information.
- Avalon Interrupts—Connects interrupt senders and the interrupt receivers of the component that service them.
- Avalon Clocks—Connects clock sources and clock sinks.
- Avalon Resets—Connects reset sources and reset sinks.
- Avalon Conduits—Connects point-to-point conduit interfaces. You can use the conduit interface type to define an arbitrary collection of signals that does not fit into any of the other Avalon interface categories.

 For more information about the Avalon interfaces, refer to the [Avalon Interface Specifications](#).

For Avalon-ST interfaces, Qsys provides adapters that allow flexibility in creating point-to-point connections. For example, the Avalon-ST data format adapter allows you to connect streaming interfaces of different widths.

For Avalon-MM interfaces, the implementation of the Qsys interconnect is based on a network-on-chip architecture. Transactions between masters and slaves are encapsulated in packets and transmitted on a network that carries the packets between masters and slaves. The master command network transports read and write command packets from master interfaces to slave interfaces. The slave response network transports read response packets from slave interfaces to master interfaces.

This chapter includes the following sections:

- “Avalon-MM Interface Components” on page 7-2
- “Avalon-ST Interfaces” on page 7-18
- “Tri-state Conduit Components” on page 7-21
- “Interrupt Interfaces” on page 7-26
- “Clock Interfaces” on page 7-28
- “Reset Interfaces” on page 7-28
- “Avalon Conduits” on page 7-29

Avalon-MM Interface Components

Qsys interconnect for memory-mapped interfaces connects Avalon-MM master and slave interfaces. It supports the following items:

- Any number of master and slave components. The master-to-slave relationship can be one-to-one, one-to-many, many-to-one, or many-to-many.
- Master and slaves of different data widths.
- Components operating in different clock domains.
- Components with different interface properties and signals. Qsys can adapt the component interfaces so that interfaces with the following types differences can be connected:
 - Interfaces that use active-high and active-low signalling
 - Interfaces with different burst characteristics
 - Interfaces with different latencies
 - Interfaces with different port signatures

Figure 7-1 is a simplified representation of the Qsys interconnect for an Avalon-MM system with multiple masters. As this figure illustrates, the underlying implementation of the master and slave connections uses a network topology. When you generate a Qsys system, Qsys implements the interconnect connectivity that you specified, replacing the point-to-point connections you created in the **Connections** column with a network topology.

Figure 7-1. Qsys interconnect—Example System

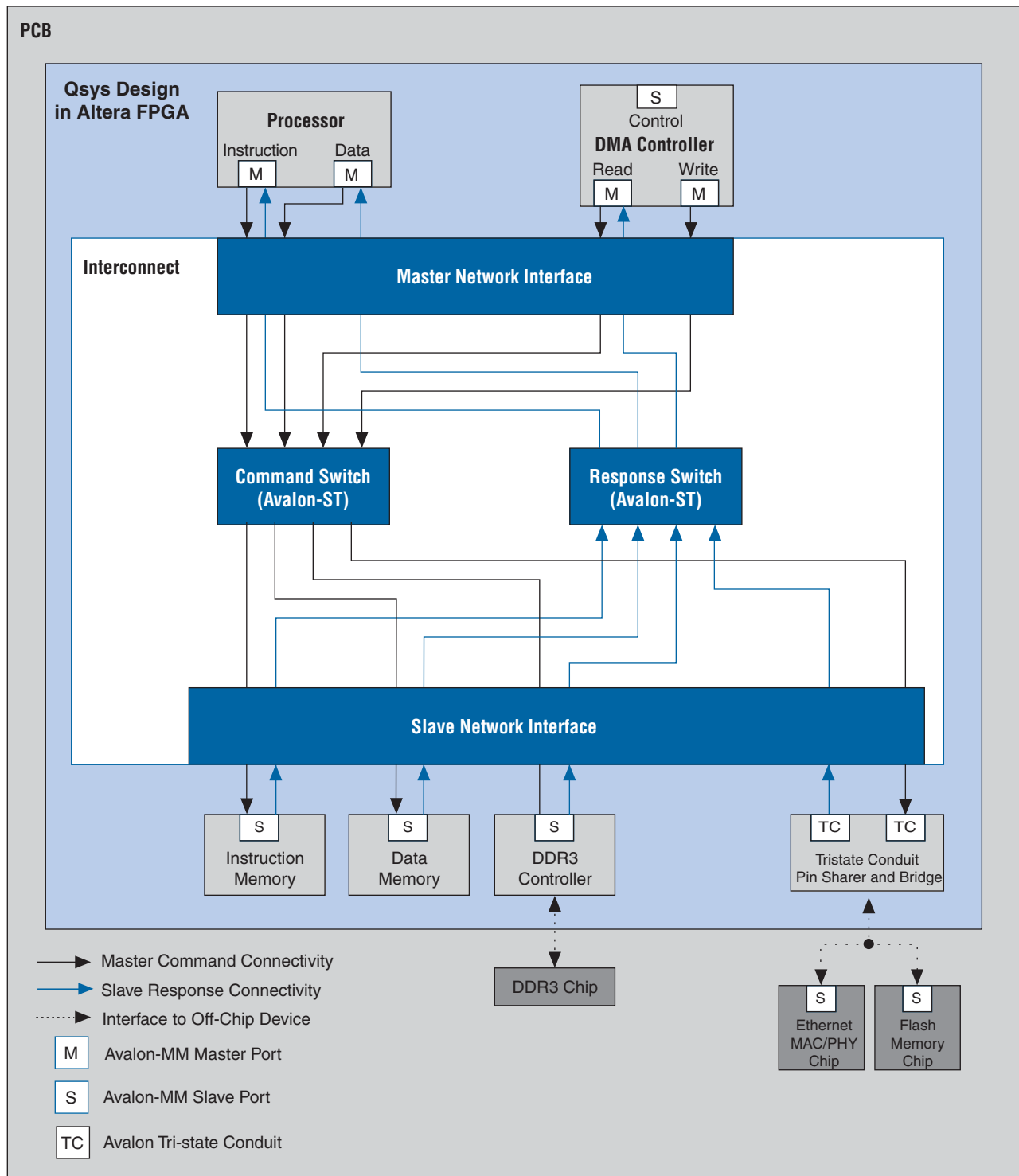


Figure 7-2 shows the format of the Qsys packet that encapsulates the Avalon-MM master commands and Avalon-MM slave responses.

Figure 7-2. Qsys Packet Format

Address	Transaction type	Data	Byte enable	Source ID	Destination ID	Byte count	Burstwrap	Protection
---------	------------------	------	-------------	-----------	----------------	------------	-----------	------------

Table 7-1 describes the fields of Qsys packet.

Table 7-1. Qsys Packet Format

Field	Description
Address	Specifies the byte address for the lowest byte in the current cycle.
Transaction_type	Indicates the transaction type. Table 7-2 lists the 5 transaction types.
Data	For command packets, carries the data to be written. For read response packets, carries the data that has been read.
Byteenable	Specifies which symbol of the data are valid. The following values are legal for Avalon-MM master and slaves transferring 32-bit data: <ul style="list-style-type: none"> ■ 1111 writes full 32 bits ■ 0011 writes lower 2 bytes ■ 1100 writes upper 2 bytes ■ 0001 writes byte 0 only ■ 0010 writes byte 1 only ■ 0100 writes byte 2 only ■ 1000 writes byte 3 only
Source_ID	The ID of the master or slave that initiated the command or response.
Destination_ID	The ID master or slave to which the command or response is directed.
Byte count	The number of bytes remaining in the transaction, including this beat. Number of bytes requested by the packet.
Burstwrap	The burstwrap value specifies the wrapping behavior of the current burst. The burstwrap value is of the form $2^{<n>-1}$. The following types are defined: <ul style="list-style-type: none"> ■ Variable wrap—Variable wrap bursts can wrap at any integer power of 2 value. When the burst reaches the wrap boundary, it wraps back to the previous burst boundary so that only the low order bits are used for addressing. For example, a burst starting at address 0x1C, with a burst wrap boundary of 32 bytes and a burst size of 20 bytes, would write to addresses 0x1C, 0x0, 0x4, 0x8, and 0xC. For a burst wrap boundary of size $<m>$, $Burstwrap = <m> - 1$, or for this case $Burstwrap = (32 - 1) = 31$ which is $2^5 - 1$. ■ Sequential—Sequential bursts increment the address for each transfer in the burst. For sequential bursts, the <i>Burstwrap</i> field is set to all 1s. For example, with a 6-bit <i>Burstwrap</i> field, the value for a sequential burst is 6'b111111 or 63, which is $2^6 - 1$. <p>In version 11.0 of the Quartus II software, adaptation logic sets a hardwired value for the burstwrap field, according the declared master burst properties. For example, for a master which declares sequential bursting, the burstwrap field is set to all 1s. Similarly, masters that declare linewrap burst have their burstwrap field set to the appropriate constant value.</p>
Protection	Access level protection. When 0, the packet has normal access. When 1, the packet has privileged access. For Avalon-MM interfaces, this field maps directly to the privileged access signal, which allows an Avalon-MM master to write to an on-chip memory ROM instance.

Table 7-2 lists the transaction type encodings.

Table 7-2. Transaction Types

Bit	Name	Definition
0	PKT_TRANS_READ	When asserted, indicates a read transaction.
1	PKT_TRANS_COMPRESSED_READ	For read transactions, specifies whether or not the read command can be expressed in a single cycle, that is whether or not it has all <code>byteenables</code> asserted on every cycle.
2	PKT_TRANS_WRITE :	When asserted, indicates a write transaction.
3	PKT_TRANS_POSTED	When asserted, no response is required.
4	PKT_TRANS_LOCK	When asserted, indicates arbitration is locked. Applies to write packets.

The fields of the Qsys packet format are variable length to minimize the resources used. However, if the majority of components in a design have a single data width, for example 32 bits, and a single component has a data width of 64 bits, Qsys inserts a width adapter to accommodate 64-bit transfers.

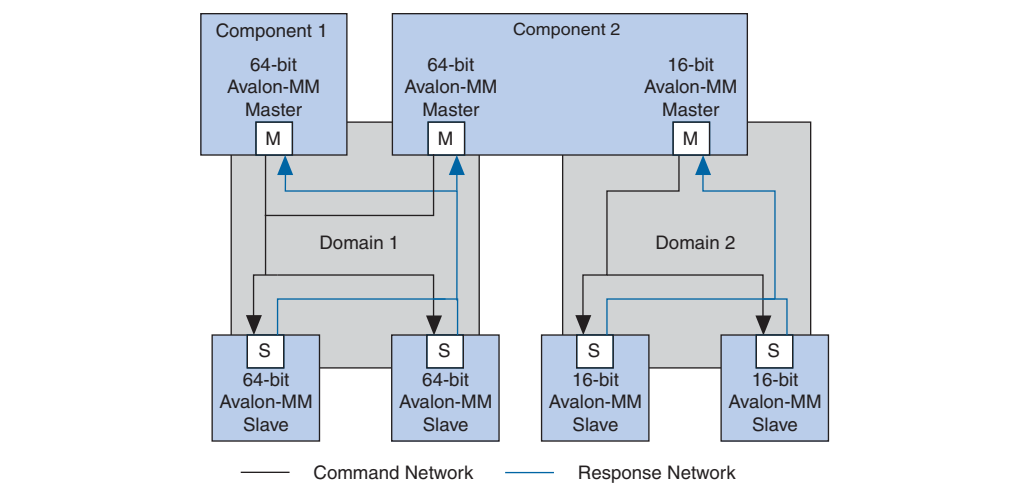
Component Interconnect Domains

A group of connected Avalon-MM masters and slaves is called an *interconnect domain*. The components in a single interconnect domain share the same packet format. The following two examples illustrate this point.

Using Two Separate Domains

Figure 7-3 illustrates the use of two separate domains. The first domain includes two, 64-bit masters connected to two, 64-bit slaves. The second domain includes one, 16-bit master connected to two, 16-bit slaves. Because the interfaces in Domain 1 and Domain 2 do not share any connections, Qsys can optimize the packet format for the two separate domains. In this example, the first domain uses a 64-bit data width and the second domain uses 16-bit data.

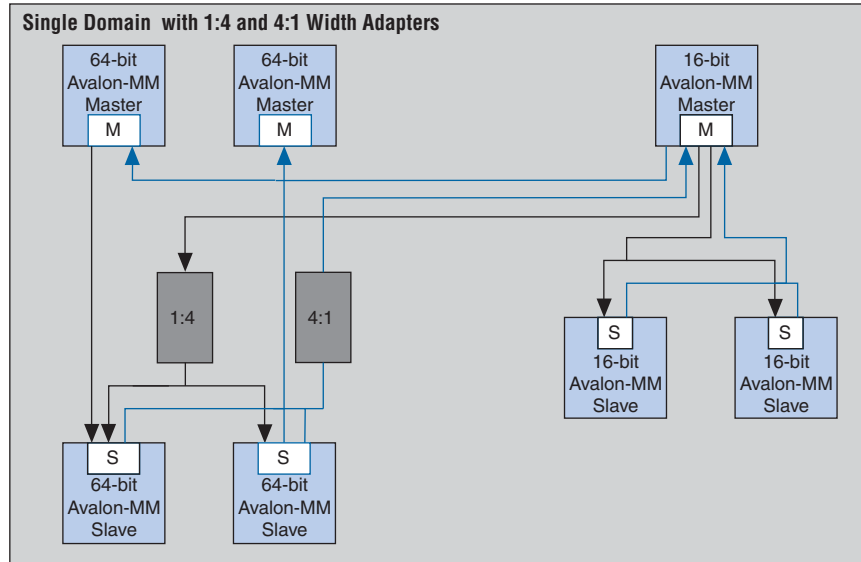
Figure 7-3. Two Domains



Using One Domain with Width Adaptation

Figure 7-4 illustrates a Qsys system that includes two, 64-bit masters that access two, 64-bit slaves. It also includes one, 16-bit Master, accessing two, 16-bit slaves and one, 64-bit slave. Because one of the masters connects to all of the slaves, Qsys creates a single domain with two packet formats: one with 64-bit data and one with 16-bit data. A width adapter manages accesses between the 16-bit master and 64-bit slaves.

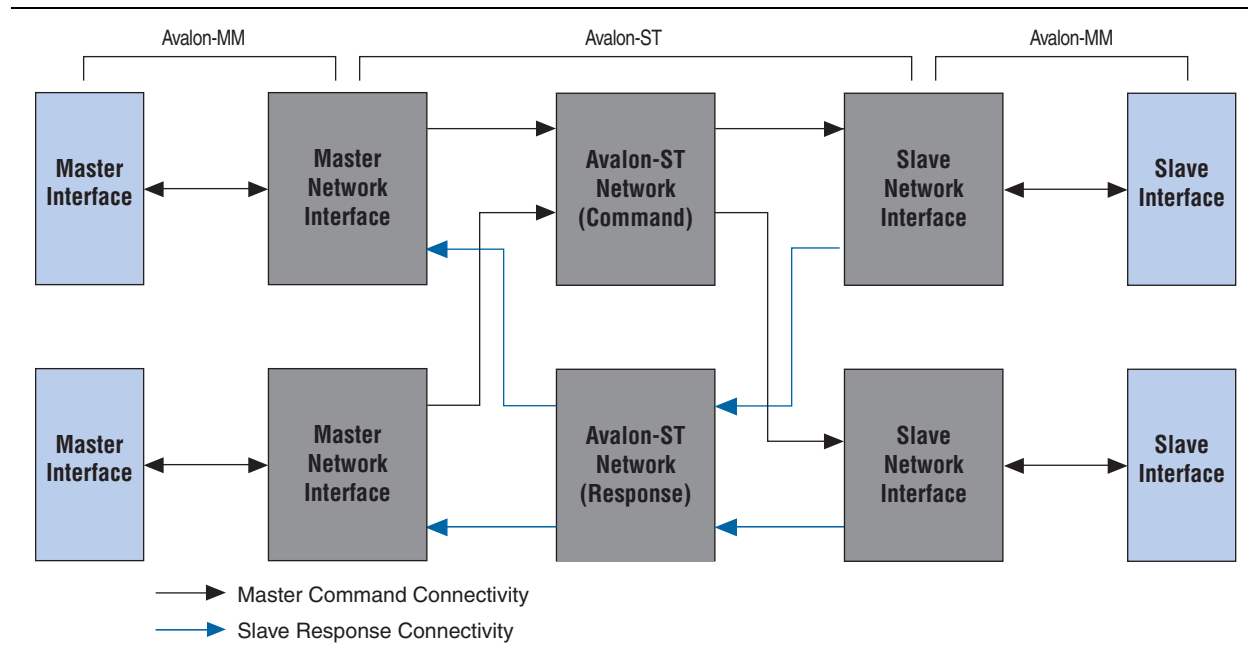
Figure 7-4. One Domain with 1:4 and 4:1 Width Adapters



Qsys Transformations

Figure 7-5 provides a more detailed view of the transformation that occurs when you generate a Qsys system with Avalon-MM master and slave components. As this figure illustrates, the Avalon-MM master and slave components connect to network interface modules that encapsulate the transaction in Avalon ST packets. The Avalon-MM interfaces have no information about the encapsulation or the function of the layer transporting the packets and simply operate in accordance with Avalon-MM protocol, using the read and write signals and transfers as defined in the *Avalon Interface Specifications*.

Figure 7-5. Qsys Transform from Avalon-MM to Avalon-ST



Master Command and Slave Response Networks

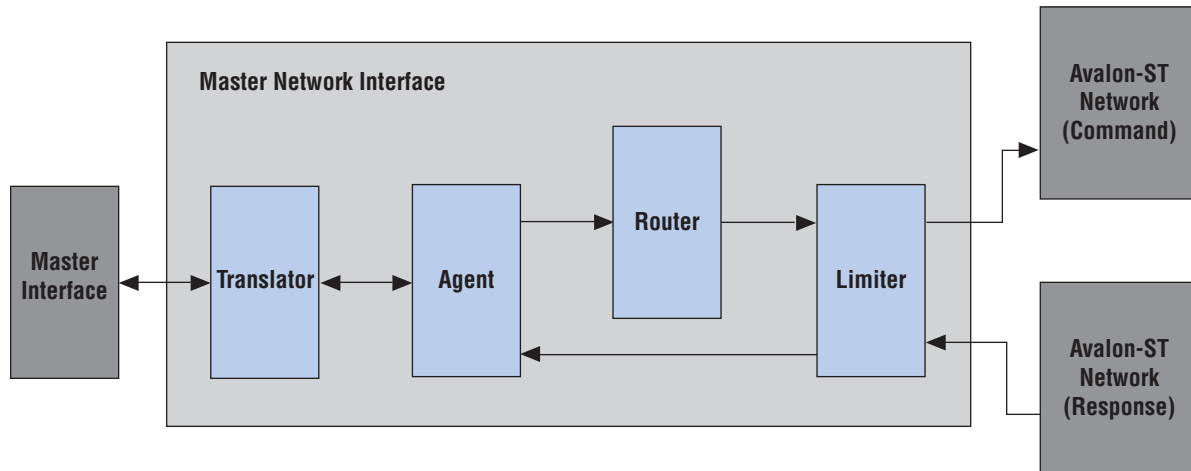
Many Qsys components implement the Qsys interconnect and network interfaces represented by the Avalon-ST Network (Command) and Avalon-ST Network (Response) blocks in Figure 7-5. All of these Qsys components are provided by Altera and included in the component library available in Qsys. They are available for you to be used stand-alone in your designs. For example, you may want to include the Avalon-ST pipeline stage in your datapath to pipeline a streaming connection, thus increasing the clock frequency of your design.

The following sections describe the components that are part of the Avalon-ST master command and Avalon-ST slave response network, including the following components:

- Merlin Master Translator
- Merlin Master Agent
- Merlin Router
- Merlin Traffic Limiter
- Merlin Slave Translator
- Merlin Slave Agent

Figure 7-6 provides a block diagram for the Master command network showing the Merlin Master Translator, Agent, Router, and Limiter.

Figure 7-6. Qsys Components in the Master Command Network



Merlin Master Translator

The Merlin Master Translator interfaces of an Avalon-MM master component. It converts the Avalon-MM master interface to a simpler representation that the Qsys network uses. It performs the following functions:

- Translates active low signalling to active high signalling
- Inserts wait states to prevent an Avalon-MM master from reading invalid data
- Translates word and symbol addresses
- Translates word and symbol burst counts
- Handles burst count timing and sequencing
- Removes unnecessary address bits

Merlin Master Agent

The Agent translates Avalon-MM master transactions into Qsys command packets and translates the Qsys Avalon-MM slave response packets into Avalon-MM responses.

Merlin Router

The Router routes command packets from the master to the slave and response packets from the slave to the master. For master command packets, the router uses the Avalon-MM address to set the `Destination_ID` and Avalon-ST channel. For the slave response packet, the router uses the `Destination_ID` to set the Avalon-ST channel. The demultiplexers use the Avalon-ST channel to route the packet to the correct destination.

Merlin Traffic Limiter

The Traffic Limiter ensures the responses arrive in order. It prevents any command from being sent if the response could conflict with the response for a command that has already been issued. By guaranteeing in-order responses, the Traffic Limiter simplifies the response network.

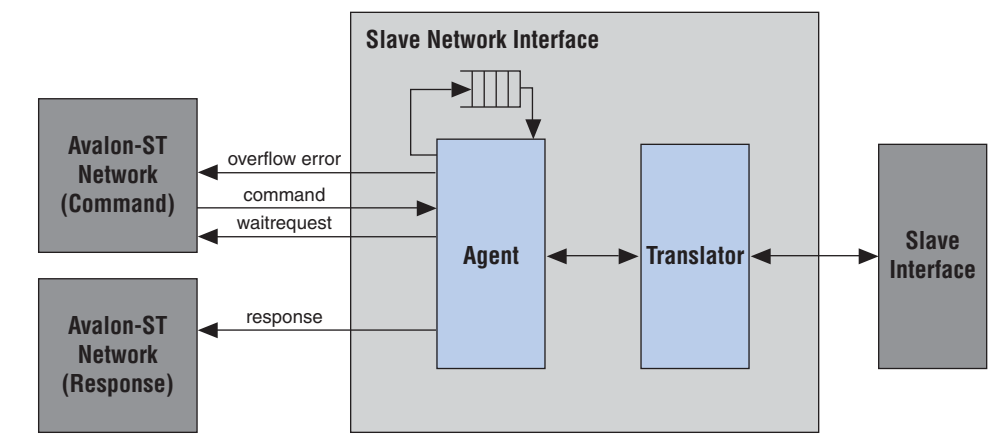
Merlin Slave Translator

The Merlin Slave Translator interfaces to an Avalon-MM slave component as [Figure 7-7](#) illustrates. It converts the Avalon-MM slave interface to a simplified representation that the Qsys network uses. An Avalon-MM Merlin Slave Translator performs the following functions:

- Drives the `begintransfer`, `beginbursttransfer`, and `writebyteenable` signals
- Supports Avalon-MM slaves that operate using fixed timing and or slaves that use the `readdatavalid` signal to identify valid data
- Translates the `read`, `write`, and `chipselect` signals into the representation that the Avalon-ST slave response network uses
- Converts active low signals to active high signals
- Translates word and symbol addresses and burstcounts
- Handles burstcount timing and sequencing
- Removes unnecessary address bits

[Figure 7-7](#) shows the Qsys components that comprise the slave response network.

Figure 7-7. Qsys Components in the Slave Response Network



Merlin Slave Agent

The Slave Agent accepts command packets and issues the resulting transactions to the Avalon interface. For pipelined slaves, an Avalon-ST FIFO stores information about pending transactions. The size of this FIFO is the maximum number of pending responses that you specify when creating the slave component.

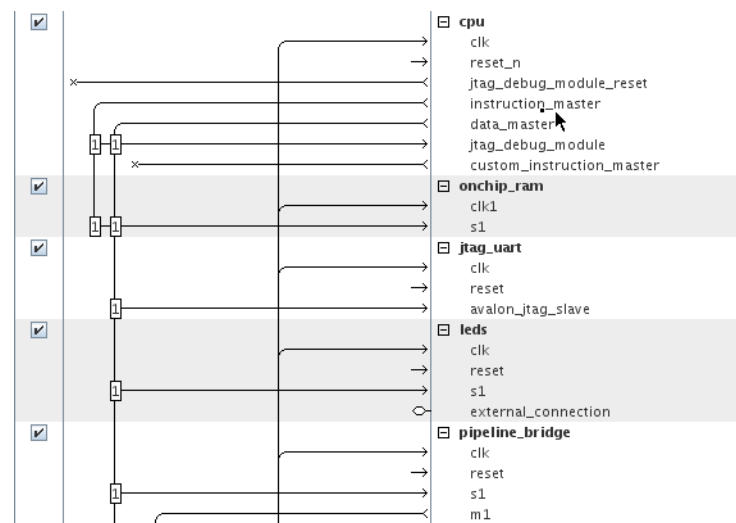
The Merlin Slave Agent also backpressures the Avalon-MM master command interface when the FIFO is full if the slave component includes the `waitrequest` signal.

Arbitration

When multiple masters contend for access to a slave, Qsys automatically inserts arbitration logic which grants access in fairness-based, round-robin order. In a fairness-based arbitration scheme, each master has an integer value of transfer *shares* with respect to a slave. One share represents permission to perform one transfer. The default arbitration scheme is equal share round-robin granting equal, sequential access to all requesting masters. You can change the arbitration scheme to weighted round-robin by specifying a relative number of arbitration shares to the masters that access a particular slave. To display arbitration settings, on the View menu, click **Show Arbitration**.

Figure 7-8 illustrates the arbitration shares.

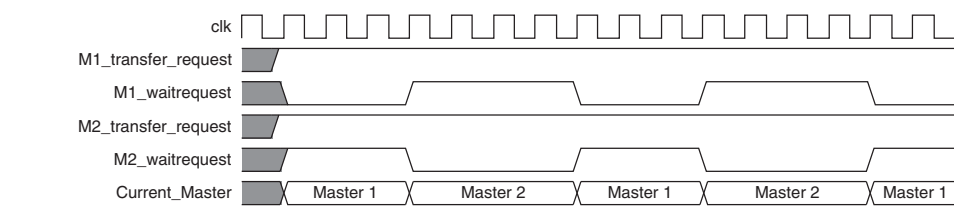
Figure 7-8. Arbitration Settings on the System Contents Tab



Arbitration Examples

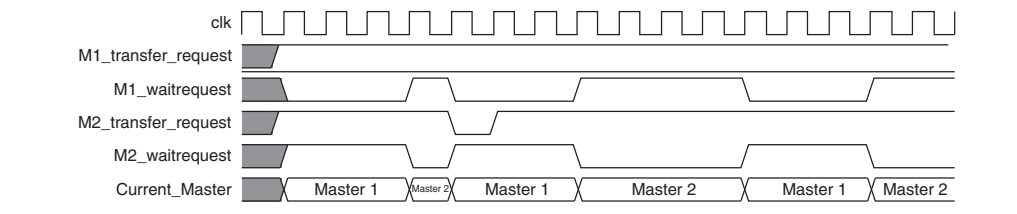
Figure 7-9 illustrates the timing for two Avalon-MM masters continuously accessing a single Avalon-MM slave to perform back-to-back transfers. Master 1 has three shares and Master 2 has four shares. The Merlin arbiter grants Master 1 access for three transfers, then Master 2 for four transfers. This cycle repeats indefinitely.

Figure 7-9. Arbitration of Continuous Transfer Requests from Two Masters



If a master stops requesting transfers before it exhausts its shares, it forfeits all of its remaining shares, and the Merlin Arbiter grants access to another requesting master as [Figure 7-10](#) illustrates. After completing one transfer, Master 2 stops requesting for one clock cycle. As a result, the arbiter grants access back to Master 1, which gets three shares.

Figure 7-10. Arbitration of Two Masters with a Gap in Transfer Requests

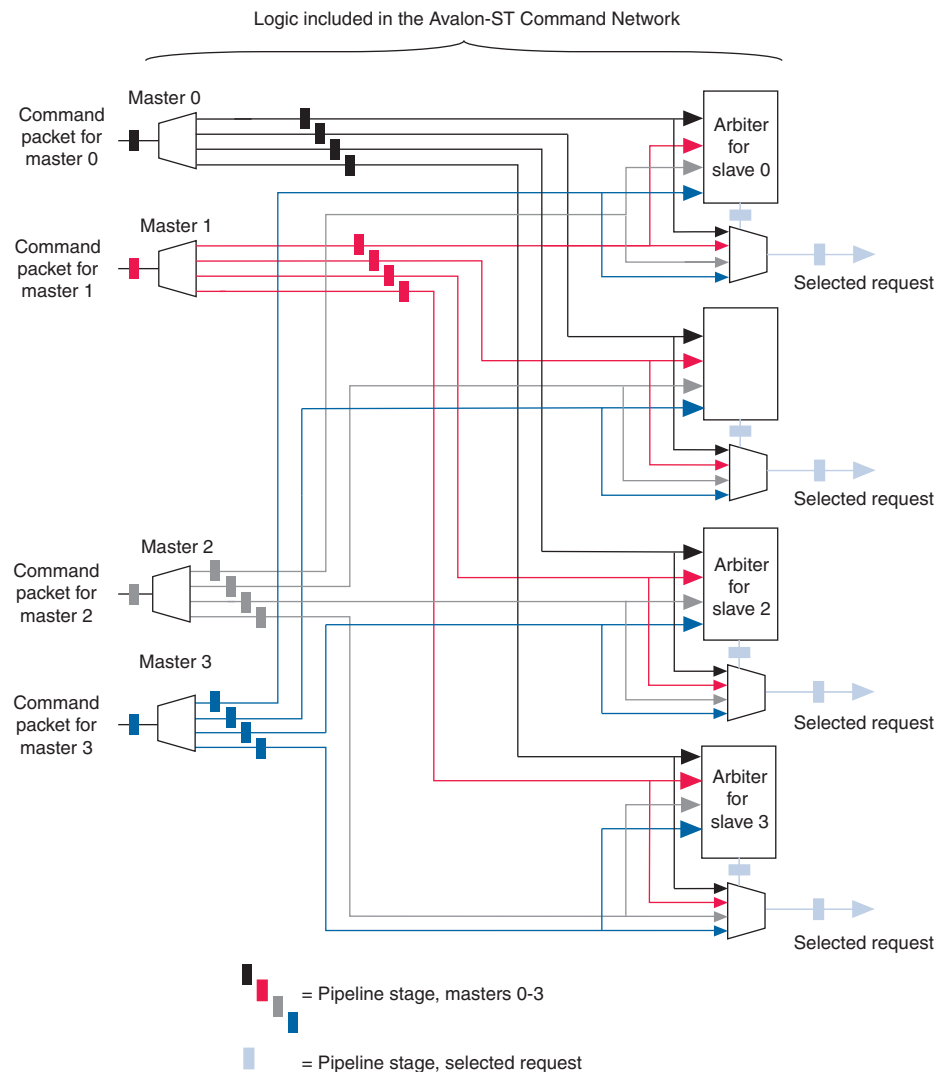


Merlin Arbiter


The input to the Merlin Arbiter is the Avalon-MM master command packet for all masters requesting access to the a particular slave. The Arbiter outputs the channel number for the selected master. This channel number controls the output of a multiplexer that selects slave device. [Figure 7-11](#) illustrates this logic.

In [Figure 7-11](#), four Avalon-MM masters connect to four Avalon-MM slaves. In each cycle, an arbiter positioned in front of each Avalon-MM slave, selects among the requesting the Avalon-MM masters.

Figure 7-11. Arbitration Logic



If you specified a **Limit interconnect pipeline stages to** parameter greater than zero on the Qsys **Project Settings** tab, the output of the Arbiter is registered. Registering this output reduces the amount of combinational logic between the master and fabric, increasing the f_{MAX} of the system.

 For more information about the **Limit interconnect pipeline stages to** parameter refer to the “Project Settings” section in the *Creating a System with Qsys* chapter in volume 1 of the *Quartus II Handbook*.

Interconnect Pipelining

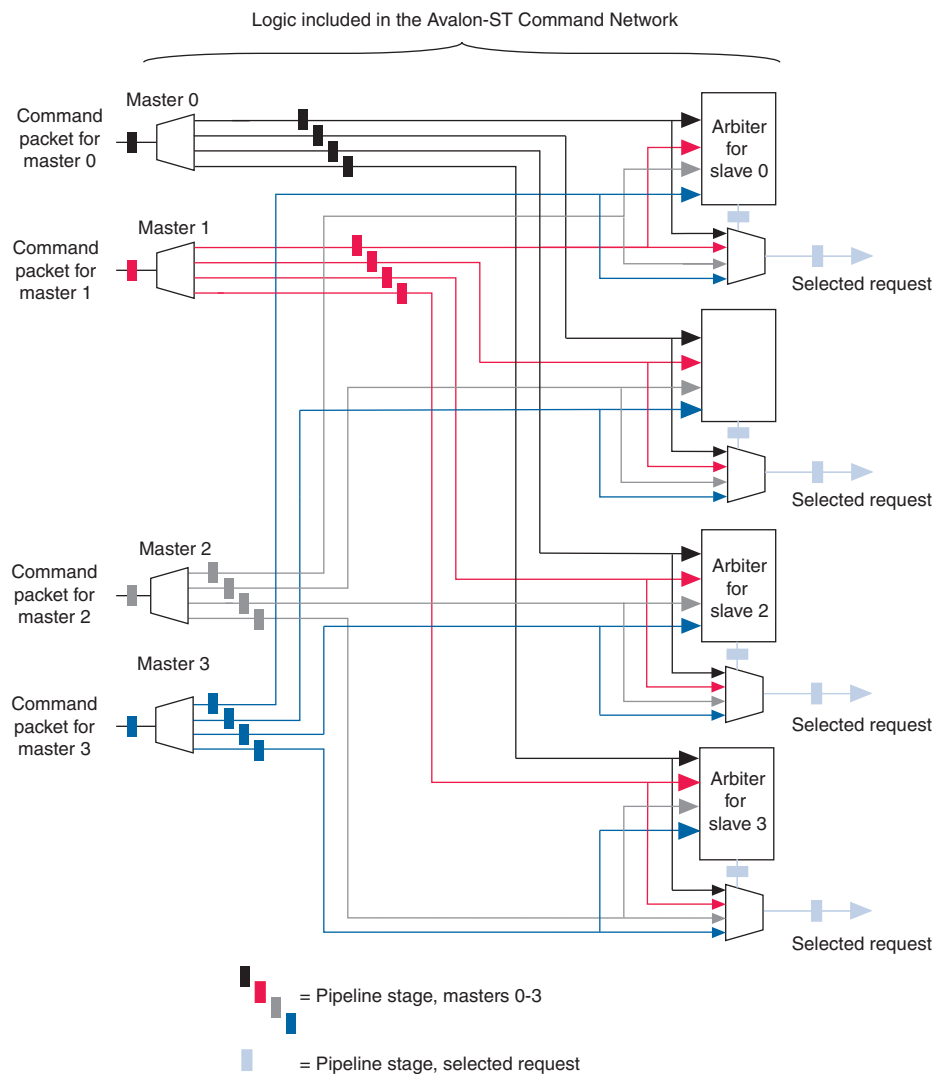
If you set the **Limit interconnect pipeline stages to** parameter to a value greater than 0 on the **Project Settings** tab, Qsys automatically inserts Avalon-ST pipeline stages when you generate your design. The pipeline stages increase the f_{MAX} of your design by reducing the combinational logic depth. The cost is additional latency and logic.

Figure 7-12 shows the placement of up to four potential pipeline stages inserted by Qsys in the following locations:

- Before the input to the demultiplexer
- At the output of the multiplexer
- Between the arbiter and the multiplexer
- At the outputs of the demultiplexer



The insertion of pipeline stages depends upon the existence of certain interconnect components. For example, in a single-slave system, no multiplexer exists; therefore multiplexer pipelining does not occur. In an extreme case, of a single-master to single-slave system, no pipelining occurs, regardless of the value of **Limit interconnect pipeline stages to**.

Figure 7-12. Pipeline Placement in Arbitration Logic

Additional Qsys Interconnect Components

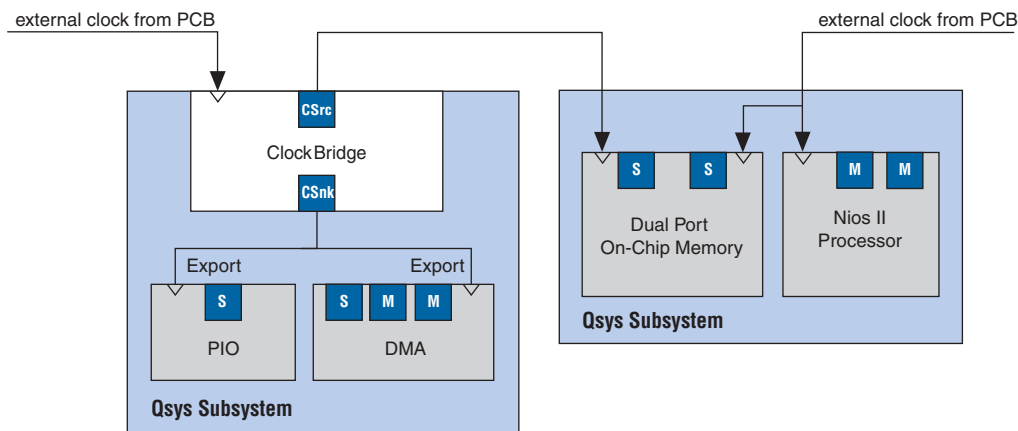
The following sections describe additional components used by the Qsys interconnect. All of these components are in the component library for use in your designs.

- “Clock Bridge” on page 7-15
- “Avalon-MM Clock Crossing Bridge” on page 7-15
- “Avalon-MM Pipeline Bridge” on page 7-15
- “Merlin Width Adapter” on page 7-16

Clock Bridge


The Clock Bridge allows you to route clocks between Qsys subsystems. You can use this bridge to connect a single clock source to the input clocks of multiple Qsys subsystems. Figure 7-13 illustrates the use of this bridge.

Figure 7-13. Clock Bridge



Avalon-MM Clock Crossing Bridge

The Avalon-MM Clock Crossing Bridge transfers Avalon-MM commands and responses between asynchronous clock domains. It uses asynchronous FIFOs to implement the clock crossing logic. The Avalon-MM Clock Crossing Bridge has a number of parameters, including parameters to control the depth of the command and response FIFOs in both the master and slave clock domains. If the number of in-flight reads exceeds the depth of the response FIFO, the Avalon-MM Clock Crossing Bridge stops sending reads. To maintain throughput for high-performance applications, increase the response FIFO depth from the default minimum depth which is twice the maximum burst size.

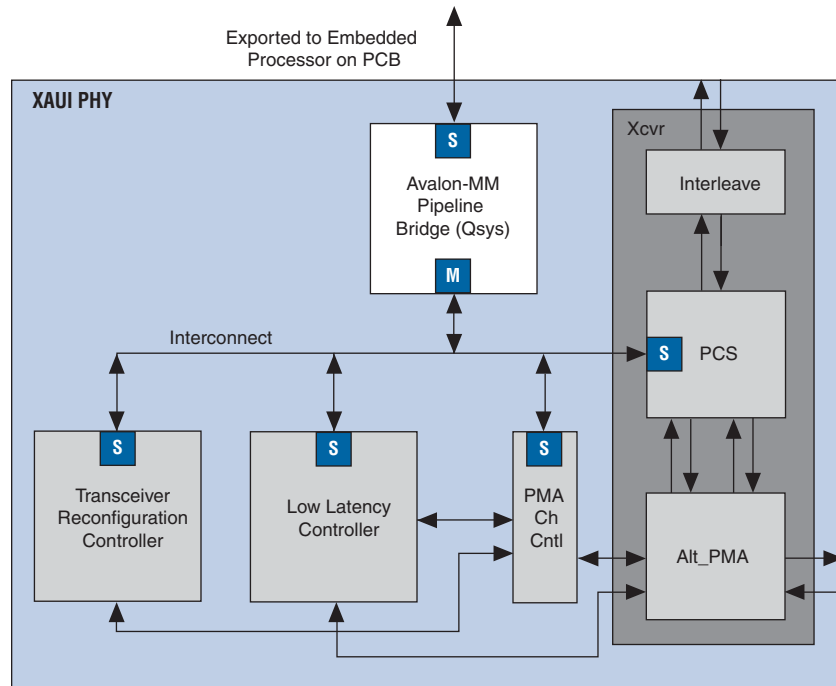
 The Avalon-MM Clock Clocking Bridge core is implemented to work with the Qsys interconnect. The legacy Avalon-MM Clock Crossing Bridge core is available for SOPC Builder systems. If you port an SOPC Builder design that includes the Avalon-MM Clock Crossing Bridge to Qsys, Qsys automatically changes the older version to the Qsys version.

Avalon-MM Pipeline Bridge

The Avalon-MM Pipeline Bridge inserts a register stage in the Avalon-MM command and response paths. It accepts commands on its Avalon-MM slave port and propagates them to its Avalon-MM master port. It provides separate parameters to turn on pipelining in the command and response networks.

Because you can turn the pipelining feature of this bridge off, you can also use the Avalon-MM bridge to export a single Avalon-MM slave interface that can be used to control multiple Avalon-MM slave devices. In this configuration, it transfers commands received on its Avalon-MM slave interface to its Avalon-MM master port. [Figure 7-14](#) illustrates its use.

Figure 7-14. Avalon Bridge



Because the Avalon-MM slave interface is exported to the pins of the device, having a single Avalon-MM slave port, rather than separate ports for each Avalon-MM slave device, reduces the pin count of the FPGA.

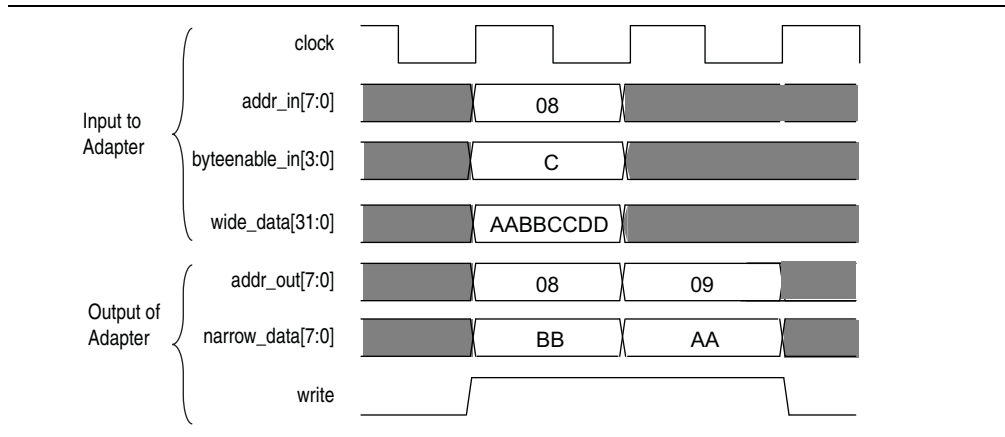
Merlin Width Adapter

The Merlin Width Adapter converts between Avalon-MM master and slaves with different data and byte enable widths. This adapter is used in the Avalon-ST domain and operates with information contained in the packet format illustrated [Figure 7-2 on page 7-4](#). It accepts packets on its sink interface with one data width and produces output packets on its source interface with a different data width. The ratio of the wider data width to the narrower width must be a power of two, such as 4:1, 8:1, and 16:1. This adapter assumes that the field ordering of the input and output packets is the same, with the only difference being the width of the data and accompanying byte enable signals.

When the Width Adapter converts from a wide data to a narrow data, the narrower data is transmitted over several beats. The first output beat contains the lowest addressed segment of the input data and byte enables. [Figure 7-15](#) illustrates the timing for a 4:1 width adapter.

When the Width Adapter converts from narrow data to wide data, each input beat's data and byte enables are copied to the appropriate segment of the wider output data and byte enables signals.

Figure 7-15. Width Adapter Timing for a 4:1 Adapter



Burst Transfers

Avalon-MM burst transactions grant a master uninterrupted access to an Avalon-MM slave for a specified number of transfers. The master specifies the number of transfers when it initiates the burst using the `burstcount` signal. Once a burst begins between a master and slave pair, arbiter logic is locked until the burst completes. For burst masters, the size of the burst is the number of cycles that the master has access to the slave, and the selected arbitration shares have no effect.

Merlin Burst Adapter

The Qsys interconnect uses the Qsys Merlin Burst Adapter to accommodate the burst capabilities of each interface in the system, including interfaces that do not support burst transfers. The maximum burst length for each interface is a property of the component interface and is independent of other interfaces in the system. Therefore, a particular master might be capable of initiating a burst longer than a slave's maximum supported burst length. In this case, the burst adapter translates the large master burst into smaller bursts, or into individual slave transfers if the slave does not support bursting. Until the master completes the burst, the arbiter logic prevents other masters from accessing the target slave. For example, if a master initiates a burst of 16 transfers to a slave with maximum burst length of 8, the burst adapter initiates 2 bursts of length 8 to the slave.

Avalon-MM masters always issue addresses that are aligned to the size of the transfer. However, in some cases, when a narrow-to-wide width adaptation is used, the resulting address may be unaligned. In the case of unaligned addresses, the Burst Adapter issues the maximum possible sized bursts, with appropriate byte enables, to bring the burst-in-progress up to an aligned slave address. Then, it completes the burst on aligned addresses.

Burst Types

The burst adapter supports variable wrap or sequential burst types to accommodate the different properties of the Avalon-MM masters. Refer to [Table 7-1 on page 7-4](#) for definitions of these burst types. Some bursting masters can issue more than one burst type.

Avalon-ST Interfaces

The interconnect for Avalon-ST connects high-bandwidth, low-latency components that use the Avalon-ST interface. This interconnect creates datapaths for unidirectional traffic including multichannel streams, packets, and DSP data. The Avalon-ST interconnect is flexible and can be used to implement on-chip interfaces for industry standard telecommunications and data communications cores, such as Ethernet, Interlaken, and video. In all cases, you can define bus widths, packets, and error conditions.

You specify how Avalon-ST source and sink ports connect in Qsys. If your source and sink interfaces have different properties, selecting **Insert Avalon-ST adapters** on the Tools menu Qsys inserts the necessary adapters which are visible in the **System Contents** tab.

Avalon-ST Examples

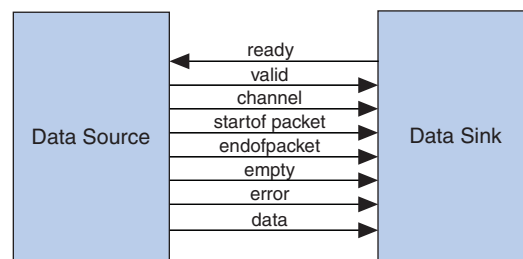
[Figure 7-16](#) illustrates the simplest system example with an Avalon-ST connection between the source and sink. This source-sink pair includes only the data signal. The sink must be able to receive data as soon as the source interface comes out of reset.

Figure 7-16. Interconnect for a Simple Avalon Streaming Source-Sink Pair




[Figure 7-17](#) illustrates a more extensive interface that includes signals indicating the start and end of packets, channel numbers, error conditions, and back pressure.

Figure 7-17. Avalon Streaming Interface for Packet Data



All data transfers using Avalon-ST interconnect occur synchronously to the rising edge of the associated clock interface. Throughput and frequency of a system depends on the components and how they are connected.

 For details about the Avalon-ST interface protocol, refer to the *Avalon Interface Specification*.

Avalon-ST Components

The Qsys component library includes a number of Avalon-ST components that you can use to create datapaths, including datapaths whose input and output streams have different properties. Generated systems that include Avalon-MM master and slave components may also use these Avalon-ST components because the generation process creates an interconnect whose structure resembles a network topology as “Qsys Transformations” on page 7-6 describes. The following sections introduce the Avalon-ST components.


Avalon-ST Handshake Clock Crosser

The Avalon-ST Handshake Clock Crossing adapter connects streams that operate at different frequencies. This adapter uses a simple hand-shaking protocol to propagate transfer control signals and responses across the clock boundary and responses in the other direction. This methodology uses fewer FPGA resources because each transfer is safely propagated to the target domain before the next transfer can begin. The Avalon-ST Handshake Clock Crosser is appropriate for low throughput connections because the handshake incurs at least four cycles of round-trip latency for every read command, limiting performance.

You can use the parameter editor for the Avalon-ST Handshake Clock Crosser to specify parameter values. Among the parameters that you can specify are the data width, whether or not to include packet support, and synchronizer depths.

Avalon-ST Pipeline Stage

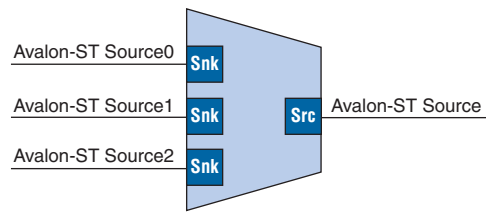
The Avalon-ST pipeline stage optionally inserts a single pipeline (register) stage in the Avalon-ST command and response datapaths. It receives data on its Avalon-ST sink interfaces and drives it unchanged on its Avalon-ST source interface.

 The Qsys component library also includes an Avalon-MM Pipeline Bridge whose data interfaces use the Avalon-MM protocol, rather than the Avalon-ST protocol.

Merlin Multiplexer

The Merlin Multiplexer accepts data on its Avalon-ST sink interface and multiplexes the data for transmission on its Avalon-ST source interface. You can parameterize the multiplexer to append channel information on the source to indicate which sink is driving the source data. The multiplexer includes internal arbitration logic which selects between inputs using a round-robin arbitration algorithm. [Figure 7-18](#) illustrates the Avalon-ST multiplexer. Among the parameters that you can specify are the option to use packet scheduling, which guarantees that the multiplexer only changes inputs at the end of a packet.

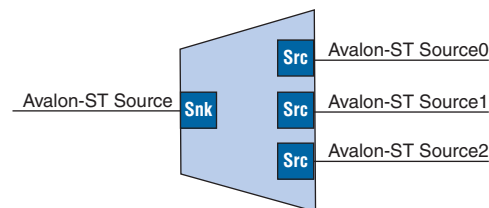
Figure 7-18. Merlin Multiplexer



Merlin Demultiplexer

The Merlin Demultiplexer accepts channelized data on its sink interface, and transmits the data on one of its source interfaces. The channel bits of the source stream indicate which port the drives the output data. [Figure 7-19](#) illustrates the Merlin Multiplexer. Among the parameters that you can specify are the number of output ports and the width of the channel signal.

Figure 7-19. Avalon-ST Demultiplexer



Avalon-ST and Avalon-MM Interfaces

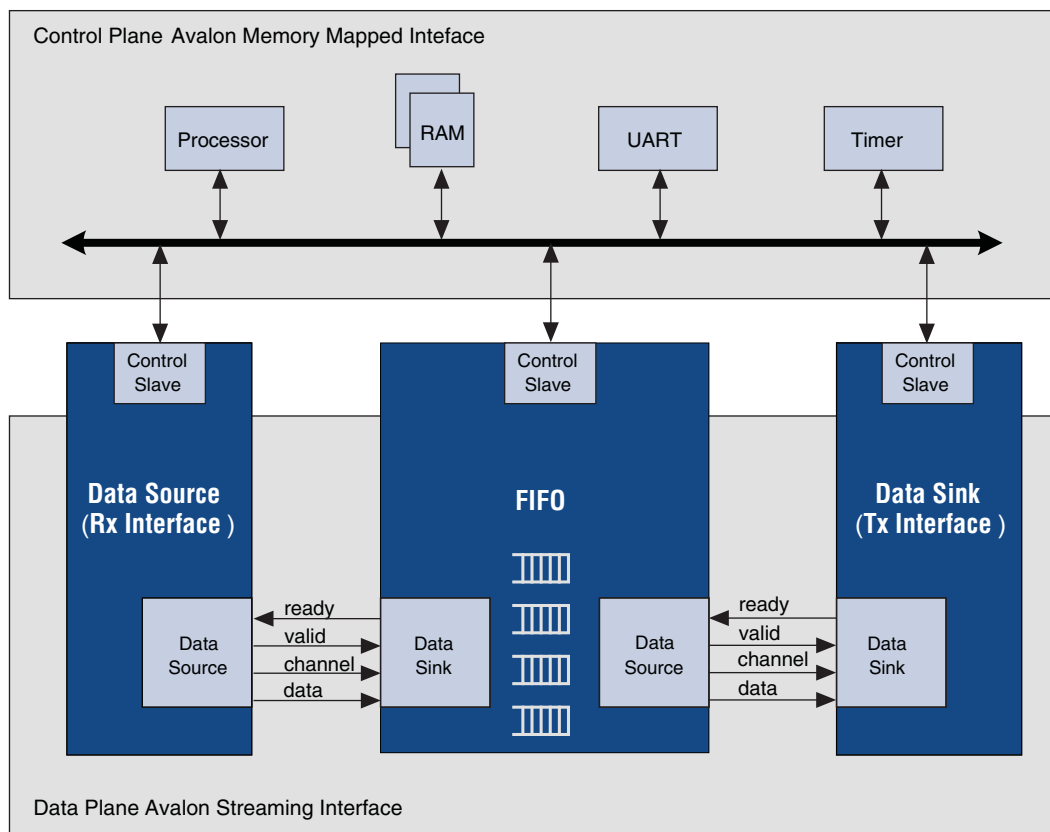
The Avalon-ST and Avalon-MM interfaces are complementary. High bandwidth components with streaming data typically use Avalon-ST interfaces for the high throughput datapath. These components can also use Avalon-MM connection interfaces to provide an access point for control. In contrast to the Avalon-MM interconnect, which can be used to create a wide variety of topologies, the Avalon-ST interconnect fabric always creates a point-to-point between a single data source and data sink, as [Figure 7-20](#) illustrates.

There are two connection pairs in this figure:

- The data source in the Rx Interface transfers data to the data sink in the FIFO.
- The data source in the FIFO transfers data to the Tx Interface data sink.

In [Figure 7-20](#), the Avalon-MM interface allows a processor to access the data source, FIFO or data sink to provide system control.

Figure 7-20. Use of the Avalon-MM Avalon-ST Interfaces



Tri-state Conduit Components

The Avalon-TC interface type allows you to design Qsys subsystems that connect to tri-state devices on your PCB. The following three components implement the tri-state conduit functionality:

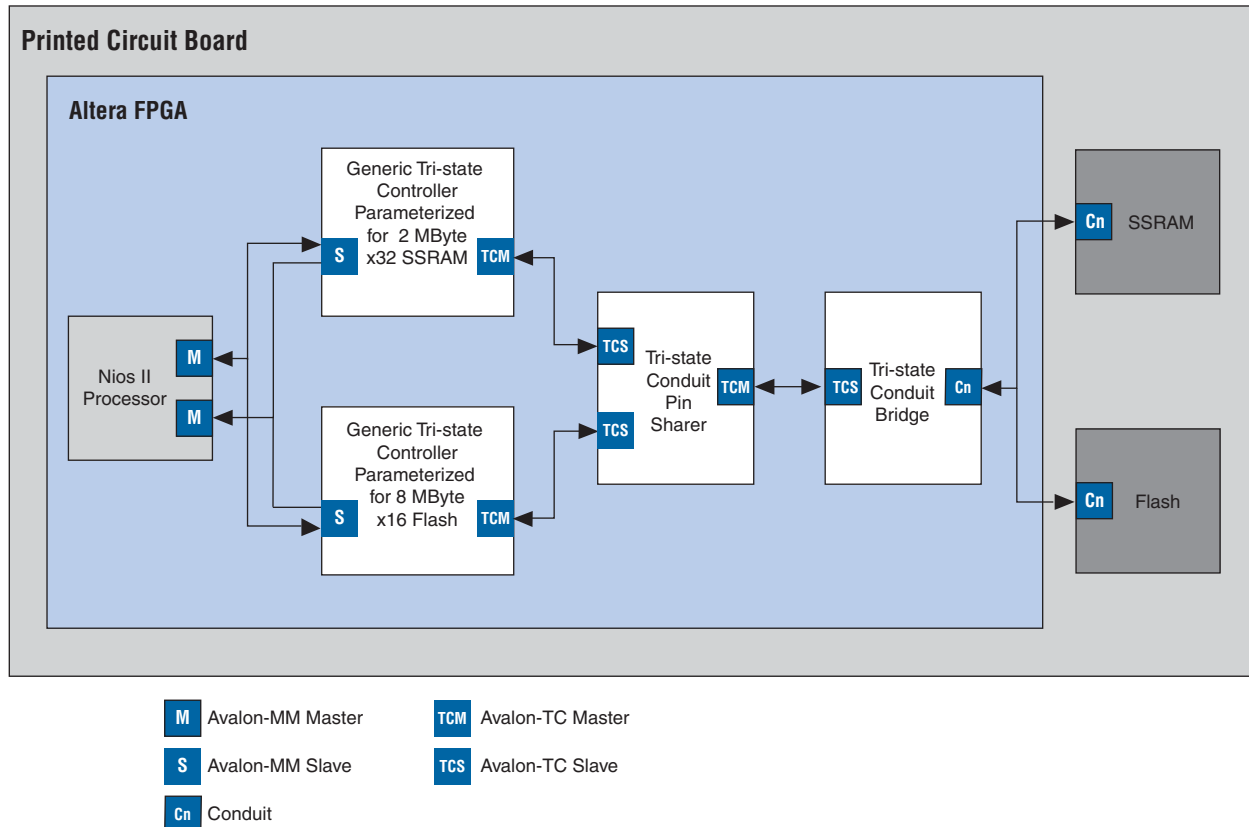
- [Generic Tri-state Controller](#)
- [Tri-state Conduit Pin Sharer](#)
- [Tri-state Conduit Bridge](#)

You can use these components to implement pin sharing, convert between unidirectional and bidirectional signals, and create tri-state controllers for devices whose interfaces can be described using the Avalon-TC signal types.

 For more information about the Avalon-TC signal types, refer to the *Avalon Tri-state Conduit Interfaces* chapter in the *Avalon Interface Specifications*.

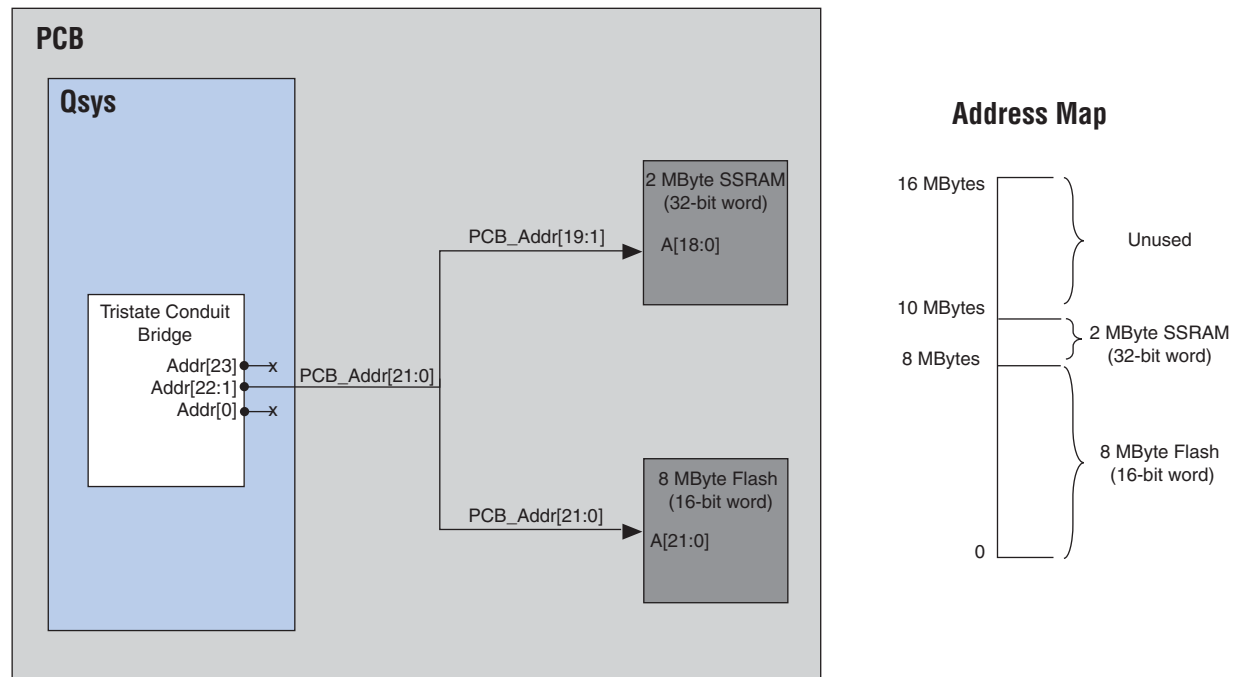
Figure 7-21 illustrates the typical use of these components. This figure includes two Generic Tri-state Conduit Controllers. The first is customized to control a flash memory. The second is customized to control an off-chip SSRAM. The Tri-state Conduit Pin Sharer multiplexes between these two controllers, and the Tri-state Conduit Bridge converts between an on-chip encoding of tri-state signals and true bidirectional signals.

Figure 7-21. Tri-state Conduit System to Control Off-Chip SRAM and Flash Devices



By default, the Tri-state Conduit Pin Sharer and Tri-State Conduit Bridge present byte addresses. Each address location in many memory devices contains more than one byte of data. In the example presented in Figure 7-21, the flash device operates on 16-bit words and must ignore the least-significant bit of the Avalon-MM address. The SSRAM memory operates on 32-bit words and must ignore the two, low-order memory bits. Because neither device requires a byte address, `addr[0]` is not routed on the PCB. Figure 7-22 shows `addr[0]` as a unconnected.

Figure 7-22. Address Connections from Qsys System to PCB



In this example design, the flash device responds to address range 0 MBytes–8 MBytes-1. The SSRAM responds to address range 8 MBytes–10 MBytes-1. The PCB schematic for the PCB connects `addr[20:2]` to `addr[18:0]` of the SSRAM device because the SSRAM responds to 32-bit word address. The 8 MByte flash device accesses 16-bit words; consequently, the schematic does not connect `addr[0]`. The chipselect signals select between the two devices.


 If you create a custom tri-state conduit master with word-aligned addresses, the Tri-state Conduit Pin Sharer does nothing to change or align the address signals.

Figure 7-23 illustrates this example system in Qsys.

Figure 7-23. Tri-state Conduit System in Qsys

System Contents		Address Map	Clock Settings	Project Settings	System Inspector	HDL Example	Generation
Use	Connections	Name	Description	Export			
<input checked="" type="checkbox"/>		nios2_qsys_0	Nios II Processor				
		data_master	Avalon Memory Mapped Master	Click to export			
		instruction_master	Avalon Memory Mapped Master	Click to export			
<input checked="" type="checkbox"/>		flash_controller	Generic Tristate Controller				
		uas	Avalon Memory Mapped Slave	Click to export			
		tcm	Tristate Conduit Master	Click to export			
<input checked="" type="checkbox"/>		SSRAM_controller	Generic Tristate Controller				
		uas	Avalon Memory Mapped Slave	Click to export			
		tcm	Tristate Conduit Master	Click to export			
<input checked="" type="checkbox"/>		tristate_conduit_pin_sharer_0	Tristate Conduit Pin Sharer				
	tcm	Tristate Conduit Master	Click to export				
	tcs0	Tristate Conduit Slave	Click to export				
	tcs1	Tristate Conduit Slave	Click to export				
<input checked="" type="checkbox"/>	tristate_conduit_bridge_0	Tristate Conduit Bridge					
	tcs	Tristate Conduit Slave	Click to export				
	out	Conduit	Click to export	tristate_conduit_out			

Generic Tri-state Controller

The Generic Tri-state Controller provides a template for a controller that you can parameterize to reflect the behavior of an off-chip device. The Generic Tri-state Controller has many parameters that you can use to customize this component such as the following examples:

- The width of the address and data signals
- The read and write wait times
- The bus-turnaround time



In calculating delays, the Generic Tri-state Controller chooses the larger of the bus-turnaround time and read latency. Turnaround time is measured from the time that a command is accepted, not from the time that the previous read returned data.

- The data hold time

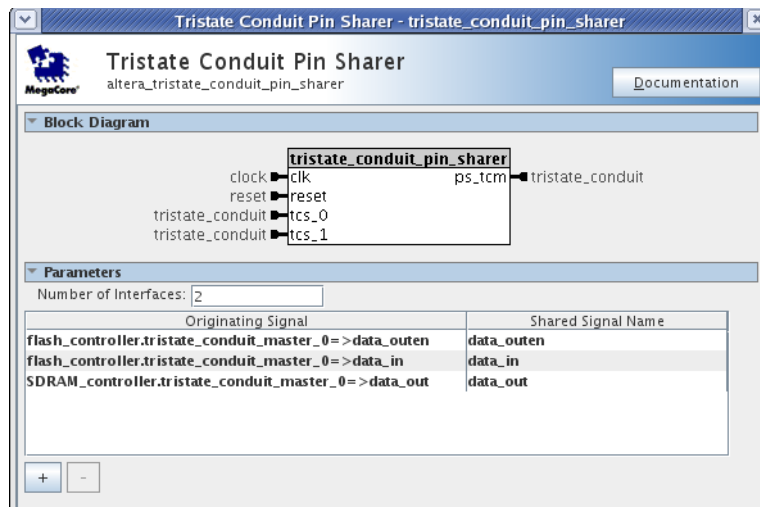
The Generic Tri-state Controller always includes the following interfaces:

- Avalon-MM slave interface—This interface connects to an Avalon-MM master, such as a Nios II processor.
- Avalon-TC master interface usually connects to the tri-state conduit slave interface of the tri-state conduit pin sharer.
- Avalon Clock sink—The component's clock reference. This interface must be connected to a clock source.
- Avalon Resets sink—This interface connects to a reset source interface.


Tri-state Conduit Pin Sharer

The Tri-state Conduit Pin Sharer multiplexes between the signals of the connected tri-state controllers. You connect all signals from the tri-state controllers to the Tri-state Conduit Pin Sharer and use the parameter editor to specify the signals that are shared. The parameter editor includes a Shared Signal Name column for you to type the shared signal name as [Figure 7-24](#) illustrates.

Figure 7-24. Specifying Shared Signals Using the Tri-state Conduit Pin Sharer



If the widths of shared signals differ, the signals are aligned on their 0th bit and the higher-order pins are driven to 0 whenever the smaller signal has control of the bus. Unshared signals always propagate through the pin sharer. The tri-state conduit pin sharer uses the round-robin arbiter that is described in [“Arbitration”](#) on page 7-10 to select between tri-state conduit controllers.

 All tri-state conduit components connected to a given pin sharer must be in the same clock domain.

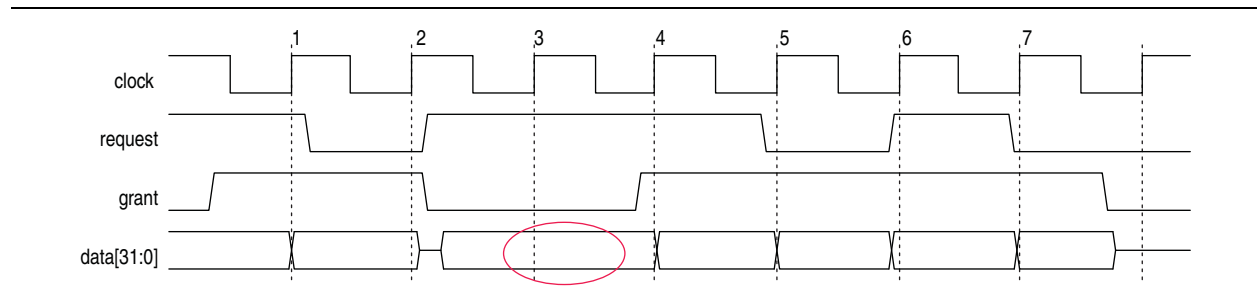
Tri-state Conduit Bridge

The Tri-state Conduit Bridge instantiates bidirectional signals for each tri-state signal while passing all other signals straight through the component. The Tri-state Conduit Bridge registers all outgoing and incoming signals, which adds two cycles of latency for a read request. You must account for this additional pipelining when designing a custom controller. During reset, all outputs are placed in a high-impedance state; outputs are enabled in the first clock cycle after reset is deasserted. The Quartus II software labels these output signals bidirectional.

Timing

[Figure 7-25](#) illustrates the arbitration timing. As this figure illustrates, a device can drive valid data in the granted cycle. [Figure 7-25](#) shows the following sequence of events:

1. In cycle one, the arbiter grants a request. The granted device drives valid data in cycles one and two.
2. In cycle 4, the arbiter grants a request. The granted device drives valid data in cycles 4 and 5.
3. In cycle 6, the arbiter grants a request. The granted device drives valid data in cycles 6 and 7.
4. Cycle 3 is the only cycle that does not contain valid data.

Figure 7-25. Arbitration Timing

Interrupt Interfaces

In systems with interrupt sender interfaces, the Qsys interconnect includes several components to implement interrupt handling. Qsys handles individual, single-bit interrupt requests (IRQs). In the event that multiple senders assert their IRQs simultaneously, the receiver logic (typically under software control) determines which IRQ has highest priority, then responds appropriately.

Using individual requests, the interrupt logic can handle up to 32 IRQ inputs connected to each interrupt receiver. With this logic, the interrupt sender connected to `interrupt_receiver_0` is the highest priority with sequential receivers being successively lower priority. You can redefine the priority of interrupt senders by instantiating the Merlin IRQ mapper component. For more information refer to the “Merlin IRQ Mapper” on page 7-27.

Assigning IRQs in Qsys

You assign IRQ connections on the **System Contents** tab of Qsys. After adding all components to the system, you connect interrupt senders and receivers. You can use the **IRQ** column to specify an IRQ number with respect to each receiver or specify not to connect the IRQ.

- ❓ For more information, refer to *Connecting Qsys Components* in Quartus II Help.

Qsys uses the following three components to implement interrupt handling:

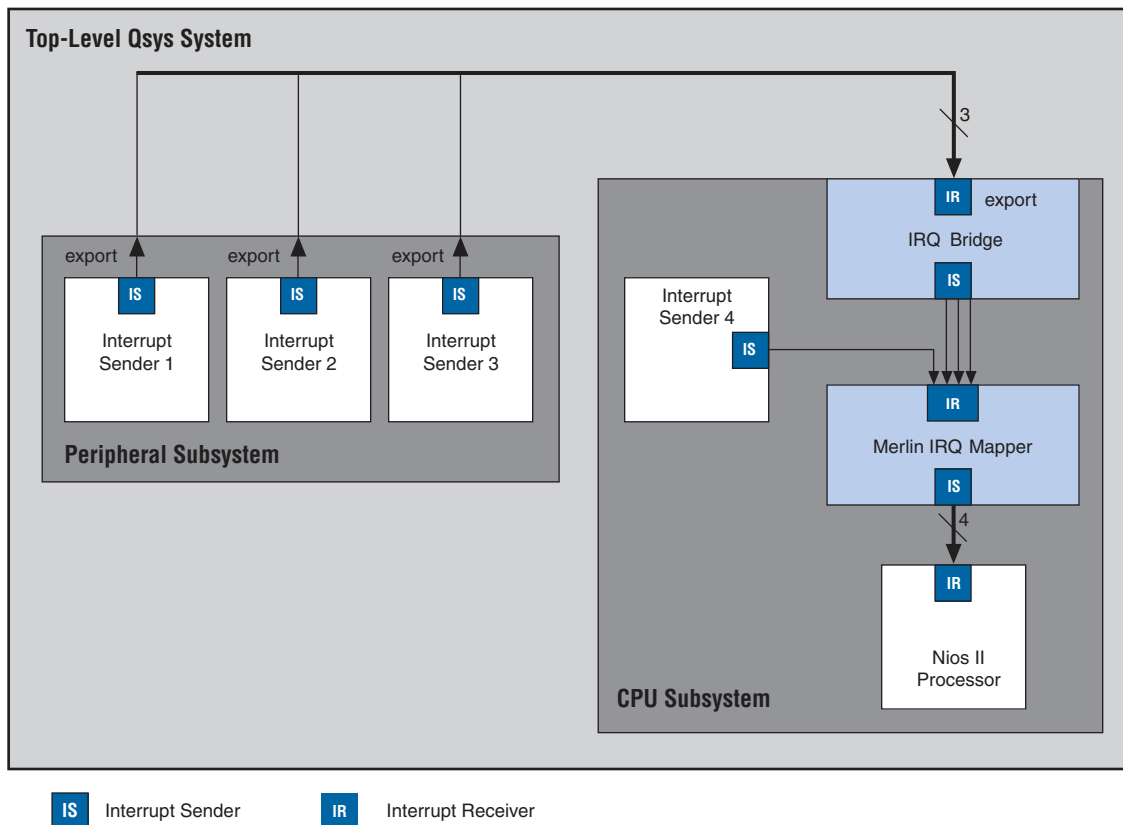
- [IRQ Bridge](#)
- [Merlin IRQ Mapper](#)
- [Merlin IRQ Clock Crosser](#)

The following sections describe these components.

IRQ Bridge

The IRQ Bridge allows you to route interrupt wires between Qsys subsystems. In [Figure 7-26](#), the Peripheral Subsystem has three interrupt senders that are exported to the top level of the subsystem. These interrupts are routed to the Merlin IRQ receiver bridge in the CPU Subsystem.

Figure 7-26. Qsys IRQ Bridge Application



Merlin IRQ Mapper

The Merlin IRQ Mapper converts individual interrupt wires to a bus. In addition, you can use the IRQ Mapper to specify the interrupt number. By default, the interrupt sender connected to the receiver0 interface of the IRQ mapper is highest priority with sequential receivers being successively lower priority. You can use the **IRQ Map** parameter in the parameter editor to remap the priority. For example, to reverse the priority of the four interrupt senders connected to the IRQ mapper in [Figure 7-26](#), you can type the following string for the **IRQ Map** parameter, 0:3, 1:2, 2:1, 0:3.

Merlin IRQ Clock Crosser

The Merlin IRQ Clock Crosser synchronizes interrupt senders and receivers that are in different clock domains. To use this component, connect the clocks for both the interrupt sender and receiver in addition to the interrupt sender and receiver interfaces. Qsys automatically inserts this component when it is required.

Clock Interfaces

You can use the **Clock Settings** tab to define external clock sources, for example an oscillator on your board. You can define separate reset sources for each clock domain, a single reset source for all clocks, or any combination in between.

Reset Interfaces

You can choose to have a single global reset domain generated by Qsys by selecting **Auto-Connect Resets** on the Tools menu. If your design requires more than one reset domain, you can implement your own reset logic and connectivity.

Single Global Reset Signal Implemented by Qsys

If you select **Auto-Connect Resets** on the Tools menu, the Qsys interconnect distributes a global reset bus. All of the reset requests are ORed together, synchronized to each clock domain, and fed to the reset inputs. The duration of the reset signal is at least one clock period.

The Qsys interconnect inserts the system-wide reset under the following conditions:

- The global reset input to the Qsys system is asserted.
- Any component asserts its reset request signal.

Multiple Reset Signals

The Qsys component library includes a reset controller and a reset bridge to implement the reset functionality. You can also design your own reset logic.



If you design your own reset circuitry you must carefully consider situations which might result in system lockup. For example, if an Avalon-MM slave is reset in the middle of a transaction, the Avalon-MM master might wait forever.

Merlin Reset Controller

If you design a system with multiple reset inputs, the Merlin Reset Controller ORs all reset inputs and generates a single reset output. The Reset Controller has the following three parameters which you can specify to customize its behavior.

- **Number of inputs**—Indicates the number of individual reset interfaces the controller ORs to create a signal reset output.
- **Output reset synchronous edges**—specifies the level of synchronization. You can select one of the following options:
 - **None**—The reset is asserted and deasserted asynchronously. You can use this setting if you have designed internal synchronization circuitry.
 - **Both**—The reset is asserted and deasserted synchronously.
 - **Deassert**—The reset is deasserted synchronously and asserted asynchronously.
- **Synchronization depth**—Specifies the number of register stages the synchronizer uses to eliminate the propagation of metastable events.

Qsys automatically inserts reset synchronizers under the following conditions:

- More than one reset source is connected to a reset sink
- There is a mismatch between the reset source's synchronous edges and the reset sinks' synchronous edges

Reset Bridge


The Reset Bridge allows you to use a reset signal in two or more subsystems of your Qsys system. You can connect one reset source to local components and export one or more to other subsystems as required. You to specify the number of reset outputs using the parameter editor.

Avalon Conduits

You can use the Avalon Conduit interface type for interfaces that do not fit any of the other Avalon interface types. You can use Avalon Conduit interfaces to group any arbitrary collection of signals. Like other interface types, you can export or connect conduit interfaces. The PCI Express link of the PCI Express IP core shown in [Figure 5-11 on page 5-22](#) is an example of the use of the conduit interface for export.

To connect two conduit interfaces inside Qsys, the following conditions must be met:

- The interfaces must match exactly with the same signal roles and widths.
- The interfaces must be the opposite directions.

 Conduits connections are always point-to-point connections.

 For more information about the Avalon Conduit interface, refer to the [Avalon Interface Specifications](#).

Summary: Qsys Interconnect Components

[Table 7-3](#) lists all of the Qsys components that implement the Qsys interconnect.

Table 7-3. Summary of Qsys Interconnect Components (Part 1 of 2)

Component Name	Typical Applications	
	Internal Qsys interconnect ⁽¹⁾	User Designs
Avalon-MM Master and Slave Network Transformation		
Merlin Master Translator	✓	—
Merlin Master Agent	✓	—
Merlin Router	✓	—
Merlin Traffic Limiter	✓	—
Merlin Slave Translator	✓	—
Merlin Slave Agent	✓	—
Avalon-ST Components		
Avalon-ST Handshake Clock Crosser	✓	✓
Avalon-ST Pipeline Stage	✓	✓

Table 7-3. Summary of Qsys Interconnect Components (Part 2 of 2)

Component Name	Typical Applications	
	Internal Qsys interconnect ⁽¹⁾	User Designs
Merlin Multiplexer	✓	✓
Merlin Demultiplexer	✓	✓
Bridges		
Clock Bridge	—	✓
Avalon-MM Clock Crossing Bridge	—	✓
Avalon-MM Pipeline Bridge	—	✓
Arbitration and Adapters		
Merlin Arbiter	✓	—
Merlin Width Adapter	✓	✓
Merlin Burst Adapter	✓	✓
Tri-state Conduits		
Generic Tri-state Controller	—	✓
Tri-state Conduit Pin Sharer	—	✓
Tri-state Conduit Bridge	—	✓
Interrupts		
IRQ Bridge	—	✓
Merlin IRQ Mapper	✓	—
Merlin IRQ Clock Crosser	✓	✓
Reset		
Merlin Reset Controller	✓	✓
Reset Bridge	—	✓

Note to Table 7-3:

- (1) These components are described to enhance your understanding of the Qsys interconnect. You probably will not need to use them in your own designs.
- (2) In this table, a ✓ means that the component is typically used for the purpose specified by the column header, a — means that the component is not typically used for the purpose specified by the column header.

Document Revision History

Table 7-4 shows the revision history for this document.

Table 7-4. Document Revision History

Date	Version	Changes
November 2011	11.0.1	Template update.
May 2011	11.0.0	Removed beta status.
December 2010	10.1.0	Initial release.



For previous versions of the Quartus II Handbook, refer to the [Quartus II Handbook Archive](#).

 Take an [online survey](#) to provide feedback about this handbook chapter.

