

This document describes the advantages of network on a chip (NoC) architecture in Altera® FPGA system design. NoC architectures apply networking techniques and technology to communications subsystems in system on a chip designs. NoC interconnect architectures have significant advantages over traditional, non-NoC interconnects, such as support for independent layer design and optimization. Altera's Qsys system integration tool, included with the Quartus® II software, generates a flexible FPGA-optimized NoC implementation automatically, based on the requirements of the application. The Qsys interconnect also provides a higher operating frequency for comparable latency and resource characteristics, with up to a 2X improvement in f_{MAX} compared to traditional interconnects.

Introduction

As FPGA device density increases to more than a million logic elements (LEs), design teams require larger and more complex systems, with increasing performance requirements, in less time. Designers can use system-level design tools to quickly design high-performance systems with a minimum of effort.

Qsys uses a NoC architecture to implement system transactions. The Qsys interconnect includes features that support high-performance operation on FPGAs, including a flexible network interconnect that implements only the minimum resources required for a given application, a packet format that varies depending on the system being supported, and a network topology that separates command and response networks for higher concurrency and lower resource utilization.

This white paper explains the Qsys network implementation, discusses its benefits, and compares the performance results between traditional and Qsys interconnect systems. These results show that the NoC implementation provides higher frequency performance with the same latency characteristics, and can provide up to twice the frequency when pipelining options are enabled.

Understanding NoC Interconnect

The NoC interconnect breaks the problem of communication between entities into smaller problems, such as how to transport transactions between nodes in the system, and how to encapsulate transactions into packets for transport. The NoC interconnect is different from traditional interconnects in one simple, but powerful way. Instead of treating the interconnect as a monolithic component of the system, the NoC approach treats the interconnect as a protocol stack, where different layers implement different functions of the interconnect. The power of traditional protocol stacks, such as TCP-over-IP-over-Ethernet, is that the information at each layer is encapsulated by the layer below it. The power of the Qsys NoC implementation comes from the same source, the encapsulation of information at each layer of the protocol stack.



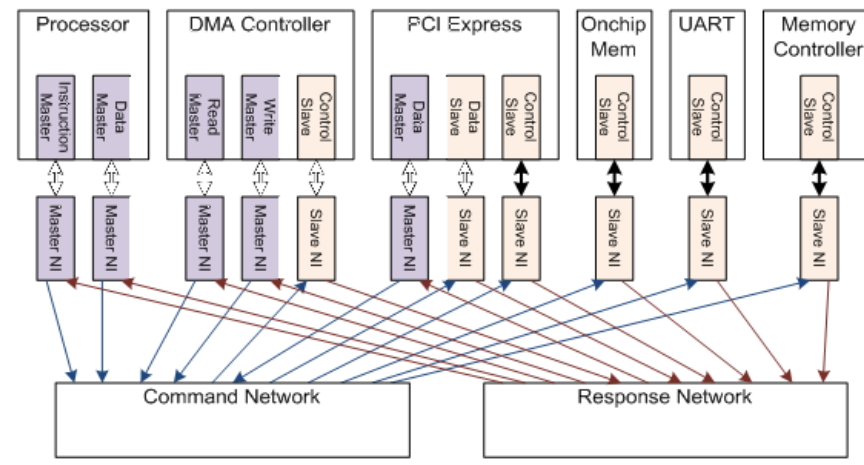
101 Innovation Drive
San Jose, CA 95134
www.altera.com

© 2011 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX are Reg. U.S. Pat. & Tm. Off. and/or trademarks of Altera Corporation in the U.S. and other countries. All other trademarks and service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



Figure 1 shows the basic topology of an NoC system. Each endpoint interface in the network, master or slave, is connected to a network interface (NI) component. The network interface captures the transaction or response using the transaction layer protocol, and delivers it to the network as a packet of the appropriate format. The packet network delivers packets to the appropriate packet endpoints, which then pass them to other network interfaces. The network interfaces then terminate the packet and deliver the command or response to the master or slave using the transaction layer protocol.

Figure 1. NoC System Basic Topology



In this system, a component such as a processor communicates with a component such as a memory controller. Each of these components uses the services of the network interfaces to communicate with one another via a transaction interface, such as Altera's Avalon[®] Memory-Mapped (Avalon-MM) interface or Advanced eXtensible Interface (AXI). The network interfaces communicate with one another to provide transaction layer services by relying on the services of the command and response networks, which provide transport services. Each component at the transport layer (within the command and response networks) recognizes the transport layer protocol, but does not need to recognize the particulars of the transactions in each packet.

Benefits of NoC Architecture

Decoupling the layers of the protocol stack has the following benefits over a traditional approach, such as advanced high performance bus (AHB) or CoreConnect:

- Independent implementation and optimization of layers
- Simplified customization per application
- Supports multiple topologies and options for different parts of the network
- Simplified feature development, interface interoperability, and scalability

Implement and Optimize Layers

A common approach to complex engineering challenges is to divide the design problem into smaller problems with well-defined interactions. With NoC interconnect, the design problem is no longer “How do I best design a flexible interconnect for a complex system?” but instead consists of the easier questions: “How do I best map transactions to packets?” and “How do I best transport packets?” Keeping the layers separate also allows you to optimize the implementation of each layer independently, resulting in better performance at that layer without having to redesign other layers. For example, designers can consider and implement a number of different transport layer topologies and implementations without having to change anything at the transaction layer.

Simplify Customization per Application

At the transport layer, commands and responses are simply packets carried by the network, and anything done at the network layer must only support the transport of these packets. This simplifies the customization of the interconnect for a given application compared to a traditional interconnect. For example, if the designer determines that the system needs pipelining or clock crossing between a set of masters and a set of slaves, the designer can add the needed components as long as they safely transport packets. The clock crossing and pipelining decisions do not need to consider the transaction layer responsibilities, such as the different transaction types, response types, and burst types.

Use Multiple Topologies and Options

NoC interconnect supports use of different optimizations and topologies for different parts of the network. For example, a design may have a set of high-frequency, high-throughput components, such as processors, PCI Express® interfaces, a DMA controller, and memory; and a second set of low-throughput peripherals such as timers, UARTs, flash memory controllers, and I²C interfaces. Such a system can be divided at the transport layer. The designer can place the high-performance components on a wide, high-frequency packet network; while the peripherals are on a less-expensive mesh network, with only a packet bridge between the two networks.

Simplify Feature Development

Interconnects must be versatile enough to support emerging new features, such as new transaction types or burst modes. If the interconnect is divided into different layers, then the addition of new features requires changes only to the layer that supports the feature. To support new burst modes, for example, only the network interface components require modification. Likewise, if a new network topology or transport technology yields higher performance, it can be substituted for the original network without requiring redesign of the entire network.

Interface Interoperability

Different intellectual property (IP) cores support different interface types, such as AMBA® AXI, AHB, and APB interfaces; as well as OCP interfaces, Wishbone interfaces, and Avalon-MM interfaces. Supporting a new interface requires implementing only the network interface to encapsulate transactions to or from interfaces of that type using the selected packet format. With this architecture, a bridge component is not needed, saving logic and latency.

Scalability

Systems with hundreds of masters and slaves are not uncommon, and traditional interconnects struggle to meet the required performance. Interconnects designed for dozens of masters and slaves cannot easily scale to support hundreds of components required by systems today. With NoC interconnect, it is relatively easy to divide the network into subnetworks, with bridges, pipeline stages, and clock-crossing logic throughout the network as required. Therefore, a multi-hop network could easily support thousands of nodes, and could even provide for a transport network spanning multiple FPGAs.

NoC System Design with Qsys

Qsys is a powerful system integration tool included as part of Altera's Quartus II development software. Qsys simplifies FPGA system design, allowing designers to create a high-performance system easily, without extensive knowledge of on-chip interconnects or networks. Qsys includes an extensive IP library from which designers can build and implement a system on a chip (SoC) in much less time than using traditional, manual integration methods. Using traditional design methods, designers write HDL modules to connect components of the system. Using Qsys, designers instantiate and parameterize system components using a GUI or a scripted system description. Qsys then generates the components and interconnect at the press of a button. Figure 2 shows an example system created in Qsys.

Figure 2. Example System Components Displayed in Qsys

System Contents		Address Map	Clock Settings	Project Settings	System Inspector	HDL Example	Generation
Use	Connections	Name	Description		Export		
<input checked="" type="checkbox"/>		[-] nios_2_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		tristate_conduit_out			

In Qsys, the system designer uses the GUI to add the desired IP components to the system, parameterize each component, and specify interface-level connections between system components. Qsys connects individual signals within connected interfaces automatically. Qsys generates the system implementation as RTL, and manages system interconnect issues such as clock domain crossing, interface width adaptation, and burst adaptation.

Qsys supports a number of different interface types, such as transaction (read and write) interfaces, streaming (packets or non-packet) interfaces, interrupts, and resets. The Qsys transaction interconnect is based on a NoC implementation that is designed specifically for FPGAs. The Qsys interconnect minimizes the use of FPGA resources, while at the same time supporting high-performance systems with high frequency and throughput requirements.

Qsys NoC Interconnect Optimized for FPGAs

The Qsys NoC interconnect has features that make it particularly well-suited to FPGAs and the systems that use them, including the minimum flexible implementation, parameterizable packet format designed to reduce adaptation, low-latency interconnect, and separate command and response networks.

Minimum, Flexible Implementation

The Qsys interconnect is not just aimed at large high-performance systems with multi-gigabit datapaths and complex bursting, it is also intended for small systems of only a few components. To support such a wide variety of systems, Qsys implements only the minimum interconnect required to meet the performance requirements for a given application.

Qsys begins by dividing the system into multiple interconnect domains. Two interfaces are in different interconnect domains if there are no connections in the system that require the system algorithm to consider them together. For example, if one master connects to two slaves, those slaves are in the same interconnect domain. For each domain, Qsys considers all the master and slave widths, and sets the network data width to be the minimum that supports full throughput for the highest throughput connection in the system, based on the clock rates of the interfaces in the domain.

In addition, Qsys adds only the interconnect components that are required for the application. For example, if there is a master in the system that is only connected to one slave, then the address decoder component is omitted. If there is a slave that is only connected to one master, then the arbiter component is omitted. If a certain type of burst adaptation is not required by that application, then support for that burst adaptation is omitted.

Parameterizable Packet Format Reduces Adaptation

In addition to minimizing interconnect resource use, Qsys determines the packet format that minimizes logic use and adaptation. For example, the address and burstcount fields in the packet are the minimum width required to support the system. The address and other fields within the packet are driven to useful and accurate values in all cycles of the packet, so the adaptation components do not have to maintain any state about the packet, and even allow the adapter to be omitted altogether in some cases.

Low-Latency Interconnect

Designers commonly associate packets with serialization, thinking that with a packet-based approach, only a portion of the entire transaction is carried in each cycle. Many NoC implementations use this approach. Such NoC implementations have a network latency on the order of 12 to 15 clock cycles, making them inappropriate for the interconnect between a microcontroller and its local memory, for example. To overcome latency issues, the components in the Qsys interconnect all have combinational datapaths. The packet format is wide enough to contain a complete transaction in a single clock cycle, so that the entire interconnect can support writes with 0 cycles of latency and reads with round-trip latency of 1 cycle. These wide connections are well supported by today's FPGAs. The system designer can change pipelining options to increase frequency at the expense of latency.

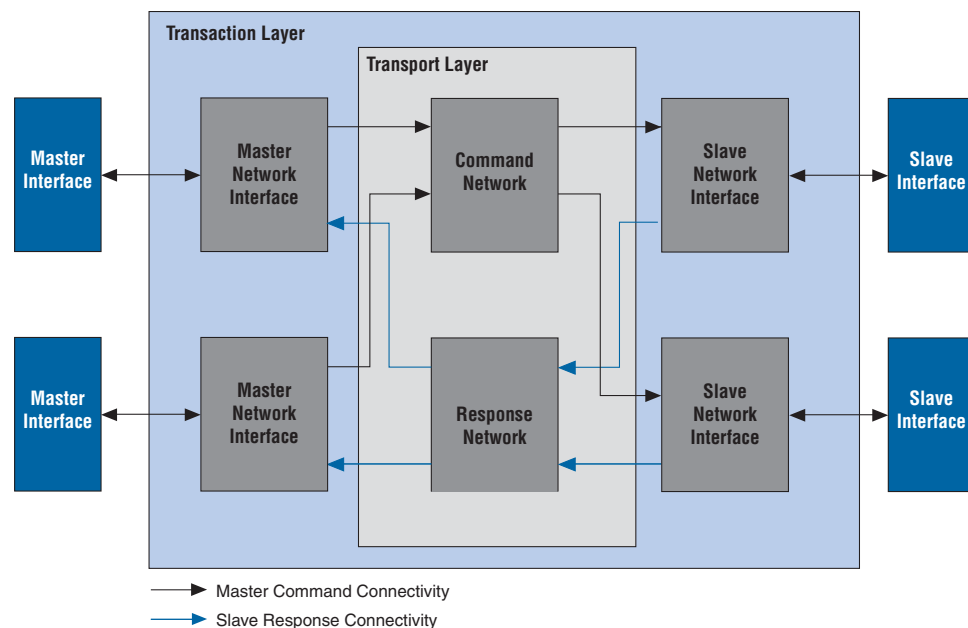
Separate Command and Response Networks

For each transaction domain, Qsys instantiates two independent packet networks, one for command traffic and one for response traffic, instead of a single network that supports both. This increases concurrency, since command traffic and response traffic do not compete for resources like links between network nodes. Qsys also allows the two networks to be optimized independently, such that even the network topology and the packet format in the two networks can be different.

Optimized Command and Response Networks

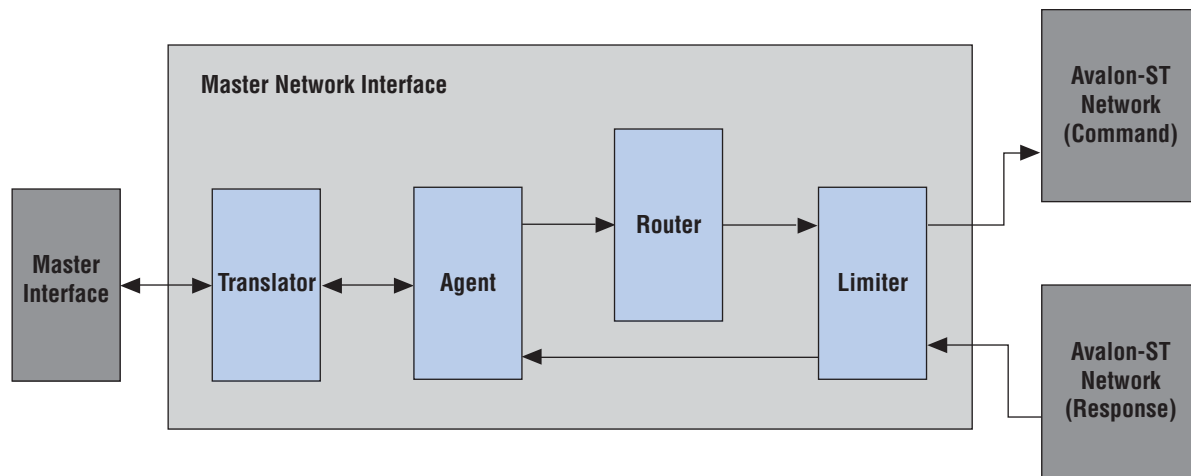
The following steps, describing a read command issued from a master to its intended slave and the response as it returns to the master, provide an overview of the command and response networks in the NoC interconnect shown in [Figure 3](#).

Figure 3. Qsys NoC Interconnect Topology



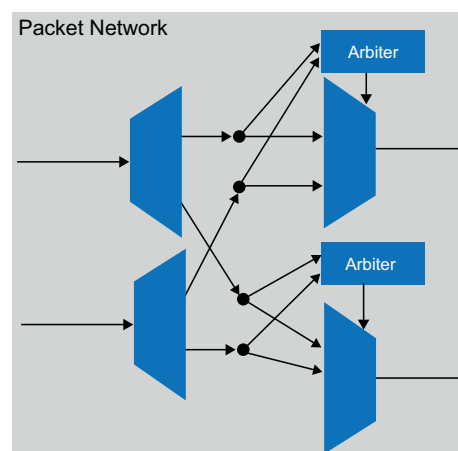
1. When a master issues a command, the first interconnect component that receives the transaction is the translator, as shown in [Figure 4](#). The translator handles much of the variability of the transaction protocol specification, such as active high versus active low signal options and optional read pipelining.

Figure 4. Master Network Interface



2. The agent is the next block to receive the command. The agent encapsulates the transaction into a command packet, and sends the packet to the command network using the transport layer. The agent also accepts and forwards to the master the response packets from the response network.
3. The router determines the address field within the packet format and the slave ID that the packet goes to, as well as the routing information for the next hop.
4. The limiter tracks outstanding transactions to different masters, and prevents commands resulting in an out-of-order or simultaneously-arriving read response.
5. Next, the component is injected into the packet network. The Qsys NoC network supports maximum concurrency, allowing all masters and slaves to communicate on any given clock cycle, as long as no two masters attempt to access the same slave, as shown in [Figure 5](#).

Figure 5. Maximum Concurrency Packet Network




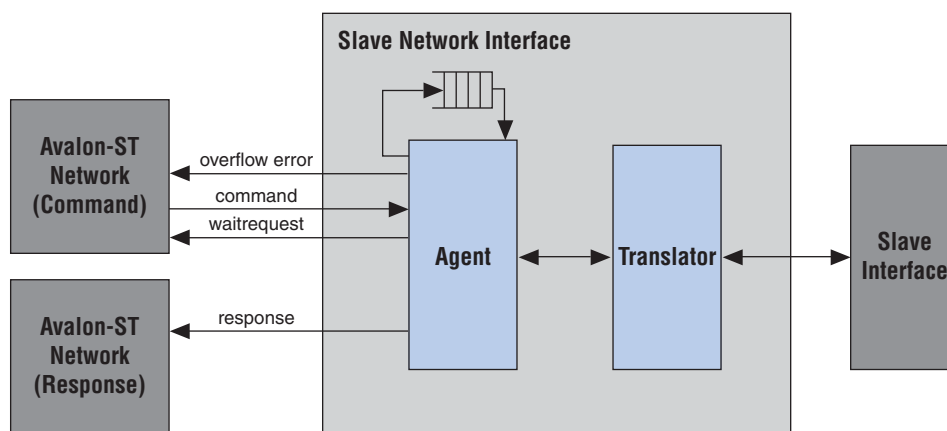
-  Note that the NoC architecture allows replacement of the packet network with any other compatible network implementation.
6. The demultiplexer is the first component that the packet encounters within the transport layer network. The demultiplexer sends the packet towards the next slave.
 7. The packet arrives at the splitter component (represented by the black dot), which then essentially copies the packet to the input of the arbiter, and to the input to the multiplexer.
 8. System designers that require application-specific arbitration, other than the weighted round robin arbitration that Qsys provides by default, can replace the Qsys arbiter with one of their own. To support this, the Qsys arbiter footprint accepts the entire packet, so that alternate arbiter implementations can use detailed transaction information to make their arbitration decision, including data-dependant arbitration.
 9. The decision from the arbiter is sent to the multiplexer, which forwards the selected packet to the slave's network interface, as shown in [Figure 6](#).

Figure 6. Slave Network Interfaces



10. Within the slave's network interface, the packet enters the slave agent component, which terminates the packet, and forwards the transaction contained therein to the slave translator. Simultaneously, the slave agent component pushes transaction information into the slave agent FIFO buffer for transactions requiring a response, such as reads and non-posted writes. The slave translator fills the same role as the master translator, accounting for all the possible variance in the interface specification. If the slave is busy and cannot accept more transactions, then the command is backpressured at the entrance of the agent.
11. When the slave responds to the read transaction, the translator forwards the response to the slave agent. The slave agent pops transaction information from the slave agent FIFO buffer, such as the originating master ID, and merges that with the transaction response to create a response packet. The read data FIFO is present to store the response in case the response network is temporarily unable to accept the response.

12. The slave router then examines the packet to determine the master ID, and assigns the local routing information.
13. The response is the same as the command, but in reverse. The response packet travels through a demultiplexer, hits an arbiter, and once selected, is forwarded through the multiplexer back to the limiter. The limiter then records that the response is received, and then sends it back to the master agent and eventually to the master in the form of a transaction response.

In addition to the components described, Qsys adds burst adapters and width adapters as needed. These are both packet components that examine the packet at the data in some of the fields to make appropriate adaptation decisions. Qsys can also add pipelining stages to help meet timing, and automatically adds handshaking or dual-clock FIFO components when masters and slaves are on different clock domains.

Performance Examples

The following examples compare the performance of two different systems: a 16-master/16-slave system, and a 4-master/16-slave burst- and width-adaptation system. This comparison illustrates how the frequency, latency, and resource use of the Qsys NoC interconnect compares to a traditional interconnect implementation. In these examples all systems are implemented on Altera's Stratix® IV devices, using the C2 speed grade. Qsys NoC interconnect system performance is compared to the traditional Avalon-MM interconnect generated for the same systems by Altera's previous generation SOPC Builder tool.

16-Master/16-Slave System

The 16-master/16-slave system is fully connected with a total of 256 connections. The simple master and slave IP components exist only to test the characteristics of the interconnect, meaning that the system is representative of a completely homogenous system, and not a typical embedded system. [Table 1](#), [Figure 7](#), and [Figure 8](#) show the frequency and resource utilization results of the traditional interconnect and different latency options of the NoC implementation.

Table 1. 16-Master/16-Slave System: Performance Results (% relative to tradition interconnect)

Interconnect Implementation	f_{MAX} MHz	Resource Usage ALMs
Traditional interconnect	131	12766
Qsys NoC, fully combinational	161 (+23%)	13999 (+10%)
Qsys NoC, 1 cycle network latency	225 (+71%)	11260 (-12%)
Qsys NoC, 2 cycle network latency	243 (+85%)	12761 (+0%)
Qsys NoC, 3 cycle network latency	254 (+93%)	14206 (+11%)
Qsys NoC, 4 cycle network latency	314 (+138%)	26782 (+110%)

Figure 7. 16-Master/16-Slave System: NoC Frequency Compared to Traditional Interconnect (MHz)

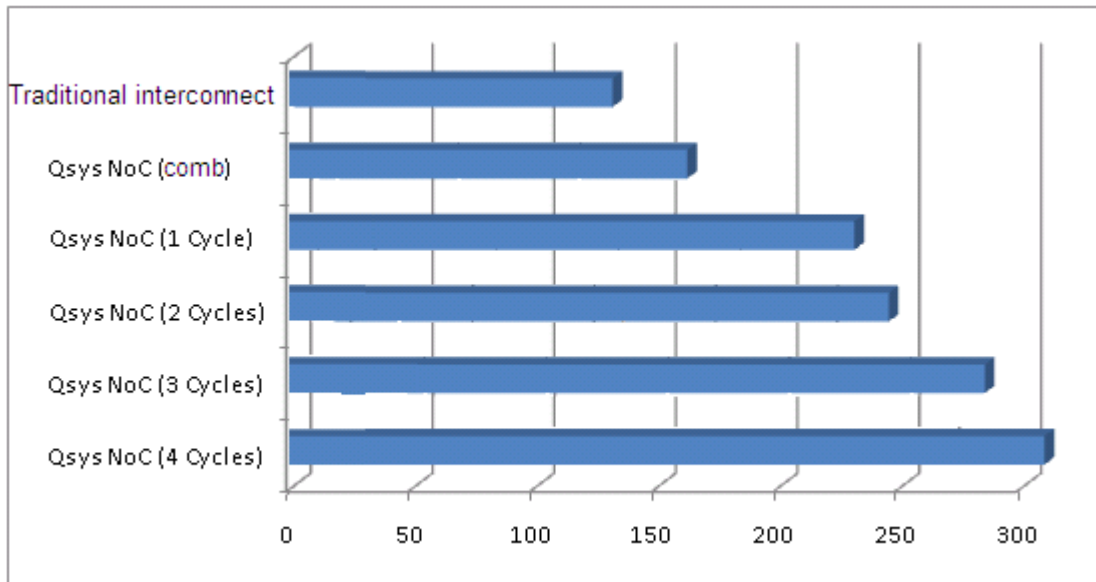
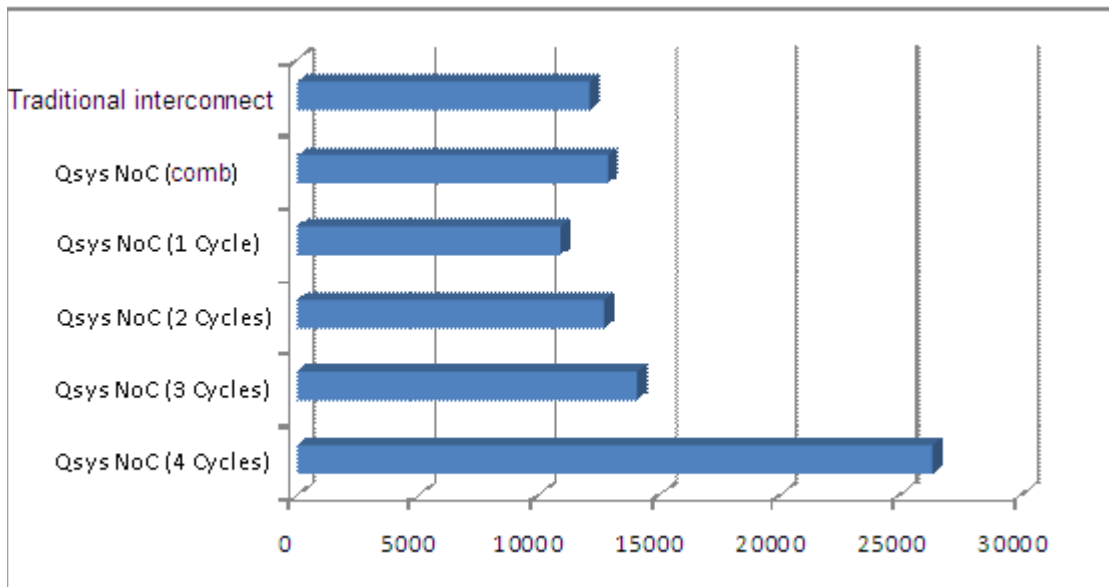


Figure 8. 16-Master/16-Slave System: NoC Resource Utilization Compared to Traditional Interconnect (ALUTs)



4-Master/16-Slave Burst- and Width-Adaptation System

The 4-master/16-slave burst- and width-adaptation system includes characteristics of typical heterogeneous systems, including masters and slaves of different widths and differences in burst support, requiring burst adaptation in the interconnect. Table 2, Figure 9, and Figure 10 show the frequency and resource utilization results of the traditional interconnect and different latency options of the NoC implementation.

Table 2. 4-Master/16-Slave System: Performance Results (% relative to tradition interconnect)

Interconnect Implementation	f_{MAX} (MHz)	Resource Usage (ALMs)
Traditional interconnect	123	11658
Qsys NoC, fully combinational	125 (+2%)	9655 (-17%)
Qsys NoC, 1 cycle network latency	150 (+22%)	9423 (-19%)
Qsys NoC, 2 cycle network latency	164 (+33%)	9847 (-16%)
Qsys NoC, 3 cycle network latency	154 (+25%)	13156 (+13%)
Qsys NoC, 4cycle network latency	171 (+39%)	16925 (+45%)

Figure 9. 4-Master/16-Slave System: Frequency Compared to Traditional Interconnect (MHz)

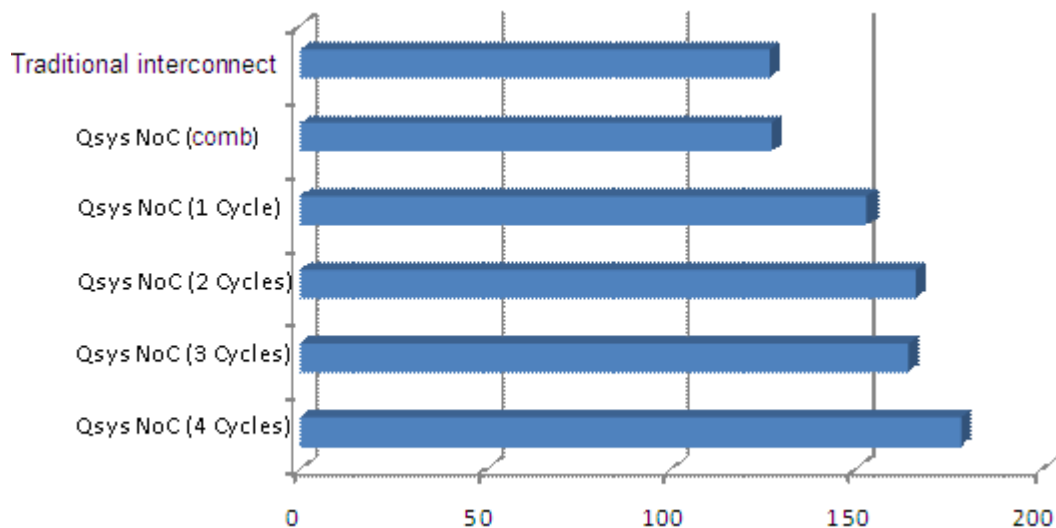
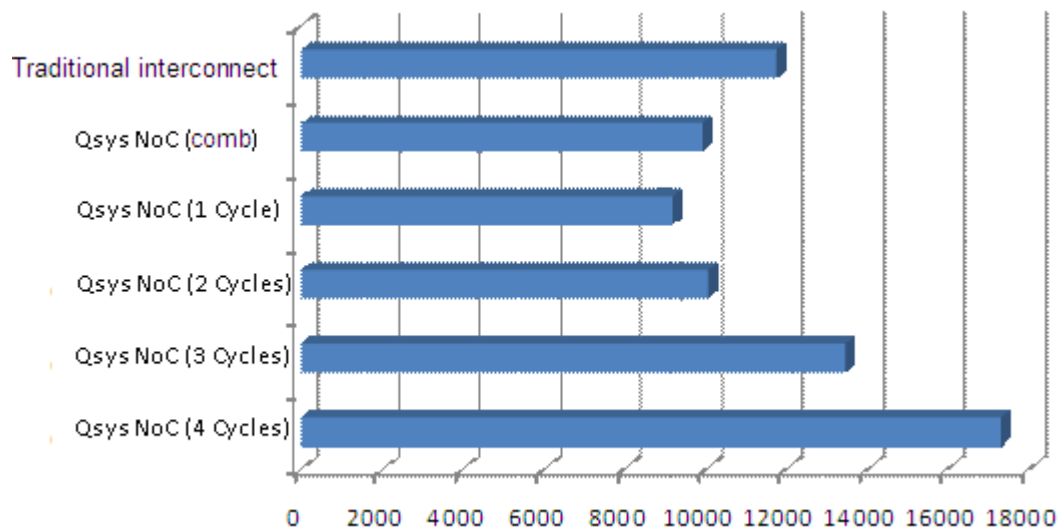


Figure 10. 4-Master/16-Slave System: Resource Utilization Compared to Traditional Interconnect (ALUTs)

Conclusion

NoC interconnect architectures provide a number of significant advantages over traditional, non-NoC interconnects, which allow for independent design and optimization of the transaction and transport protocol layers. The Qsys system integration tool generates an exceedingly flexible FPGA-optimized NoC implementation, based on the requirements of the application. The Qsys NoC interconnect provides a higher operating frequency for the same latency and resource characteristics, with up to a 2X improvement in f_{MAX} compared to traditional interconnects.

Further Information

- Qsys Software Support page of the Altera website:
<http://www.altera.com/support/software/system/qsys/sof-qsys-index.html>
- *System Design with Qsys* section in volume 1 of the *Quartus II Handbook*
http://www.altera.com/literature/hb/qts/qsys_section.pdf
- *AN632: SOPC Builder to Qsys Migration Guidelines*
<http://www.altera.com/literature/an/an632.pdf>
- *Qsys System Design Tutorial*
http://www.altera.com/literature/tt/tt_qsys_intro.pdf

Acknowledgements

- Kent Orthner, Sr. Manager, Software & IP, Altera Corporation

Document Revision History

Table 3 shows the revision history for this document.

Table 3. Document Revision History

Date	Version	Changes
April 2011	1.1	Updated performance and resource usage information.
January 2011	1.0	Initial release.

