WHITE PAPER

Intel® FPGA

Applying the Benefits of Network on a Chip Architecture to FPGA System Design

Authors

Kent Orthner
Senior Manager, Software and IP
Intel® Corporation

Abstract

This document describes the advantages of network on a chip (NoC) architecture in Intel® 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. The Platform Designer (formerly Qsys) system integration tool, included with the Intel Quartus® Prime software, generates a flexible FPGA-optimized NoC implementation automatically, based on the requirements of the application. The Platform Designer 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.†

Table of Contents

Abstract 1
Introduction 1
Understanding NoC Interconnect. 1
Benefits of NoC Architecture 2
- Implement and Optimize Layers. 2
- Simplify Customization per Application . 2
- Use Multiple Topologies and Options. . . . 3
- Simplify Feature Development 3
- Interface Interoperability 3
- Scalability. 3

NoC System Design with Platform Designer (formerly Qsys). 3
Platform Designer NoC Interconnect Optimized for FPGAs 4
- Minimum, Flexible Implementation. 4
- Parameterizable Packet Format Reduces Adaptation 4
- Low-Latency Interconnect 4
- Separate Command and Response Networks 4

Optimized Command and Response Networks 5
Performance Examples 7
- 16-Master/16-Slave System. 7
- 4-Master/16-Slave Burst- and Width-Adaptation System 7

Conclusion. 9
Where to Get More Information . . 9

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.

The Platform Designer uses a NoC architecture to implement system transactions. The Platform Designer 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 Platform Designer network implementation, discusses its benefits, and compares the performance results between traditional and the Platform Designer 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 Platform Designer NoC implementation comes from the same source, the encapsulation of information at each layer of the protocol stack.

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.

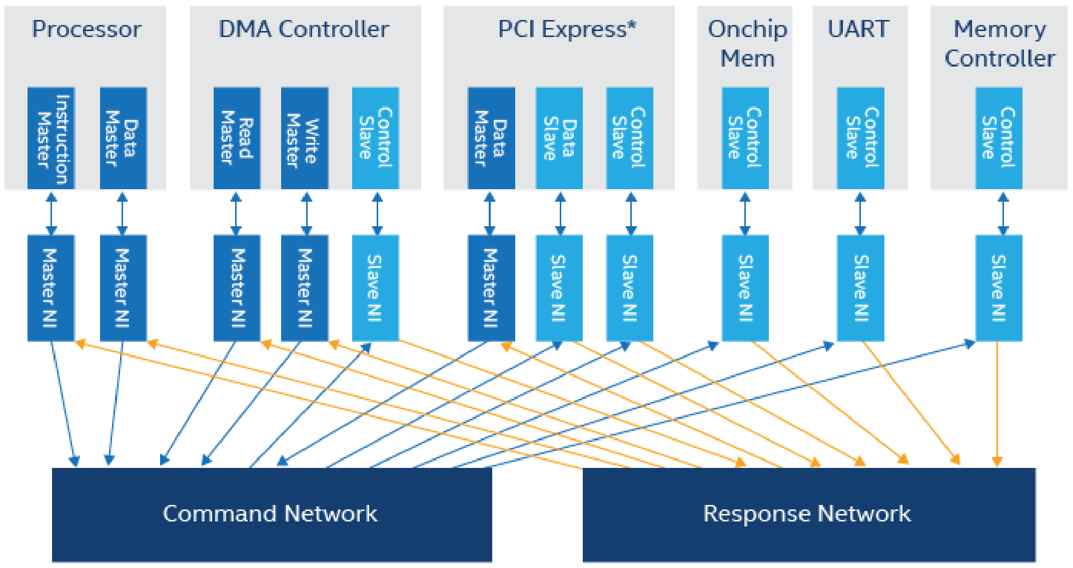


Figure 1. NoC System Basic Topology

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 Intel'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 I^{2}C interfaces. Such as 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

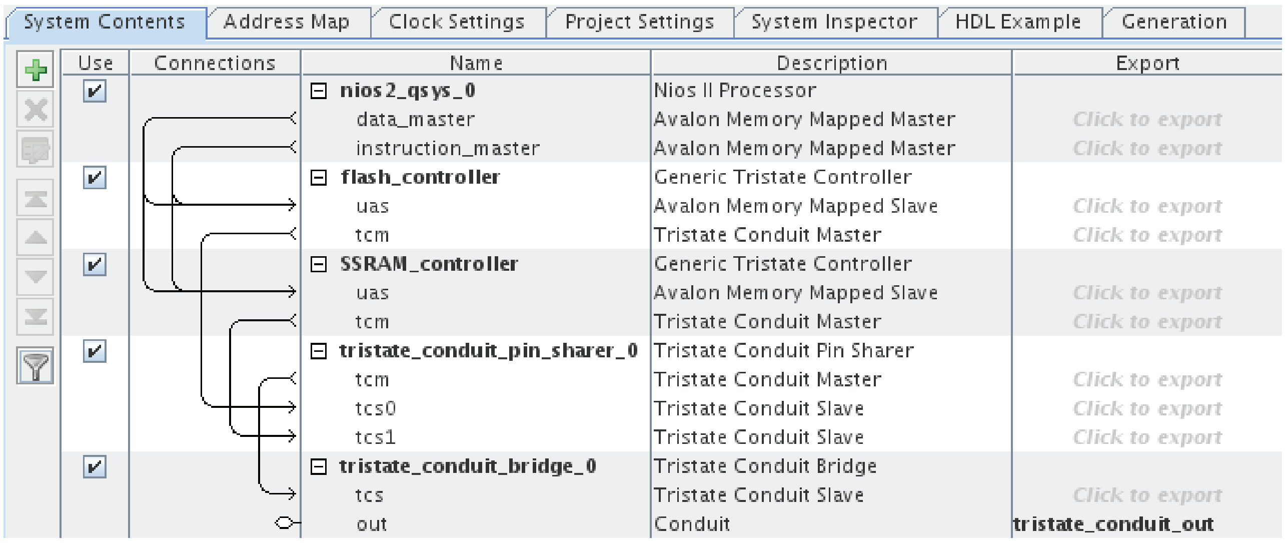
Different intellectual property (IP) cores support different interface types, such as AMBA* AXI*, AHB*, and APB* interfaces; as well as OCP interfaces, Wishbone interfaces, and Avalon-MM interfaces. Supporting a new interface requires implementing only the network interface to encapsulate transactions 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 Platform Designer (formerly Qsys)

The Platform Designer is a powerful system integration tool included as part of Intel Quartus Prime software. The Platform Designer simplifies FPGA system design, allowing designers to create a high-performance system easily, without extensive knowledge of on-chip interconnects or networks. The Platform Designer 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 the Platform Designer, designers instantiate and parameterize system components using a GUI or a scripted system description. The Platform Designer then generates the components and interconnect at the press of a button. Figure 2 shows a system example created in the Platform Designer.



Use	Connections	Name	Description	Export
✓		⊟ **nios2_qsys_0**	Nios II Processor	
		data_master	Avalon Memory Mapped Master	Click to export
		instruction_master	Avalon Memory Mapped Master	Click to export
✓		⊟ **flash_controller**	Generic Tristate Controller	
		uas	Avalon Memory Mapped Slave	Click to export
		tcm	Tristate Conduit Master	Click to export
✓		⊟ **SSRAM_controller**	Generic Tristate Controller	
		uas	Avalon Memory Mapped Slave	Click to export
		tcm	Tristate Conduit Master	Click to export
✓		⊟ **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
✓		⊟ **tristate_conduit_bridge_0**	Tristate Conduit Bridge	
		tcs	Tristate Conduit Slave	Click to export
		out	Conduit	**tristate_conduit_out**

Figure 2. Example System Components Displayed in the Platform Designer (formerly Qsys)

In the Platform Designer, 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. The Platform Designer connects individual signals within connected interfaces automatically. The Platform Designer generates the system implementation as RTL, and manages system interconnect issues such as clock domain crossing, interface width adaptation, and burst adaptation.

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

Platform Designer NoC Interconnect Optimized for FPGAs

The Platform Designer 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 Platform Designer 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, The Platform Designer implements only the minimum interconnect required to meet the performance requirements for a given application.

The Platform Designer 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, the Platform Designer 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, the Platform Designer 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, the Platform Designer 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 Platform Designer 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, the Platform Designer 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. The Platform Designer 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 and overview of the command and response networks in the NoC interconnect shown in Figure 3.

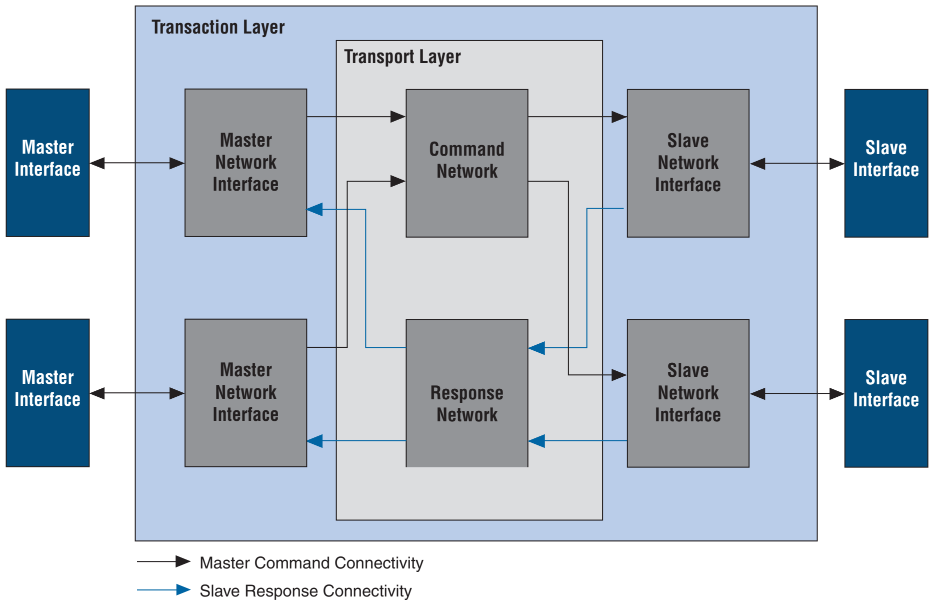


Figure 3. Platform Designer (formerly Qsys) NoC Interconnect Topology

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.

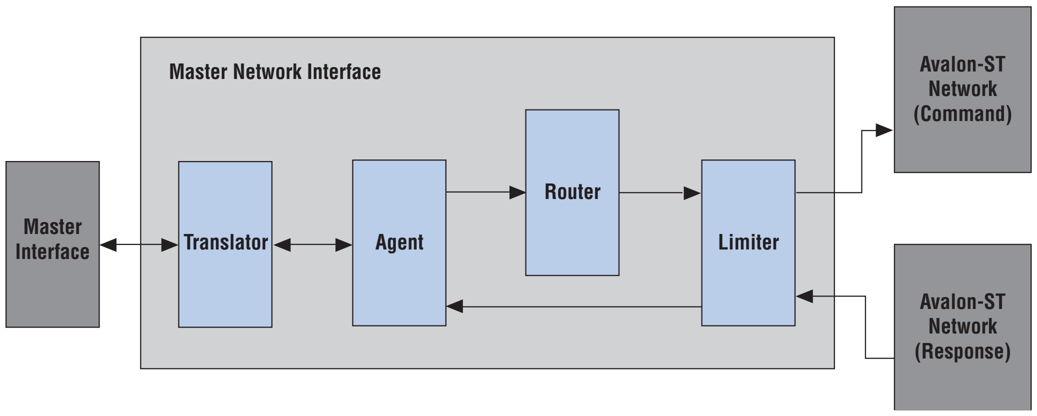


Figure 4. Master Network Interface

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 Platform Designer 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.

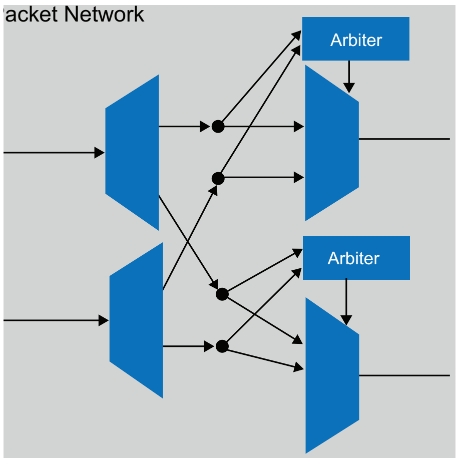


Figure 5. Maximum Concurrency Packet Network

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 the Platform Designer provides by default, can replace the Platform Designer arbiter with one of their own. To support this, the Platform Designer 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.

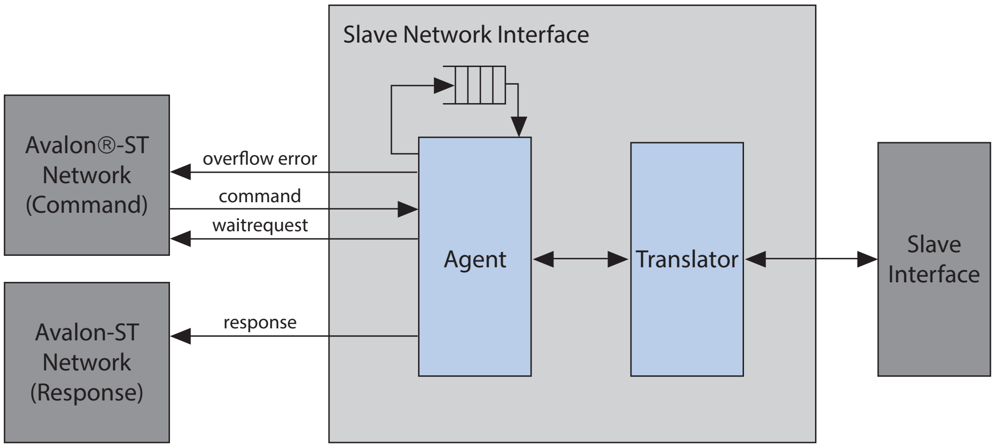


Figure 6. Slave Network Interfaces

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, the Platform Designer 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. The Platform Designer 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 Platform Designer NoC interconnect compares to a traditional interconnect implementation. In these examples all systems are implemented on Stratix® IV devices, using the C2 speed grade. The Platform Designer NoC interconnect system performance is compared to the traditional Avalon-MM interconnect generated for the same systems by the 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.

NTERCONNECT IMPLEMENTATION	F_{MAX} MHZ	RESOURCE USAGE (ALMS)
Traditional interconnect	131	12766
Platform Designer NoC, fully combinational	161 (+23%)	13999 (+10%)
Platform Designer NoC, 1 cycle network latency	225 (+71%)	11260 (-12%)
Platform Designer NoC, 2 cycle network latency	243 (+85%)	12761 (+0%)
Platform Designer NoC, 3 cycle network latency	254 (+93%)	14206 (+11%)
Platform Designer NoC, 4 cycle network latency	314 (+138%)	26782 (+110%)

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

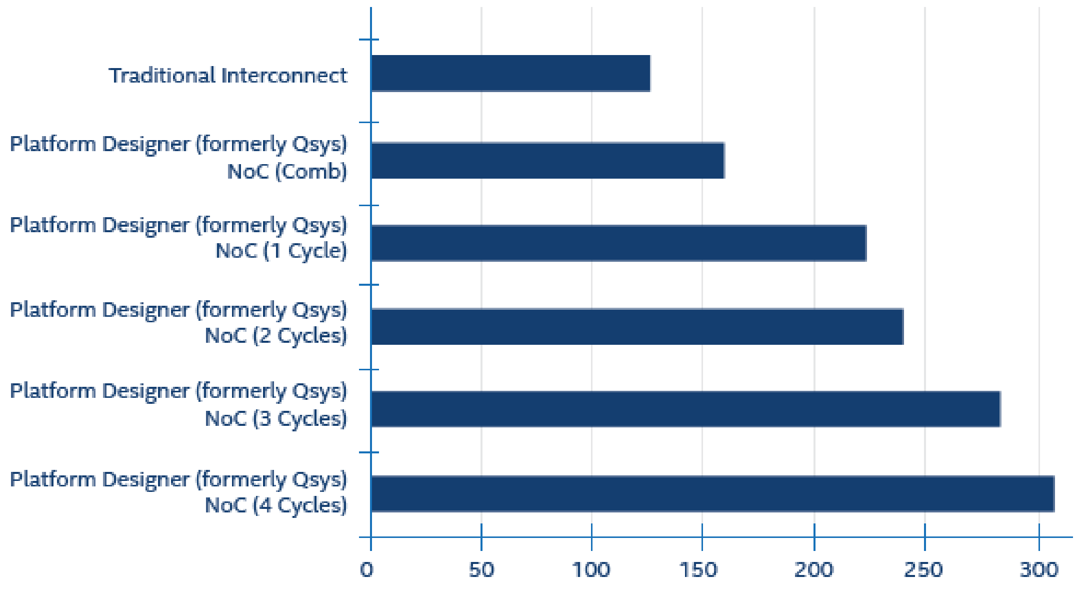


Figure 7. 16-Master/16-Slave System: NoC Frequency Compared to Traditional Interconnect (MHz)

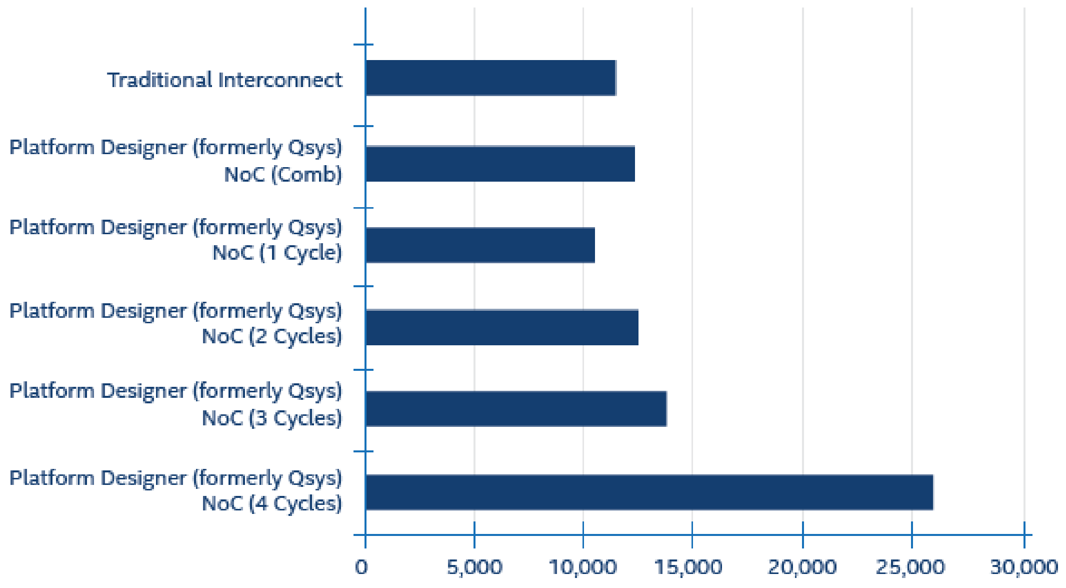


Figure 8. 16-Master/16-Slave System: NoC Resource Utilization Compared to Traditional Interconnect (ALUTs)

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.

NTERCONNECT IMPLEMENTATION	F_{MAX} MHZ	RESOURCE USAGE (ALMS)
Traditional interconnect	123	11658
Platform Designer NoC, fully combinational	125 (+2%)	9655 (-17%)
Platform Designer NoC, 1 cycle network latency	150 (+22%)	9423 (-19%)
Platform Designer NoC, 2 cycle network latency	164 (+33%)	9847 (-16%)
Platform Designer NoC, 3 cycle network latency	154 (+25%)	13156 (+13%)
Platform Designer NoC, 4 cycle network latency	171 (+39%)	16925 (+45%)

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

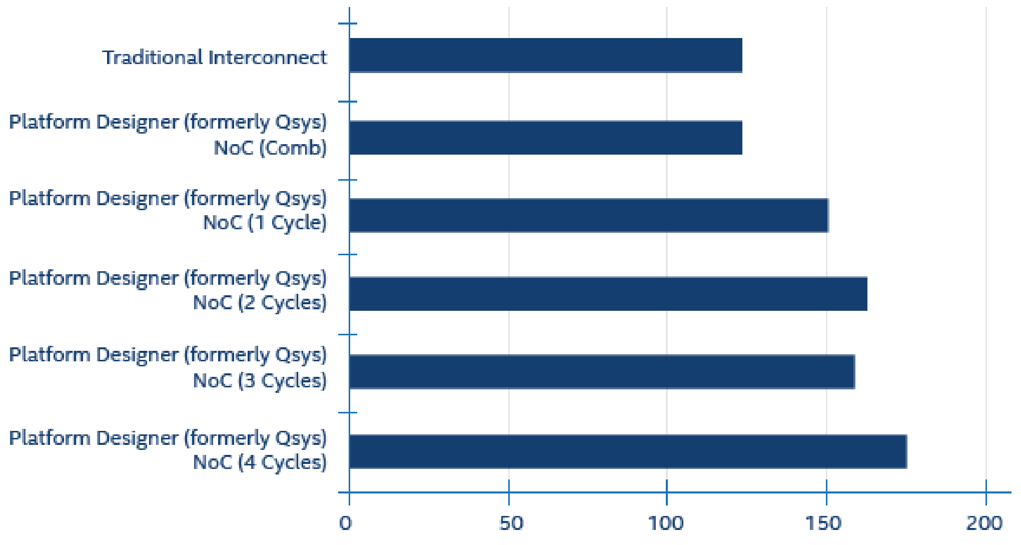


Figure 9. 4-Master/16-Slave System: Frequency Compared to Traditional Interconnect (MHz)

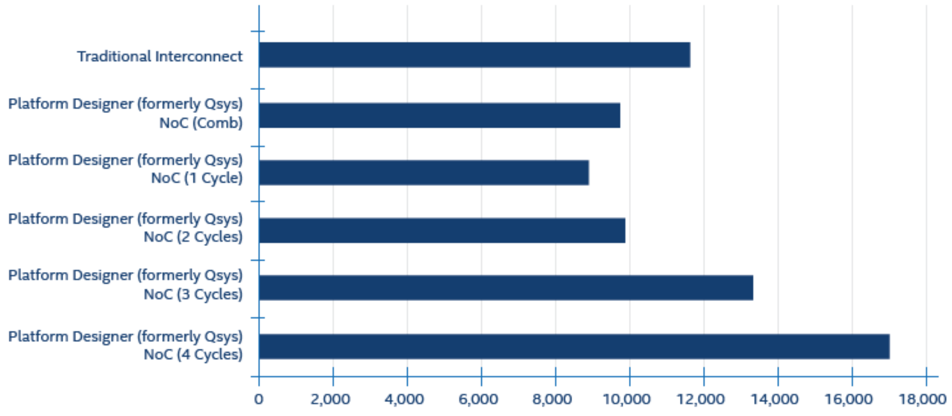


Figure 10. 4-Master/16-Slave System: Resource Utilization Compared to Traditional Interconnect (ALUTs)

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 Platform Designer generates an exceedingly flexible FPGA-optimized NoC implementation, based on the requirements of the application. The Platform Designer 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.[†]

Where to Get More Information

- Platform Designer (formerly Qsys):
  www.altera.com/products/design-software/fpga-design/quartus-prime/features/qts-qsys.html
- Platform Designer (formerly Qsys) Tool Support: www.altera.com/support/support-resources/design-software/qsys.html
- AN632: SOPC Builder to Qsys Migration Guidelines: www.altera.com/literature/an/an632.pdf

[†] Tests measure performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

WP-01149-1.2