RapidIO™: An Embedded System Component Network Architecture

Architecture and Systems Platforms
Motorola Semiconductor Product Sector
7700 West Parmer Lane, MS: PL30
Austin, TX 78729

Abstract

This paper describes RapidIO, a high performance low pin count packet switched system level interconnect architecture. The interconnect architecture is intended to be an open standard which addresses the needs of a variety of applications from embedded infrastructure to desktop computing. Applications include interconnecting microprocessors, memory, and memory mapped I/O devices in networking equipment, storage subsystems, and general purpose computing platforms. This interconnect is intended primarily as an intra-system interface, allowing chip to chip and board to board communications at giga-byte per second performance levels. Supported programming models include globally shared distributed memory and message-passing. In its simplest form, the interface can be implemented in an FPGA end point. The interconnect architecture defines a protocol independent of a physical implementation. The physical features of an implementation utilizing the interconnect are defined by the requirements of the implementation, such as I/O signalling levels, interconnect topology, physical layer protocol, error detection, etc. The interconnect is defined as a layered architecture which allows scalability and future enhancements while maintaining compatibility.

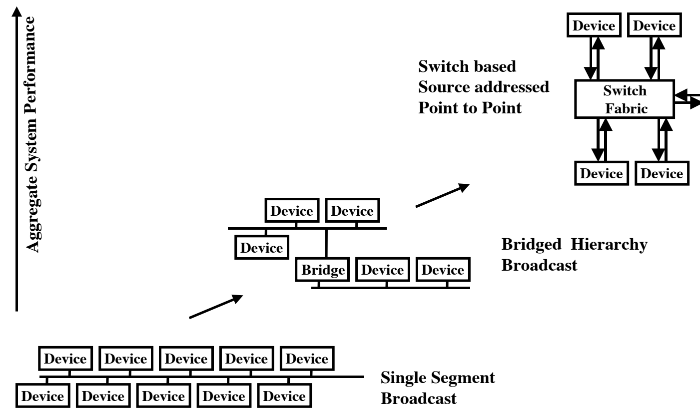
1 Introduction

Computer and embedded system development continues to be burdened by divergent requirements. On one hand the performance must increase at a nearly logarithmic rate, while on the other hand system cost must stay the same or decrease. Several applications, such as those found in telecommunications infrastructure equipment, are also burdened with increasing capabilities while decreasing the board size and ultimately the floor space which equipment occupies.

The connection fabric between microprocessors and peripherals has traditionally been comprised of a hierarchy of shared buses (Figure 1). Devices are placed at the appropriate level in the hierarchy according to the performance level they require. Low performance devices are placed on lower performance buses which are bridged to the higher performance buses so as to not burden higher performance devices. Bridging is also done to address legacy interfaces.

The need for higher levels of bus performance are driven by two key factors. First, the need for raw data bandwidth to support higher peripheral device performance demands. Second, the need for more system concurrency. Overall system performance has increased due to the application of distributed DMA and processing.

FIGURE 1 System topology trends



Over the past several years the shared multi-drop bus has been exploited to its full potential. Many techniques have been applied, such as increasing frequency, widening the interface, pipelining transactions, splitting transactions, and allowing out of order completion. Continuing to work with a bus in this manner creates several design issues. Increasing bus width, for example, reduces the maximum achievable frequency due to skew between signals. More signals will also result in more pins on a device, resulting in a higher product cost and a reduction in the number of interfaces the device can provide.

Another issue is the desire to increase the number of devices that can communicate directly with each other. As frequency and width increase, the ability to have more than

a few devices attached to a shared bus becomes a difficult design challenge. In many cases, system designers have inserted a hierarchy of bridges to reduce the number of loads on a single bus.

To address the needs of present and future systems, an embedded system component network architecture is proposed. The architecture is for a point-to-point, moderately parallel, packet-based interconnect. The inteconnect is focused as a processor, memory, and memory mapped I/O interface optimized for use inside a chassis. In embedded system applications it must have limited to no impact on the software infrastructure that operates over it. It can be implemented in few gates, offers low transaction latency and high bandwidths. The *RapidIO* interconnect architecture is intended to be an open standard architecture.

There are similar movements in the industry away from shared buses, towards fabrics. Proposed fabrics optimized for I/O include InfiniBand [5] .

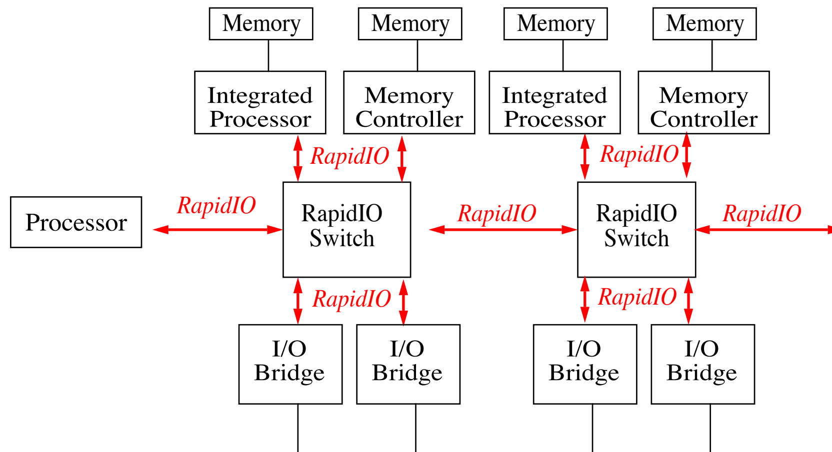
InfiniBand is targeted as a System Area Network (SAN) interconnect. A SAN is used to cluster systems together to form larger highly available systems. SANs usually connect whole computers together within distances of up to 30 meters. Transactions through a SAN are typically handled through software drivers using message channels or remote direct memory access (RDMA).

RapidIO differs from InfiniBand in that it is targeted toward embedded, in the box communications. *RapidIO* can be thought of as the mechanism to connect devices together to form the computer system. InfiniBand connects these computer systems together to form the SAN.

System Applications

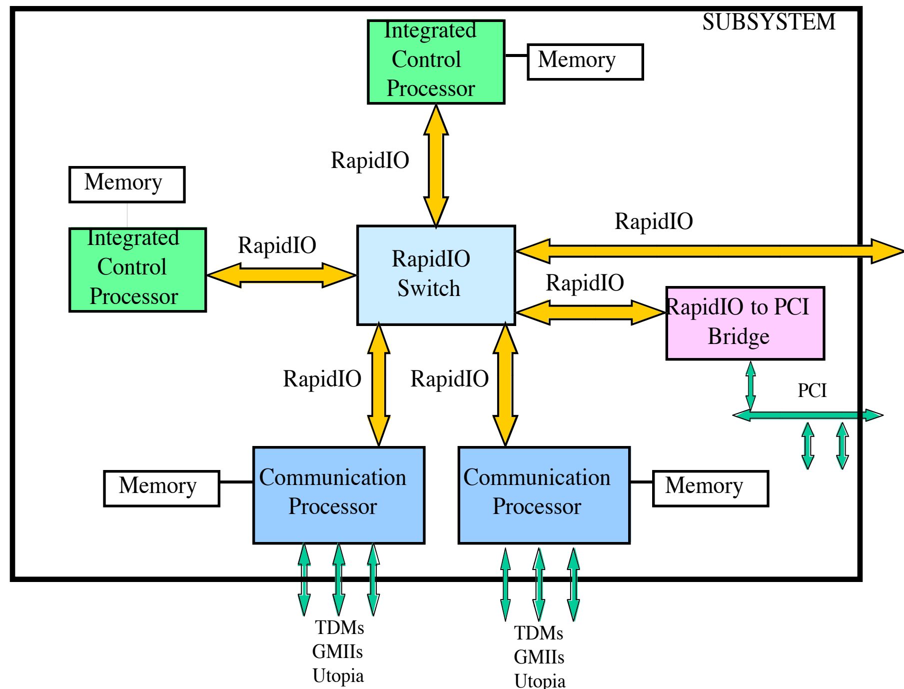
The *RapidIO* interconnect is targeted as a device level interface for use in environments where multiple devices must work in a tightly coupled load/store environment. Figure 2 illustrates a generic system containing memory controllers, processors, and I/O bridges connected using *RapidIO* switches.

FIGURE 2 *RapidIO* allows a variety of devices to be interconnected



Applying this generic system block diagram to real applications the following examples are presented. In the first example (Figure 3), communications engines are connected with control processors to deliver a high performance network protocol subsystem. This system enables direct memory access between communication and control processors.

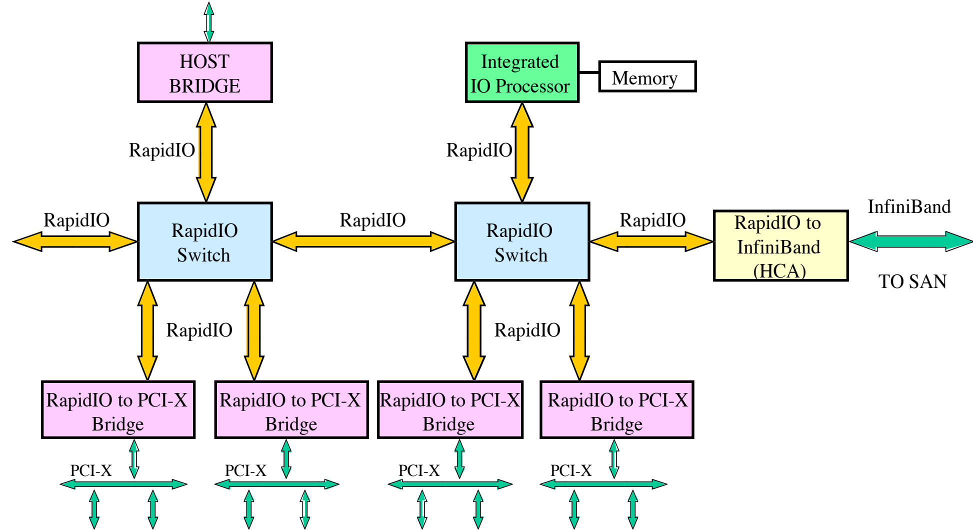
FIGURE 3 *RapidIO* furnishes a fabric to attach a distributed network protocol management subsystem



In computing, the PCI bus [6] is ubiquitous. Enterprise storage applications, for example, use PCI to bridge multiple disk channels to a system. As disk throughput has increased so has the need for higher system throughput (a la PCI-X). To meet the electrical requirements of higher bus frequencies the number of devices per bus segment has decreased. Therefore to connect the same number of devices, more bus segments are required. These applications require higher bus performance, more device fan-out, and greater device separation. PCI-to-PCI bridge devices could be used to solve this problem but this would come at the cost of many pins and a limit in the total transmission distance between devices.

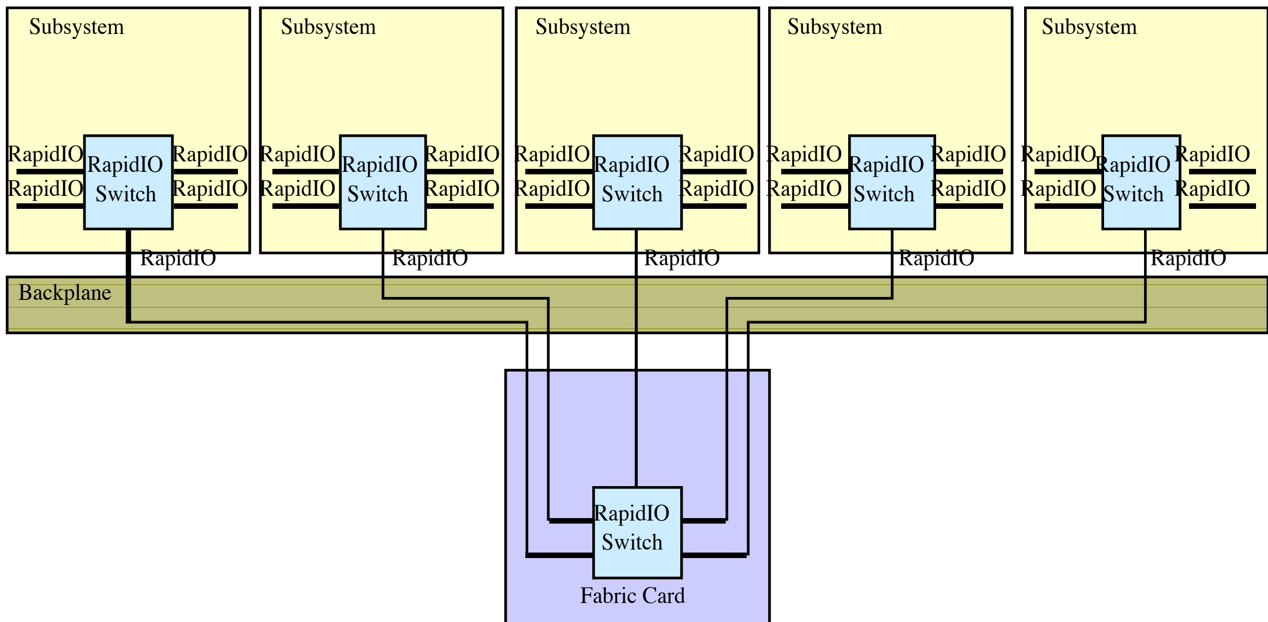
RapidIO can be used for transparent PCI-to-PCI bridging allowing for a flattened architecture utilizing fewer pins with greater transmission distances. Figure 4 shows one such bridged system. In this example, several PCI-X bridges are connected together using *RapidIO* switches. An InfiniBand Host Channel Adapter can be provided if the system is to become part of a wider System Area Network.

FIGURE 4 *RapidIO* as a PCI bridging fabric reduces device pin count



Many systems require the partitioning of functions into field replaceable units. These printed circuit boards have traditionally been interconnected using multi-drop interfaces like VME or Compact PCI. Higher system level performance can be achieved by using *RapidIO* as shown in Figure 5. For hot swap applications, the removal of a device has no electrical impact on other devices as it would in a shared bus environment.

FIGURE 5 *RapidIO* spans boards in a a passive backplane application



Philosophy

The architecture of *RapidIO* was driven with certain principal objectives:

- Focus on applications that connect devices which are within the box or chassis: As an example, since devices are not intended to be connected with several meters of cable the interface can use a simple signalling and flow control scheme.
- Limit the impact on software: Many applications are built on a huge legacy of memory mapped I/O. In such applications the interconnect must not be visible to software. Offering more abstracted interfaces requires a large software re-engineering effort.
- Confine protocol overhead: Because of the dependency on latency and bandwidth it is important that the protocol use no more transaction overhead than is absolutely necessary.
- Partition the specifications to limit the necessary function set to only those needed for the application: This limits design complexity and enables future enhancements without impacting the top to bottom specification.
- Manage errors in hardware: In meeting the high standards of availability, yet limiting the impact to performance and software infrastructure, it is important that the interconnect be able to detect errors. The transmission media should have the capability of detecting multiple bit errors. Going a step further the interconnect must utilize hardware mechanisms to survive single bit errors and most multiple bit errors.
- Limit silicon footprint: Many applications require the use of a high performance interconnect but have a limited transistor budget. It is important that the interface be able to fit within a small transistor count. As an example, it is important that it fit within a commonly available FPGA technology.
- Build from common process technology I/O: Since this interface is targeted for intra-system communications, it should not require separate discrete physical layer hardware. This means that the I/O technology should be power efficient and limit the use of exotic process technology.
- Choose a memory coherency scheme optimized for small (16 or fewer memory controllers) distributed memory systems: Most applications of shared memory usually involve few processing elements.

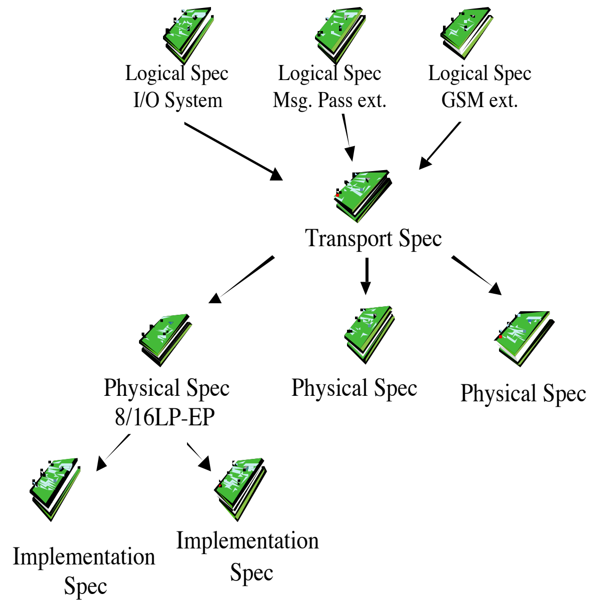
RapidIO is specified in a 3-layer architectural hierarchy (Figure 6). A Logical specification defines the overall protocol and packet formats. This is the information necessary for end points to process a transaction. A Transport specification provides the necessary route information for a packet to move from end point to end point. A Physical specification contains the device level interface such as packet transport mechanisms, flow control, electrical characteristics, and low level error management.

FIGURE 6 *RapidIO* is architected in a partitioned hierarchy of Logical, Transport, and Physical specifications

Logical Specification
- Information necessary for the end point to process the transaction. (i.e. Transaction type, size, physical address)

Transport Specification
- Information to transport packet from end to end in the system. (i.e. Routing Address)

Physical Specification
- Information necessary to move packet between two physical devices. (i.e. Electrical interface, flow cntl)



This partitioning provides the flexibility to add new transaction types to a logical specification without requiring modification to the transport or physical layer specifications.

The *RapidIO* feature set and protocols are based upon a number of considerations for both general computing and embedded applications. These considerations are broken down into three categories; functional, physical, and performance.

Functional Considerations

The *RapidIO* architecture is targeted toward memory mapped distributed memory systems and subsystems. A message passing programming model is supported as well as an optional globally shared distributed memory programming model. This enables both general purpose multiprocessing and distributed I/O processing to co-exist under the same protocol.

Message passing and DMA devices can improve the interconnect efficiency if larger non-coherent data quantities are encapsulated within a single packet, so *RapidIO* supports a variety of data sizes within the packet formats. Since the message passing programming model is fundamentally a non-coherent non-shared memory model, *RapidIO* can assume that portions of the memory space are only directly accessible by a processor or device local to that memory space. A device attempting to access memory space which is not locally owned must do so using software maintained coherency methods or through a local device controlled message passing interface.

For the globally shared memory programming model, a directory based coherency mechanism is chosen. *RapidIO* furnishes a variety of ISA specific cache control operations such as multiple cache line size support, block flushes, data cache block zeroing, and TLB synchronization mechanisms.

Physical Considerations

The protocol and packet format are independent of the topology of the physical interconnect. The protocol works whether the physical interconnect is a point-to-point, ring, bus, switched multidimensional network, duplex serial connection, etc. There is no dependency on the bandwidth or latency of the physical fabric. The protocol handles out of order packet transmission and reception. There is no requirement for geographical addressing; a device's identifier does not depend on its location in the address map, but can be assigned by other means. The physical interface contains the signal definitions, flow control and error management.

Initially an 8-bit and 16-bit parallel (8/16 LP-EP), point-to-point interface is deployed. An 8/16 LP-EP device interface contains a dedicated 8- or 16-bit input port with clock and frame signals, and a 8- or 16-bit output port with clock and frame signals. A source synchronous clock signal clocks packet data on rising and falling edge. The frame signal provides a control reference. Differential signalling is used to reduce interface complexity, provide robust signal quality, and promote good frequency scalability across printed circuit boards and connectors.

Performance Considerations

Packet headers are as small as possible to minimize the control overhead and are organized for fast, efficient assembly and disassembly. As the amount of data included in a packet increases, packet efficiency increases. *RapidIO* supports data payloads up to 256 bytes. Messages are very important for embedded control applications, so a variety of large and small data fields and multiple packet messages are supported.

Packet-based cache coherence requires a large amount of control overhead, so an interventionist (cache to cache) protocol saves a large amount of latency for memory accesses that cause another cache to provide the requested data.

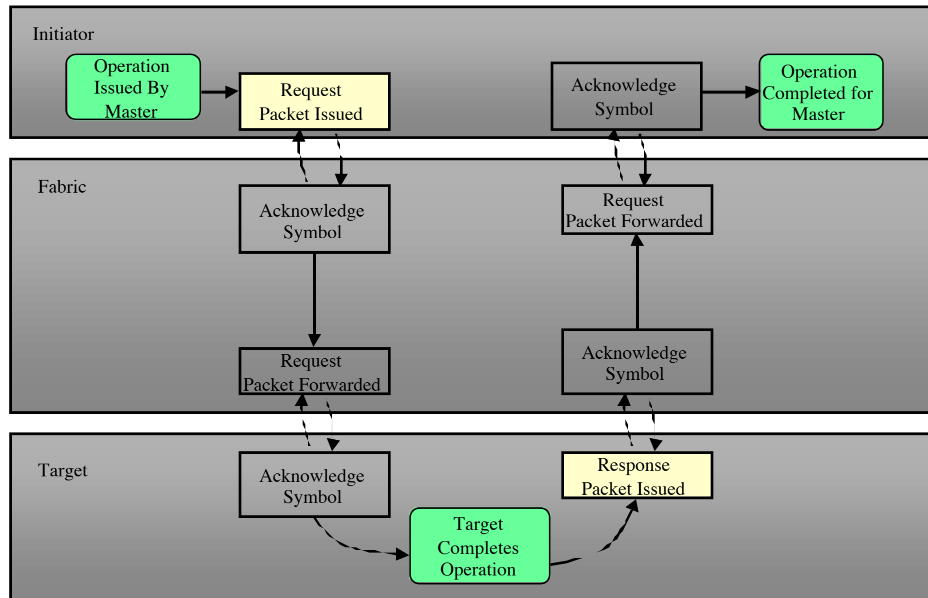
Multiple transactions are allowed concurrently in the system, not only through the ability to pipeline transactions from a single device, but also through spatial reuse of interfaces between different devices in the system. Without this, a majority of the potential system throughput is wasted. The sustainable bandwidth target for the initial deployment is 1 gigabyte per second per device pair with headroom for future growth toward multiple gigabytes per second.

2 Protocol Overview

Packets and Control Symbols

RapidIO transactions are based on request and response packets. Packets are the communication element between end point devices in the system. A master or initiator generates a request packet which is transmitted to a target. The target then generates a response packet back to the initiator to complete the transaction. *RapidIO* end points are typically not connected directly to each other but instead have intervening connection fabric devices. Control symbols are used to manage the flow of transactions in the *RapidIO* physical interconnect. Control symbols are used for packet acknowledgement, flow control information, and maintenance functions. Figure 7 shows how a packet progresses through the system.

FIGURE 7 Transactions are constructed with request and response packet pairs

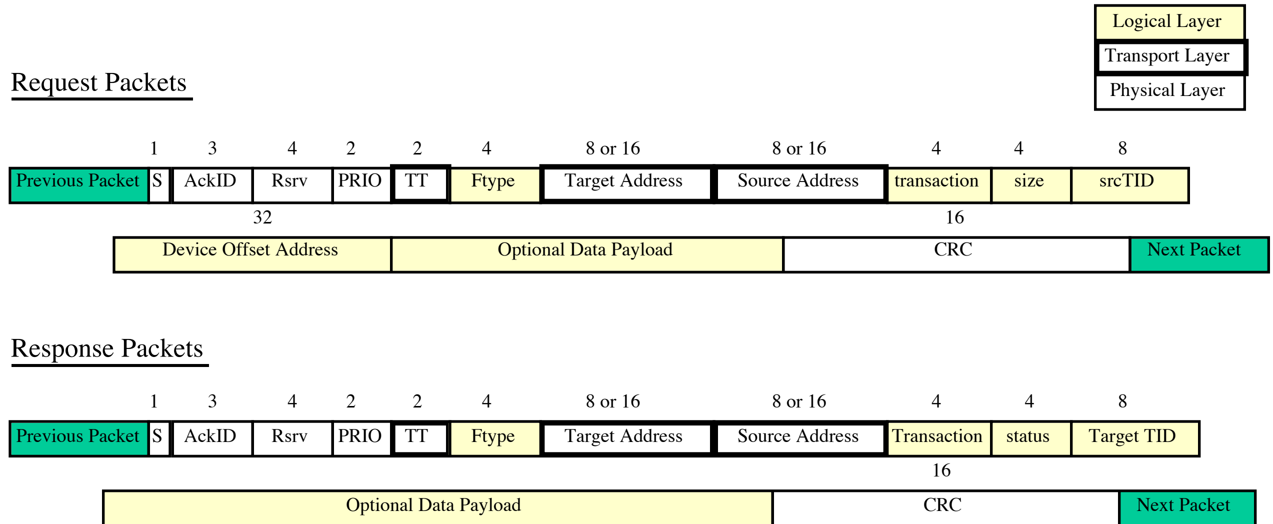


In this example, the initiator begins a transaction in the system by generating a request packet. The request is sent to a fabric device which in turn replies with an acknowledge control symbol. The packet is forwarded to the target through the fabric device. This completes the request phase of the transaction. The target completes the transaction and generates a response packet. The response packet returns through the fabric device using control symbols to acknowledge each hop. Once the packet reaches the initiator and is acknowledged, the transaction is considered complete.

Packet Format

The *RapidIO* packet is comprised of fields from the three-level specification hierarchy as shown in Figure 8.

FIGURE 8 The *RapidIO* packet contains fields from the specification hierarchy.



The request packet begins with physical layer fields. The “S” bit indicates whether this is a packet or control symbol. The “AckID” indicates which packet the fabric device should acknowledge with a control symbol. *RapidIO* supports up to 8 unacknowledged packets between two adjacent devices. The “PRIO” field indicate the packet priority used for flow control. The “TT”, “Target Address”, and “Source Address” fields indicate the type of transport address mechanism used, the device address where the packet should be delivered, and where the packet originated. The “Ftype” and “transaction” indicate the transaction that is being requested. The “size” is an encoded transaction size. *RapidIO* transaction data payloads range from 1-byte to 256 bytes in size. The “srcTID” indicates the transaction ID. *RapidIO* devices may have up to 256 outstanding transactions between two end points. For memory mapped transactions the “Device Offset Address” follows. For write transactions a “Data Payload” completes the transaction followed by a 16-bit CRC.

Response packets are very similar to request packets. The “Size” field is replaced by the “status” field which indicates whether the transaction was successfully completed. The “TargetTID” is the corresponding request packet transaction ID.

Transaction Formats and Types

One of the attributes of a software transparent interconnect is the requirement for a rich set of transaction functions. The *RapidIO* transaction is described through two fields;

the format type “Ftype”, and the “Transaction”. To ease the burden of transaction deciphering, transactions are grouped by format as shown in Table 1.

TABLE 1. *RapidIO* Transactions are grouped in format groups

Ftype	Class	Transaction Examples	Logical Specification[a]
0, 15	User	User Defined	All
1	Intervention Request	Read from current owner	GSM
2	Non-Intervention Request	Read from home, Non-coherent read, IO read, TLB sync, Atomic	GSM, IOS
5	Write Request	Cast-out, Flush, Non-coherent write, Atomic swap	GSM, IOS
6	Streaming Write	Steam write	IOS
8	Maintenance	Configuration, control, and status register read and write	All
10	Doorbell	In-band Interrupt	MSG
11	Message	Mailbox	MSG
13	Response	Read and write responses	All
3,4,7,9, 12,14	Reserved		

a. Logical Specifications:, IOS - basic input/output system, GSM - globally shared memory extensions, MSG - message passing extensions

Flow Control

Flow control is an important aspect of any interconnect. The goal is for devices to be able to complete transactions in the system without being blocked by other transactions. Bus based interconnects use arbitration algorithms to be sure that devices make forward progress and that urgent transactions take precedence over less urgent ones. With switch based interconnects, transactions enter the interconnect at different points in the system and there is no centralized arbitration mechanism. This creates the need for more complex methods to manage transaction flow.

One of the objectives of *RapidIO* is to limit overhead and complexity as much as possible especially in the area of flow control. For *RapidIO*, flow control is specified as part of the physical specification. This is because transaction flow is largely dependent on the physical interconnect and system partitioning. The *RapidIO* 8/16 LP-EP physical layer specification defines each packet as having a transaction priority. Each transaction priority is associated with a transaction flow. There are three transaction flows defined. Transaction flows allow higher priority transactions to go ahead of lower priority transactions. All transactions are ordered within a transaction flow and complete in a first come first serve basis. *RapidIO* also describes three types of flow control mechanisms; retry, throttle, and credit based.

The retry mechanism is the most simple mechanism and is required not only for flow control but also a component of hardware error recovery. A receiver which is unable to accept a packet because of a lack of resources or because the packet was corrupt, may respond with a retry control symbol. The sender will retransmit the packet.

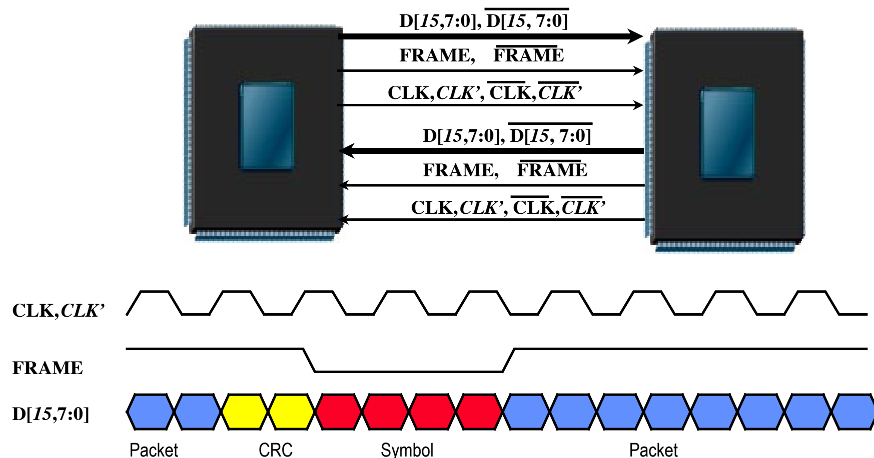
The throttle mechanism makes use of *RapidIO*'s idle control symbol. Idle symbols may be inserted during packet transmission. This allows a device to insert wait-states in the middle of packets. A receiving device can also send a throttle control symbol to the sender requesting that it slow down by inserting idle control symbols.

The credit based mechanism is useful for devices that implement transaction buffer pools, especially fabric devices. In this scheme certain control symbols contain a buffer status field which represents the current status of the receiver's buffer pool for each transaction flow. A sender only sends packets when it knows the receiver has a buffer available to store it to.

3 Physical Interface

The *RapidIO* logical specification is defined to be physical layer independent. This means that the *RapidIO* protocol could be transmitted over anything from serial to parallel interfaces, from copper to fiber media. That said, the protocol is optimized to operate through byte wide granularity parallel point-to-point interfaces. The first physical interface considered and defined is known as the 8- or 16-bit link protocol end point specification (8/16 LP-EP). This specification is defined as having 8 or 16 data bits in each direction along with clock and frame signals in each direction (Figure 9.)

FIGURE 9 The *RapidIO* 8/16 LP-EP physical layer provides for full duplex communications between devices.



8/16 LP-EP is a source synchronous interface which means that a clock is transmitted with the associated data. Source synchronous clocking allows longer transmission distances at higher frequencies. Two clock pairs are provided for the 16-bit interface to

help control skew. The receiving logic is able to use the receive clock for re-synchronization of the data to its local clock domain.

The FRAME signal is used to delineate the start of a packet or control symbol. It operates as a no return to zero (NRZ) signal where any transition marks an event.

Electrical Interface

RapidIO adopts the IEEE 1596.3 Low Voltage Differential Signals (LVDS)[4] standard as basis for the electrical interface. LVDS is a low swing (250 to 400 mV) constant current differential signalling technology which is targeted toward short distance board level applications. LVDS is technology independent and can be implemented in CMOS. Differential signals provide improved noise margin, immunity to externally generated noise, lower EMI, and reduced numbers of power and ground signal pins. LVDS has a simple receiver based termination of 100 ohms.

The target frequencies of operation are from 250MHz to more than 1GHz. Data is sampled on both edges of the clock. The resulting data rates in each direction scale to 2 Gbyte/sec for the 8-bit and 4 Gbytes/sec for the 16-bit interfaces. The targeted transmission distance is 30 inches of trace over standard printed circuit board technology. The transmission environment is intended for use in backplane applications traversing connector pairs.

Power Considerations

A concern in any system is power consumption. This is especially true of high component density chassis applications. This has been a problematic issue with typical single-ended interfaces. Frequencies are rising faster than voltages are lowering resulting in increased power dissipation. Because LVDS is low swing and because it uses a constant current source, the power consumption remains relatively constant over a wide operating frequency range.

4 Protocol Extensions

Message Passing

When data must be shared amongst multiple processing elements in a system, a protocol must be put in place to maintain ownership. In many systems, especially embedded, this protocol is often managed through software mechanisms. If the memory space is accessible to multiple parties, then locks or semaphores are used to grant access to one party. In other cases, processing elements may only have access to locally owned memory space. In these “shared nothing” machines, a mechanism is required to pass data from the memory of one processing element to another. This can be done using software visible mailbox hardware.

RapidIO provides a message passing extension which is useful in shared nothing machines. The *RapidIO* message passing logical extension protocol describes transactions to enable mailbox and doorbell communications. A *RapidIO* mailbox is a port

through which one device may send a message to another device. The receiving device controls where the message is placed when it arrives. A *RapidIO* message can consist of up to 16 packets of up to 256 bytes each for a total 4 kbytes. A receiver can have 1 to 4 addressable message queues to capture inbound messages.

The *RapidIO* doorbell is a light weight port based transaction which can be used for in-band interrupts. A doorbell message has a 16-bit software definable field which can be used for a variety of messaging purposes between two devices.

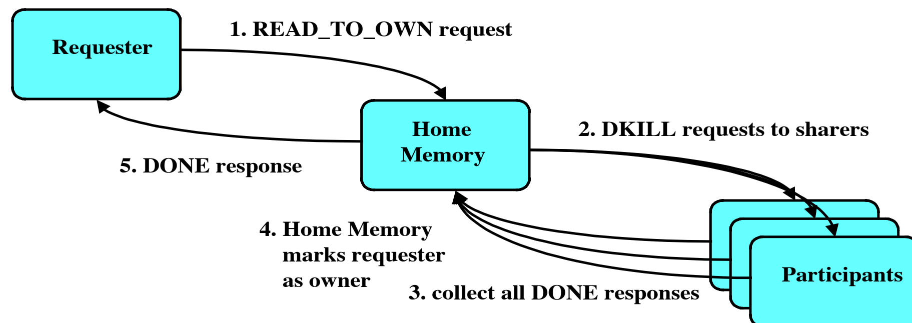
Globally Shared Memory

One of the optional protocol extensions offered in *RapidIO* is support for a globally shared distributed memory system. This means that memory may be physically located in different places in the machine yet may be shared amongst different processing elements. Typically mainstream microprocessor architectures have addressed shared memory using transaction broadcasting sometimes known as bus based snoopy protocols. These are typically implemented through a centralized memory controller for which all devices have equal or uniform access.

Massively parallel, super computers, and cluster machines which have distributed memory systems must use a different technique for maintaining memory coherency. A broadcast snoopy protocol in these machines is not efficient given the number of devices that must participate and the latency and transaction overhead involved. Therefore a different mechanism is required to keep track of where the most current copy of data resides. Such machines use more complex coherency mechanisms such as coherence directories or distributed link lists. These schemes are often referred to as cache coherent non-uniform memory access (CC-NUMA) protocols. Examples include the Stanford DASH machine[2] , the MIT Alewife Machine [1] and Scalable Coherent Interface (SCI)[3] .

For *RapidIO*, a more simple directory based coherency scheme is chosen. For this method each memory controller is responsible for tracking where the most current copy of each data element resides in the system. A directory entry is maintained for each device in the system which is participating in the coherency domain. A simple coherency state tracking of Modified, Shared, or Local (MSL) is tracked for each element. Figure 10 shows an example of a “read with intent of modification” request to a memory controller. For this example other devices in the system have shared copies of the data. The memory controller indexes the requested data in its directory and subsequently issues appropriate invalidation transactions to the sharers. At the completion of this set of operations the memory controller forwards the latest copy of the data to the requestor to complete the transaction.

FIGURE 10 The memory controller is responsible for managing coherency in the directory based scheme.



To reduce the directory overhead required, the architecture is optimized around small clusters of 16 processors known as coherency domains. With the concept of domains, it is possible for multiple coherence groupings to coexist in the interconnect as tightly coupled processing clusters.

Future Extensions

RapidIO is partitioned to support future protocol extensions. This can be done at the user level through the application of user definable transaction format types, or through future encoding in reserved fields. *RapidIO* is architected so that switch fabric devices do not need to interpret packets as they flow through, lending itself toward forward compatibility.

5 Maintenance and Error Management

RapidIO steps beyond traditional bus based interfaces by providing a rich set of maintenance and error management functions. These enable the initial system discovery, configuration, error detection and recovery methods.

Maintenance

A maintenance transaction common to all logical specifications is used to access a predefined maintenance port in each device. This port is outside of the system physical address map. The registers contain information about the device, including its capabilities and memory mapped requirements. Also included are error detection and status registers such as watchdog timer settings and pointer status. For more complex multi-bit errors, software may make use of these registers to recover or quiesce a *RapidIO* device.

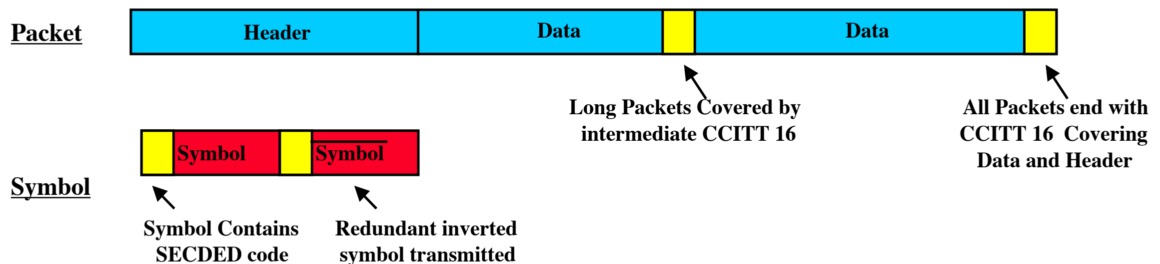
Error coverage

The mean time between failure (MTBF) and mean time to repair (MTTR) of a system are often critical considerations in embedded infrastructure applications. It is intolerable for an error to go undetected. It is also desirable to recover from these errors with mini-

mal system interruption. Because of these factors and since *RapidIO* operates at very high frequencies, it is necessary to provide strong error coverage in the protocol. Much of the *RapidIO* error coverage is handled through the physical layer specification. This allows different coverage strategies depending on the physical environment without affecting the transport and logical specifications.

For the 8/16 LP-EP physical layer the objective is to detect and recover from moderate burst errors. Because this physical layer is intended as a board level interconnect, it is assumed that the interface will not suffer from a high bit error rate as it might if it were traversing a cable. Several error detection schemes are deployed to provide coverage for 8/16 LP-EP. Packets are covered using a CCITT16 cyclic redundancy check (CRC), and control symbols are covered using a single error correct, double error detect (SECDED) code along with redundant inverted transmission as shown in Figure 11.

FIGURE 11 Error detection is accomplished through a variety of schemes.



The FRAME signal is covered against inadvertent transitions by treating it as a no-return-to-zero NRZ signal. It must stay in the new state for more than 1 clock to be considered valid.

Error Recovery

The control symbol is at the heart of the hardware error recovery mechanism. Should a packet be detected with a bad checksum, control symbols are sent to verify that both sender and receiver are still synchronized and a packet retry is issued. Should a transaction be lost, a watchdog time-out would occur and the auto recovery state machines would attempt to re-synchronize and retransmit.

If an interface fails severely, it may not be able to recover gracefully in hardware. For these extreme case, *RapidIO* hardware can generate a software trap and a higher level error recovery protocol can be run. Software may query the maintenance registers to reconstruct the current status of the interface and potentially restart it. An in-band device reset command is provided for even more extreme conditions.

6 Performance

RapidIO is intended as a processor and memory interface where both latency and bandwidth must be considered. Separate clock and frame signals are provided to eliminate

encoding and decoding latency. Source routing and a transaction priority mechanism limits blocking of packets, especially those of a critical nature. Large packets of up to 256 bytes and response-less stream write transactions move larger volumes of data with less transaction overhead.

Packet Structures

The *RapidIO* packet is structured to promote simplified construction and parsing of packets in a wider on-chip parallel interface, limiting the amount of logic operating on the narrower high frequency interface. Packets are organized in byte granularities and further in 32-bit word alignments. In this way fields land consistently in specific byte lanes on the receiving device, limiting the need for complex parsing logic.

Source Routing and Concurrency

Traditional bus-based systems, such as those using PCI, have relied on address broadcasting to alert targets of a transaction. This is effective since all devices monitor a common address bus and respond when they recognize a transaction to their address domain. Unfortunately, only one master can be broadcasting an address at a time.

Switch-based systems can employ two methods to route transactions, broadcast or source routing. For the broadcast scheme a packet is sent to all connected devices. It is expected that one and only one device actually responds to the packet. The advantage of the broadcast scheme is that the master does not need to know where the receiver resides. The disadvantage is that there is a large amount of system bandwidth wasted on the paths for which the transaction was not targeted.

To take full advantage of available system bandwidth, *RapidIO* employs source routing. This means that each packet has a source specified destination address which instructs the fabric specifically where the transaction is to be routed. With this technique only the path between the sender and receiver is burdened with the transaction. This method leaves open bandwidth on other paths for other devices in the system to communicate with each other concurrently. This scheme does not preclude the use of broadcast or multicast routing.

Packet Overhead

A performance concern in any tightly coupled intra-system interconnect is the transaction overhead required for the interface. Such overhead includes all bytes sent to complete a transaction such as arbitration, addresses, acknowledgements, error coverage, etc. Figure 12 shows some typical transactions and the number of bytes required to complete the transaction. It is important to remember that *RapidIO* is a full duplex interface and therefore the interface can be fully pipelined with several outstanding transactions at various stages of completion. Reply overhead does not contend with sending overhead. This is different from traditional buses which require turnaround cycles and arbitration phases that add to the overhead.

FIGURE 12 Transaction overhead includes the total bytes sent in each direction to complete a transaction.

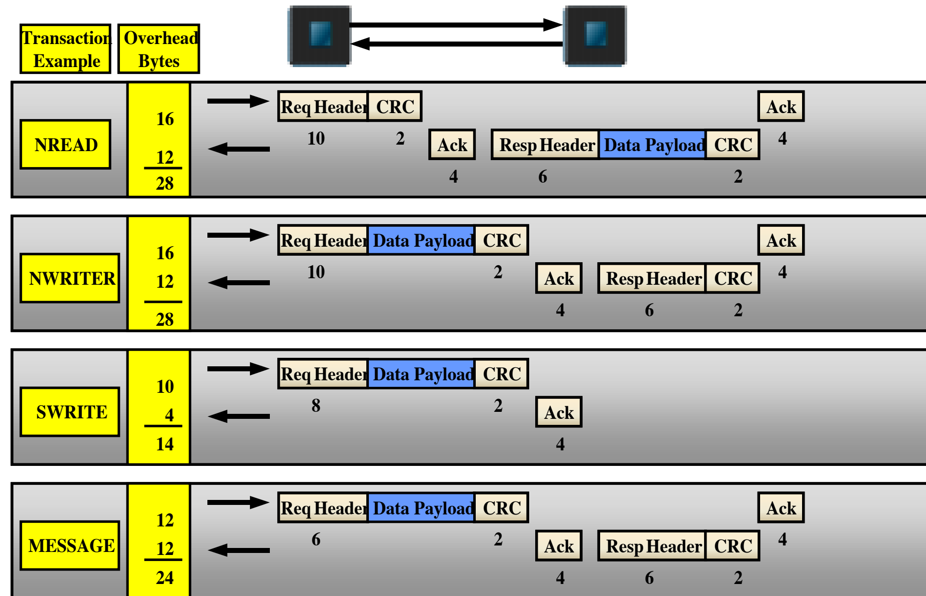


Table 2 compares the efficiencies of typical multidrop buses to *RapidIO*. For this case, a 64-bit PCI bus and a 64-bit PowerPC 60x processor bus[7] are used as examples of parallel buses. The efficiency here is defined as the total transaction overhead divided by the data payload delivered. It includes all of the bits of information that were necessary to complete the transaction such as arbitration, address, and flow control bits. In this example, *RapidIO*, PCI, and 60x are found to have very similar efficiencies for small transactions. Further, both *RapidIO* and PCI are found to have increasing efficiencies as the transaction size is increased. This is because the larger transactions require the same transaction information as the small transaction. In both cases the signal pins used to deliver control information are used to deliver data. The PowerPC bus does not multiplex control and data, but instead is highly pipelined. It allows concurrent address and data tenures thereby reducing the arbitration overhead and serialization that may occur on a multiplexed interface; of course this comes at the price of many additional signal pins.

TABLE 2. The *RapidIO* bus efficiency improves as the transaction payload size is increased.

Interface	Transaction	Signals	Clks/ Tenure	Transaction Overhead (bytes)	Bytes Transferred	Efficiency
RapidIO	32B Read	20	44	35	32	48%
	32B Write	20	44	35	32	48%
	256B Read	20	270	37	256	87%
	256B Write	20	270	37	256	87%
PPC 60x w/ MPX ext.	32B Read	133	4	34.5	32	48%
	32B Write	133	4	34.5	32	48%
	256B Read	133	32	276	256	48%
	256B Write	133	32	276	256	48%
PCI-64	32B Read	87	6	33.25	32	49%
	32B Write	87	6	33.25	32	49%
	32B Deferred Read	87	9	65.875	32	33%
	256B Deferred Read	87	37	146.375	256	64%
	256B Write	87	34	113.75	256	69%

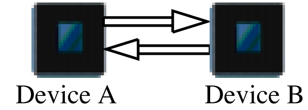
Notes:
- Efficiency is the ratio of total bytes for a tenure to data transferred.
- PPC 60x assumes a fully pipelined 64 bit data bus
- PCI and PPC 60x do not include overhead due to bus turnarounds or arbitration

Transaction Latency

In load/store environments, where processors or other devices are dependent on the results of a transaction before proceeding, latency becomes a key factor to system performance. It is assumed that the transactions of interest here are small byte or cache-line oriented transactions. Since *RapidIO* is narrower than traditional parallel buses, a transaction requires more clock cycles for data transmission; also *RapidIO* has extra overhead associated with routing and error management. However, *RapidIO* has limited to no contention or arbitration, higher operating frequency, and a concurrent reply path.

Two simple examples are presented to illustrate the latency through a typical *RapidIO* system. In the first example, (Table 3) two end point devices are connected directly together. Four transactions are shown representing data payloads of the smallest packet granularity (64-bit) and a cache-line granularity (32-byte) for both read and write cases. The latency numbers include the time to assemble a packet, transmit, receive, synchronize, and disassemble. A clock frequency of 500MHz is used where data is sampled on both edges of the clock. The reader is cautioned in comparing these numbers to traditional interfaces since the sequencing hardware necessary to generate and receive a transaction is not usually accounted for. For read transactions the important number is the total request and response time. For write transactions only the request latency is necessary.

TABLE 3. *RapidIO* latency is considered from the assembly to extraction of the packet.



Example 1: **Direct device to device**

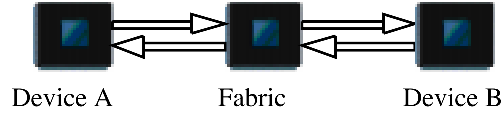
Given an 8/16 LP-EP interconnect running at 500MHz clock rate (1Gb/s/pin)
- Estimated time required to assemble, transmit and disassemble packet.
- Does not include device latency outside of RapidIO interfaces.

Width	Transaction		Total (ns)	Req / Resp Assembly Disassembly	Packet Transmit	Req / Resp Assembly Disassembly
8-bit	64 bit NREAD		80			
		Request	38	12	12	14
		Response	42	14	16	12
	64 bit NWRITER		80			
		Request	46	12	20	14
		Response	34	14	8	12
	32 byte NREAD		104			
		Request	38	12	12	14
		Response	66	14	40	12
	32 byte SWRITE		116			
		Request	68	12	42	14
16-bit	64 bit NREAD		66			
		Request	32	12	6	14
		Response	34	14	8	12
	64 bit NWRITER		66			
		Request	36	12	10	14
		Response	30	14	4	12
	32 byte NREAD		78			
		Request	32	12	6	14
		Response	46	14	20	12
	32 byte SWRITE		48			
		Request	48	12	22	14

In the second example, a switch fabric device is added between the two end points (Table 4.) It is assumed that the fabric does full store and forward. It is further assumed a very simple port routing scheme is used in the switching fabric.

TABLE 4. The addition of a cross bar fabric between devices adds additional latency

Example 2: **Device to store and forward crossbar fabric to device**



Given an 8/16 LP-EP interconnect running at 500MHz clock rate (1Gb/s/pin)
- Estimated time required to assemble, transmit and disassemble packet.
- Does not include device latency outside of RapidIO latency

Width	Transaction		Total(ns)	to / from LSRU	Req / Resp Assembly Disassembly	Packet Transmit	Fabric Latency	Packet Transmit	Req / Resp Assembly Disassembly
8-bit	64 bit NREAD		160						
		Request	76		12	12	26	12	14
		Response	84		14	16	26	16	12
	64 bit NWRITER		160						
		Request	92		12	20	26	20	14
		Response	68		14	8	26	8	12
	32 byte NREAD		208						
		Request	76		12	12	26	12	14
		Response	132		14	40	26	40	12
	32 byte SWRITE		136						
		Request	136		12	42	26	42	14
16-bit	64 bit NREAD		132						
		Request	64		12	6	26	6	14
		Response	68		14	8	26	8	12
	64 bit NWRITER		132						
		Request	72		12	10	26	10	14
		Response	60		14	4	26	4	12
	32 byte NREAD		156						
		Request	64		12	6	26	6	14
		Response	92		14	20	26	20	12
	32 byte SWRITE		96						
		Request	96		12	22	26	22	14

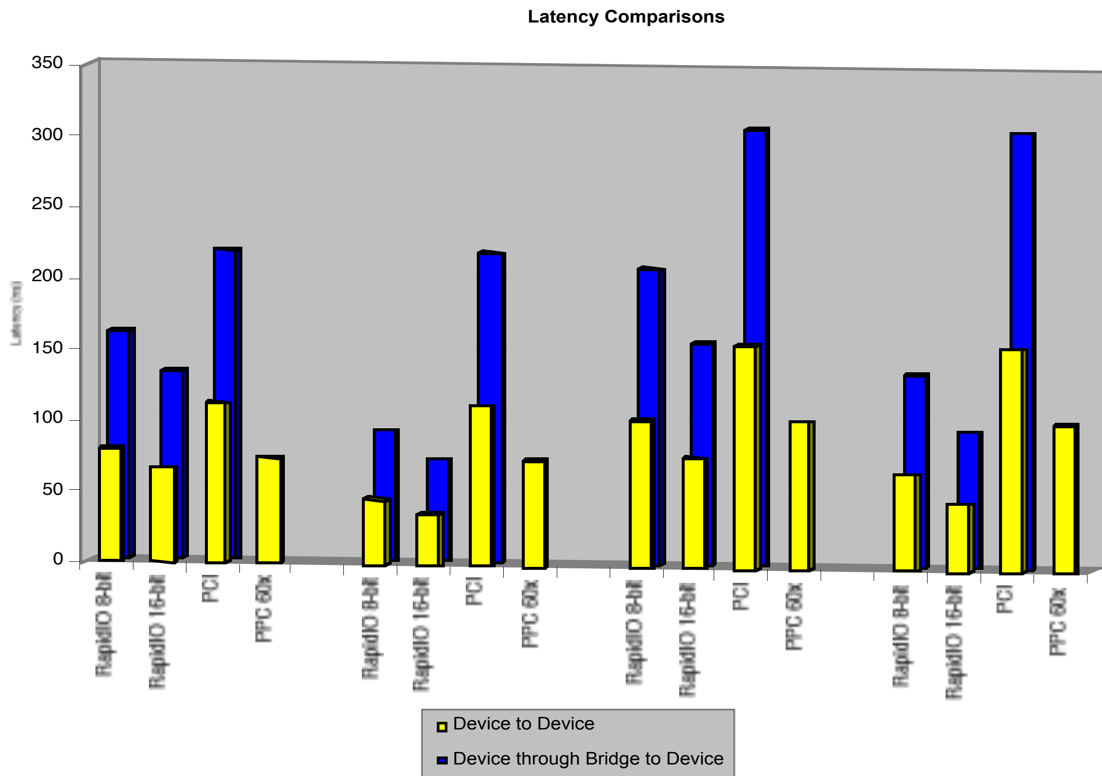
Finally Table 5 summarizes a comparison of different bus latencies. In this case RapidIO is compared to a PCI bus and PowerPC bus. The 64-bit read is shaded to contrast the different interconnect latencies with a similar transaction. For both the PCI and PowerPC bus it is assumed that an arbitration sequence must be completed before a cycle is issued. The PCI bus does not allow split transactions. If a PCI target can not service a read transaction immediately, it must retry the master. After retrying the master, the target is allowed to complete the read transaction in the background so that when the master attempts the transaction in the future the data will be available. This is called a deferred read transaction. Deferred read transactions are shown since a PCI target device can rarely service a read before being forced to disconnect.

TABLE 5. RapidIO transactional latency compares favorably to buses

Interconnect	Configuration	Transaction	Transactional Latency (ns)
RapidIO 500Mhz 8-bit	Master to Target	64-bit Read	80
16-bit		64-bit Read	66
8-bit		64-bit Write	46
16-bit		64-bit Write	36
8-bit		32-byte Read	104
16-bit		32-byte Read	78
8-bit		32-byte Write	68
16-bit		32-byte Write	48
RapidIO 500Mhz 8-bit	Master to Target Through 1 level Switch Fabric	64-bit Read	160
16-bit		64-bit Read	132
8-bit		64-bit Write	92
16-bit		64-bit Write	72
8-bit		32-byte Read	208
16-bit		32-byte Read	156
8-bit		32-byte Write	136
16-bit		32-byte Write	96
PCI Bus, 64-bit, 66MHz	Master to Target	64-bit Read	113
		64-bit Deferred Read	210
		64-bit Write	113
		32-byte Read	158
		32-byte Deferred Read	255
		32-byte Write	158
PCI Bus, 64-bit, 66MHz	Through PCI to PCI Bridge	64-bit Read	218
		64-bit Deferred Read	405
		64-bit Write	218
		32-byte Read	308
		32-byte Deferred Read	495
		32-byte Write	308
PowerPC 60x/MPX Bus, 64-bit, 100MHz	Master to Target	64-bit Read	75
		64-bit Write	75
		32-byte Read	105
		32-byte Write	105

The latency to traverse a switch fabric is considered since *RapidIO* devices will typically be connected with these devices. A comparison is made to that of a PCI-to-PCI bridged system. Figure 13 summarizes the latency comparisons in a chart format. In these examples the PowerPC bus does not have a bridge consideration. Applications have typically not implemented processor mezzanine bridges but rather bridge to an I/O interface such as PCI.

FIGURE 13 A graphical comparison of some latencies



It is conceivable that a *RapidIO*-based system with several devices could be implemented with a fairly flat topology resulting in very low transaction latency.

7 Summary

The *RapidIO* interconnect provides a robust packet-switched system level interconnect. It is a partitioned architecture that can be further enhanced in the future. It enables higher levels of system performance while maintaining or reducing the implementation costs. A *RapidIO* end point can be implemented in a small silicon footprint. A proven industry standard signaling scheme (LVDS) is used for the first physical interface. Error management includes the ability to detect multi-bit errors and survive most multi-bit and all single bit errors. The protocol overhead and transaction latency are very comparable to current bus technologies.

References

[1] A. Agarwal, et. al. "The MIT Alewife Machine: Architecture and Performance," International Symposium on Computer Architecture, 1995.

[2] D. Lenoski, et. al., "The DASH Prototype: Implementation and Performance," Computer Systems Laboratory, Stanford University.

[3] "IEEE Std. 1596-1992, IEEE Standard for Scalable Coherent Interface (SCI)," IEEE Computer Society, 1992.

[4] "IEEE Std. 1596.3-1996, IEEE Standard for Low-Voltage Differential Signals (LVDS) for Scalable Coherent Interface (SCI)," IEEE Computer Society, 31 July 1996.

[5] InfiniBand Trade Association, 5440 SW Westgate Dr, Suite 217, Portland OR, 97221.

[6] "PCI Local Bus Specification, Rev.2.2," PCI Special Interest Group 1999.

[7] "PowerPC Microprocessor Family: The Bus Interface for 32-Bit Microprocessors," G522-0291-00, Motorola Inc. 1997.