RapidIO: The Interconnect Architecture for High Performance Embedded Systems

RapidIO™

The Embedded System Interconnect

Table of Contents

Abstract 1

Introduction 2

- Why RapidIO? 2
- Interconnect Landscape 4
- Where Will It be Used? 5
- Philosophy 6

RapidIO Protocol Overview 8

- Packets and Control Systems 8
- Packet Format 9
- Transaction Formats and Types 10
- Message Passing 10
- Globally Shared Memory 11
- Future Extensions 11
- Flow Control 11

Physical Interface 13

- Parallel Electrical Interface 13
- The Serial RapidIO Controller 14
- Link Protocol 14
- Enhanced Flow Control 14
- PCS and PMA Layers 14
- Electrical Interface 15

Maintenance and Error Management 16

- Maintenance 16
- System Discovery 16
- Error Coverage 16
- Error Recovery 16

Performance 17

- Packet Structures 17
- Source Routing and Concurrency 17
- Packet Overhead 18
- Bandwidth 19
- Operation Latency 19

Summary 20

RapidIO™

The Embedded System Interconnect

Dan Bouvier
System Architecture Manager
Motorola, Semiconductor Product Sector
Chair, RapidIO Steering Committee

Abstract

This paper describes RapidIO, a high performance, low pin count, packet switched system level interconnect architecture. The interconnect architecture is an open standard which addresses the needs of a wide variety of embedded infrastructure applications. 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 with performance levels ranging from 1 gigabit per second to 60 gigabits per second. Two families of RapidIO interconnects are defined: A parallel interface for high performance microprocessor and system connectivity and a serial interface for serial back-plane, DSP and associated serial control plane applications. The serial and parallel forms of RapidIO share the same programming models, transactions, and addressing mechanisms. Supported programming models include basic memory mapped I/O transactions, port-based message passing, and globally shared distributed memory with hardware-based coherency. RapidIO also offers a very high degree of error management and provides a well-defined architecture for recovering from and reporting transmission errors. The RapidIO interconnect is defined as a layered architecture which allows scalability and future enhancements while maintaining backward compatibility.

Introduction

Computer and embedded system development continues to be burdened by divergent requirements. On one hand the system performance must continue to increase at a nearly exponential rate, while on the other hand the system cost must remain constant or even decrease. Several applications, such as those found in networking and telecommunications infrastructure equipment, are also burdened with increasing capability and reliability requirements.

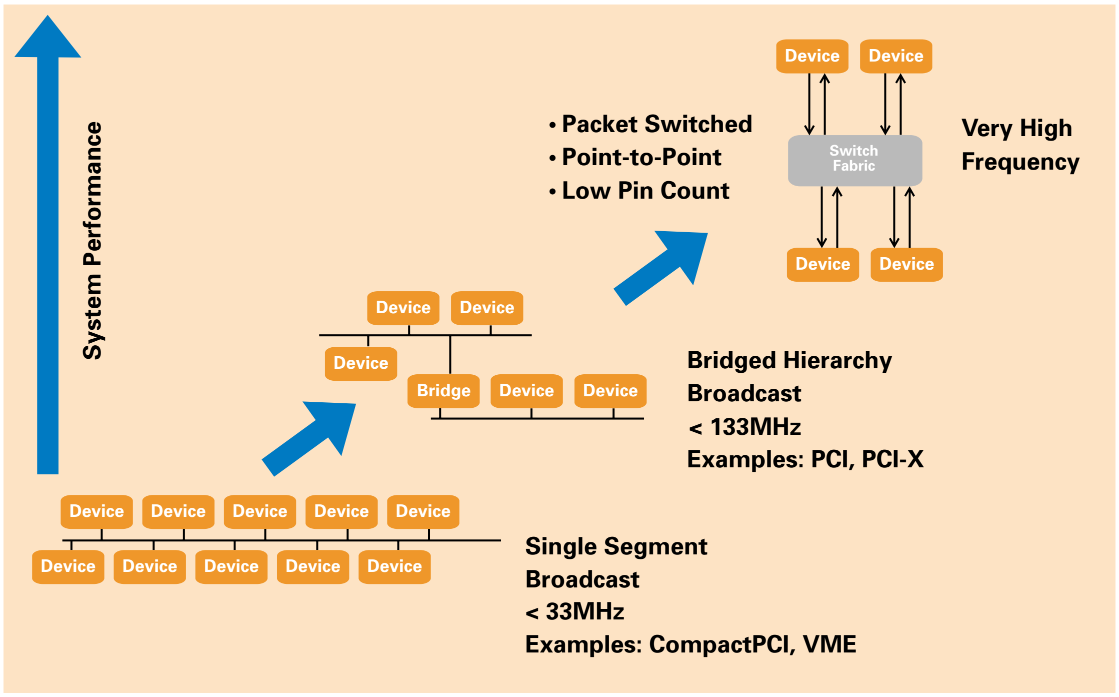
The connections between microprocessors and peripherals have traditionally been composed of a hierarchy of buses (Figure 1). Devices are placed at the appropriate level in the hierarchy according to the performance level they require. Low performance devices are placed on lower performance buses, which are bridged to the higher performance buses so as to not burden the higher performance devices. Bridging may also be used to address legacy interfaces.

The need for higher levels of bus performance is driven by two key factors. First, the need for higher raw data bandwidth to support higher peripheral device performance requirements, second the need for more system concurrency. The overall system bandwidth requirements have also increased because of the increasing use of DMA, smart processor-based peripherals and multiprocessing in systems.

Why RapidIO?

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, traces on boards and larger connectors, resulting in a higher product cost and a reduction in the number of interfaces a system or device can provide.

Figure 1



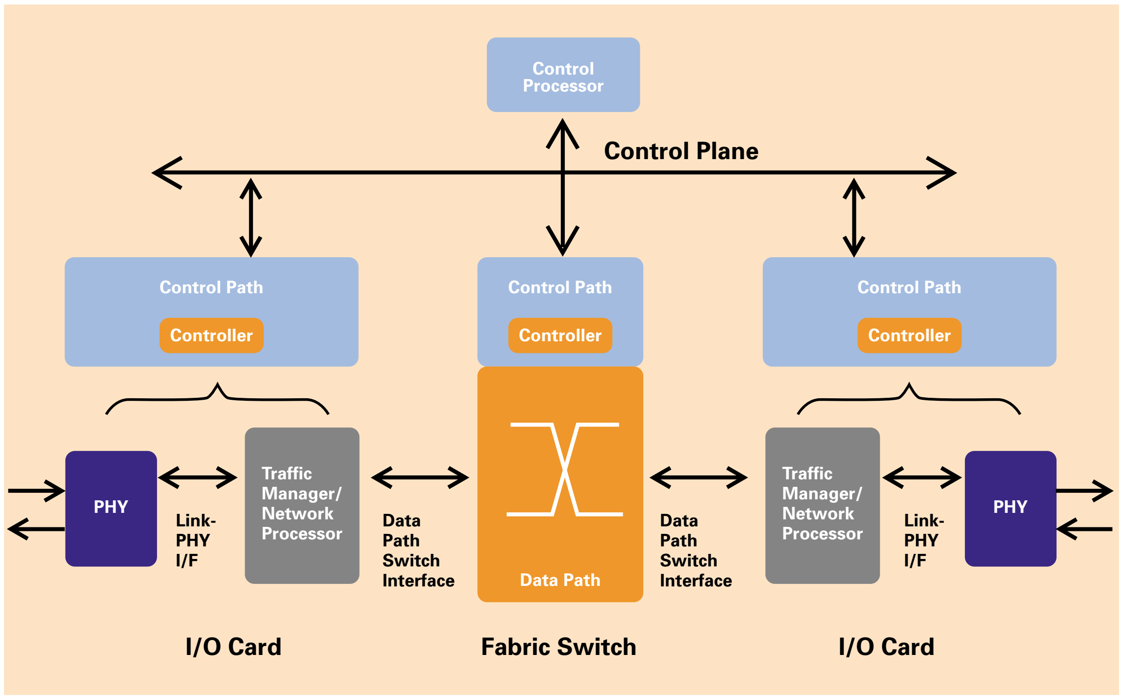
Higher system performance levels require adoption of point-to-point switched interconnects.

Worsening the situation 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.

Developed as an open standard, RapidIO is designed to addresses the needs of present and future high performance embedded systems. In embedded system applications, RapidIO has just a limited to no impact on the software infrastructure that operates over it. It can be implemented with relatively few transistors and it offers low operation latency and high bandwidths.

High performance embedded equipment often contains separate control plane and data-forwarding planes as shown in Figure 2. The data-forwarding plane is responsible for moving data through the system while the control plane manages the data movement. RapidIO provides all of the necessary attributes to be useful in the control plane. RapidIO, in some applications, may also be useful as the basis for data-forwarding plane implementations.

Figure 2



RapidIO is targeted toward control plane applications in network equipment.

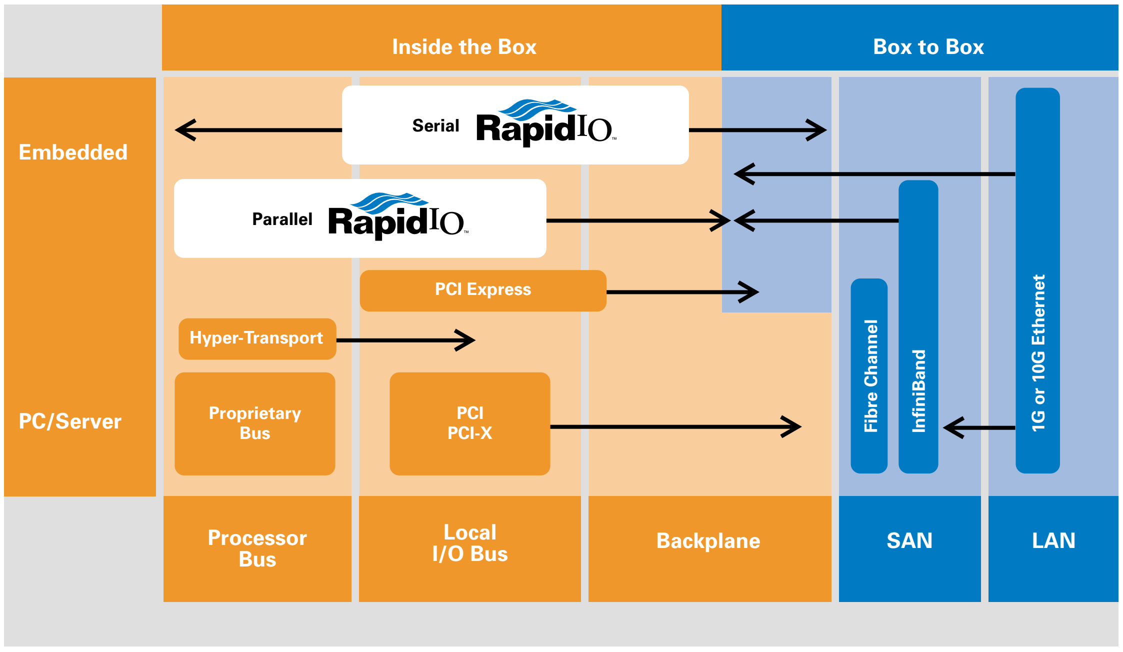
Interconnect Landscape

RapidIO is categorized as an intra-system interconnect as shown in Figure 3. Specifically, RapidIO is targeted at the intra-system interconnect applications in the high performance embedded equipment market. This market has distinctly different requirements from the desktop and server computer spaces. The embedded market has historically been served by a number of different vendors. The openness of the RapidIO Trade Association is well suited to this environment. The RapidIO Trade Association counts among its current members nearly two dozen leading vendors of microprocessors, DSPs, FPGAs, ASICs and embedded memories.

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. Operations through a SAN are typically handled through software drivers using message channels or remote direct memory access (RDMA). InfiniBand competes more directly with Fibre Channel and Ethernet-based system area networking technologies such as iSCSI.

Hyper-Transport and PCI Express share some common characteristics with RapidIO but are more appropriately described as point-to-point versions of PCI. While they maintain compatibility with the PCI interconnect architecture from a software viewpoint, which is very important for desktop computer markets, they do not offer the scalability, robustness, and efficiency required by embedded systems developers.

Figure 3



RapidIO connects processors, memory, and peripherals within a subsystem or across a backplane.

Where Will It be Used?

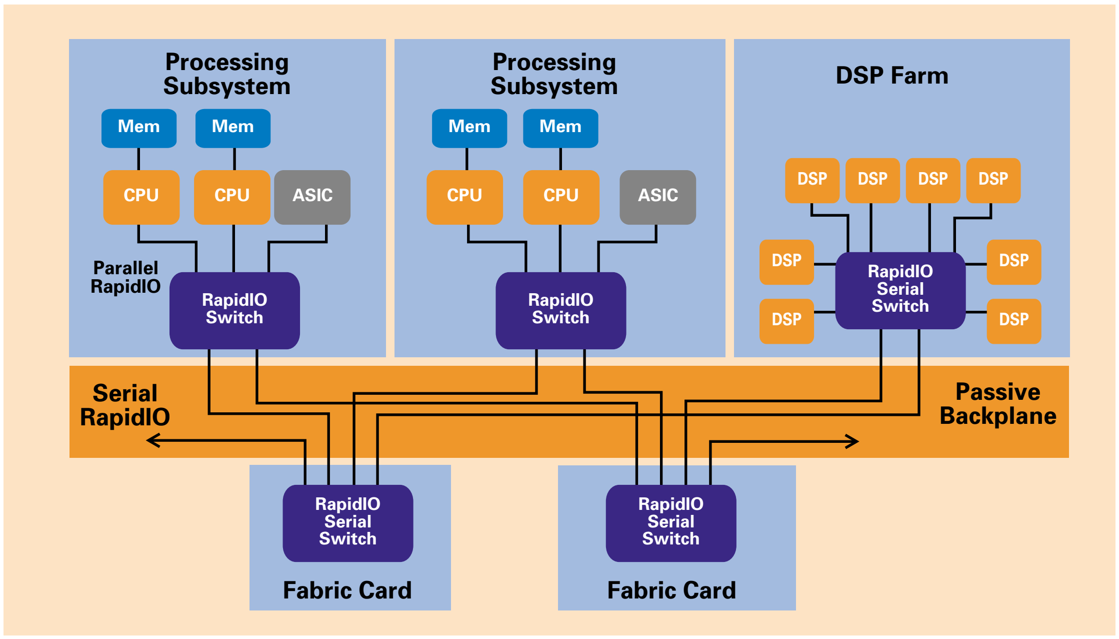
The RapidIO interconnect is targeted for use in environments where multiple devices must work in a tightly coupled architecture. Figure 4 illustrates a generic system containing memory controllers, processors, and I/O bridges connected using RapidIO switches.

In computing applications the PCI bus is frequently used. Enterprise storage applications, for example, use PCI to connect multiple disk channels to a system. As disk throughput has increased so has the need for higher system throughput. To meet the electrical requirements of higher bus frequencies, the number of supportable devices per bus segment must be decreased. In order 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 only within a tree-shaped hierarchy that increases system latency and cost as more PCI devices are added to the system.

RapidIO can be used for transparent PCI-to-PCI bridging allowing for a flattened architecture utilizing fewer pins and offering greater transmission distances. Figure 5 shows one such bridged system. In this example, several PCI-X bridges are connected together using RapidIO switches. An InfiniBand or Ethernet Channel Adapter can be provided if the system is to become part of a wider System Area Network.

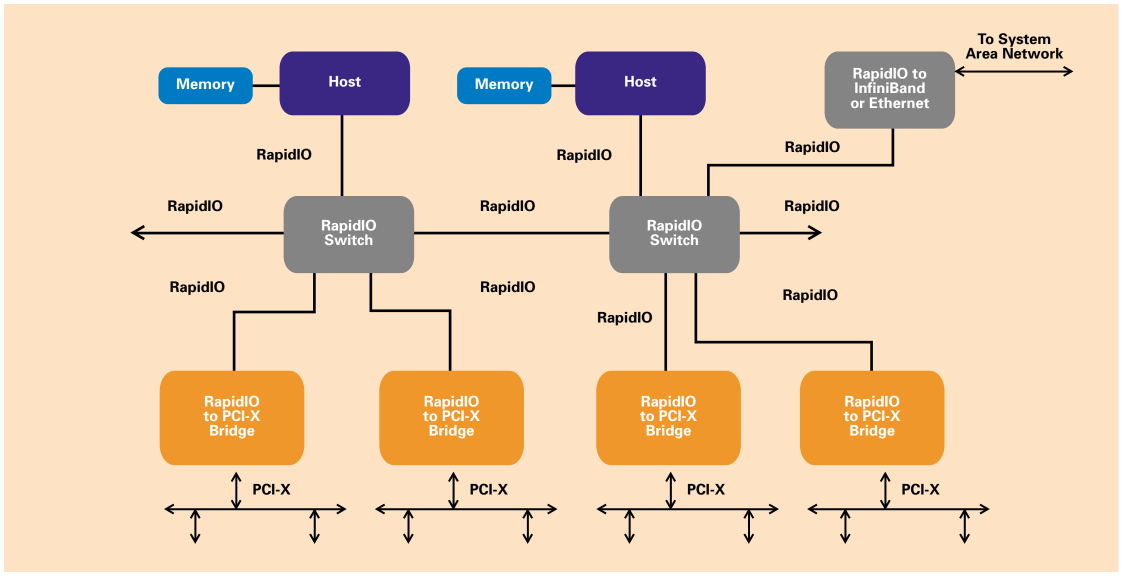
Many systems require the partitioning of functions into field replaceable units. These printed circuit boards have traditionally been interconnected using multi-drop interfaces like VME or PCI. Higher system level performance can be achieved by using RapidIO. RapidIO is well suited for hot swap applications, as well, because RapidIO's point-to-point topology enables the removal of devices with little or no electrical impact to neighboring devices or subsystems.

Figure 4



RapidIO allows a variety of devices to be interconnected.

Figure 5



RapidIO as a PCI bridging fabric reduces device pin count.

Philosophy

The architecture of RapidIO was driven by certain principal objectives:

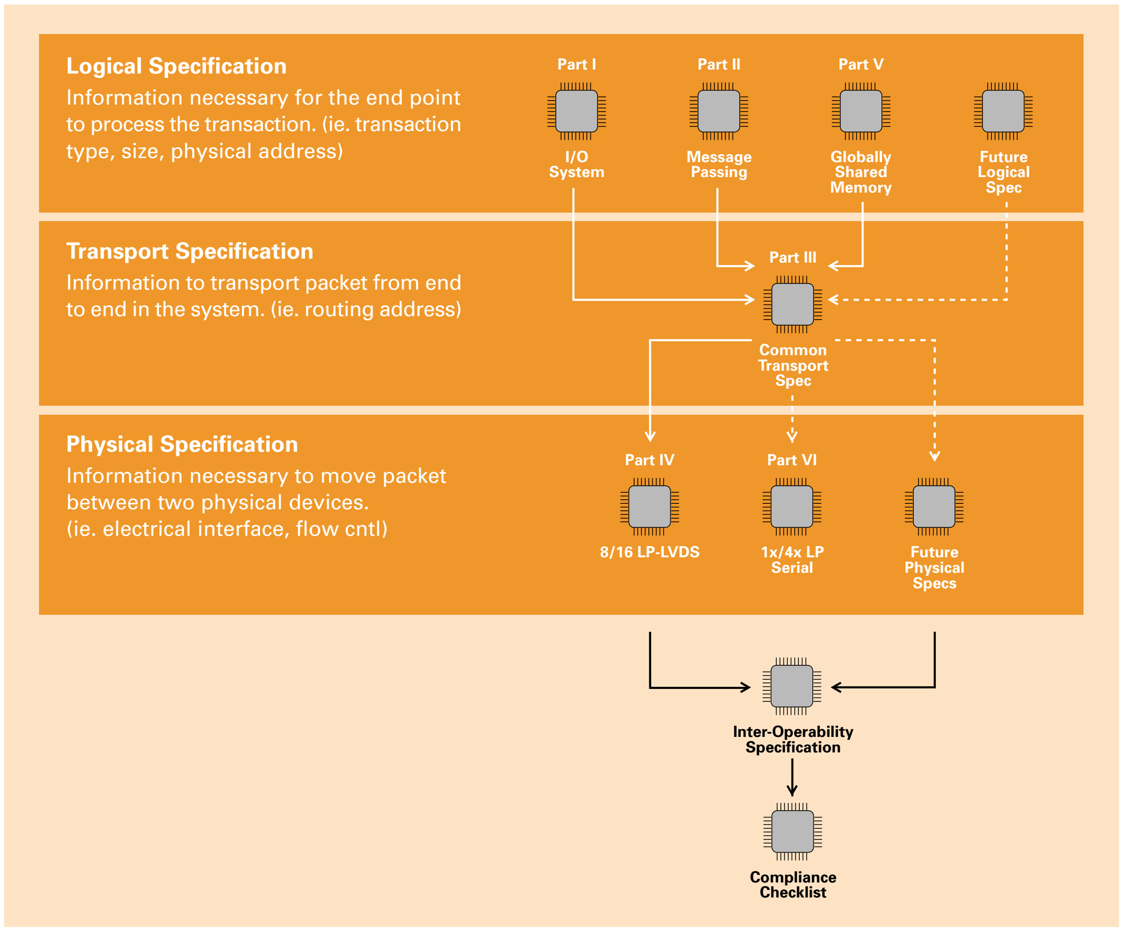
- *Focus on applications that connect devices which are within the box or chassis:*
  As an example, since devices are not intended to be connected with several meters of cable, the interface can use a simple signaling and flow control scheme.
- *Limit the impact on software:*
  Many applications are 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 typical application dependence 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 many 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 commonly available FPGA technology.

- ***Leverage 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. Further, the interface should leverage existing industry-standard I/O technology and the support infrastructure associated with such standards.

- ***RapidIO is specified in a 3-layer architectural hierarchy (Figure 6):*** The logical layer specification defines the overall protocol and packet formats. This is the information necessary for end points to initiate and complete a transaction. The transport layer specification provides the necessary route information for a packet to move from end point to end point. The physical layer specification describes the device level interface specifics such as packet transport mechanisms, flow control, electrical characteristics, and low-level error management.

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

Figure 6


Logical Specification
Information necessary for the end point to process the transaction. (ie. transaction type, size, physical address)
Part I
I/O System
Part II
Message Passing
Part V
Globally Shared Memory
Future Logical Spec
Transport Specification
Information to transport packet from end to end in the system. (ie. routing address)
Part III
Common Transport Spec
Physical Specification
Information necessary to move packet between two physical devices. (ie. electrical interface, flow cntl)
Part IV
8/16 LP-LVDS
Part VI
1x/4x LP Serial
Future Physical Specs
Inter-Operability Specification
Compliance Checklist

The RapidIO specifications hierarchy allows flexibility in usage and future specification development.

RapidIO Protocol Overview

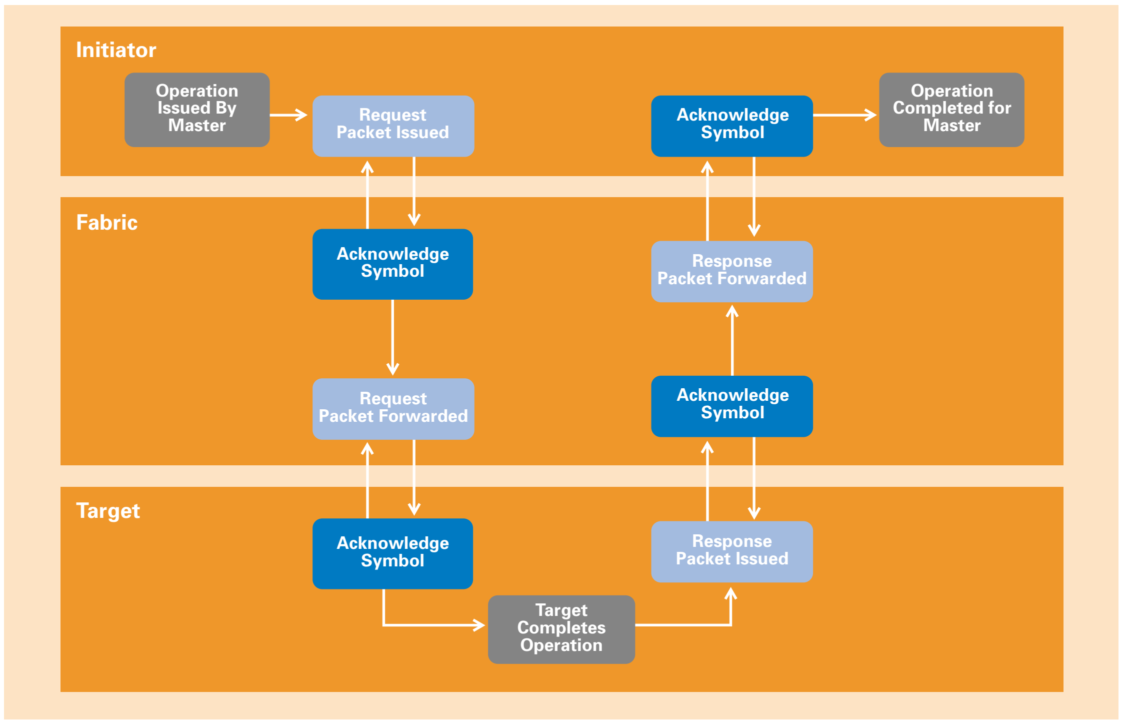
Packets and Control Symbols

RapidIO operations are based on request and response transactions. Packets are the communication element between end point devices in the system. A master or initiator generates a request transaction, which is transmitted to a target. The target then generates a response transaction back to the initiator to complete the operation. The RapidIO transactions are encapsulated into packets, which include all of the necessary bit fields to ensure reliable delivery to the targeted end point. 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 transactions progress through the system.

In this example, the initiator begins an operation in the system by generating a request transaction. The request packet is sent to a fabric device, which is acknowledged with a control symbol. The packet is forwarded to the target through the fabric device. The target completes the request transaction and generates a response transaction. 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 operation is considered complete.

Figure 7



Transactions are constructed with request and response packet pairs.

Packet Format

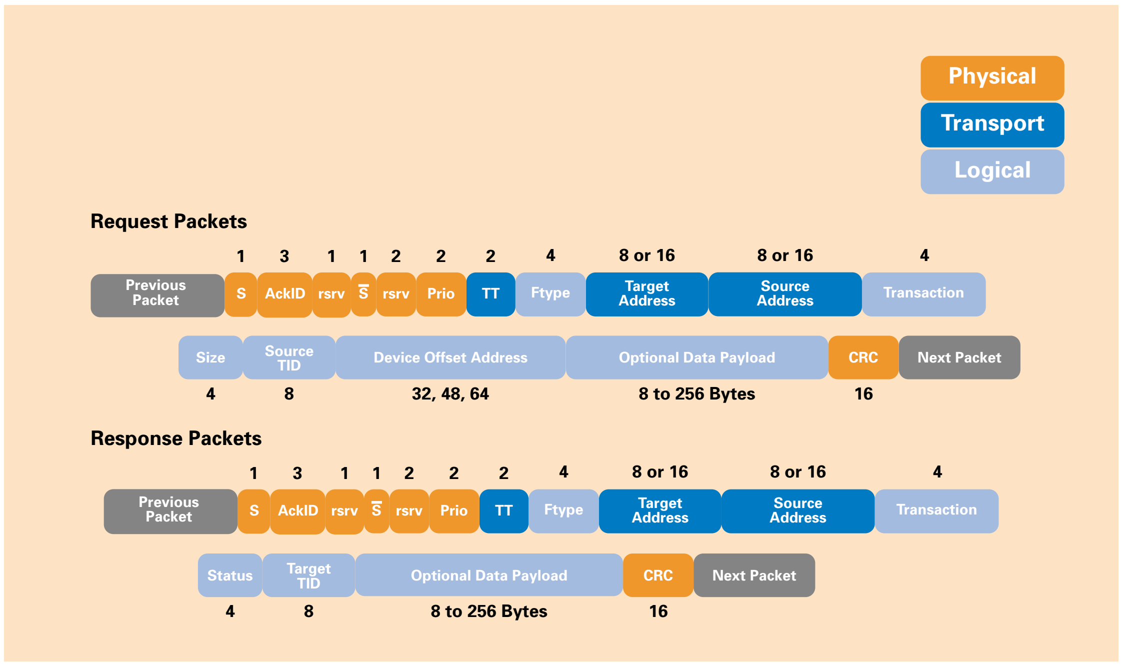
The RapidIO packet is comprised of fields from the three-level specification hierarchy. Figure 8 shows typical request and response packets. Certain fields are context dependent and may not appear in all packets.

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. The "PRIO" field indicates 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 "Status" field indicates whether the transaction was successfully completed. The "TargetTID" corresponds to the request packet source transaction ID.

The Serial RapidIO physical layer bits are slightly different than those used in Parallel RapidIO packet headers.

Figure 8



The RapidIO packet contains fields from the specification hierarchy.

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

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 embedded systems this protocol is managed through software mechanisms. If the memory space is accessible to multiple devices, then locks or semaphores are used to grant access to one device. In other cases, processing elements may only have access to locally owned memory space. In these “shared nothing” machines, a mechanism is needed to pass data from the memory of one processing element to another. This can be done using software visible mailbox hardware.

RapidIO provides a useful message passing facility. The RapidIO message passing protocol extensions describe transactions that 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 determines what to do with the message after it arrives. A RapidIO message can range in length from zero to 4096 bytes. A receiver can have 1 to 4 addressable message queues to capture inbound messages.

The RapidIO doorbell is a lightweight 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.

Table 1

Functions	Transaction Types
I/O non-coherent functions	NREAD (read non-sharable memory)
	NWRITE, NWRITE_R, SWRITE (write non-sharable memory)
	ATOMIC (read-modify-write to non-sharable memory)
Port-based functions	DOORBELL (generate an interrupt)
	MESSAGE (write to port)
System support functions	MAINTENANCE (read or write configuration, control, and status registers)
User defined functions	Open for application specific transactions
Cache coherence functions	READ (read globally shared cache line)
	READ_TO_OWN (write globally shared cache line)
	CASTOUT (surrender ownership of globally shared cache line)
	IKILL (instruction cache invalidate)
	DKILL (data cache invalidate)
	FLUSH (return globally shared cache line to memory)
	IO_READ (read non-cacheable copy of globally shared cache line)
OS support functions	TLBIE (TLB invalidate)
	TLBSYNC (force completion of TLB invalidates)

RapidIO has a rich set of Transaction Formats.

Globally Shared Memory

One of the 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 cached amongst different processing elements.

For RapidIO, a directory-based coherency scheme is specified to support this. 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 participating in the coherency domain. Simple coherency states of Modified, Shared, or Local (MSL) are tracked for each element. Figure 9 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.

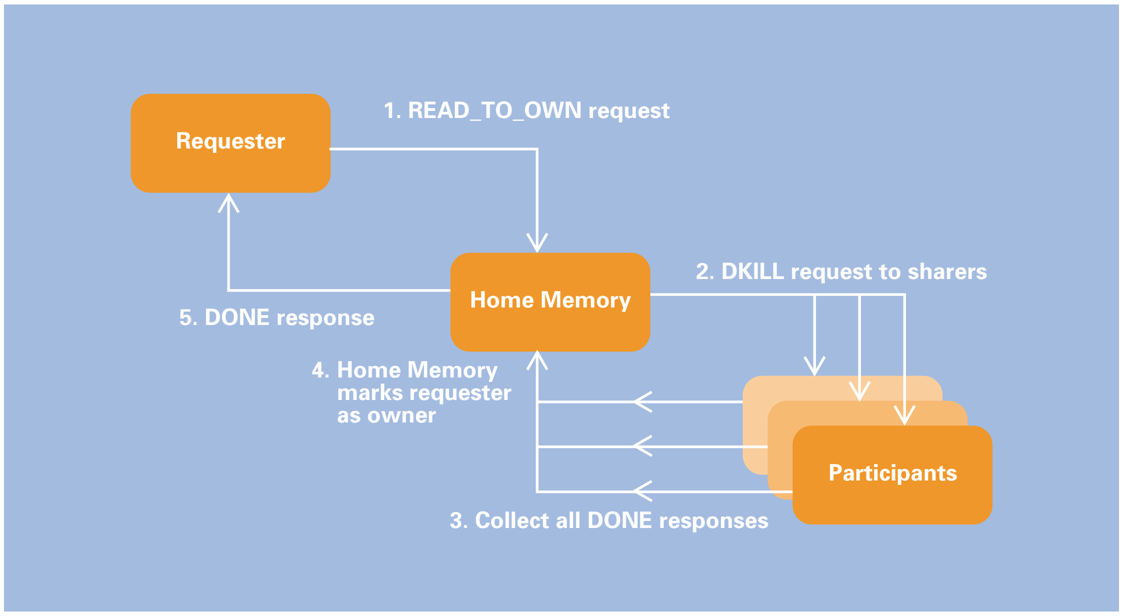
Future Extensions

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 of reserved fields. RapidIO is architected so that switch fabric devices do not need to interpret packets as they flow through, making future forward compatibility much easier to achieve.

Flow Control

Flow control is an important aspect of any interconnect. The objective 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 higher priority transactions take precedence over lower priority ones. With switch-based interconnects, transactions enter at different points in the system and there is no opportunity to employ a centralized arbitration mechanism. This creates the need for alternate methods to manage transaction flow in the system.

Figure 9


Requester
1. READ_TO_OWN request
Home Memory
2. DKILL request to sharers
5. DONE response
4. Home Memory marks requester as owner
Participants
3. Collect all DONE responses

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

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. Each RapidIO packet carries with it a transaction priority. Each transaction priority is associated with a transaction request flow. There currently are three transaction request flows defined. Transaction request flows allow higher priority requests to move ahead of lower priority requests. Transaction ordering rules are managed within a transaction request flow but not between flows.

Transaction request flows make it possible for system implementers to structure traffic through the machine to guarantee deterministic behavior. For example, an application may require that certain time critical operations be delivered with a guaranteed latency. This data can be identified as a higher priority request flow. To handle critical data, the switch fabric may contain queuing structures that allow the movement of higher priority traffic ahead of lower priority traffic. These queuing structures may include interval timers to guarantee that flows from different source ports targeted to a common destination port are given opportunity to move critical data through – resulting in isochronous delivery behavior for certain data classes.

RapidIO also describes three types of flow control mechanisms; retry, throttle, and credits. The retry mechanism is the simplest mechanism and is required not only for flow control but also as a component of hardware error recovery. A receiver, unable to accept a packet because of a lack of resources or because the received packet was corrupt, may respond with a retry control symbol. The sender will retransmit the packet.

The throttle mechanism makes use of the idle control symbol. Idle control symbols may be inserted into a packet 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 sending device 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 that 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 available buffer space. This information enables switch fabric devices to discover downstream blocking conditions and to make better packet output selections.

Physical Interface

The RapidIO logical packet description 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. The first physical interface considered and defined is known as the 8- or 16-bit link protocol end point specification (8/16 LP-LVDS). This specification is defined as having 8 or 16 data bits in each direction along with clock and frame signals in each direction (Figure 10).

The 8/16 LP-LVDS interface is a source synchronous interface. This means that a clock is transmitted along 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 into its local clock domain.

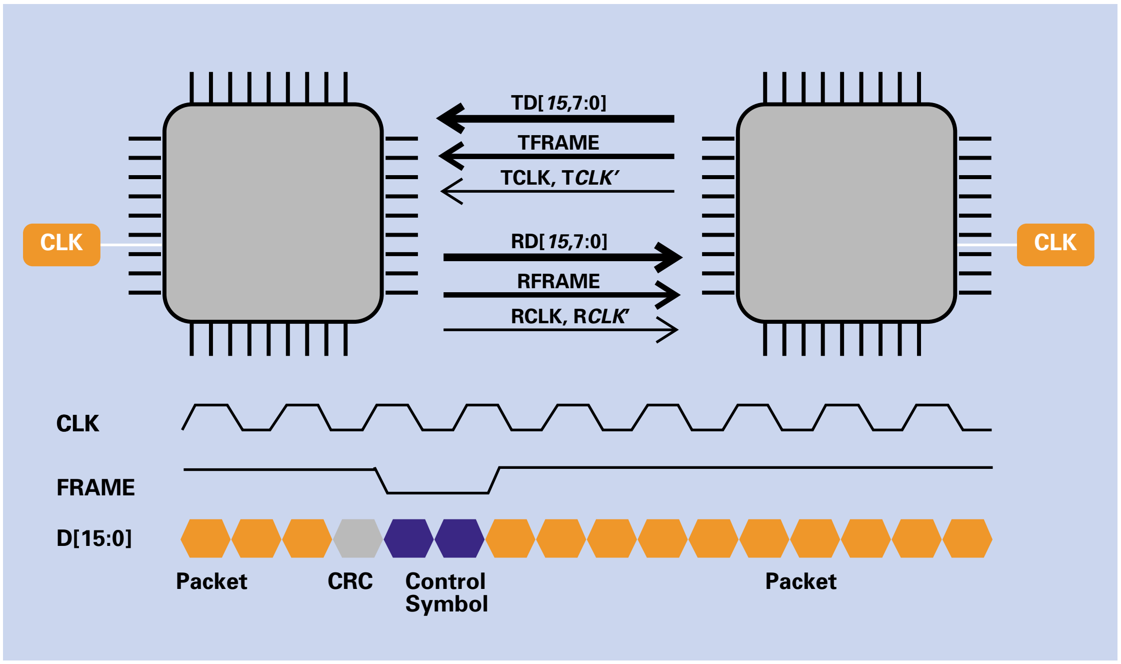
The FRAME signal is used to delineate the start of a packet or control symbol. It operates as a non return to zero (NRZ) signal where any transition marks an event.

Parallel Electrical Interface

RapidIO adopts the IEEE 1596.3 Low Voltage Differential Signals (LVDS) standard as the basis for the electrical interface. LVDS is a low swing (250 to 400 mV) constant current differential signaling technology targeted toward short-distance printed circuit board 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. At higher frequencies the RapidIO specification recommends additional source termination to reduce signal reflection effects.

The target frequencies of operation are from 250MHz to more than 1GHz. Data is sampled on both edges of the clock. The resulting bandwidth scales to 4 GByte/sec for the 8-bit and 8 GBytes/sec for the 16-bit interfaces.

Figure 10



The RapidIO 8/16 LP-LVDS physical layer provides for full duplex communications between devices.

The Serial RapidIO Controller

Since the Serial RapidIO specification is only defined in the physical layer (RapidIO technology defines the physical layer as the electrical interface and device-to-device link protocol), most of the controller remains the same. As a result, much of the design knowledge and verification infrastructure are preserved. This eases system level switching between parallel and serial links.

During the initial development stages of the Serial RapidIO specifications the designers decided to preserve as many of the concepts found in the RapidIO parallel specification as feasible. The parallel specification includes the concept of packets and in-band control symbols. These were delineated and differentiated by both a separate frame signal and an “S” bit in the header. In the serial link specification this delineation is accomplished using spare characters (“K-codes”) found in the 8B / 10B encoding technique. In this way, the sending device indicates to the receiving link partner the start of a packet, end of packet, or embedded control symbol using these codes.

Link Protocol

A unique feature of parallel RapidIO technology is that packet transmission is managed on a link-by-link basis. In the past, with synchronous buses, a mastering device had to exchange handshake signals with the target device. These signals indicated whether a transaction was acknowledged and accepted by the target device.

With a source synchronous interface like the RapidIO specification defines, it is not practical to rely on a synchronous handshake since the receive port of a link is decoupled from the sending port. Therefore, many interconnects have ignored this issue and rely on an end-to-end handshake to guarantee delivery. However, this has the disadvantage of not allowing precise detection and recovery of errors and forces far longer feedback loops for flow control.

To address this issue RapidIO technology uses embedded control symbols for link level communication between devices. Packets are explicitly tagged between each link with a sequence number otherwise known as AckID. The AckID is independent of the end-to-end transaction ID. Using control symbols, the receiving device indicates for each packet whether it has been received along with additional buffer status information. Receiving devices can immediately detect a lost packet, and through control symbols, can re-synchronize with the sender and recover it without software intervention. The receiving device then forwards the packet to the next switch in the fabric, and so on, until the final target of the packet is reached.

Enhanced Flow Control

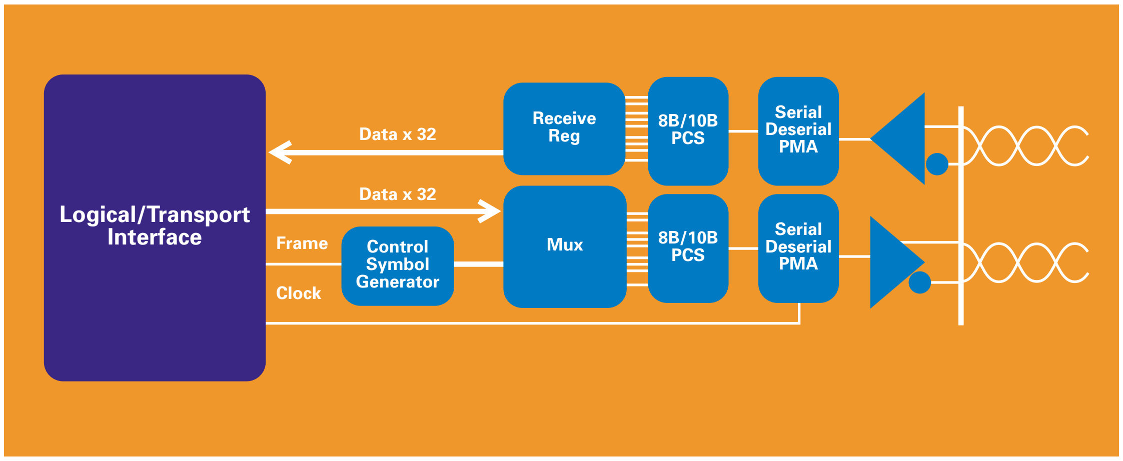
Serial RapidIO technology allows for the longer transmission distances and thus longer loop latencies in providing feedback between the receiver and transmitter on a link. Consequently the Serial physical layer specification increases the number of “AckID” values from 8 to 32. Additionally, the Serial RapidIO specification now defines a Transmitter-Controlled flow control scheme whereby the receiving port provides information to its link partner about the amount of buffer space it has available. With this information, the sending port can allocate the use of the receive buffers of the receiving port. The sending port does not have to be concerned that one or more of the packets shall be forced to retry.

PCS and PMA Layers

The Serial RapidIO specification uses a Physical Coding Sublayer (PCS) and Physical Media Attachment (PMA) sublayer to organize packets into a serial bit stream at the sending side and to extract the bit stream at the receiving side (this terminology is adopted from IEEE 802.3).

Besides encoding for transmission and decoding for reception, the PCS function is also responsible for idle sequence generation, lane striping, lane alignment, and de-striping on reception. The PCS uses 8B / 10B encoding for transmission over the link.

Figure 11



The RapidIO Serializer/Deserializer function integrates directly with the logic and transport layers.

The PCS layer also provides the mechanisms for automatically determining the operational mode of the port as either 1-lane or 4-lane, and provides for clock difference tolerance between the sender and receiver without requiring flow control.

The PMA function is responsible for serializing 10-bit parallel code-groups to / from a serial bit stream on a lane-by-lane basis. Upon receiving data, the PMA function provides alignment of the received bit stream to 10-bit code-group boundaries, independently on a lane-by-lane basis. It then provides a continuous stream of 10-bit code-groups to the PCS, one stream for each lane. The 10-bit code groups are not observable by layers higher than the PCS.

Electrical Interface

Serial RapidIO technology uses differential current steering drivers based on those defined in the 802.3 XAUI specifications. This signaling technology was developed to drive long distances over backplanes.

For the Serial RapidIO technology, two transmitter specifications were designated: a short run transmitter and a long run transmitter. The short run transmitter is used mainly for chip-to-chip connections either on the same printed circuit board or across a single connector such as that for a mezzanine card. The minimum swings of the short run specification reduce the overall power used by the transceivers. A user can further reduce the power by lowering the termination voltages.

The long run transmitter uses larger “voltage swings” that are capable of driving across backplanes. This allows a user to drive signals across two connectors and common printed circuit board material.

To ensure interoperability between drivers and receivers of different vendors and technologies, AC coupling must be used at the receiver input.

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. A discovery mechanism with certain similarities to PCI is used to find devices in the system and configure them.

Maintenance

A maintenance operation is supported by all RapidIO devices and is used to access predefined maintenance registers within each device. 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. For more complex multi-bit errors, software may make use of these registers to recover or shut down a RapidIO device.

System Discovery

Through the use of the maintenance operation, a RapidIO host may traverse a system and configure each device in the network. Since RapidIO allows complex system topologies such as meshes, a mechanism is provided so that the host can determine if it has been to a node before. For high reliability applications, RapidIO allows more than one host to configure the machine.

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. RapidIO applications require reliable packet delivery. It is intolerable for an error to go undetected or for a packet to be dropped. Further, once detected, it is desirable to recover from these errors with minimal system interruption. Because of these factors and since RapidIO operates at very high frequencies, it is necessary to provide strong error coverage. 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.

Several error detection schemes are deployed to provide coverage. Packets are covered using a CCITT16 cyclic redundancy check (CRC). Packets are transmitted and acknowledged following a strict sequence using the “AckID” field.

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 occurs and the auto recovery state machines attempt to re-synchronize and retransmit.

If an interface fails severely, it may not be able to recover gracefully in hardware. For this extreme case, RapidIO hardware can generate interrupts so that system software can invoke a higher level error recovery protocol. Software may query the maintenance registers to reconstruct the current status of the interface and potentially restart it. An in-band device reset control symbol is provided.

Performance

RapidIO is intended as a processor and memory interface where both latency and bandwidth must be considered. RapidIO may also be used to carry data streams in which deterministic delivery behavior is required. Separate clock and frame signals are provided to eliminate encoding and decoding latency. Source routing and transaction priority tagging limit the blocking of packets, especially those of a critical nature. Large data payloads of up to 256 bytes and response-less stream write operations 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 assembly and parsing logic.

Source Routing and Concurrency

Traditional bus-based systems, such as those using PCI, have relied on address broadcasting to alert targets of an operation. 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.

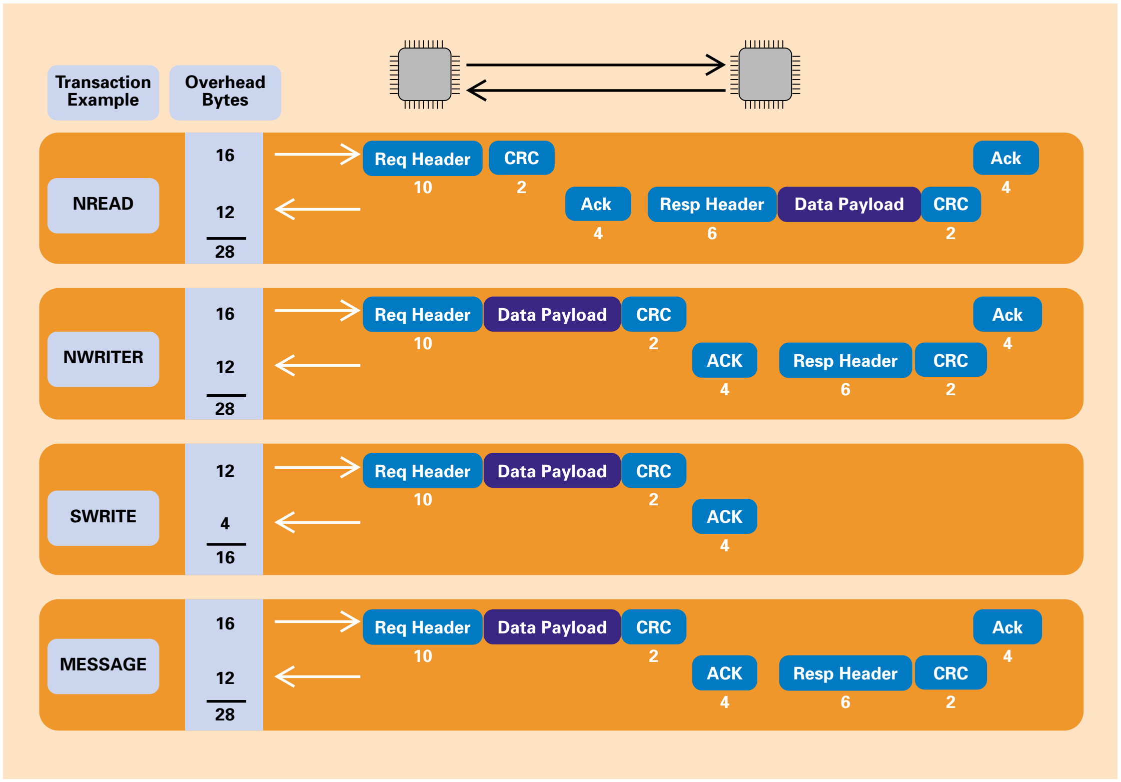
Switch-based systems can employ two methods to route transactions: broadcast or source routing. For the broadcast scheme a packet is sent to all connected devices. It is expected that one and only one device actually will respond to the packet. The advantage of the broadcast scheme is that the master 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 that 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. The switch devices make decisions on a much smaller number of bits resulting in small route tables and lower latency. 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 RapidIO operations and the number of bytes required to complete each operation. It is important to remember that RapidIO is a full duplex interface and therefore the interface can be fully pipelined with several outstanding operations 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



Operation overhead includes the total bytes sent in each direction to complete an operation.

Bandwidth

The RapidIO 8/16LP-LVDS specification presently defines four operating frequencies and two widths. The Serial RapidIO specification defines three operating frequencies and two widths (x1, x4). Together RapidIO offers sustained data payload bandwidths ranging from just under a gigabit per second for the 1.25 GHz 1x serial interface up to 60 gigabits per second for the 1 GHz x16 parallel interface. Table 2 shows the operating points for both serial and parallel RapidIO with peak and sustained full duplex data bandwidth.

Operation Latency

In load/store environments, where processors or other devices are dependent on the results of an operation before proceeding, latency becomes a key factor to system performance. It is assumed that the operations of interest here are small byte or cache-line oriented operations. Since RapidIO is narrower than traditional parallel buses, an operation requires more clock cycles for data transmission; also RapidIO has extra overhead associated with routing and error management. However, RapidIO being a point-to-point technology with separate send and receive paths has virtually no contention or arbitration overhead, can operate at higher frequencies and can send and receive data simultaneously. The combination of these factors leads to operation latency comparable to existing parallel buses.

Table 2

Parallel RapidIO

Clock Rate	8-bit Mode			16-bit Mode		
	PEAK	Sustained 32 byte Op	Sustained 256 byte Op	PEAK	Sustained 32 byte Op	Sustained 256 byte Op
250 MHz	8 Gb/s	4 Gb/s	7.5 Gb/s	16 Gb/s	8 Gb/s	15 Gb/s
500 MHz	16 Gb/s	8 Gb/s	15 Gb/s	32 Gb/s	16 Gb/s	30 Gb/s
750 MHz	24 Gb/s	12 Gb/s	22.5 Gb/s	48 Gb/s	24 Gb/s	45 Gb/s
1 GHz	32 Gb/s	16 Gb/s	30 Gb/s	64 Gb/s	32 Gb/s	60 Gb/s

Serial RapidIO

Clock Rate	1-bit Wide			4-bit Wide		
	PEAK	Sustained 32 byte Op	Sustained 256 byte Op	PEAK	Sustained 32 byte Op	Sustained 256 byte Op
1.25 GHz	2 Gb	1 Gb	1.8 Gb	8 Gb	4 Gb	7.2 Gb
2.5 GHz	4 Gb	2 Gb	3.6 Gb	16 Gb	8 Gb	14.4 Gb
3.125 GHz	5 Gb	2.5 Gb	4.5 Gb	20 Gb	10 Gb	18 Gb

The RapidIO physical layers offer a wide range of performance levels to meet the needs of applications.

Summary

The RapidIO interconnect provides a robust packet-switched system level interconnect. It provides a partitioned architecture that can be enhanced in the future. It enables higher levels of system performance while maintaining or reducing implementation costs. A RapidIO end point can be implemented in a small silicon footprint. Proven industry-standard signaling schemes (LVDS, XAUI) are used for the physical interfaces. Error management includes the ability to detect multi-bit errors and survive most multi-bit and all single bit errors. Even with all these capabilities, RapidIO's protocol overhead and latency are comparable to current bus technologies and significantly better than local area network based fabric technologies such as Ethernet.

WWW.RAPID**IO**.ORG