

RapidIO vs. PCIe Peer Communications

By Barry Wood,
Chair of the RapidIO Technical Working Group

The summary of the Tech. Note:

- PCIe supports “peer-to-peer” communication for small, tightly coupled, hierarchical systems.
- PCIe “peer-to-peer” communication mechanisms require significant software overhead
- RapidIO supports simple, reliable, efficient true peer-to-peer communication, implemented in hardware that scales from small to large (thousands of processors) systems.

Rather than reiterating the points above (or publishing the text below), RapidIO marketing folks ‘sound-bite’ this as ‘PCIe does not support peer-to-peer communication’.

The rest of the story...

PCIe specifies that switches must support “peer to peer communication”, which they define as the ability to route packets headed upstream back downstream if the packets address matches the BAR configuration on a downstream port. This is not consistent with the usual network definition of peer-to-peer communications, whereby software entities at the same level in a network stack or with the same responsibilities in a network communicate.

For small systems, the ability to reroute packets back downstream is sufficient to allow multiple processors to communicate. For simple and infrequent communications between processors where overhead, scalability and software evolution is not an issue, PCIe will work. Typically, however, the processors in these systems are not organized as true peers. The root complex controls most communication, and usually acts as the source or target for communication, as is the case of the graphics GPU board. In this application the GPUs are very tightly coupled from a software perspective.

When multiprocessing platforms larger than a PC are designed, they typically require multiple cooperating control entities, and may use M+N sparing, load sharing, and other elegant hardware and software mechanisms which require software entities on different processors to behave as true peers in order to boost the capacity of the system and/or reduce the cost while maintaining high reliability and efficiency. Ideally, the same hardware and software design paradigms are used for systems of various sizes in order to leverage hardware and software investments. Software is partitioned to allow independent evolution of different functional areas, and to allow flexibility when mapping software components to different hardware platforms.

PCIe “peer to peer communication” generally does not scale to meet the requirements of these systems, as PCIe is based on a flat address space shared hierarchically among all system components. Software built for address oriented messaging systems tends to be more difficult to design, verify, and evolve. Vendor-specific support for non-transparent bridging and/or opaque address spaces does not change this basic reality.

RapidIO vs. PCIe Peer Communications

By Barry Wood,

Chair of the RapidIO Technical Working Group

Even from a strict address-oriented perspective, RapidIO is far more flexible than PCIe, as RapidIO routes packets based on a destination ID rather than according to a global address map known to every processor in the system. This allows the RapidIO protocol to be more efficient and flexible than PCIe, even in small systems. Even with this added flexibility for read/write transactions, most RapidIO users rely on messaging in small systems to communicate effectively and efficiently between processors.

In large systems RapidIO really shines, because RapidIO also supports a messaging paradigm in hardware. This enables the simple, reliable, and efficient peer-to-peer software communication that scales from small systems to large systems.

- **Simple** – Software determine the recipients destination ID. Hardware transfers the message.
- **Reliable** – RapidIO guarantees packet delivery, and every Message packet generates a logical layer response to confirm reception
- **Efficient** – messaging is implemented in hardware, and so does not take significant processor overhead

RapidIO supports four kinds of messaging, all of which have efficient hardware support and scale to support all sizes of system:

- **Doorbell** (Type 10) – Designed for exchanging events, but more capable than PCIe MSI as Doorbells can be queued up by the receiver or the transmitter can be told to try again later if the queue is full.
- **Message** (Type 11) – Transfer up to 4KB segmented over up to 16 packets, each of which requires a logical layer response. Message packets can be retried if queues are full, or if there are insufficient hardware resources to assemble multiple multi-segment messages.
- **Data Streaming** (Type 9) – Transfer up to 64KB segmented over many packets. Data Streaming packets cannot be retried – there must be hardware resources in place to handle every multi-segment message.
- **Flow Control** (Type 7) – Used to perform flow control for messaging. It can also be used to request ownership of resources, such as a data streaming context, or some other hardware or software resource.