Overcoming Latency in PCIe Systems Using PLX

by Jack Regula, Chief Technology Officer, PLX Technology (www.plxtech.com)

Overcoming PCI Express (PCIe) latency isn't simply a matter of choosing the lowest-latency components from among those suitable for an embedded-system design, but it's a good place to start. It's also a matter of architecting operations to reduce or eliminate the sensitivity of system performance to latency. It's impossible to mask all the latency, so the less there is to begin with, the better.

Defining Latency

Latency is the delay between starting and completing an action. For a switch, it's the time between the start-of-packet (SoP) symbol on an input pin and the SoP symbol on an output pin for the same packet forwarded through the switch. From an endpoint's perspective, the latency includes the packet transmission time, since it can't use the data until it has seen the cyclic redundancy check (CRC) at the end and checked for errors. At the highest level, the overall task latency, which may include multiple switch latencies, is what really matters. At issue is whether resources are idled during the waiting period implied by a task or transfer latency, and whether the waiting time prevents a deadline from being met.

A PCIe switch's latency can be decomposed into the time required to receive the header, a pipeline delay and a queuing delay. The pipeline delay is the length of time for a packet to traverse an otherwise empty switch and is solely a function of the switch's design. The queuing delay depends to a large extent upon the traffic pattern but can also be dependent on flow control credits, as well as the switch's arbitration and scheduling policies. Deficiencies in a switch's implementation or architecture most often show up when dealing with flows consisting primarily of short packets. Therefore, a switch's performance should be evaluated with short packets, long packets, and, of course, with a packet mix and flow pattern representative of the application.

Latency and Throughput

Engineers often try to extract full-wire-speed performance from the system interconnect but that can be a mistake if low latency is also required. The closer the egress link of a switch port is to being saturated, the deeper the queue in the buffers behind it. With large buffers, it's seldom necessary to throttle back an input, so maximum throughput is obtained. The price is the latency of the queues that develop.

With traffic patterns that lend themselves to mathematical analysis, (e.g. uniform distribution and Poisson arrival times), the average queue depth can be estimated from queuing theory. Without getting into the mathematical details, Figure1 provides a rough guide as to the queue depth that will develop behind a switch egress port, based on the number of ports feeding an output and the degree of utilization of its output link. For simplicity, equal-sized packets were assumed. Keep in mind that on the order of five percent of the link is consumed in DLL overhead.

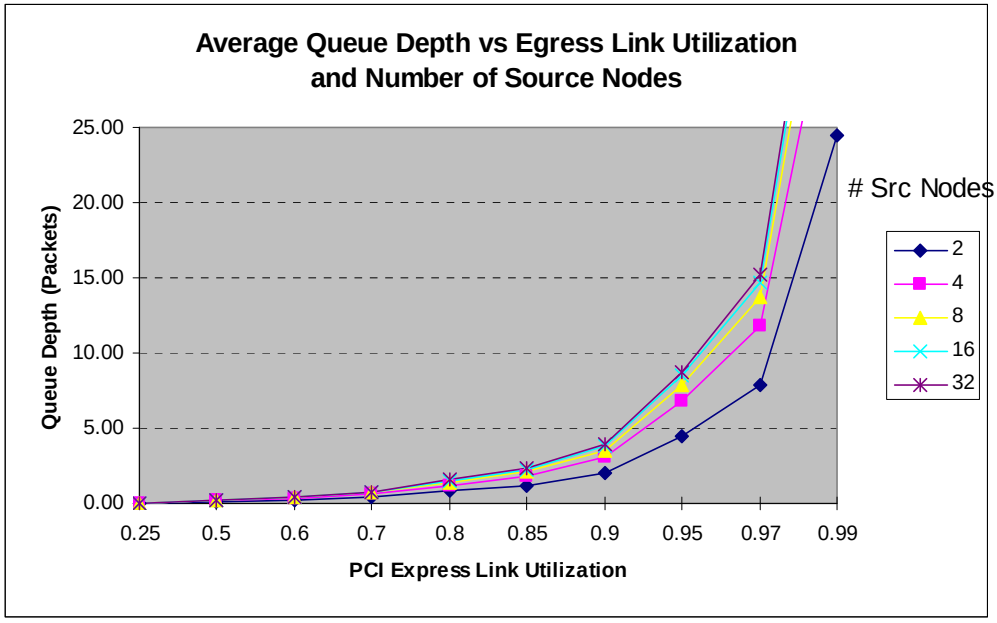

Figure 1: Average Queue Depth Behind PCIe Switch Egress Port

Link Width and Latency

There is another, less-esoteric relationship between bandwidth and latency. A switch cannot forward a packet until it has seen enough of the packet's header to determine its egress port. The wider the input link, the less time required to see the complete header. On an x16 link, the entire header may be visible in a single clock, depending upon how the SoP symbol aligns on the link. On a x8 link, it takes as few as two clock cycles to see the entire header. Each halving of the link width doubles this component of the switch latency.

The situation is more complicated when the egress link is wider than the ingress link. A switch can't initiate cut-through on a packet until it has received enough of it so that the faster egress link won't run dry of packet to send before the rest of the packet comes in. Roughly speaking, if the egress is twice the width of the ingress, then half the packet must be received before forwarding starts. Ironically, using an egress link that is wider than the ingress link will increase the latency measured to the SoP symbol, but decrease it when measuring to the end-of-packet (EoP) symbol. An endpoint can't make use of the packet until it checks the CRC at the end of the packet.

.

Thus, using wider links can have three beneficial effects – reducing the cut-through latency, the queuing delay, and the packet transmission time. As can be seen in Figure 1, doubling the output link width can shrink the queue depth from near-maximum to nearly empty.

Latency Sensitivity of Reads

A read is generally considered to be a blocking operation in that once a read request is initiated no additional instructions in its thread of processing can be undertaken until it is completed. Simple applications have the following work flow:

1. Make a read request
2. Wait for data
3. Process the data
4. Loop back to 1

In this simple example, the latency of the read directly affects the throughput. If the read latency is much smaller than the processing time, then latency isn't a problem. When it's not small, users look for ways to mask the latency by doing useful work during it. A multithreaded processor could switch threads, for example, doing some other work during the latency. Optimizing compilers issue the read early to minimize the wait.

Bus interface units often have an ability to issue multiple read requests before being forced to wait for a completion. If, for example, N outstanding read requests are supported, and the completion to the first read request arrives before the Nth read request is sent, then latency is said to have been masked and full throughput can be achieved after that initial waiting period. In practice, devices have varying degrees of ability to mask latency so in a system, such as a PC or server where there is no control as to what is plugged into an open slot, latency is always an issue.

Bridging Legacy PCI Devices to PCIe

When bridging PCI to PCIe, the bridge must make a guess as to how much data the device will consume on a read. If the bridge guesses wrong, performance suffers. An advanced bridge will use the version of the PCI read command as a hint. In response to a simple MemRd, it will fetch only a single bus width of data. In response to a RdLin command, it will typically prefetch a cache line of data. Use of the RdMult command on PCI should result in the prefetch of multiple cache lines. After prefetching the data, the bridge should retain it in a cache after an initial disconnection by the PCI device in case the device returns for more data. When the PCIe-to-PCI bridge's prefetch policy isn't adequate, it can help to insert a PCI-to-PCI bridge in the path to the device. The bridge can be configured, for example, to translate a MemRd or RdLin command into a Read Multiple command, and to keep the data longer in its internal prefetch cache. For both PCI-to-PCI and PCIe-to-PCI bridges, it's necessary to do device-specific configuration to enable advanced prefetch features.

DMA I/O and Read Latency

The DMA I/O subsystem at the heart of PCs and servers is inherently latency-sensitive. I/O is accomplished using a DMA controller in each I/O device to move data between it

.

and main memory located next to the CPU. The DMA controller follows a chain of descriptors located in memory. Each descriptor describes a unit of work assigned to the DMAC, requiring the DMAC to move a block of data from the device to memory or from memory to the device. The DMAC reads a descriptor, then assigns a DMA engine to do the data movement dictated by the descriptor. While the data is being moved, it reads the next descriptor. If the DMA engine completes its assignment before the next descriptor read completes, it is forced to idle for lack of work. Typically, a workload for tasks such as networking consists of a mix of short and long data blocks (packets) to be moved to and from memory. When the data block is relatively long, latency is masked. For short blocks, such as those used for Ethernet control packets, descriptor read latency can lead to a loss of throughput. To avoid this, a sophisticated DMAC may read several descriptors ahead and maintain a cache of prefetched descriptors. However, there is always a limit to the size of the cache and to the number of descriptors available to be prefetched. A particular device may be capable of masking descriptor read latency when directly attached to a Northbridge (NB) but its throughput may suffer when it is connected to the NB through a switch. System designers are best advised to use the lowest-latency switches available to maximize the performance of their I/O subsystems.

Accelerators and Switch Latency

An increasingly common usage model is the attachment of multiple accelerators to a processor complex to increase performance for certain applications. Examples are the use of graphics processors for floating point acceleration. In the accelerator model, the host processor offloads a computation to the accelerators, then waits for the result. It may or may not have useful work to perform while waiting. Only if the waiting time is less the time required to complete the operation without an accelerator is there a gain in throughput.

The amount of time the host waits is the sum of:

1. Synchronization time at start of computation
2. Time for accelerator to read data from memory
3. Time for accelerator to produce the result
4. Time for accelerator to write the result back to memory
5. Synchronization time at end of computation

Each of these operations, except for the computation time itself (#3) includes the interconnect latency. All the usual games of masking latency with concurrency apply. Nevertheless, when you consider that typical accelerators operate in the GHz range while interconnect latency is generally greater than 150 nanoseconds, you can see it is necessary to offload a relatively long computation in order to gain throughput by offloading work to the accelerator. Every step decrease in switch or interconnect latency widens the range of problems to which accelerators may be profitably applied.

Summary

Given enough time and resources, engineers can usually figure out how to mask any fixed amount of latency. Often this effort consumes most of their development time and contributes significantly to the end cost of their product. No more dramatic example of

.

this exists than the die area consumed by cache, cache controllers, and support for multiple threads on high-end microprocessor chips.

Efforts to mask latency achieve varying degrees of success. System interconnects have varying degrees of latency. In practice, we see some devices showing latency sensitivity in some slots of some systems. These disturbing observations lead to additional effort to root cause and find ways to reduce the latency sensitivity, thus extending the time to market.

The availability of low-latency switches from PLX Technology makes the job of everyone producing a PCIe-based infrastructure easier. These industry-leading switches drop latency to as low as 110ns, or 87 percent lower than competing devices on the market. Low-latency switches such as these should be the first choice of system engineers interested in producing high-performance systems.

.