QoS for High-Performance and Power-Efficient HD Multimedia

Ashley Stevens
Solutions Architect

April 2010
Market drivers

Consumer expectations are placing ever greater demands on the multimedia requirements of today’s consumer electronic devices. Advanced yet intuitive graphical user interfaces drive the demand for Graphics Processor Unit (GPU) hardware whilst pervasive HD video on everything from phones to TV drives demand for video acceleration. And with the incorporation of HDMI video interfaces on everything from set top boxes and PVRs down to phones and even cameras, full 1080p graphics and video is becoming a requirement on almost every consumer electronic device today. Cameras with high speed motion capture for slow motion replay place even greater demands on the system. Ensuring all these demands are consistently met whilst minimizing cost and maximizing battery life are the challenges for today’s SoC designer. Inconsistent video frame rates, dropped frames and stuttering audio are both accepted and expected by PC users but are unacceptable in a quality consumer product.

Memory solutions for multimedia – UMA vs video RAM

There are two fundamental approaches – separate video memory or Unified Memory Architecture (UMA). In the separate video memory system there’s a memory system just for graphics and video and another for the CPU sub-system. This is the classic PC architecture with a separate video card. A clear disadvantage with this approach is the additional cost of the separate memory sub-system. It also makes for more complex memory management since there are two different ‘kinds’ of memory in the system (multimedia memory and ‘normal’ memory); or else it results in memory that can’t easily be used since it’s dedicated to another purpose. For greatest flexibility and lowest cost, especially in multi-function devices like smart-phones where workloads are highly variable, a system with a unified memory system (UMA) is to be preferred. For the rest of this paper we shall consider a unified memory architecture example; though advances in packaging may enable separate video memory for consumer devices, by packaging wide data interface (Wide I/O) memory face-2-face or chip-on-chip within the same package as the SoC.

![Diagram of Traditional PC architecture with separate video RAM for GPU and CPU](image)
Modern SoC Unified Memory Architecture (UMA)

SDRAM Fundamentals - Rows, Columns & Banks

Before going further it’s worth a quick recap on how SDRAM is organized. SDRAM’s contain several banks. Each bank consists of rows and columns of memory. The address is multiplexed into two parts, the Row and Column address. To perform any access an ‘activate command’ is first used to open a particular Row. Once open, Column read or write commands are used to read or write within that Row. To access a different Row within the same bank, the Row must be closed (pre-charged), and then the new row opened using a row-activate command. There’s a latency delay between row open and column read and from column read to data. SDRAM chips typically contain 4 or 8 banks. To enable a degree of parallelism each bank can be accessed independently. Thus it possible to hide the latency of row open commands by interleaving access between banks. Various address line to physical DRAM address lines mappings are possible; the column addresses do not necessarily need to be the least significant address lines of the processor. To maximize latency-hiding memory accesses should be spread equally amongst banks; often achieved by mapping the bank control lines to mid-order address lines at MMU page granularity (eg 4KB).
Because opening a row is time-consuming and energy-intensive SDRAM controllers often aim to maximize ‘in-row’ accesses. Memory accesses can be performed in a more optimal order than they are received from the interconnect. Upcoming accesses to rows already open are prioritized ahead of accesses that would require a bank to be closed, pre-charged and re-opened. This can save time and reduce energy consumption and help to enable high SDRAM utilization, but can adversely affect latency for out-of-page accesses.

To enable the SDRAM controller to schedule accesses in an efficient manner by prioritizing in-row accesses over those that require a new row-activate command, the controller typically contains a queue of pending transactions which can be performed out-of-order. Transactions within the queue are not handled by the SDRAM controller in the order they were received but in the order that maximizes in row accesses, providing no express ordering is required to manage ordering hazards. In ARM® AMBA® AXI™ transactions are tagged as originating from a particular master with a master-ID and transactions with different IDs are considered independent. Alternatively, controllers may have multiple slave ports dedicated to individual masters, a popular approach for AMBA® AHB™ SDRAM controllers since AHB doesn’t support multiple outstanding transactions.

What bandwidth is required for 1080p graphics and video?

Both graphics and video memory bandwidths depend on the frame rate, complexity and GPU and video engines being used. A normal target is 30 frames per second or fps. At this rate motion appears smooth and life-like. For the purposes of further analysis we’ll assume 1.5GB/s for GPU bandwidth at 1080p30 which is achievable for a bandwidth-efficient GPU. For video we assume a worst-case figure of 500MB/s. Any design must be capable of handling the worst case for video to avoid dropped video frames which is unacceptable. Finally a 1080p 60Hz video controller requires approximately 500MB/sec just for screen refresh at 32bpp.

<table>
<thead>
<tr>
<th>Device</th>
<th>Typical bandwidth 1080p30</th>
</tr>
</thead>
<tbody>
<tr>
<td>HD GPU</td>
<td>1.5GB/s</td>
</tr>
<tr>
<td>HD Video Codec H.264</td>
<td>500MB/s</td>
</tr>
<tr>
<td>1080p 60Hz 32bpp video refresh</td>
<td>500MB/sec</td>
</tr>
</tbody>
</table>

The above bandwidths means that 1080p 60Hz graphics with screen refresh can typically require around 2GB/sec. If video and graphics are used simultaneously (eg video as overlay texture within a GPU generated image) then 2.5GB/s could be required. Video post-processing (eg de-interlace of an interlaced video source) can increase bandwidth still further, as can high frame-rate video encode. However for the rest of the article we’ll take as an example a mobile device (smart-phone, tablet or netbook) application (which doesn’t have video post-processing and scaling) requiring in the region of 2 – 2.5GB/s for the multimedia sub-system (GPU/VPU and screen refresh). In addition there would be system-dependent bandwidth required for DMA and the CPU.
Memory types available – DDR2, LP-DDR2 & DDR3

Today the choice of SDRAM memory for consumer devices is primarily between DDR2 (lowest cost), LP-DDR2 (lowest power) or DDR3 (highest mainstream performance\(^1\)). It’s expected that DDR3 will achieve price-per-bit parity with DDR2 sometime during the course of 2010. For mobile phone applications LP-DDR2 will be used, for netbook and home use (STB, DTV) the choice is between DDR2 and DDR3. In real systems, actual SDRAM efficiency will be significantly less than the peak theoretical, and could be as little as half in some scenarios.

<table>
<thead>
<tr>
<th></th>
<th>LP-DDR2</th>
<th>DDR2</th>
<th>DDR3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Clock speeds</strong></td>
<td>333MHz, 400MHz, 533MHz*</td>
<td>267MHz, 333MHz, 400MHz, 533MHz</td>
<td>533MHz, 667MHz, 800MHz*</td>
</tr>
<tr>
<td><strong>Data rates</strong></td>
<td>667Mbps, 800Mbps, 1066Mbps*</td>
<td>533Mbps, 666MHz, 800Mbps, 1066MHz</td>
<td>1,066Mbps, 1,333Mbps, 1,600Mbps*</td>
</tr>
<tr>
<td><strong>Peak Theoretical bandwidth x32</strong></td>
<td>3.2GB/sec @ 400MHz clock</td>
<td>4.26GB/sec @ 533MHz clock</td>
<td>5.33GB/sec @ 667MHz clock</td>
</tr>
</tbody>
</table>

* Speeds defined but not common as of time of writing therefore not used for bandwidth calculation

Differing Master Requirements

There are two fundamental quality measures of the effective memory system as seen by each master; that is, latency and bandwidth. The requirements placed on these two qualities vary enormously by master type.

**CPU – Latency-sensitive**

When a read memory access is performed by the CPU there is normally little further processing that can be accomplished before the CPU simply stops and waits for the memory transaction to complete. For this reason, the CPU is said to be latency-sensitive, meaning that performance is directly impacted by effective memory latency, even though the overall external memory bandwidth requirement (after the effect of caching) may be comparatively low. The term ‘effective memory latency’ here is used to describe the memory latency as seen by the master within the system and environment it finds itself in, not just the raw memory latency of the system memory controller. The ‘effective memory latency’ is longer than the raw memory controller latency due to the effects of arbitration and other masters in the system.

---

\(^1\) There are higher performance memories like GDDR3/4/5 but these are specialist devices for high-end PC graphics cards and are not considered here
Execution time penalty of increased memory latency – Cortex-A8 running Nitehawk browser on Linux

**GPU & VPU – Bandwidth-sensitive**

In contrast to a CPU, a Graphics Processing Unit (or GPU) requires very high memory bandwidth but is not typically latency sensitive. The reason: A GPU typically performs the same (or very similar) operations on huge numbers of individual data elements, whether they’re pixels (sometimes called fragments) in the case of rendering, or triangles (sometimes called polygons) in the case of geometry processing. Whilst there are many individual calculations to be performed, there’s no dependence on previous results. The GPU does not need to stop and wait for the results of a previous memory access; it can simply move on to another pixel or triangle and begin processing that. (Simplistic immediate-mode GPU renderers may have a requirement on a low-latency read-modify-write access but tile-based renderers don’t perform this type of memory access). Video Processing Units (VPU) that accelerate video encode and decode are similar to GPU in their memory requirements — they may have high memory bandwidth requirements (though not quite as high as GPU) but their lack of inter-dependencies means they should be latency-insensitive provided they receive sufficient bandwidth over a frame-period. In summary provided the GPU and VPU receive sufficient long-term memory bandwidth (where ‘long-term’ means over the period of a video frame), then the latency of each individual access isn’t critical.
Video display (LCD / HDMI output) – Latency-critical (and bandwidth-sensitive)

A video display continually scans the frame buffer, normally in a linear incrementing fashion. Data must be output at a constant rate and latency is critical: If the video FIFO underflows then the screen image breaks up which is a catastrophic failure. Thus video displays are highly latency-critical, even if their bandwidth requirement is moderate in comparison with a GPU or even VPU (although an uncompressed 1080p 60Hz display at 32bpp takes around 500MB/s which isn’t insignificant). Since the video display address pattern is highly predictable (normally linear incrementing) it can be made less sensitive to latency by increasing the size of the FIFO buffer, support for multiple outstanding transactions to the memory controller or increasing burst size. But whatever size the FIFO buffer is, for each and every transaction there’s a specific latency requirement, in addition to a long-term average latency requirement over a frame-time interval. The individual transaction latency requirement makes up the latency-critical quality and the long-term average latency requirement is defined by the bandwidth requirement and thus forms the bandwidth-sensitive quality.

What’s the difference between Latency-sensitive and Latency-critical?

A latency-sensitive master, like a CPU, has its performance directly impacted by memory latency: the longer the memory latency the lower the performance. In contrast a latency-critical master has a specific maximum latency that’s acceptable for each individual transaction, and if that latency is not achieved a catastrophic failure results (eg the screen breaks up). But so long as that minimum requirement is met, reducing it further doesn’t improve performance.

Effects of SDRAM controller scheduling in multimedia systems

Scheduling mechanisms in SDRAM controllers are used to maximize efficiency by prioritizing in-row accesses. The goal is to minimize bank opening and closing, which reduces time wasted opening and closing and pre-charging as well as minimizing row-activate energy usage. Scheduling makes use of a buffer of pending transactions in the SDRAM controller. SDRAM bandwidth efficiency is defined as the actual bandwidth achieved as a percentage of peak theoretical bandwidth. The actual efficiency can never be 100% (some time must always be taken for SDRAM refresh) but optimized scheduling can help approach the peak value.

\[
\text{Efficiency} \% = \frac{\text{Actual Bandwidth}}{\text{Peak Theoretical Bandwidth}} \times 100
\]

It’s generally accepted wisdom that effective scheduling can attain efficiencies well in excess of 50%, often expectations are 80-85% or higher. But whilst scheduling maximizes efficiency, left unchecked it can result in undesirable consequences. A non latency-sensitive master that has a high bandwidth requirement, supports many outstanding transactions and/or produces many in-row accesses can block latency-sensitive or latency-critical masters for prolonged periods of time potentially causing serious system performance issues.
Outstanding transactions and effects in interconnect

‘Outstanding transactions’ refers to the ability of a master to start a new transaction before a prior one has completed. It’s often assumed that the more outstanding transactions the better, since it would appear to increase the likely-hood of hiding SDRAM latency. But it turns out it’s possible to have too much of a good thing! Consider as an example a fully-connected cross-bar interconnect. Arbitration occurs at the target when more than one master accesses the same slave simultaneously. An SDRAM controller contains a queue that can accept multiple outstanding transactions. For example, the ARM DMC-341 and DMC-342 SDRAM controllers can be configured to support up to 16 outstanding transactions but are often configured for 12 as that’s a practical limit to achieve timing-closure. But some masters such as a GPU can issue far more than this. The result is that transactions back up from the SDRAM queue into the interconnect and the interconnect and masters accessing the slave stall. At this point, a high-priority QoS master can’t issue any more transactions to the SDRAM slave as arbitration stages of the interconnect are locked-up. The result is that the QoS mechanism of the SDRAM controller fails – it’s not possible to limit the time a QoS master has to wait to be serviced. What’s needed is a mechanism to ensure this doesn’t occur, or at least limit the amount it can happen to an absolute minimum. To achieve this we need to ensure there’s always space in the SDRAM controller queue and the interconnect for a QoS high-priority transaction. This means limiting the number of outstanding transactions from high-throughput masters in the system to ensure this is the case. We call this ‘regulation’. There’s three main ways to accomplish this regulation.

Queuing Theory and Little’s Law

Queuing Theory is the mathematical study of queues. Little’s Law states that:

\[ N_T = R_T \cdot L_T \]

Where \( N_T \) is the long-term average elements in the queue, \( R_T \) is the long-term average arrival rate and \( L_T \) is the time an element spends in the system. In the case of analyzing a network interconnect with an SDRAM controller, \( N_T \) is the queue length, \( R_T \) the transaction arrival rate and \( L_T \) the latency of a transaction (time waiting for it). Thus we can say that

**Queue length = Arrival rate * Latency**

If we need to limit the queue length to a defined maximum, we can either directly regulate it by delaying non latency-sensitive transactions occurring if they would cause the queue length to exceed a limit, or we can attempt to otherwise regulate the arrival rate or the latency to indirectly influence queue length.

One approach would be to limit the latency by using an over-provisioned system in which bandwidth hugely exceeds what’s needed, (eg. by a factor of 3-4x or more). Whilst this may be feasible in simple, low-end microcontroller systems where the CPU and other masters are of limited performance, today’s complex multimedia systems cannot economically take this approach. We’ve seen that systems supporting 1080p graphics and video may need 1.75 – 2.35GB/s of memory bandwidth just for graphics.
and video; with other masters including CPU the requirement can be >2.5GB/s. A likely low-cost and low-power candidate memory system would be 32-bit LP-DDR2 at 400Mbps offering 3.2GB/s peak theoretical bandwidth. 32-bit DDR2 (low cost but not low power) offers 4.26GB/s peak theoretical bandwidth and 32-bit DDR3 (moderate cost and power) 5.33GB/s. To avoid the cost of moving to a 64-bit memory system (or dual-channel 32-bit) we require a system enabling from about 50% (DDR3) to nearly 80% (LP-DDR2) of peak theoretical bandwidth whilst simultaneously maintaining low latency for latency-critical and latency-sensitive masters and sufficient bandwidth for bandwidth-sensitive devices.

To achieve this we need to introduce Quality of Service (QoS) mechanisms.

QoS mechanisms

1. Regulation of outstanding transactions

We’ve seen that it’s necessary to regulate the number of outstanding transactions in a system to ensure the interconnect doesn’t back up with pending outstanding transactions. The simplest way to do this is to directly regulate the number of outstanding transactions. Each master can be assigned a maximum number of outstanding reads or writes individually. Each interconnect arbitration point can also be assigned a shared maximum outstanding limit. This better copes with the case where it’s desirable to limit a collection of masters overall, but wish to enable that limit to be shared dynamically according to system needs. In the ARM CoreLink™ Interconnect (NIC-301) with Advanced Quality of Service (QoS-301) transaction regulation limits are fully programmable via registers within the interconnect. This enables in-silicon tuning and doesn’t require a fixed design-time decision to be made. A limitation of using integer transaction limits is the granularity of tuning of average queue depth is coarse. To enable finer tuning of queue-depth the QoS-301 supports fractional average outstanding limits.

2. Regulation of address issue rate

Earlier we looked at Little’s Law. Little’s Law says:

\[ \text{Queue length} = \text{Arrival rate} \times \text{Latency} \]

Thus we can indirectly regulate the queue length by controlling the arrival rate. We can re-arrange Little’s Law to to get

\[ \Rightarrow \quad \text{Arrival rate (issue rate)} = \frac{\text{Queue length (outstanding transaction)}}{\text{Latency}} \]

For a fixed transaction size the arrival rate is proportional to bandwidth and regulating issue rate equates to regulating master bandwidth. In practice it only approximates bandwidth since in most systems the transaction size is not fixed. The SDRAM controller burst size is fixed but long bursts on the bus can cause multiple fixed-burst accesses on the SDRAM interface. Additionally some masters such may dynamically alter transaction size according to system conditions.

In the NIC-301 with QoS-301 plug-in the issue rate can be limited per master to limit master bandwidth. This scheme is good for masters where there’s a defined maximum bandwidth that should never be
exceeded. For example, a memory to memory DMA copy could be limited to a maximum bandwidth. It could be used to limit a GPU but in cases where the system was otherwise lightly loaded the GPU couldn’t use the spare system bandwidth above that limit. Thus it’s inflexible for complex multi-function devices. Note that issue rate regulation is sometimes known as TSPEC (from Traffic SPECification) as this is terminology used in QoS for networking.

Regulation of latency (via transaction priority)

So far we’ve looked at regulating outstanding transactions to directly control the queue length, and we’ve looked at regulating the arrival rate (i.e. master issue rate) to indirectly control the queue length. The other variable in Little’s Law that we can manipulate is the latency. We can’t directly control the latency but we can indirectly influence it using a dynamic priority scheme. This assigns a priority to each transaction which flows through the interconnect to the SDRAM controller. The priority is raised and lowered to attempt to maintain a long-term average latency. For this scheme to be effective the SDRAM controller slave needs to support the priority field information and arbitrate transactions according to the priority requested. This enables indirect control of issue rate and therefore indirectly influences the queue length as per Little’s Law.

Case study

ARM studied a system consisting of an ARM Cortex™-A9 dual-core CPU with ARM CoreLink™ PL310 level-2 cache controller, and HD-capable GPU and video engine (VE). To increase simulation speed and enable faster turnaround of results, the CPU, GPU and video engine (VE) were replaced with VPE master models. VPE stands for Verification and Performance Exploration which is the ARM solution for generation and capture of stimulus within an RTL simulation world. VPE enables the generation of traffic profile patterns that match those of any master with greatly enhanced simulation speed compared with using the full RTL of a master such as a CPU or GPU. More details on the Study can be found in the full conference paper: Tune&Bruce 2010²

² Tune&Bruce2010: How to tune your SoC to avoid traffic congestion Andrew Tune & Alistair Bruce DesignCon 2010
Double-buffering for GPU and Video

GPU and video systems are typically double-buffered. There are two frame buffers, the current one being scanned to the display and the next one (known as the back buffer) being rendered by the GPU. At some point the GPU completes rendering the back buffer and then waits for a vsync interval (end of frame flyback period) to swap buffers. While waiting for the vsync period to swap buffers the GPU is idle, and may be powered down. If the GPU takes more than a frame time to render the frame then the previous frame is repeated until the GPU has completed rendering. Thus the GPU transaction profile appears as a long period of intense memory transactions whilst rendering, followed by a period of idle waiting for the next vsync period. In the scenario modeled, the GPU was assumed to require 50MB/frame at a target frame rate of 30fps resulting in 1500MB/s on average over a frame-period. In practice, GPU activity consisted of a period of intense activity at significantly higher peak bandwidth, followed by a period of idle waiting until the next frame period. Note that since the LCD refresh rate was 60Hz this means that each GPU rendered frame was displayed two times by the LCD controller and the target maximum render time was two frame internals so 33.3ms.
Scenarios modeled:

- Unconstrained

- Outstanding transaction regulation – GPU constrained to 3 read and 1 write

- Address issue regulation (TSPEC) – GPU limited to 2.4GB/s

At the time of the study there was no suitable memory controller available for latency regulation.

The results presented here are only a brief summary of those in the paper. The reader is referred to the paper for the complete results including write latencies and results for other scenarios such as GPU idle but VE active.
Bandwidth (MB/s) – Unregulated baseline scenario

It can be seen that in the unregulated baseline scenario the GPU consumer 2734MB/s during the active phase, which lasted for 55% of the frame time. Due to the high GPU bandwidth the CPU was only able to get 32MB/s during GPU activity and average 91MB/s over a complete frame period.
With the GPU constrained to 3 read and 1 write transaction the GPU bandwidth during the active phase was reduced 3% to 2664MB/s and the active phase increased slightly to 56% of a frame period. This small change resulted in a large increase in CPU bandwidth, up to 99MB/s during the GPU active phase and 127MB/s over a complete frame period. Increased bandwidth for the CPU translates directly to increased CPU performance. Since the GPU still renders the frame within a frame time no GPU performance drop is observed.
With the GPU regulated using issue rate regulation, with it limited to 2441MB/s the GPU active phase increased to 61% of a frame period. CPU bandwidth increased to 119MB/s during the GPU active phase and 136MB/s average over a complete frame period. The further increase in CPU bandwidth increases CPU performance with no GPU performance drop since the GPU still renders well within a frame interval.

Bandwidth (MB/s) Against Time – Address Issue Rate Regulation
Average Read Latencies (cycles) for various QoS schemes

A very significant reduction in read latency was observed with regulation.

Average Read Queue Length (Outstanding transactions)
Further Work

Issue rate regulation enables fine-grained control of QoS but puts a hard limit on the bandwidth of a regulated master. In contrast, outstanding transaction regulation is more adaptive to system needs but had a very coarse grained control. To enable finer tuning of QoS using outstanding transaction regulation, ARM has implemented fractional outstanding transaction regulation, which alternately switches between two integer values using a defined mark:space ratio to emulate non-integer outstanding transaction limits.

**Issue Rate (TSPEC) regulation lacks adaptivity to system conditions**

Latency regulation via dynamic priority is a promising technique which should be highly adaptive to system needs avoiding over-regulation in lightly-loaded systems. It should achieve low latencies without the fixed bandwidth limit imposed by issue rate regulation, but requires a memory controller supporting priority information. ARM is developing such a memory controller but it wasn’t complete at the time of the study.
What’s the practical difference between Address Issue Regulation and Latency Regulation?

Address issue regulation (TSPEC) limits a master to a fixed maximum bandwidth, but can’t control the minimum. Chances are in most systems the bandwidth it gets will be less than defined, and potentially a lot less. Latency Regulation attempts to maintain a certain specific long-term average latency (which translates to bandwidth as we have seen through Little’s Law).

As a rough analogy, imagine driving a long toll road. You take ticket on entry and then when you exit the freeway you hand the ticket in at the toll and pay the toll. Since your start and end times are recorded, it’s possible to calculate your average speed. Now you don’t want that average to exceed the speed limit or you could be in trouble! So one approach is to ensure you never exceed the speed limit. This is analogous to the address issue regulation. But when there’s other traffic your likely average speed A to B will be lower than the speed limit, and if you encounter any serious traffic jams then it could be very much lower than the limit. If the limit is 70mph and you can never go faster than that (imagine a truck fitted with a speed limiter as an example), any time lost in jams can never be made up. You control the maximum speed you will go, but have no control of the average speed A to B which could be very much lower.

Latency regulation attempts to regulate the effective latency to a defined target and is thus more like a target average speed A to B for the toll road. If your target average A to B is 70mph, you can start out (with no other traffic around) doing a steady 70mph. But then you hit a jam which lowers your average. With dynamic priority, after getting past the jam you could speed up to 90mph for a few miles to get the overall average speed back to the target 70mph average (the author doesn’t endorse doing this on the public highway!). Thus latency regulation attempts to get as close as possible to the latency target within the limits of the system and the traffic within it and thus controls issue rate. You won’t maintain a faster average than requested, but shouldn’t have a slower average either, unless it’s physically impossible due to other (higher priority) traffic in the system.

Effects of QoS on system performance and SDRAM controller efficiency

The goal of QoS is to prevent transaction backing up into the interconnect by regulating the number of outstanding transactions in the system, either directly or indirectly. QoS must be used sparingly as overly-restrictive policies may result in a reduced average SDRAM queue length leading to fewer opportunities to hide SDRAM latency. This could reduce SDRAM controller memory efficiency. Careful tuning of the QoS system is needed to ensure sufficient transactions are available to maintain SDRAM efficiency without the queue become so long that latency suffers. Achieving such tuning correctly at design-time is quite challenging, so a scheme that enables post-silicon tuning and optimization in software is highly desirable. A further benefit of programmability of QoS is that it enables modal operation where different QoS parameters are applied in different use-case scenarios. The introduction of QoS should minimally reduce SDRAM efficiency if ‘tuned’ correctly to minimally affect efficiency.
ARM CoreLink™ Network Interconnect (NIC-301) and Advanced Quality of Service (QoS-301)

The ARM CoreLink NIC-301 Network Interconnect provides a highly-configurable interconnect solution supporting AMBA 3 AXI, AHB and APB interfaces. NIC-301 creates a network of interconnect switches with automatic insertion of clock domain and bus protocol conversion bridges. NIC-301 supports a global, per-master QoS priority field that’s passed through the interconnect and used at each arbitration point as well as being exported to slaves such as SDRAM controllers that can use the QoS information to arbitrate internally. QoS values may be fixed, driven from a master, or may be controlled under software from registers within the NIC-301. The network interconnect is easily configured within the AMBA Designer configuration tool graphical user interface (GUI). Extensive support is provided for configuring the number of outstanding transactions (read, write and combined total) at each slave port enabling a degree of support for a fixed QoS mechanism.

From the case study we have seen achieving optimal performance is a fine balancing act between minimizing latency for latency-sensitive and latency-critical masters whilst attempting to maintain sufficient SDRAM controller bandwidth efficiency. To enable post-silicon QoS tuning and optimization ARM offers the CoreLink Advanced Quality of Service (QoS-301) plug-in, a companion product for NIC-301 supporting advanced, in-silicon programmable QoS optimization. The QoS-301 add-on supports three QoS mechanisms:

1. **Hierarchical switch network for the NIC-301 in the AMBA Designer Configuration GUI**
   - Programmable outstanding transactions
   - Address issue rate regulation (also known as TSPEC bandwidth regulation)
   - Latency regulation (to be used in conjunction with a priority-enabled SDRAM controller)

   The three regulator types can be configured per slave interface of the network interconnect enabling per-master regulation. The three schemes may be enabled at run-time either separately or together. For example programmable outstanding transaction regulation (which prevents queues backing up into the interconnect) may be combined with issue rate or latency regulation which limit long-term average
bandwidth but not the latency of individual transactions. Using these mechanisms, bandwidth-hungry masters like GPUs can be constrained in their issue rate to ensure low-latency memory access for latency-sensitive masters like ARM processors. All programming can be done at run-time in-silicon enabling tuning and optimization of system performance on the bench, reducing the need for costly re-spins.

QoS system architecture and design is complex and so the ARM VPE (Verification and Performance Exploration) tool is available to quickly investigate various scenarios. Each simulation in the QoS case-study took just minutes using VPE compared with hours or days using a full RTL simulation of a complete system including CPU and GPU.

Summary

Today's consumer products must support 1080p graphics and video. Cost constraints on the memory system mandate high SDRAM bandwidth efficiency but this directly impacts system latency. Latency-sensitive CPU performance suffers and latency-critical real-time devices like video displays may fail unless latency is controlled. To enable high SDRAM utilization without massive and costly over-provision of bandwidth or very high effective memory latencies, today’s systems need QoS mechanisms. From Little’s Law of queues we’ve deduced there are three possible approaches to SDRAM queue-length management. We’ve explored the three types of QoS mechanism, each of which is supported in the CoreLink Advanced Quality-of-Service (QoS-301) product. QoS enabled lower memory latency for the latency-sensitive CPU whilst maintaining sufficient total system bandwidth. System simulation can be speeded-up massively by using ARM VPE models instead of a full RTL-level simulation, but QoS tuning is so complex that in-silicon programmable optimization is needed and is supported by the NIC-301 with QoS-301. Further work on latency regulation-based QoS is warranted once the new ARM SDRAM controller supporting QoS priority is available.

Acknowledgments

The author would like to thank Andrew Tune and Alistair Bruce who performed the QoS system case-study. More information can be found in their DesignCon 2010 paper.