XBOX 360 SYSTEM ARCHITECTURE

THIS ARTICLE COVERS THE XBOX 360'S HIGH-LEVEL TECHNICAL REQUIREMENTS, A SHORT SYSTEM OVERVIEW, AND DETAILS OF THE CPU AND THE GPU. THE AUTHORS DESCRIBE THEIR ARCHITECTURAL TRADE-OFFS AND SUMMARIZE THE SYSTEM'S SOFTWARE PROGRAMMING SUPPORT.

Jeff Andrews
Nick Baker
Microsoft Corp.

Microsoft's Xbox 360 game console is the first of the latest generation of game consoles. Historically, game console architecture and design implementations have provided large discrete jumps in system performance, approximately at five-year intervals. Over the last several generations, game console systems have increasingly become graphics supercomputers in their own right, particularly at the launch of a given game console generation.

The Xbox 360, pictured in Figure 1, contains an aggressive hardware architecture and implementation targeted at game console workloads. The core silicon implements the product designers' goal of providing game developers a hardware platform to implement their next-generation game ambitions. The core chips include the standard conceptual blocks of CPU, graphics processing unit (GPU), memory, and I/O. Each of these components and their interconnections are customized to provide a user-friendly game console product.

Design principles

One of the Xbox 360's main design principles is the next-generation gaming principle—that is, a new game console must provide value to customers for five to seven years. Thus, as for any true next-generation game console hardware, the Xbox 360 delivers a huge discrete jump in hardware performance for gaming.

The Xbox 360 hardware design team had to translate the next-generation gaming principle into useful feature requirements and next-generation game workloads. For the game workloads, the designers' direction came from interaction with game developers, including game engine developers, middleware developers, tool developers, API and driver developers, and game performance experts, both inside and outside Microsoft.

One key next-generation game feature requirement was that the Xbox 360 system must implement a 720p (progressive scan) pervasive high-definition (HD), 16:9 aspect ratio screen in all Xbox 360 games. This feature's architectural implication was that the Xbox 360 required a huge, reliable fill rate.

Another design principle of the Xbox 360 architecture was that it must be flexible to suit the dynamic range of game engines and game developers. The Xbox 360 has a balanced hardware architecture for the software game pipeline, with homogeneous, reallocatable hardware resources that adapt to different game genres, different developer emphases, and even to varying workloads within a frame of a game. In contrast, heterogeneous hardware resources lock software game pipeline performance in each stage and are not reallocatable. Flexibility helps make the design "futureproof." The Xbox 360's three CPU cores, 48 unified shaders, and 512-Mbyte DRAM main memory will enable developers

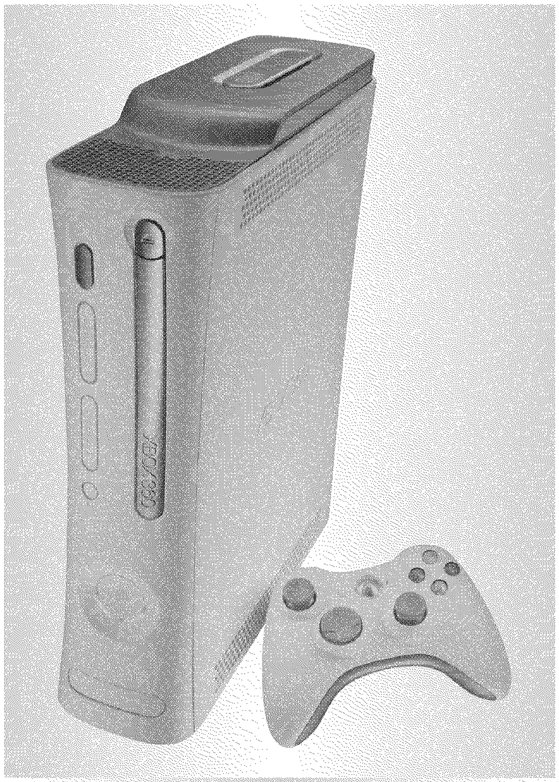
Figure 1. Xbox 360 game console and wireless controller.

to create innovative games for the next five to seven years.

A third design principle was programmability; that is, the Xbox 360 architecture must be easy to program and develop software for. The silicon development team spent much time listening to software developers (we are hardware folks at a software company, after all). There was constant interaction and iteration with software developers at the very beginning of the project and all along the architecture and implementation phases.

This interaction had an interesting dynamic. The software developers weren't shy about their hardware likes and dislikes. Likewise, the hardware team wasn't shy about where next-generation hardware architecture and design were going as a result of changes in silicon processes, hardware architecture, and system design. What followed was further iteration on planned and potential workloads.

An important part of Xbox 360 programmability is that the hardware must present the simplest APIs and programming models to let game developers use hardware resources effectively. We extended programming models that developers liked. Because software developers liked the first Xbox, using it as a working model was natural for the teams. In listening to developers, we did not repackage or include hardware features that developers did not like, even though that may have simplified the hardware implementation. We considered the software tool chain from the very beginning of the project.

Another major design principle was that the Xbox 360 hardware be optimized for achievable performance. To that end, we designed a scalable architecture that provides the greatest usable performance per square millimeter while remaining within the console's system power envelope.

As we continued to work with game developers, we scaled chip implementations to result in balanced hardware for the software game pipeline. Examples of higher-level implementation scalability include the number of CPU cores, the number of GPU shaders, CPU L2 size, bus bandwidths, and main memory size. Other scalable items represented smaller optimizations in each chip.

Hardware designed for games

Figure 2 shows a top-level diagram of the Xbox 360 system's core silicon components. The three identical CPU cores share an 8-way set-associative, 1-Mbyte L2 cache and run at 3.2 GHz. Each core contains a complement of four-way single-instruction, multiple data (SIMD) vector units.[1] The CPU L2 cache, cores, and vector units are customized for Xbox 360 game and 3D graphics workloads.

The front-side bus (FSB) runs at 5.4 Gbit/pin/s, with 16 logical pins in each direction, giving a 10.8-Gbyte/s read and a 10.8-Gbyte/s write bandwidth. The bus design and the CPU L2 provide added support that allows the GPU to read directly from the CPU L2 cache.

As Figure 2 shows, the I/O chip supports abundant I/O components. The Xbox media audio (XMA) decoder, custom-designed by Microsoft, provides on-the-fly decoding of a large number of compressed audio streams in hardware. Other custom I/O features include

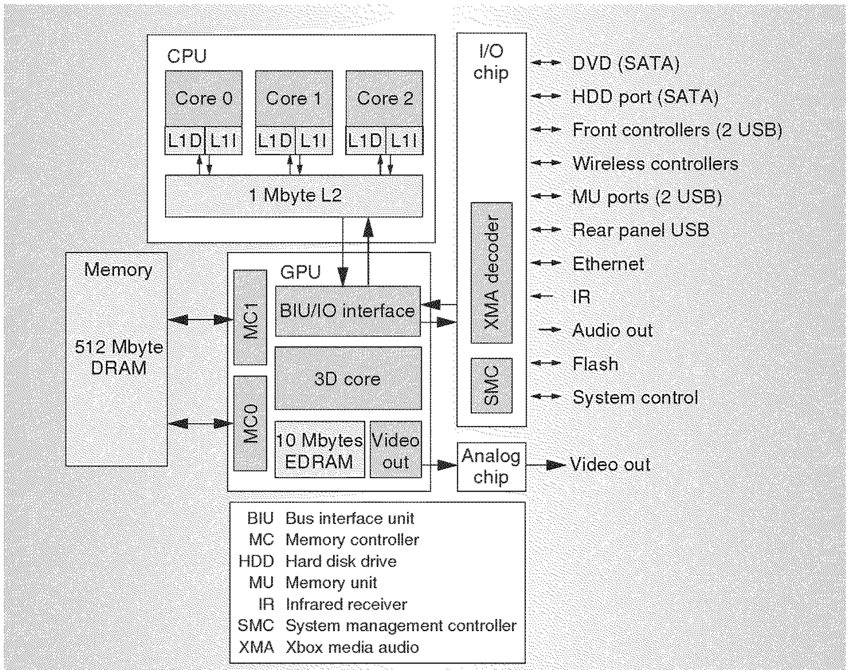

Figure 2. Xbox 360 system block diagram.

the NAND flash controller and the system management controller (SMC).

The GPU 3D core has 48 parallel, unified shaders. The GPU also includes 10 Mbytes of embedded DRAM (EDRAM), which runs at 256 Gbytes/s for reliable frame and z-buffer bandwidth. The GPU includes interfaces between the CPU, I/O chip, and the GPU internals.

The 512-Mbyte unified main memory controlled by the GPU is a 700-MHz graphics-double-data-rate-3 (GDDR3) memory, which operates at 1.4 Gbit/pin/s and provides a total main memory bandwidth of 22.4 Gbytes/s.

The DVD and HDD ports are serial ATA (SATA) interfaces. The analog chip drives the HD video out.

CPU chip

Figure 3 shows the CPU chip in greater detail. Microsoft's partner for the Xbox 360 CPU is IBM. The CPU implements the PowerPC instruction set architecture,[2-4] with the VMX SIMD vector instruction set (VMX128) customized for graphics workloads.

The shared L2 allows fine-grained, dynamic allocation of cache lines between the six threads. Commonly, game workloads significantly vary in working-set size. For example, scene management requires walking larger, random-miss-dominated data structures, similar to database searches. At the same time, audio, Xbox procedural synthesis (described later), and many other game processes that require smaller working sets can run concurrently. The shared L2 allows workloads needing larger working sets to allocate significantly more of the L2 than would be available if the system used private L2s (of the same total L2 size) instead.

The CPU core has two-per-cycle, in-order instruction issuance. A separate vector/scalar issue queue (VIQ) decouples instruction issuance between integer and vector instructions for nondependent work. There are two symmetric multithreading (SMT),[5] fine-grained hardware threads per core. The L1 caches include a two-way set-associative, 32-Kbyte L1 instruction cache and a four-way set-associative, 32-Kbyte L1 data cache. The write-through data cache does not allocate cache lines on writes.

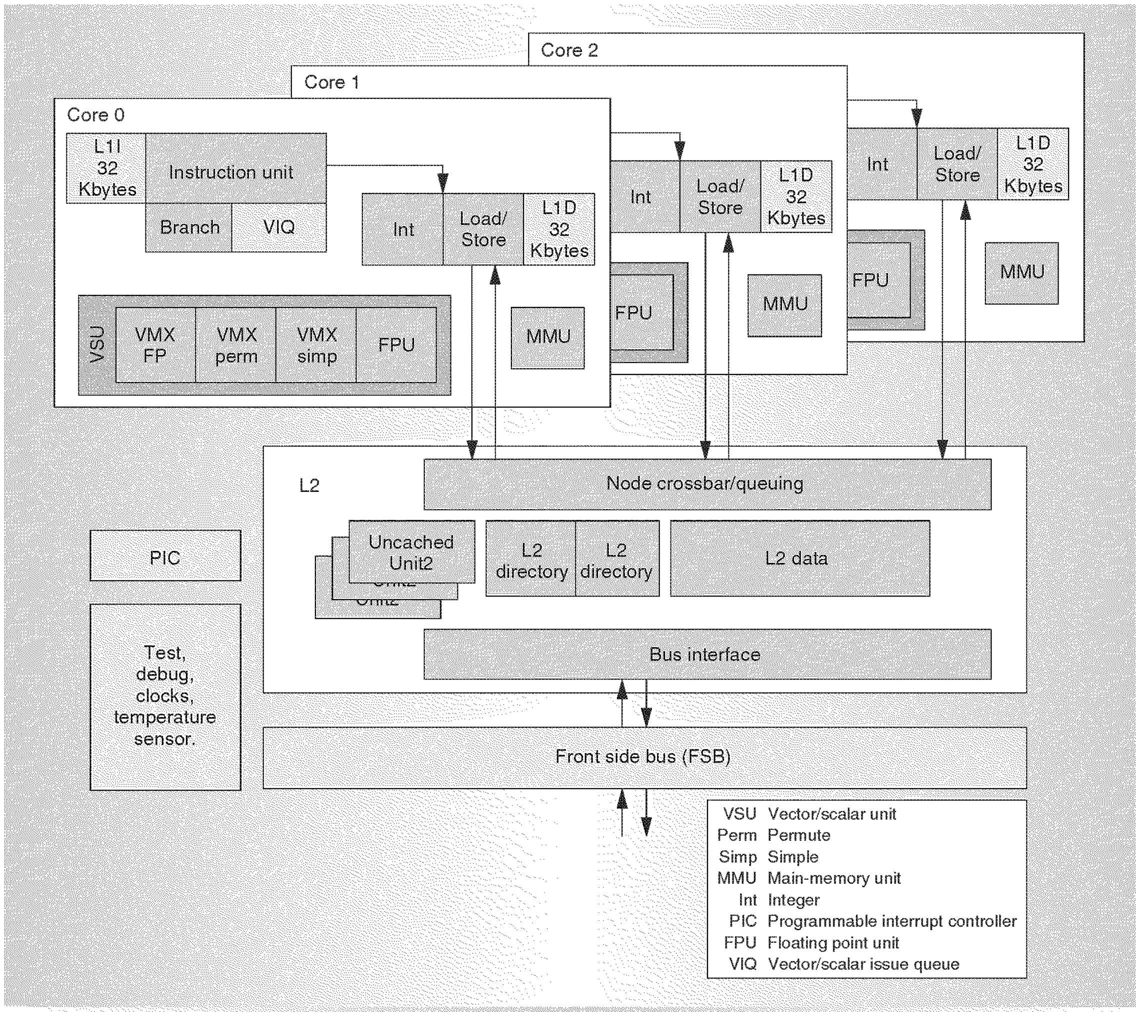

Figure 3. Xbox 360 CPU block diagram.

The integer execution pipelines include branch, integer, and load/store units. In addition, each core contains an IEEE-754-compliant scalar floating-point unit (FPU), which includes single- and double-precision support at full hardware throughput of one operation per cycle for most operations. Each core also includes the four-way SIMD VMX128 units: floating-point (FP), permute, and simple. As the name implies, the VMX128 includes 128 registers, of 128 bits each, per hardware thread to maximize throughput.

The VMX128 implementation includes an added dot product instruction, common in graphics applications. The dot product implementation adds minimal latency to a multiply-add by simplifying the rounding of intermediate multiply results. The dot product instruction takes far less latency than discrete instructions.

Another addition we made to the VMX128 was direct 3D (D3D) compressed data formats,[6-8] the same formats supported by the GPU. This allows graphics data to be generated in the CPU and then compressed before being stored in the L2 or memory. Typical use of the compressed formats allows an approximate 50 percent savings in required bandwidth and memory footprint.

CPU data streaming

In the Xbox, we paid considerable attention to enabling data-streaming workloads, which are not typical PC or server workloads. We added features that allow a given CPU core to execute a high-bandwidth workload (both read and write, but particularly write), while avoiding thrashing its own cache and the shared L2.

First, some features shared among the CPU cores help data streaming. One of these is 128-byte cache line sizes in all the CPU L1 and L2 caches. Larger cache line sizes increase FSB and memory efficiency. The L2 includes a cache-set-locking functionality, common in embedded systems but not in PCs.

Specific features that improve streaming bandwidth for writes and reduce thrashing include the write-through L1 data caches. Also, there is no write allocation of L1 data cache lines when writes miss in the L1 data cache. This is important for write streaming because it keeps the L1 data cache from being thrashed by high bandwidth transient write-only data streams.

We significantly upgraded write gathering in the L2. The shared L2 has an uncached unit for each CPU core. Each uncached unit has four noncached write-gathering buffers that allow multiple streams to concurrently gather and dump their gathered payloads to the FSB yet maintain very high uncached write-streaming bandwidth.

The cacheable write streams are gathered by eight nonsequential gathering buffers per CPU core. This allows programming flexibility in the write patterns of cacheable very high bandwidth write streams into the L2. The write streams can randomly write within a window of a few cache lines without the writes backing up and causing stalls. The cacheable write-gathering buffers effectively act as a bandwidth compression scheme for writes. This is because the L2 data arrays see a much lower bandwidth than the raw bandwidth required by a program's store pattern, which would have low utilization of the L2 cache arrays. Data transformation workloads commonly don't generate the data in a way that allows sequential write behavior. If the write gathering buffers were not present, software would have to effectively gather write data in the register set before storing. This would put a large amount of pressure on the number of registers and increase latency (and thus throughput) of inner loops of computation kernels.

We applied similar customization to read streaming. For each CPU core, there are eight outstanding loads/prefetches. A custom prefetch instruction, extended data cache block touch (xDCBT), prefetches data, but delivers to the requesting CPU core's L1 data cache and never puts data in the L2 cache as regular prefetch instructions do. This modification seems minor, but it is very important because it allows higher bandwidth read streaming workloads to run on as many threads as desired without thrashing the L2 cache. Another option we considered for read streaming would be to lock a set of the L2 per thread for read streaming. In that case, if a user wanted to run four threads concurrently, half the L2 cache would be locked down, hurting workloads requiring a large L2 working-set size. Instead, read streaming occurs through the L1 data cache of the CPU core on which the given thread is operating, effectively giving private read streaming first in, first out (FIFO) area per thread.

A system feature planned early in the Xbox 360 project was to allow the GPU to directly read data produced by the CPU, with the data never going through the CPU cache's backing store of main memory. In a specific case of this data streaming, called Xbox procedural synthesis (XPS), the CPU is effectively a data decompressor, procedurally generating geometry on-the-fly for consumption by the GPU 3D core. For 3D games, XPS allows a far greater amount of differentiated geometry than simple traditional instancing allows, which is very important for filling large HD screen worlds with highly detailed geometry.

We added two features specifically to support XPS. The first was support in the GPU and the FSB for a 128-byte GPU read from the CPU. The other was to directly lower communication latency from the GPU back to the CPU by extending the GPU's tail pointer write-back feature.

Tail pointer write-back is a method of controlling communication from the GPU to the CPU by having the CPU poll on a cacheable location, which is updated when a GPU instruction writes an update to the pointer. The system coherency scheme then updates the polling read with the GPU's updated

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming inter-face) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.