Write Combining Memory Implementation Guidelines

Order Number: 244422-001

November 1998

Table of Contents

WHY SHOULD I READ THIS PAPER?

If you are someone who cares about graphics performance in a system based on the P6 family processor (NOTE: Through out this document, *P6 family processor* is used to refer to Pentium® II processor, Pentium Pro processor, Intel® Celeron™ processor and Pentium II Xeon™ processor), then this document is important reading. It is specifically targeted at IHV's and OEM's and articulates key hardware & software considerations needed to obtain graphics performance in a P6 family processor system. The key focus of this document is on PCI-based 2D and 3D graphics accelerators.

The dynamic execution architecture introduced with the Pentium Pro processor offers unprecedented performance levels for Intel Architecture (IA) software. However, obtaining commensurate graphics I/O performance levels requires clear understanding and use of the processors WC (write-combining) technology. This document will explain WC technology basics as well as key considerations for how you should implement it in your HW & SW. Properly implemented, WC technology enables the P6 family processor to achieve graphics I/O performance unachievable with the Pentium® Processor.

As P6 family processors volume continue to ramp, P6 family processors graphics performance and benchmarks will increase in their use and visibility. Please read this document carefully and ensure that you are using WC technology to get optimized graphics performance - and to ensure that your graphics systems and subsystems remain competitive. OEMs are aware of the performance they can achieve with WC enabled graphics controllers, and will demand that level of performance.

Processor/System Configuration	**Typical Bandwidth(MB/s)**
Processor Writes to UC/PCI memory	8
Processor Writes to UC/PCI memory with PCI posting enabled	20
Processor Writes to WC/PCI memory without PCI posting enabled	40
Processor Writes to WC/PCI memory with PCI posting enabled	100+

Table 1. UC = UnCachable memory, WC = Write Combining memory. See system configuration data at end.

INTRODUCTION

This document provides technical and strategic details to enable graphics controller vendors to take advantage of the performance gains provided by the P6 family processor WC memory type. The WC memory improves the processor write performance by combining the individual writes that a processor may make to a particular memory region (like a video controllers frame buffer) into a burst write containing many aggregated individual writes. It is easy to see the performance advantages of WC when for example we change a processor that writes 32 pixels to a frame buffer via 32 individual bus transactions into a processor which writes all 32 pixels at the same time with little to no incremental transaction overhead. However, WC is not naturally compatible with all graphics controller designs particularly those with FIFO based interfaces. This document also outlines a graphics controller interface that is WC compatible and is based on a memory mapped register interface, possibly double buffered for the best performance.

GRAPHICS PERFORMANCE ON P6 FAMILY PROCESSOR

The bus architecture of the P6 family processor is optimized for burst performance. This allows the processor to achieve a very high memory bandwidth. Since graphics writes from the processor are most often pixel writes and as such tend to be 8-bit, 16-bit or 32-bit quantities rather than full cache lines, a processor would normally be unable to run burst cycles for graphics operations. In previous Intel Architecture processors, like the Pentium processor, graphics-like data writes have been sent to the system bus and have been grouped or combined into burst writes in the chip set. The uncached transaction performance characteristics of the P6 family processor implementation have made that architectural model less effective. Because of this, the P6 family processor was designed with a new caching method, or *memory type*, that allows internal buffers of the processor to be used to combine smaller (or partial) writes automatically into larger burstable cache line writes. The table above shows typical bandwidth results for Pentium Pro processor system with differing configurations varying from 8MB/s with all tuning options disabled through to sequential WC memory writes which can saturate a 33MHz PCI subsystem. These performance numbers could be contrasted with Pentium processor, 430 PCIset uncached PCI performance of ~70MB/s. Cache line like bursts to graphics space then have the potential for being combined into even longer bursts by the chip set. To further enhance graphics performance, the 82450/82440 PCIsets were designed for the Pentium Pro processor with features such *as Outbound Posting* (OBP), *Burst Write Assembly* and the ability to run Memory Write Invalidate PCI bus commands.

For applications to harness the maximum performance of the P6 family processor it is essential that operating system and driver software allow the system bus to be utilized, as the initial architecture design intended.

WRITE COMBINING

Once a memory region has been defined as having the WC memory type, accesses into the memory region will be subject to the architectural definition of WC:

> WC is a weakly ordered memory type. System memory locations are not cached and coherency is not enforced by the processor's bus coherency protocol. Speculative reads are allowed. Writes may be delayed and combined in the write combining buffer to reduce memory accesses..

What does this really mean? Writes to WC memory are not cached in the typical sense of the word cached. They are delayed in an internal buffer that is separate from the internal L1 and L2 caches. The buffer is not snooped and thus does not provide data coherency. The write buffering is done to allow software a small window of time to supply more modified data to the buffer while remaining as non-intrusive to software as possible. The size of the buffer is not defined in the architectural statement above. The Pentium Pro processor and Pentium II processor implement a 32 byte buffer. The size of this buffer was chosen by implementation convenience rather than by performance optimization. The buffer size optimization process may occur in a future generation of the P6 family processor and so software should not rely upon the current 32 byte WC buffer size or the existence of just a single concurrent buffer. The WC buffering of writes has another facet, data is also collapsed e.g. multiple writes to the same location will leave the last data written in the location and the other writes may be lost.

So if the data is delayed inside the processor how do pixels get to the screen where I want them? On the current P6 family processors, once software writes to a region of memory that is addressed outside of the range of the current 32byte buffer the data in the existing buffers will automatically get forwarded to the system bus and written to memory. Therefore software that writes more than one 32byte buffers worth of data will ensure that the data from the first buffers address range is forwarded to memory. The last buffer written in the sequence may be delayed by the processor a little longer unless deliberately propagated to memory by software. The main message here is that despite the fact that data is delayed in the processor the natural software operations of moving data will cause the data in the WC buffers to be written to memory. A caution at this stage: Software developers should not rely on the fact that there is only one active WC buffer at a time. If software cares about data being delayed developers **must** deliberately empty the WC buffers and not assume the hardware will. The WC buffer is also propagated to memory with the occurrence of a range of external events, a subset of which is listed below. In the event that pixels are being written to a linear frame buffer it is worth noting that there may be a small number of them delayed in the last WC buffer, assuming that the buffer is not deliberately emptied. The maximum delay time for these pixels would be dictated by arrival time of the next interrupt which typically occur at a higher rate than the video frame refresh rate. At worst the missing pixels would be presented to the screen at the next frame refresh.

What does the "buffer is propagated to memory mean"? Once the processor has started to move the data in the WC buffer it must make a bus transaction style decision based on how much of the buffer contains valid data. If the buffer is full e.g. all 32 bytes are valid, the processor will execute a burst write transaction on the bus that will result in all 32 bytes being transmitted on the data bus in 4 bus clocks. This burst can be repeated without any clocks delay i.e. sustained data at a rate of 32bytes of data every 4 bus clocks. . If one or more of the WC buffer's bytes are invalid e.g. have not been written by software, then the processor will start to move the data to memory using "partial write" transactions on the system bus. There will be a maximum of 4 partial write transactions for each WC buffer sent to memory. Each partial write will take one data bus clock and may contain up to 8 bytes (one chunk) of information, the validity of each byte in the chunk being indicated by the appropriate byte enable signals for the transaction.

Will the WC buffer be propagated to memory with the same predictable data order each time? Generically no. Once the WC buffer contents have started to be propagated to memory the data is subject to the weak ordering semantics of it's definition. Ordering is not maintained between the successive allocation/deallocation of WC buffers e.g. writes to WC buffer 1 followed by writes to WC buffer 2 may appear as buffer 2 followed by buffer 1 on the system bus. When a WC buffer is propagated to memory as partial writes there is no guaranteed ordering between successive partial writes e.g. partial write for chunk 2 may appear on the bus before partial write for chunk 1 or visa-versa.

What system bus ordering is guaranteed for WC? The only elements of WC propagation to the system bus that are guaranteed are those provided by transaction atomicity. On the P6 family processor a completely full WC buffer will always be propagated as a single burst transaction. In a WC buffer propagation where the data will be propagated as partials, all data contained in the same chunk (0 mod 8 aligned) will be propagated simultaneously.

How do I deliberately propagate the WC buffer to system memory? The architectural memory model of WC was defined such that a UC load or store would operate as a double-sided fencing operation. This implies that once a UC store is issued by the processor all preceding WC activity is complete and no following WC activity has yet started. Many other events in the processor are architecturally defined such that they cause the WC buffers to be deallocated internally and propagated to the bus. For a full list refer to chapter 11 of the *Pentium® Pro Processor Developers Guide Volume 3* (242692-001) A subset of the list is:

- A WBINVD instruction is executed.
- The FLUSH pin is activated.
- Whenever the processor detects a hardware context switch, via a write to CR3.
- Before every *INPUT/OUTPUT* instruction.
- Before every interrupt is serviced.
- Before the IRET instruction.
- Before locked atomic RMW instructions.
- Before any UC memory type load or store.

WC PERFORMANCE GAIN EXPECTATIONS

The amount of performance increase derived from a WC implementation will vary from a lower bound of 0 to an unknown upperbound. Intel has observed an actual performance increase of over 1000%. The 1000% number will not be typical however. There are a number of issues that will determine what level of performance increase will be seen. These include, the graphics controller architecture, the OS, and the mechanism the application uses to place pixels on the screen. First, today's graphics controllers must be able to support processor generated WC transfers directly to the linear frame buffer. Second depending upon the video requirements of the application there may not exist a bottle neck that is slowing graphics performance at the interface of the P6 family processor and the PCI graphics device e.g. the applications compute to video display ratio is very high. Any given graphics controller may show a wide range of performance variation with various different OS implementations. This is because in some cases the application is accessing the video subsystem by commands through the API that are accelerated by the graphics controller. In this case all of the writes would be occurring to an area of the graphics controller that requires strong memory ordering, and therefore no performance increase would be given by WC. In the same system, there may exist another application that is written such that it directly interacts with the framebuffer. In this case a significant improvement would be observed because the linear frame buffer is an area of memory that can be mapped as a WC region. Different OS vendors have provided different graphics architecture strategies over the years that will also demonstrate this differing behavior.

Graphics performance should improve rapidly as graphics controller, and OS vendors take advantage of the dynamic execution architecture of Intel processors. Industry standard benchmarks, today, do not reflect one significant attribute that a WC enabled graphics controller will bring. This attribute is lower processor utilization at potentially the same or higher graphics performance. Even in systems where there is no bottleneck from the processor to the graphics controller that directly impacts graphics performance, as in the case where the CPU can supply data to the GC faster than it can consume the data already, system performance may still improve. This is because the graphics task is consuming less of the processor and system bus bandwidth. With multitasking OSs other useful work is possible with that incremental processor and bus bandwidth. If the graphics controller operates correctly with WC enabled, there should be no situation where performance would ever deteriorate below that of the UC performance of the controller.

WC DEPLOYMENT STRATEGY

The fundamental goal in deploying a new feature like WC memory type throughout the PC industry is the development of a usage architecture that guarantees continuous and successful operation without user intervention, such as in a Plug & Play like model.

It is NOT possible to simply enable the WC memory type with all graphics controllers. WC is a "weakly ordered" memory type that removes the guaranteed order of stored data arrival. Strongly ordered data delivery is a concept upon which the hardware of many of today's graphics controllers are reliant. There are memory ranges in some existing graphics controllers where weakly ordered data has no negative effect on functionality. After the video controller hardware has been determined to be compatible with the "weakly ordered" WC memory type, the following steps are required.

1. Determine the base address and size of the linear frame buffer in the memory range allocated to the graphics subsystem.

2. Search the MTRRs (0 through 5) to determine the first unused one and also to guarantee that this range has not already been defined WC (although there is no jeopardy in defining the same range twice, it does waste MTRRs.)

3. Load the base address of the frame buffer into the Physical Base register of the MTRR and calculate the Physical Mask and load it into the appropriate register in the MTRR and enable the MTRR with the WC memory type.

There are eight MTRR registers in the Pentium Pro processor and Pentium II processor. These registers determine the processor and bus behavior for each address used or accessed by the processor. These registers are a global resource shared by all elements of software. In the future an operating system will need to arbitrate access to the registers as they are requested by different device drivers and if necessary reconfigure the system to maximize resource coverage. Intel has documented the upper two of the eight total register pairs as being reserved for operating system usage. The initial 6 register pairs are open to system BIOS configuration. Refer to the *Pentium® Pro Processor Developers Guide Volume 3,* Chapter 11 section 11 for details on the MTRRs and their initialization. This chapter also includes an example memory mapping on page 11-20. Further MTRR initialization examples are included at the end of this paper.

√ **Note:** *Any software attempting to directly configure a MTRR pair must start a scan from MTRRPhysMask[0] through to MTRRPhysMask[5] and select the first entry that has the enable flag (bit 11) clear on each processor installed in the system. It is not possible to assume that any given register pair is unused or available without first testing their validity. MTRRs must remain uniform across all processors installed in a system.*

There is a broad range of potential WC platform support and enabling models. The following list is prioritized in terms of implementation suitability:

1. Operating System & Video Driver Support

2. System BIOS Support

3. Direct Video Driver Support

4. Separately Executed Configuration Utility

5. Video BIOS Support

Operating System & Video Driver Support

This is the Intel preferred solution. Intel has been working with Operating System vendors to develop WC based application programming interfaces (APIs) that allow the video driver developer to have the MTRRs initialized and managed by the OS. This approach removes the need for multiple WC intelligent video drivers from the IHV, with the WC intelligence being provided by the OS. Obviously this strategy is only operable for new OS's or updates to existing operating systems. The first operating system to deliver a WC capable API was Windows*NT(4.0).

WindowsNT V4.0

To enable WC memory from the video driver: In the adapter specific miniport driver the call to IOCTL_VIDEO_MAP_VIDEO_MEMORY should have the InIoSpace flags set to include at least the following:

VIDEO_MEMORY_SPACE_MEMORY | VIDEO_MEMORY_SPACE_P6CACHE

such that when the calls are made to:

VideoPortMapMemory() and to
VideoPortGetDeviceBase()

the InIoSpace flags are propagated through.

Note: For WindowsNT 4.0 the only supported WC, MTRR, initialization method is via the operating system based API. Direct video driver manipulation of the MTRR registers is not supported.

Windows*95

Microsoft has defined a set of APIs that will allow drivers to initialize and utilize the WC memory type. This API set are planned to be made available in a future version of Windows95.

API Name	Routine Description	Arguments	Return Value
_MTRR_Get_Version	Get MTRR support version.	None	STC if no support. CLC if supported
_MTRRSetPhysicalCacheTypeRange	This function sets a physical range to a particular cache type. If the system does not support setting cache policies based on physical ranges, no action is taken.	PhysicalAddress The starting address of the range being set NumberOfBytes The length, in bytes, of the range being set CacheType The caching type for which the physical range is to be set.	STATUS_SUCCESS If success, the cache attributes of the physical range have been set (or feature not supported or not yet initialized). STATUS_NO_SUCH_DEVICE If FrameBuffer type is requested and is not supported. STATUS_UNSUCCESSFUL/STATUS_INTERNAL_ERROR Requested Physical Range is below 1M or other error.

Other Operating Systems

Other operating systems will be handled in a case by case basis where each have the opportunity to implement one of the above listed options. System OEMs are strongly encouraged to engage with their sources for video controllers, boards and drivers with a goal of acquiring WC capable solutions. As an interim solution OEMs could implement their own system BIOS based solutions, see system BIOS support section below.

System BIOS Support

Providing WC support via the System BIOS is a viable process for a restricted number of video cards i.e. not a general purpose solution. The system OEM would need to specifically identify the presence of the video card or group of cards for which they know it is safe to enable WC and set up the MTRRs accordingly. This model suffers from the general LFB relocation and MTRR management issues raised above but still forms the basis for a solution under direct OEM control. OEMs should be aware that they must first detect the graphics controller that is in the system before setting any MTRRs for the frame buffer. If BIOS senses an unknown graphics controller e.g. one inserted by a user after system shipment, there is little chance that the second graphics controller would map to the same frame buffer area and WC should not be configured. See the √ note above.

Direct Video Driver Support

It would be possible for video device drivers, executing at a privilege level of 0, to detect the presence of the P6 family processor and to enable WC memory type for the video card. The primary disadvantages of this mechanism are that processor sensitive device drivers are required and that the operating system has no knowledge that performing a LFB relocation has other side effects. This option has the advantage that a properly implemented solution will allow a graphics controller IHV to supply WC enabled driver solutions for any OS the user has chosen. It will almost certainly be necessary for the graphics controller IHV to support users of OS that have been superseded by their manufacturer but are still in use in some environments. See the √ note above.

Windows*3.1/DOS

Intel knows of no plans to update Windows3.1, for this problem, in the future. DOS does not use a video driver concept that lends itself as a base for WC support. For Windows3.1 the preferred implementation mechanism would be via a video driver update (#3 above) with the IHV specific configuration application (#4 above) as a secondary choice. The IHV specific, configuration application is seemingly the only choice for DOS users. Intel will work with the video IHVs by providing them access to a parameterless configuration utility in source and binary form that will be easily modifiable by the IHV to tailor it to their requirements.

Separately Executed Configuration Utility

It would be possible to implement WC support via an, IHV specific, configuration application. This is a viable alternative when the OS does not have native support for WC. See the Windows3.1/DOS section above.

Video BIOS Support

The controller Video BIOS is technically a potential point of WC support. Practically, supporting WC in a video BIOS would require an update to all existing hardware in service and as such has not been seriously considered as a viable alternative.

In all of the above listed solutions, it will be necessary for the OEM and the graphics controller vendor to work closely together in identifying the graphics controllers that support WC and the mechanism for positive identification of the correct address range for each physical memory size and graphics mode.

AD HOC Solutions

There are software utilities available on bulletin boards and the Internet that may or may not work for a particular graphics controller. Since these software utilities have not been validated by either Intel or specific system suppliers, Intel is strongly discouraging the use of these programs, encouraging the users to instead contact their OEM or graphics controller IHV for a validated solution.

Paged Video Region(A0000h)

There are numerous applications that still access the video subsystem through the 128KB region mapped at A0000h and use this window to page through a linear frame buffer positioned in high memory. Given that the video controller is normally capable of being used in conjunction with WC memory types the paged video region could be mapped as a WC memory type also. This is not supported in any of the operating systems developing WC API support nor in the application developed by Intel. The intentional lack of WC support for this video region stems from the complexities of maintaining compatibility with previous generations of hardware and software e.g. When strict VGA compatibility is required WC memory should NOT be mapped into this space. VGA compatibility requires in-order completion of writes that trigger and control side effects within the VGA subsystem. If the video controller is operated out of VGA compatible mode then WC mapping is a possibility. This complication restricts the WC implementation model for the paged video region to specific video drivers or a separate IHV specific configuration application ONLY.

HARDWARE DESIGNING FOR WC

Graphics Accelerators(GA)

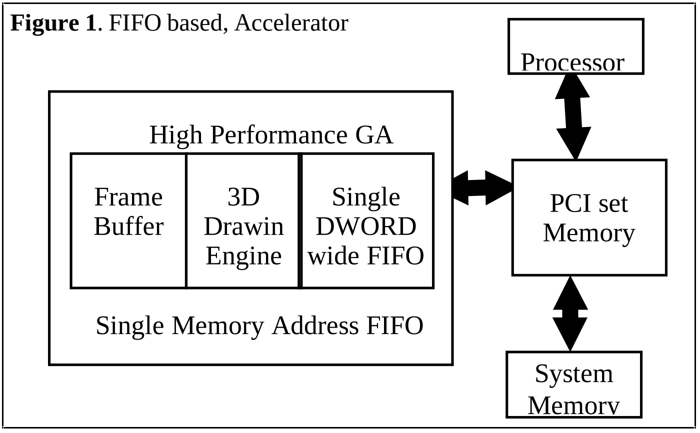
This section addresses the design characteristics of Graphics Accelerators without distinguishing between 2D or 3D architectures. 2D architectures will be addressed specifically later. The 2D and 3D controllers perform very similar processes and can easily be treated for the most part as having a common system architecture. The GA is essentially a coprocessor for performing graphics operations. In either a 2D or 3D environment, the program generates graphics by supplying geometry that describes the scene to be drawn to some API. For a simple linear frame buffer based card, the processor does all of the work to turn the geometry into finished pixels, and writes the finished pixels into the linear frame buffer region or display memory on a simplistic graphics card. The primary function of the simple graphics card is to turn the contents of this display memory into video so it can be displayed on a monitor.

For a GA , the processor performs some numerical calculations (FP or Integer) on the geometry given to the API, then writes the results of these calculations (still a geometric description of what is to be drawn, but now in a card specific format) to the GA, and it is the GA (not the processor) which renders the geometry to finished pixels that are then stored into the display memory. Unlike a simple graphics controller, the processor will rarely interact directly with the display memory on the GA, all changes to the screen are accomplished via the command protocol.

Current high performance GA use one of three I/O models:

1. FIFO (Figure 1)
2. Memory mapped control registers (Figure 2)
3. Bus Mastering model (Figure 3)

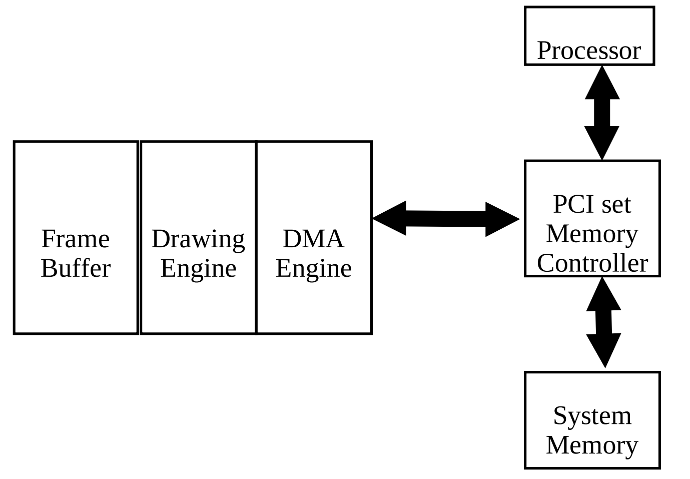
Figure 1. FIFO based, Accelerator



1. FIFO based models were once very common for high performance GA because the coding efficiency of the inner loops in a FIFO based I/O model, coupled with data only moving once across the system bus (compared to twice in a Bus Master Model) gives what designers feel is the highest performing most scaleable implementation. The byte stream from the processor is written to a single memory address. With this type of GA subsystem software synchronization between the processor and the GA is an area that has always required careful attention. Optimum synchronization is required so that the execution performance of both processor and GA is maintained and not wasted through inefficient polling techniques or hardware retries. Pure FIFO based models require strict in order memory semantics which are only possible on a P6 family processor when using the UnCached (UC) memory type. I/O to a FIFO requires a UC memory type and as such will not allow effective use of the P6 family processor bus bandwidth nor will it yield the best compute capability. **Intel recommends that IHV's who have FIFO based GAs either redesign them to be bus masters or ensure that they are not integrated in P6 family processor based systems.**

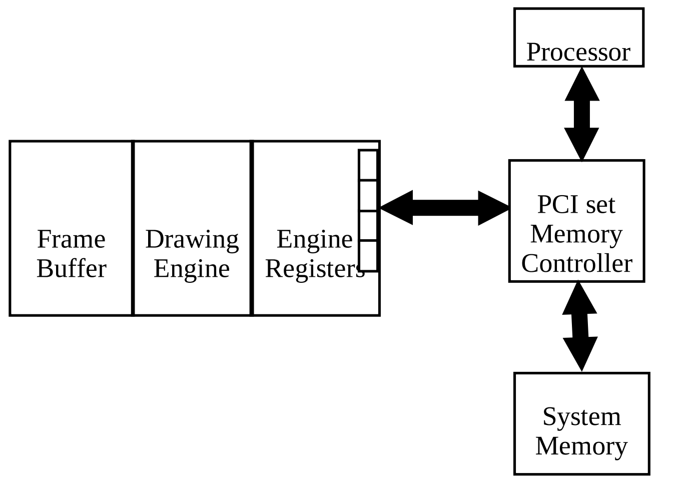
2. **The memory mapped control register** I/O model is similar to the FIFO based model, however, instead of writing a byte stream to a single memory address, different registers are mapped to different memory locations. The device driver writer modifies the contents of specific registers to program the card for the graphics operation desired and then writes a value to a specific register to initiate the command. In this programming model data can be sent to the accelerator in any order as long as all registers are set to the correct values before the start command is sent to the accelerator. This architectural model may be able to utilize and gain the performance advantage of WC memory. If accelerator registers are contiguous in memory WC should be implemented and there should be a significant, throughput, performance gain. It is important for the video driver to ensure WC buffer serialization is complete before signaling the card that drawing should commence. As with the FIFO model careful attention to GA/processor synchronization is required.

Figure 2. Memory mapped control registers based, Accelerator



3. **Bus Mastering model**: The application creates a high level command execute buffer, resident in main system memory, that the video driver converts into a graphics controller specific command/data stream. When the buffers are either full or reach some high water mark, a command is sent to the accelerator instructing its DMA engine to transfer the buffer to the GA card and execute its contents. While this transfer and execution is taking place, the processor is free to concurrently build another execute buffer in main system memory overlapping the construction of one set of

Figure 3. Bus Mastering based, Accelerator



commands with the delivery and execution of another. Synchronization between processor and GA is again important however given the ability to increase the relative buffer sizes can be reduced in significance. Bus mastering in this way ensures that data is streamed to the GA in sequential order. It should be noted that adding bus mastering capabilities to a GA with a FIFO interface would be an effective way of ensuring an in order data flow as is required for the FIFO design. Bus mastering GAs are currently the highest performing of the I/O models noted above, this is possibly due to the lack of WC support in today's controllers.

Bus mastering across the P6 family processor bus increases the bandwidth requirements of the system bus and the memory system. If the system memory buffers are mapped as Write Back(WB) memory type, data has the potential to flow across the system bus many times:

- From the application buffer to the processor when the GA specific execute buffer is first written.
- From the processor cache to memory as the GA specific execute buffer is eventually evicted
- From the memory system to the accelerator during the bus mastering transfer.

It is possible the data will only flow twice if it is still modified or exclusive in the processor cache during the bus mastering transfer, but studies show this case to occur infrequently. As triangle rates increase this increased burden on the memory system and system bus will limit the data access rates and therefore execution capabilities of the processors, preventing them from accessing the instructions and data they require to perform non-graphics related work. In FIFO and control register models data only flows once across the system bus in all cases, easing the burden on the system bus and memory system.

Intel Recommended Graphics Accelerator Architectures

As system technology develops the architectural design of add-in modules will need to change to provide a high performance, correctly balanced system. This section provides a high level roadmap, identified by system technologies, for add-in module design.

Pentium Pro processor/440FX PCIset time period

This is basically a recommendation for today's technology. Current graphics accelerators do not uniformly support the WC memory type due to their FIFO architecture. The recommendation for this time period is for graphics controller architectures to make the move to the bus mastering model with both application and driver specific execute buffers resident in normal system memory.

Pentium II Processor card/440LX PCIset time period

The 440LX introduces AGP as a new technology which will provide a mechanism for the bus mastering model to operate with a slightly lower bandwidth requirement from main system bus. AGP will allow the mapping of a portion of main system memory as WC memory type and provide the GA access to this memory space via the GART. The GART/WC element of system memory will be used for texture data but could be used to host the GA specific execute buffers required in the bus mastering model. In this environment the data flow would then be:

1. Application creates command buffer, in WB system memory, and hands the buffer off to the graphics driver
2. Graphics driver translates the system memory resident command buffer into the GART/WC resident GA specific execute buffer and initiates the DMA process.

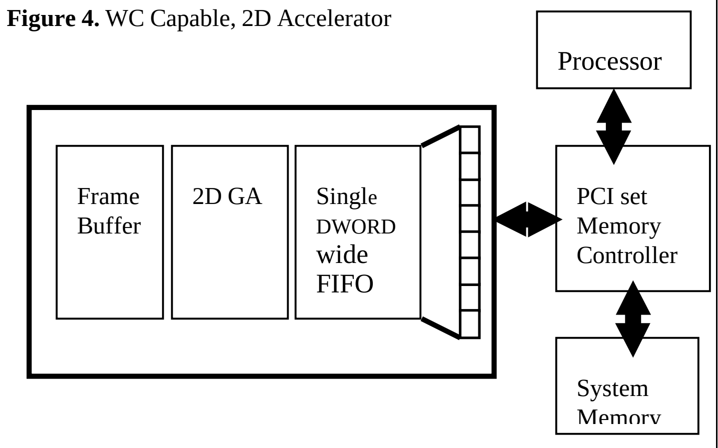
- Once the data is resident in GART/WC space the snoop traffic that would be associated with a DMA read of the graphics specific execute buffer resident in WB memory space is eliminated
- Cache efficieny reduction associated with multiple buffers resident in WB memory are reduced.
- Bus mastering scatter gather functionality that would normally be required may be provided by the GART interface thus reducing GA complexity.

Future Processor card/Future chipset time period

Once the next generation of both processor and chipset are available increases in bus bandwidth and the processors computational capabilities will show benefits from a true read once write once mechanism for buffer conversion. At this stage migration of the controller specific execute buffer from GART/WC memory to the GA's own memory space should be completed. This will allow the graphics driver to read the application buffer in WB system memory and write the controller specific buffer in WC memory on the GA itself. This would require the minimum of system and I/O bus bandwidth and allow for scaleable graphics architectures. See the *Processor - Graphics Accelerator Synchronization* section below for implementation guidelines.

2D GA Specific Considerations

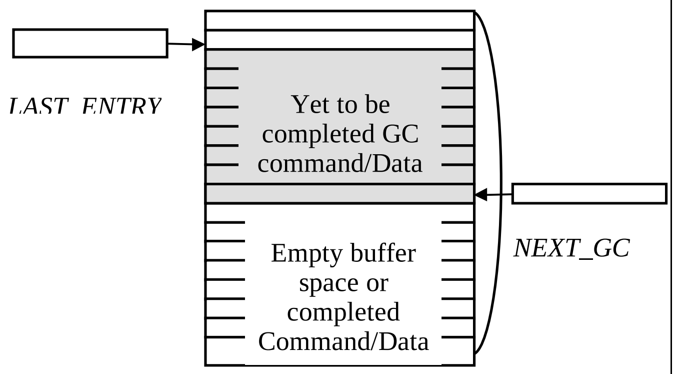
Figure 4. WC Capable, 2D Accelerator



The primary, system visible, difference between 2D and 3D GA's is the amount of data that must pass between the processor and the GA per drawing command. The 2D commands require approximately 8 DWORDS to sent from the processor to the GA. The 3D commands may require more than three times that number. The implication here is that in a bus mastering architecture the initialization overhead of 2D is >3X that of 3D. Based on this data and the fact that a WC buffer is 8 DWORDS in length, Intel believes that a 2D GA would be best implemented with a WC capable front end. This front end would look like an 8 DWORD communication buffer in a WC mapped memory space. Once all 8 DWORDs of the buffer have been written they could be passed sequentially through the existing FIFO design. The driver should be written to ensure that the WC buffer is written without pre-emption and therefore would maintain the ordering required.

Processor - Graphics Accelerator Synchronization

Figure 5. GC Synchronization Buffer


LAST_ENTRY
Yet to be completed GC command/Data
Empty buffer space or completed Command/Data
NEXT_GC

Processor to GA synchronization is a critical area for attention during the design phase for both the hardware and driver software of a graphics subsystem. It is clear that the smaller the communication buffers used to communicate with the GA the more often software synchronization will need to occur and therefore the more opportunity for improving performance with the correct synchronization algorithm. The days when simple polling were sufficient are gone as is the luxury of having an interrupt deliver an appropriate message. Interrupts can be a very dramatic performance loss to a processor when delivered with a high frequency, as would be required from a small buffer based graphics subsystem. Any attempt to allow the I/O bus to handle “excess data” with the bus retry mechanism is a good way to reduce overall system throughput and is a poor substitute for efficient design. It is clear that what is required is a synchronization mechanism that not only indicates that the I/O device is busy but also how busy it is e.g. how much buffer space is left for processor communication. Intel believes that the best interface for a graphics controller would be based on a circular communication buffer structure with a producer/consumer pointer protocol for synchronization without locking. This requires:

1. A reasonable size circular command/data buffer that is memory mapped and eventually, (for future designs would be in the graphics controller memory domain) useable with the WC memory type. The initial implementation of this recommendation would use a system memory resident buffer structure. The processor stores a combination of sequential graphics controller commands and data into this buffer.

2. A pointer (actually a displacement) into the command/data buffer. Pointer *NEXT_GC* is the next data/command to be handled by graphics controller. The graphics controller advances this as it processes data/commands. *NEXT_GC* can be read by the device driver, executing on the processor, but not written.

3. A second pointer *LAST_ENTRY* is the address+1 of last byte of data/command written to the buffer for the graphics controller to process. This is pointer is updated by the device driver after writing a sequence of commands and data into the buffer. This pointer resides in a UC memory space.

This mechanism is detailed in figure 5. The operating sequence is: The device driver reads control registers (*LAST_ENTRY* and *NEXT_GC*). If there is enough room, between these two pointers, for the pending command/data sequence that requires to be processed the device driver writes the command/data to the buffer. The device driver then updates the *LAST_ENTRY* pointer (this is done as a UC STORE). The latter flushes the on-chip USWC buffer before doing the UC-store ensuring that the WC based command/data buffer is updated and the processor based WC buffer is emptied. Copies of *LAST_ENTRY* and *NEXT_GC* are also kept by the device driver in regular memory to avoid unnecessary reads to the control register.

Future Graphics Accelerators

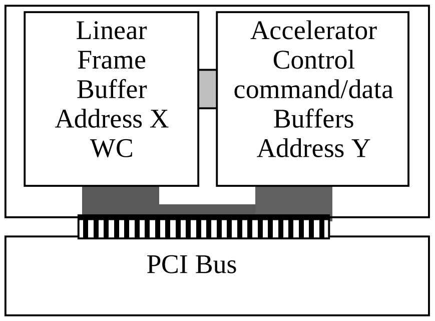
The primary considerations for the design of a WC capable graphics accelerators are:

1. Design a GC interface that can accept full speed I/O bus transactions without stalling.

Have the Linear Frame Buffer and any graphics controller specific command/execute buffers located in separately decoded regions as in Figure 6.

2. Have each of the separately addressable regions individually assigned memory space allocations at boot time by the video/system BIOS. Do not decode a single block of memory that contains both frame buffer and controller registers/buffers.

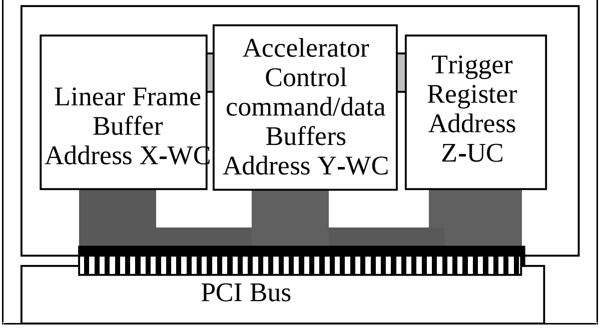
Figure 6. Separate LFB and Accelerator Registers


Linear Frame Buffer Address X WC
Accelerator Control command/data Buffers Address Y
PCI Bus

1. Design the graphics controller buffers to be accessible with no ordering requirements. Implementing a separately addressable trigger register (as in Figure 7.) will remove the need for video drivers to explicitly flush the WC buffers.

2. Ensure that the graphics controller command/execute buffers are sufficiently large or the GA needs to process data as fast as the bus can send it so that concurrency is maintained between the processor and the graphics subsystem.

Figure 7. Separate LFB, Accelerator Registers and Trigger register


Linear Frame Buffer Address X-WC
Accelerator Control command/data Buffers Address Y-WC
Trigger Register Address Z-UC
PCI Bus

3. Support bus mastering to and from all of the different memory spaces, system memory, linear frame buffer, GART etc.

NOTE: Hardware should support a minimum granularity and separation for differing memory types of 4Mbytes for maximum flexibility given the presence of 4MB pages.

By following these guidelines the different memory type requirements of the linear frame buffer and the graphics controller registers can be recognized by the video drivers and each assigned the correct memory type in the P6 family processor. This will result in the highest possible graphics accelerator foundation for the P6 family processor architecture.

WC INITIALIZATION EXAMPLES

This section only provides example values for MTTR initialization. Driver writers should reference the *Pentium® Pro Processor Developers Guide Volume 3* for the full details on how to detect the presence of, availability and initialization of MTRRs in a potentially multiprocessor based system.

A LFB located at A0000000h, 4MBytes in length and mapped WC:

MTRRPhysBasen = 0A0000001h

MTRRPhysMaskn = FFFC00800h

A LFB located at FD000000h, 8MBytes in length and mapped WC:

MTRRPhysBasen = 0FD000001h
MTRRPhysMaskn = FFF800800h

A LFB located at 80000000h, 32MBytes in length and mapped WC:

MTRRPhysBasen = 080000001h
MTRRPhysMaskn = FFE000800h

A LFB located at FD000000h, 8Kbytes in length and mapped WC:
Note This mapping should not be requested for processor steppings prior to a CPUID return value greater than 617h.

MTRRPhysBasen = 0FD000001h
MTRRPhysMaskn = FFFFFE800h

TEST SYSTEM CONFIGURATION

The performance measurements noted in table 1 above were measured on the following system configuration

Intel Alder systemboard, 200MHz Pentium Pro processor with 256Kbyte L2 cache.

64Mbytes main system memory
Adaptec 7880 SCSI subsystem
Seagate Barracuda 1G HDD
Matrox Millennium MGA graphics controller
3DFx Obsidian 2200 3D accelerator
WindowsNT 3.51

The video bandwidth measurement software, developed by Intel, BW32L.EXE is supplied in the attached package. The test software was run with a block size of 4Kbytes