Choosing a Platform Architecture for Cost Effective MPEG-2 Video Playback

---

Platform Architecture Labs/Platform Technical Marketing
Desktop Products Group
Intel Corporation

April 1996

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

MPEG is an international standard for video compression/decompression promoted by ISO. Implementations of MPEG CODECs, or MPEG-enabled platforms may require licenses from various entities, including Intel Corporation. Intel makes no representation as to the need for licenses from any entity. No licenses, either express, implied or by estoppel are granted herein. For information on licensing Intel patents, please contact Intel.

Intel retains the right to make changes to these specifications at any time, without notice.

* Other brands and names are the property of their respective owners.

Choosing a Platform Architecture for Cost Effective MPEG-2 Video Playback

Paul Buckley, Bob Faber, Rick Mangold, Tim Mostad, Joe Nardone
Platform Architecture Labs/Platform Technical Marketing
Intel Corp. Desktop Products Group
Revision 1.0, 4/12/96

Introduction

Adding movie playback to personal computers is a natural step in the continuing evolution of PCs as entertainment devices. MPEG-1 has recently become established on PCs as the standard for video playback, particularly in some geographies. Partly because of inexpensive, widely available content and partly because of a limited installed base of VCRs, MPEG-1 on PCs became an attractive alternative in Asia/Pacific countries and to some extent in Europe. In the U.S., low cost, widely available cable programming and the inherently inferior MPEG-1 picture quality, among other factors, prevented MPEG-1 from gaining any real momentum. However, with the advent of MPEG-2 with its higher resolution picture and with the high density and low cost of new DVD (originally Digital Video Disk or Digital Versatile Disk) drives, a significant new video playback capability is on the horizon.

Left to evolve on their own, MPEG-2 and DVD would probably take a path that would preclude them from having a major impact on the personal computer for some time. From the PC industry's perspective, the worst case scenario would have expensive drives (>$400 U.S.) with integrated MPEG-2 and audio decode with only composite video and 4 or 5-channels of analog audio output. These devices would be the equivalent of today's laser discs. Some amount of PC integration would probably occur but a major opportunity to create new uses and capture new users would be lost.

To maximize the potential for movie playback of MPEG-2 content from DVD drives in PCs, the industry should recognize that there are potential problems and then coordinate to produce the desired results. This paper details efforts underway by Intel's Platform Architecture Labs, one of the main drivers behind system architecture initiatives such as PCI, Plug and Play, and DMI, to provide this capability to the industry.

We have two main objectives:

1. Ensure that DVD and MPEG-2 playback technologies develop for cost effective integration into personal computers.
2. Accelerate the availability of DVD drives and MPEG-2 movie content.

These are very broad, aggressive objectives and require significant commitment to accomplish. Comprehensive programs at Intel are in place to address related issues at all levels, from digital copyright issues to disc production issues. The successful outcome of those programs (whose details are not covered here) must be matched with platforms and software that both have the necessary performance and are cost effective in implementation. The bewildering choices of technologies leaves open the possibility for false starts and dead ends. The industry cannot afford limited or proprietary implementations if we are to be successful in accelerating the availability of MPEG-2 and DVD on PCs. This paper represents some of Intel's commitment in the form of basic research aimed at reducing the burden of assimilating these new technologies by identifying the best way to implement the system architecture.

At Intel, the process of architecting PC platform solutions is referred to as "partitioning". This process is less than intuitive. Partitioning requires a global platform understanding and view. We have developed an extensive tool set that enables us to monitor all aspects of system operation, from the way an instruction gets executed on the processor, to its eventual affect on, for examples, the frame buffer or the cache on the disk drive. This whitepaper is based on results from studies done in Intel's Platform Architecture Labs. It reflects our view of how you can partition your platform to deliver the best performance MPEG-2 movie playback from a DVD drive for the lowest cost.

Accelerating DVD

We believe that DVD drives will develop in two stages. The drive specification is fairly solid: data encoding, track format, and file structure are defined (see Table 1 below) but it will take an intermediate step to get the bus interface to its eventual, lowest cost, goal. Initially the target is bus mastering, ATOP drives that support DMA mode 2 on the standard IDE bus. This will enable rapid integration into existing system architectures. The next step in our plan is to help move the interface electronics to the PC chip set or highly integrated peripheral chip to further cost reduce DVD drives.

The table below compares CD-ROM and DVD-ROM drives. The increased capacity of DVD-ROM drives enables large amounts of video data to be stored on a single disk. The increased resolution brings laserdisk quality video to the PC. The DVD-Movie specification calls for a bit rate of 4.7Mbps continuous transfer, similar to today's 4X CD-ROM drives (both the DVD spec and the MPEG-2 spec allow for higher data rates; 4.7Mbps is specifically for DVD-Movies).

	Standard CDs	**DVD-Video CDs**
CD-ROM Xfer Rate	2-4X	~ 6X (~ 5 Mbps) for DVD-Movie disks
CD-ROM Capacity	650 MB	4.7 GB for single layer/single sided DVD disks
CD Data Rate	1.5 Mb/s fixed	4.7 Mbps (DVD-Movie)
Video Algorithm	MPEG-1 (ISO 11172-2)	MPEG-2 (ISO 13818-2) Main Profile @ Main Level
Video Support	Progressive	Progressive or Interlaced
Video Format	YUV4:2:0	YUV4:2:0
Video Quality	VHS Quality	Broadcast Quality - CCIR 601
NTSC	352x240x30fps	704x240x60 fields/s (interlaced or progressive)
PAL/SECAM	352x288x25fps	704x288x50 fields/s (interlaced or progressive)
Video Aspect Ratio	4:3	4:3 or 16:9
Video Compression	~ 26:1	~ 26:1
Audio	MPEG-1 (ISO 11172-3) 2 channel layer-2 stereo	Dolby AC-3 w/NTSC in US and Japan MPEG-2 5.1 channel (ISO 13818-3) w/PAL in Europe
Audio Sample Rates	32, 44.1, 48 kHz	16, 22.05, 24, 32, 44.1, and 48 kHz
Audio Compression	~6:1	~6:1, but with support for 5.1 channels
Video Run Time	74 min. VHS Quality	133 min./layer with 3 languages + 4 subtitles
Audio Run Time	74 min. CD quality	~8 CD quality CDs

Table 1: Video Playback Comparison

Since the cost reduction strategy is not complete, not much more can be said about DVD at this time. A future design guide will build on this white paper to add more DVD information. For the purposes of this document, it is assumed that cost-effective DVD drives will be in the market in the required time frame and will deliver an MPEG-2 data stream per the above resolution and performance.

Audio Playback

Audio is an important element of the movie playback process, at least equal to that of the video quality. While the eye will tolerate dropped frames or slight picture degradation, the ear is much more sensitive. There is no equivalent phenomena to visual persistence for sound. This means listeners are very intolerant to dropouts but it has been shown that some amount of degradation will be tolerated. CD quality audio is sampled at up to 48 kHz resulting in a frequency bandwidth well beyond the range of human hearing. This bandwidth results in rich sound with excellent clarity but very few people can even detect

the difference between sound sampled at the full rate and sound sampled at 22 kHz. As long as the data stream is maintained without break-ups and the audio is synchronized with the video during playback, there is some amount of flexibility in the way audio decoding can occur on an MPEG-2 playback system.

The target for sound reproduction is always at the full sample rate but if the goal is to keep the platform cost low, then we can use the flexibility that we have with sound reproduction to make system implementation tradeoffs. Flexibility is especially important since there are at least a couple of audio encoding formats in the DVD drive specification (see Table 1 above). The DVD specification doesn't reflect the reality of the PC market where two speakers is typical and multi-channel standards like Dolby's AC3 will probably be mixed down to two channels for playback. For low cost, the ideal decoder implementation would be done on the CPU and output to a high quality, split digital/analog CODEC. This would allow for the system to adjust to the CPU work load and still maintain high audio fidelity.

Other options, like dedicated audio decoders either on the motherboards or an add-in card or even embedded in speakers on the end of a USB cable, are also possible. The wealth of options promises to provide many solutions at differing price/performance points but also means that there is still a lot of work to be done before the all of the options are well enough understood to provide specific design guidance. For the purposes of the whitepaper, like DVD drives, we will assume that the appropriate audio subsystem at the right performance points are available. Many of the tradeoffs and implementation details will be included. in our upcoming design guide. Look for more information on Intel's website (www.intel.com).

This white paper will explore some of the issues involved in MPEG-2 video playback on the PC, based on the analysis thus far. When the analysis is complete, a Design Guide will be published which is meant to provide a method of integrating DVD/MPEG-2 Movie Playback capabilities into the PC platform.

Part I. Platform Partitioning for DVD/MPEG-2: A Balanced System Approach

System Cost/Performance

If the problem was strictly to implement MPEG-2 decode on PCs as quickly as possible, the answer would be clear. Dedicated decoder ICs are already available from a few vendors and others are nearly ready with several more. With the right amount of memory and associated support logic, MPEG-2 decode can be added to a PC platform today. However, as with most things, the price has to be right. With demand in the PC market being highly price elastic, the more a system costs, the smaller the volume that will be sold. Adding significant cost to a platform to include MPEG-2 playback will counter the opportunity to increase the total available market through new uses such as MPEG-2 movie playback.

The trick is to add cost, when necessary, to the right places. The "right places" are the ones that yield general performance benefits and make the platform run better for most applications as opposed to just a few or one. Some of these things need to be part of the basic platform architecture and others can be options which can be added at the point of sale or left for future user upgrade via hardware or software.

For MPEG-2, adding cost to the right places will result in the ability to do partial or full software MPEG-2 decode and playback. Software CODECs may not always offer the highest performance but they are always the least expensive and are of high enough quality to replace dedicated hardware for the mass market. Best of all, they can be added and/or inexpensively upgraded later as needed.

Support For Real-time Processing

As we discovered when we examined what it takes to do software decode of MPEG-1, poor system design causes substandard playback performance. The problem was caused by an incomplete understanding of how a system operates in real-time, particularly in light of the data processing requirements of today's emerging multimedia applications. Where does the microprocessor spend its time? How much value does cache have when executing real applications? Previous generations of PC benchmarks reflected single task environments executing programs that could tolerate long latencies. New real-time, multimedia, applications like MPEG playback required a more sophisticated "platform" approach.

To help characterize the problem we developed a model called the "balanced system approach". It divides system performance elements into the quadrants of a 2x2 matrix:

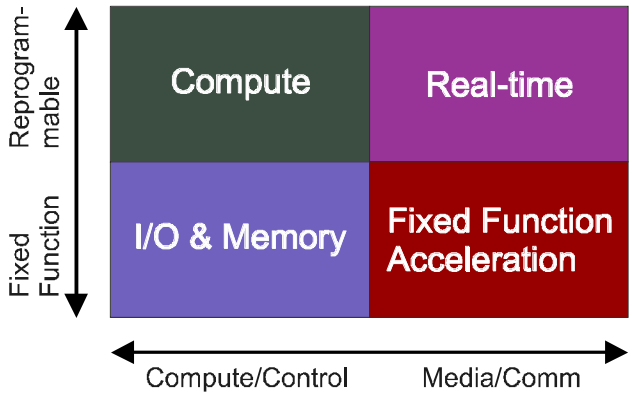

Figure 1. Elements of a Balanced System

We can then approximately map possible system features into the matrix:

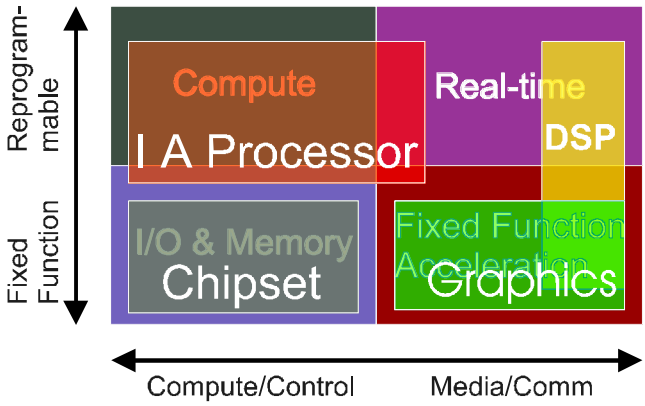

Figure 2. Mapping System Components to the Model

The microprocessor has traditionally been the focus of the platform, serving roles in each quadrant but optimized for integer computing chores like running spreadsheets and word processors. I/O and memory are the general system resources that, along with the processor comprise the core of the system. Over time I/O and memory support functions have moved from discrete logic into integrated chip sets. More recently fixed function accelerators have been added to systems, mostly in response to the unique

requirements of graphical user interfaces. The new, real-time part of the system model is enabled by commodity, low latency operating systems like Windows* 95 and Windows NT*. Until now, platforms have not really had to contend with the difficulties of dealing with real-time data streaming from multiple sources to different destinations.

Also as you can see from the diagram above, in our view, the real-time element of the system has not been adequately addressed in the past or even in current generations of systems. Digital signal processors have served part of the job by integrating some fixed functions along with some amount of reprogrammability to do things like adaptive filters for telephony. This role is relatively specialized but will continue for point solutions needing the highest performance where cost is less of a concern. An inexpensive, general purpose, real-time processing capability was unavailable until now.

With the introduction of MMX™ technology, Intel has provided a key element to complete the balanced system picture for dealing with real-time data processing (see the picture below). The single instruction, multiple data (SIMD) instructions added to all future generations of Intel architecture microprocessors help to streamline data processing in multimedia systems; reducing or eliminating the need for dedicated processors for many applications. For more information on MMX technology, see our web site at www.intel.com.

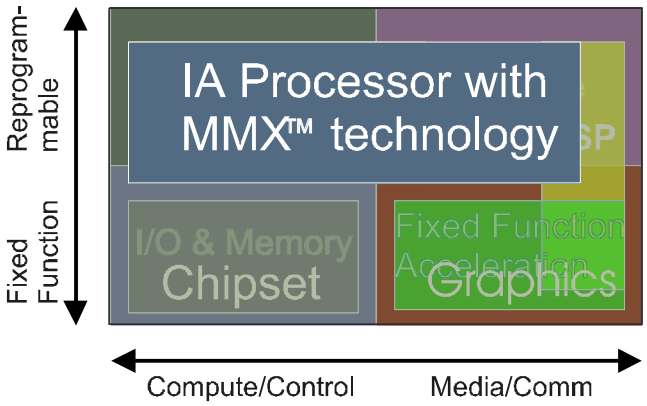

Figure 3. MMX Technology adds Media Processing

The main design challenge for system designers is to ensure that the additional processing power that MMX technology offers can be used effectively. This is done by adding performance to the system in a balanced way, i.e. the right supporting system elements in each of the quadrants of the model. Now let's look at how to create a balanced system for MPEG-2 video playback.

MPEG-2 Playback System Operation

To know how to improve system performance and spend the system budget wisely, a more detailed understanding of MPEG-2 soft playback is helpful. The first step is to look at what is happening in the system.

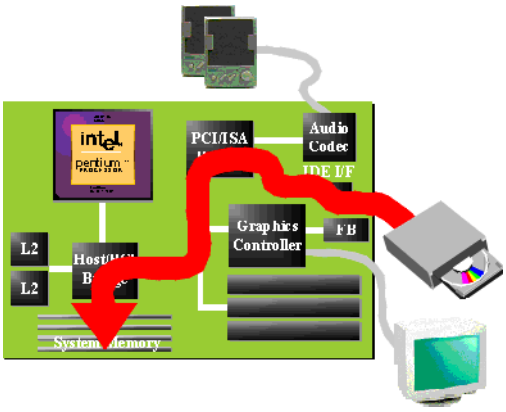

Figure 4. Data Coming in from a DVD-ROM

Data streams off of a source such as a DVD drive at a data rate of 4.7 Megabits per second and is written into the system memory array by a bus mastering IDE controller. Bus mastering is required to eliminate the burden from the CPU from doing a simple control function like doing data movement. It then has more bandwidth to do more valuable system functions like soft decode.

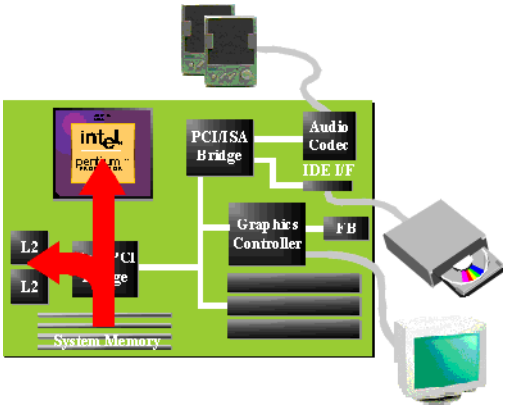

Figure 5. Processor Decodes Data in Memory

The processor begins to parse the data stream, dividing the work between video and audio compression. As data is read from main memory, the cache controller saves a copy to speedup later accesses.

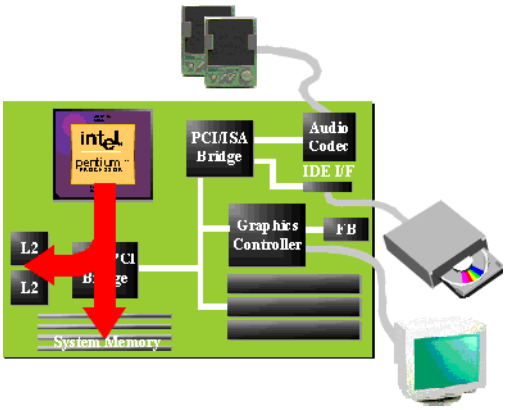

Figure 6. Decoded Data is Written to Memory

The processor begins the decompression process which, for video, includes algorithms such as inverse Huffman, inverse Discrete Cosine Transforms, and other computations. These are highly integer intensive algorithms, well suited for processing on a Pentium® processor. The set of information that is needed to be repeatedly accessed by these algorithms, in this case for decoding a frame of video, is called the "working set". For MPEG-2, as we will show later, the working set is large and quite complex.

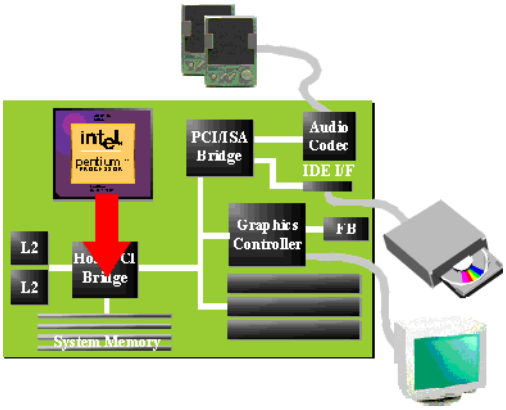

Figure 7. Write to Frame Buffer is Posted in Chip Set

After a video frame is decoded, it is written out to the frame buffer, but there is an important intermediate step. To improve the transfer efficiency on the PCI bus, the chip set buffers writes and then efficiently bursts them across the PCI bus. How this works and its impact upon system performance is rather interesting and something we examined in detail.

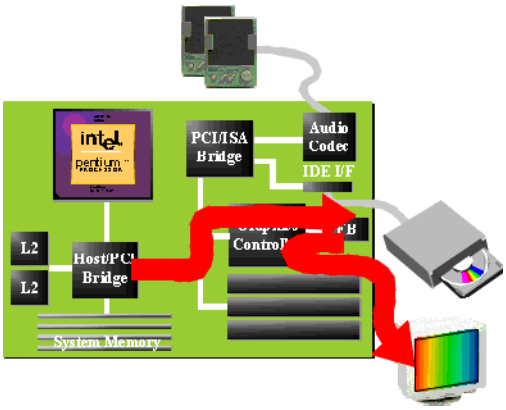

Figure 8. Chip Set Bursts Video Data over PCI to Frame Buffer

The chip set then bursts packed pixel information into the frame buffer. Concurrently, the graphics controller is reading video memory to refresh the screen. This concurrent operation limits the available frame buffer bandwidth significantly which is another area for potential performance improvement as we see later.

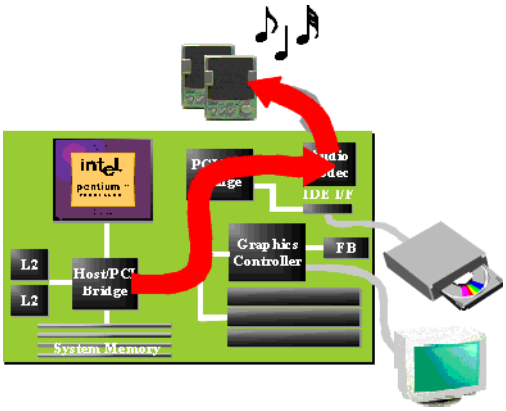

Figure 9. Audio Data Written to PCI/ISA Bridge

Interleaved with the video frame buffer writes, are writes to the audio CODEC to replay the audio portion of the program.

There are a couple of key things to note from the above description. All transfers go through the chip set. It handles reads and writes to and from main memory, L2 cache, and the frame buffer and audio CODEC on the PCI bus. Secondly, the large MPEG-2 working set causes a significant "thrashing" of the level 2 cache. Thrashing is caused by a significant number of level 2 cache misses; the result is increased bus utilization "wasted" in an attempt to keep the L2 cache full.

Understanding System Stalls

If you think of the MPEG-2 connection from the CPU doing soft decode to the frame buffer as one continuous pipe with YUY2 data flowing through it, then a stoppage at any point along the way will halt everything behind it. If the system (hardware and software architecture) is designed such that the processor is stopped by stalls at the frame buffer then soft decode of MPEG-2 (or any other high data rate real-time application) cannot work. The frame buffer has a much more limited bandwidth than the processor which means the processor can easily overwhelm it. The frame buffer also becomes periodically busy as the graphics controller does a screen refresh, causing the data in the pipe to halt.

In order to be as flexible as possible and capable of connecting to a broad number of device types, PC's are intentionally not designed as a single, continuous data pipe. Chip sets and peripherals have evolved to include buffering to break the single pipe into segments so that each can start and stop without necessarily affecting others before it. Buffering in each segment allows the previous pipe to accumulate some data until the next segment in the pipe frees up. PCI chip sets work this way. Processor write buffering in the chip set enablesbursting data onto the PCI bus and reduces the effects of stalls at the destination from affecting the processor.

Bursting also has the benefit of increasing the transfer rate across the bus. The processor writes data into the chipset as fast as it is instructed to do, even if the result is a series of single-write transfers with contiguous addresses. If the addresses are indeed contiguous, the chip set is able to group together all the contiguous writes and burst the data in a single burst transfer. The signal overhead for a burst transaction remains constant, so the greater the burst length, measured in bytes, the higher the transfer rate (see figure 10).

One can use the target data rate to determine what the ideal transfer rate is, and hence what the desired burst length is. For example, the required display data rate for an MPEG-2 video stream in CCIR601 format is 15.2 MB/s. If the goal is to have this load consume no more than 15% of all bus activity, then the data transfer rate should be at least 100 MB/s. This (from the given assumptions in the chart below) indicates that bursts should be at least 20 DWORDS or greater in length. Note, however, that buffer overflow at the target (e.g., a PCI graphics controller) can cause the bus to stall, which in turn can cause the processor to halt. Longer burst length is not advantageous in all situations. This will be discussed in greater detail later in the paper.

Taking advantage of the burst capabilities of the PCI bus and PCI chip set is critical for applications involving large amounts of data, like decoding MPEG-2 video.

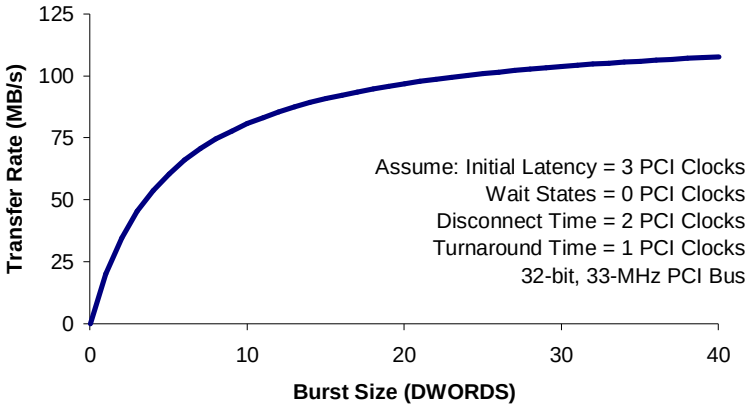

Figure 10. PCI Transfer Rate as a Function of Burst Size

Still, because the data rates of each segment of the pipe are not equal, stalls will occur when the bus is heavily loaded. The key to making the most efficient use of the environment is to understand the stalls, and determine if anything can be done about them. Changing the speed of one or more of the pipes might provide the answer. This is likely to add cost, but, if done correctly it will also benefit overall system performance, which is easier to justify. Another solution might be as simple as optimizing the length and the frequency of the bursts, much like managing automobile traffic in a city. If you avoid gridlock, much higher volumes of traffic can be handled. These optimizations can either be hardware or software but generally only require tuning of the software algorithms which makes them an inexpensive way to gain significant performance.

Part II. MPEG-2 Analysis

First Look: Soft MPEG-2 Playback

The exact requirements to do MPEG-2 software playback are not yet well understood since MPEG-2 soft CODECs aren't available in the market today; work is underway but it won't be until later in 1996 that products appear. OEMs, however, are having to design their 1997 systems now; most have MPEG-2 on their list of requirements. The decision of whether to include dedicated hardware or be able to count on quality software-only playback is critical to how systems get partitioned. In the absence of having soft MPEG-2 to analyze, we studied soft MPEG-1 running on existing platforms and extrapolated to derive our conclusions.

Interest in MPEG-1 playback on PCs grew over most of 1995. At the time it was generally accepted that hardware acceleration was needed to achieve the maximum frame rate (30 frames per second for NTSC source material). The common belief still is that good software MPEG-1 playback is not feasible on anything less than a 133 MHz Pentium processor. We performed a study on a well designed system and were able to show that quality soft MPEG-1 playback at 30 fps was indeed possible using a Pentium processor 100 MHz. For more detail see our whitepaper "Designing a Low Cost, High Performance Platform for MPEG-1 Video Playback" on Intel's website (www.intel.com).

Our Analysis

We used a number of analysis tools to examine bus-level system performance for software MPEG-1 players. With these tools we have estimated the number of CPU core clocks and amount of CPU bus traffic required to run MPEG-2 in software. The variations with player and chipset are relatively small and do not detract from the central findings. The data presented here is for an Intel 82430VX-based system with 16MB SDRAM and 256KB pipelined burst L2 cache. The frame size for the data is 352x240 at 30 frames per second (fps).

Two different MPEG-1 decoders are represented in the data. A scalar version was run on a Pentium processor 166 Mhz platform (PP-166). An early version of a decoder using MMX technology was run on a prototype P55C processor running at 166 MHz. Whether or not the processor is stalled or executing is measured by using the Pentium processor performance monitoring registers. By measuring the unrestricted[1] frame rate of a particular decoder, the core clocks/frame is easily calculated.

In order to predict MPEG-2 performance based on MPEG-1 profile data, a method of scaling from MPEG-1 to MPEG-2 is necessary. The simplest, most obvious way to do this is to translate the increased resolution and data rate of MPEG-2 into an MPEG-1 frame rate. Since the algorithms and data structures remain similar while the frame resolution is increased by a factor of 4x, a rough approximation of MPEG-2 performance can be calculated by predicting the MPEG-1 performance at 120 fps (30 fps x 4).

MPEG-1 Playback *CPU Performance Profile*	**(units)**	**PP-166 MHz**	**P55C-166 MHz**
Core Speed	MHz	166	166
Core Stalls	% time	53	62
Core Executing	% time	47	39
Unrestricted Frame Rate	fps	61.7	83.5
Core Clocks per Frame	MHz/frame	1.26	0.77
Core Clocks at 120 fps	MHz	152	92

Table 2. Processor Utilization Profile for MPEG-1 Playback

Table 2 summarizes the frame rate and core utilization for unrestricted MPEG-1 playback without audio. Using core clocks/frame as a scale factor to MPEG-2 frame sizes, it is estimated that 92 Mhz of a P55C-166 are required to play MPEG-2 video. For the processor, then, the projection is that MPEG-2 video decoding on the host processor consumes ~1/2 of a Pentium processor 200 MHz with MMX technology.

The next area to examine is the host bus of the system. The host bus controls all data movement throughout the system, whether it be directly to L2/DRAM memory, the PCI bus for high-speed functions, or the ISA bus for peripheral add-in cards. Since the processor is involved, at one level or another, in all data transactions, the question to look at now is whether or not a Pentium processor 200 Mhz with MMX technology, which is projected to be 50% consumed with computational decoding activity, is capable of performing all the data movement tasks with the other 50%. Note: this assumes that the processor halts anytime there is an outstanding bus cycle. This is in fact untrue; in some situations, the processor can continue executing out of register memory and L1 cache, even while waiting for a bus cycle to return other data. Analysis of the workload in this manner thus becomes a “worst-case” scenario.

[1] unrestricted frame rate means the decoder is allowed to run “flat out;” decoding and displaying frames as fast as the system will allow. Audio playback and synchronization is disabled.

For the bus utilization analysis, several assumptions are made concerning scaling from MPEG-1 to MPEG-2. First, that all bus traffic will scale by a factor of four. This will be true for the volume of the frame buffer traffic. To map from volume of traffic to time on the bus requires that the access characteristic of the frame buffer remain constant. In fact, they may deteriorate with the volume of data, but this is clearly an area that must be improved. The volume of read and write data are likely to scale by the same factor, and will scale as a measure of bus utilization if the memory/L2 characteristics remain constant. This requires that the L2 hit rate remain constant. The code read volume is small, and has been assumed to scale from MPEG-1 to MPEG-2.

To examine the bus, a scaling factor is measured similar to the one above in Table 2. For the bus, however, the important factor is total bus utilization. This is measured against frame rate to produce a "Bus Utilization per Frame" for a given platform.

MPEG-1 Playback *Bus Utilization Profile*	**units**	**PP-166**	**PP-166 w/ MMX Technology**
Code Reads	% time	8.4	3.4
Data Reads	% time	25.3	29.0
Data Writes	% time	14.6	9.3
Frame Buffer Writes	% time	15.5	19.7
Total Bus Util	% time	63.8	61.4
Unrestriced Frame Rate	fps	61.7	83.5
Bus Utilization per Frame	% time	1.03	0.74
Bus Utilization at 120 fps	% time	124	88

Table 3. Bus Utilization for MPEG-1 Playback

As shown in Table 3, the bus utilization per frame is 0.74% for MPEG-1. For MPEG-2, (MPEG-1 at 120 fps), the bus utilization is predicted at 88%, clearly indicating a saturated condition. In order to implement software-based MPEG-2 playback on today's volume PC platforms, then, work must clearly be done to lower the bus utilization required for MPEG-2 decoding. This can be done by increasing the transfer rate performance of the various memory subsystems, and optimizing decoders for fast accesses. Identifying and investigating methods for such improvements will be the focus of the remainder of this document.

A Closer Look: Identifying the System Bottlenecks

A closer look at the system activity will help identify areas for increasing efficiency or other optimizations. First lets take a look at what the MPEG-2 decoder activity consists of. The chart below shows a breakdown of the different types of host bus cycles that occur in the decoding of MPEG-2 video.

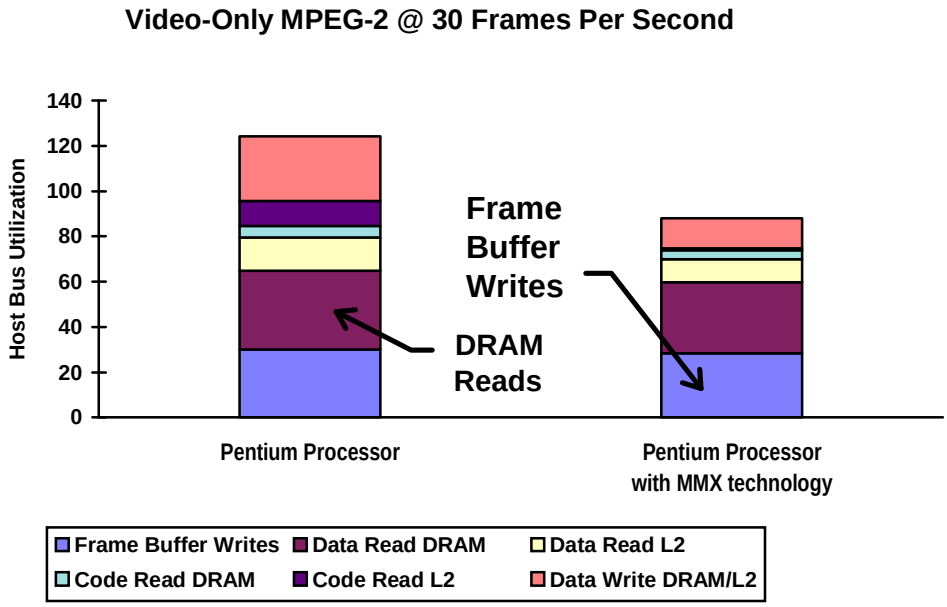

Figure 11. MPEG-2 Decoder Activity at the Bus Level

As you can see, two bus cycle types stand out as consuming the largest portion of bus cycle time. Writes into frame buffer memory consume the most, taking up nearly 30% of the host bus cycles, while reads into main memory take up another 30%. Of the remaining activities, none consumes more than ~10% alone. Appropriately, then, these two bus activities will become the focus for the remainder of this paper. We will discuss at length possible optimizations, in both hardware and software, as well as point out future technologies and architectural enhancements which will address the needs of applications incorporating MPEG-2 video.

Part III. System Bottleneck Identification and Analysis

A first look at software-based MPEG-2 decoding concluded with two major system bottlenecks: memory read performance (or memory read bus utilization) and frame buffer write performance (or frame buffer write bus utilization). These activities have been shown to take up a very large proportion of the overall bus traffic; they will thus become the first target area for improvement. The remainder of this paper will focus on some of our initial efforts at reducing the amount of bus utilization required for each of these activies. Further research is continuing, and in Q2'96, Intel will publish the conclusions in the form of a DVD/MPEG-2 Platform Design Guide.

Memory Read Performance

To understand the memory read activity, it is important to first understand the MPEG decoding process in general. With MPEG-1, data is read in from the CD-ROM at a rate of about 300 KB/sec (most MPEG-1 video clips are compressed to a range between 1.5 Mbps - 1.8Mbps). If one second of data carries 30 video frames, then each frame consists, on average, of roughly 10 KB.

From this one frame of data, the processor will decode the pixel data. MPEG-1 and MPEG-2 video uses the planar YUV12 format, so the processor will build three planes of pixel data; one Y plane (the *luminance* plane), and a U and V plane (the *chrominance* planes). For planar YUV12 (or YUV4:2:0), the chrominance planes are sub-sampled 2:1 in both the horizontal and vertical directions; the luminance

plane is not sub-sampled. The U and V planes, then, are each ¼ the size of the Y plane. Once the three planes, or buffers, are constructed, the data is written to an offscreen surface in frame buffer memory. There, the pixels are combined to form a single frame of video data.

To calculate the size of the array used to store a single plane of data, we can work backwards from the size of the decompressed final image. A single frame of MPEG-1 video contains 352 x 240 = 84,480 pixels. At 12 bpp, a single frame requires 126,720 Bytes, or 127 KB. At a ratio of 4:1:1 for the Y, U, and V planes, the buffers are roughly 85 KB, 21 KB, and 21 KB respectively.

For MPEG-2 the two changes are in the input buffer from CD-ROM and the increased resolution. For DVD/Video MPEG-2, the data rate will average ~4.7Mbps. This is roughly 600 KB/s from the DVD-ROM; divided into frames, this yields an input buffer of about 20 KB. The larger CCIR601 resolution of 704 x 480 also increases the working set size, giving the following working set size approximations:

Buffer Size for Decoding a Single Frame	**MPEG-1 (KB)**	**MPEG-2 (KB)**
Input Buffer	10	20
Y Plane	85	340
U Plane	21	84
V Plane	21	84
Total	137	528

Table 4: MPEG-1 and MPEG-2 Working Set Size Comparison

As you can see, constructing an MPEG-1 frame requires (at the barest minimum) about 137 KB of memory. This size, or portions thereof, would fit fairly comfortably within a 256 KB L2 cache. This is ideal, because access times for a 256 KB PBSRAM L2 Cache memory are twice as fast as EDO DRAM Memory at 66 MHz (see Table 6).

Notice that the problem becomes much more acute when decoding MPEG-2. The buffer size estimates above are nearly 300% larger for MPEG-2 than MPEG-1.

The above exercise really only applies to *index* frames, or I-frames, where all the data required to decode a particular frame is within the compressed frame. MPEG compression also utilizes *predicted* frames (P-frames), and *bi-directional* frames (B-frames). For P-frames, some of the information required to decode the frame has been compressed in the bit stream in a previous I-frame. This would require storing that previous I-frame until the P-frame can be completed. For B-frames, two other frames—an I-frame and a P-frame—must be accessed to complete the frame. If we modify the above table for frame type, then, we see the following:

Buffer Sizes for Different Frame Types	**MPEG-1 (KB)**	**MPEG-2 (KB)**	**% of Total Frames***
I frame	137	528	10
P frame	274	1056	30
B frame	411	1584	60

* Percent of total frames calculated using a "typical" series: I-B-B-P-B-B-P-B-B-P

Table 5: MPEG-1 and MPEG-2 Working Set Size Comparison

In reality, MPEG video decoders operate on much smaller data types, called *macroblocks*, but the calculations above serve to illustrate the decoding complexity and read-intensive nature of an MPEG decoder. Operating on macroblocks will be more closely examined later in this paper. The above exercise serves to illustrate the critical need for an L2 cache for platforms capable of performing MPEG-2

decoding. The Pentium Pro Processor and processors in the P6 family will have improved L1 and L2 cache bandwidth, which will benefit MPEG-2 performance considerably. See the table below comparing a platform based on the Pentium Processor with MMX technology to a P6 Family processor with MMX technology:

Read Bandwidth (MB/s)	**Pentium Processor with MMX Technology 200 MHz Platform**	**P6 Family Processor with MMX Technology 233 MHz[2] Platform**	**% Increase**
L1 Cache (Internal)	1150	1370	19
L2 Cache (PB SRAM)	460	760	65
DRAM Memory (EDO)	235	235	--

Table 6. Read Bandwidth Improvements on Future Platforms

Our MPEG-1 application profile data showed an L2 cache hit rate of only ~45% on memory reads, projected to a Pentium Processor with MMX technology at 200 MHz. Hence, in addition to requiring a fast L2 cache, MPEG-2 decoders need to optimize for fast data reads—from L2 cache and DRAM. One way to optimize for fast data reads is to do a working set analysis like we did previously (using the appropriate variables and data sizes) and optimize for high hit rates in the L2 cache. Another is to include a "prefetch" mechanism in appropriate sections of code to maximize DRAM performance. The latter will be looked at here in more detail.

Optimizing for DRAM Read Performance

A critical factor for DRAM memory read performance is the ability to pipeline DRAM accesses. For example, for a 16 MB EDO DRAM array and an Intel 430FX PCIset, DRAM read performance is limited to 160 MB/s when back-to-back accesses cannot be pipelined. This happens during row-miss accesses, and the DRAM read pattern is limited to 10-2-2-2-10-2-2-2-10-2-2-2. With EDO page-hit timings, successive reads can be pipelined and a 10-2-2-2-3-2-2-2-3-2-2-2 pattern can be achieved. In this case, the read bandwidth increases from 160 MB/s to 235 MB/s (see Table 6).

Bandwidth	**PP-200 w/ MMX Technology L1 (MB/s)**	**256 KB PB SRAM L2 (MB/s)**	**EDO DRAM @66 MHz (MB/s)**
Read	1150	460	235* 160**
Write (QWORD)	595	175	175

* pipelined page hit timings
** row miss timings

Table 7: Relative Bandwidth of L1, L2, and DRAM Memory

Page-hit accesses can only be maintained if the processor can initiate a new read cycle before the DRAM interface has completed the previous read. For small code loops that perform successive reads, a prefetch mechanism can be employed that will ensure that successive reads can be pipelined for maximum performance. The prefetch algorithm is designed to insure that nearly all memory reads for tightly coded loops will result in L2 cache hits. Although such a mechanism increases the total number of instructions executed, the improved average memory read speed can result in overall code speedup. The result, for EDO DRAM, is a potential increase in read bandwidth from 160 MB/s to 235 MB/s, an improvement of nearly 50%.

[2]performance estimates of future products are subject to change

To see how such a prefetch mechanism might be implemented, look at the sample code segment in Figure 12. The following algorithm interleaves the values from two byte streams into a single bytes stream.

```
Setup       mov       esi, Source1          ;pointer to first input array
            mov       ebx, Source2          ;pointer to second input array
            mov       ecx, SrcBufferSize    ;length of the output array
            shr       ecx, 3                ;number of quadwords in SrcBufferSize

MainLoop    movq      mm0, [esi]            ;load first operand into mm0 (8 bytes)
            movq      mm1, [ebx]            ;load second value into mm1 (8 bytes)
            movq      mm2, mm1              ;duplicate mm1 into mm2
            punpklbw  mm1, mm0              ;interleave low  bytes into mm1
            add       esi, 8                ;increment first array pointer by 8 bytes
            punpkhbw  mm2, mm0              ;interleave high bytes into mm2
            add       ebx, 8                ;increment second array pointer by 8 bytes
            movq      [edi], mm1            ;store mm1 into SrcBuffer
            movq      [edi+8], mm2          ;store mm2 into SrcBuffer
            add       edi, 16               ;increment edi by 16 bytes for next result(s)
            dec       ecx                   ;decrement loop counter
            jnz       MainLoop              ;repeat MainLoop until complete
```

Figure 12. An example code loop

This code loop serves as an example of one which might be optimized for fast memory accesses by implementing a prefetch mechanism in software. The cache line size in the Pentium Processor architecture is 32 Bytes. A packed register in the MMX architecture consists of 8 bytes, so a cache line can hold 4 packed registers. The prefetch algorithm above, will walk down the length of the operand arrays, one at a time, and read each cache line into L2 cache. No operations will be performed; instead, the first value in each cache line will simply be "touched" to insure that that particular cache line is pulled into L2 memory. See below how we've added the twoprefetch loops:

```
Setup1      mov       esi, Source1          ;pointer to first input array
            mov       ebx, Source2          ;pointer to second input array
            mov       ecx, SrcBufferSize    ;length of the array
            shr       ecx, 5                ;number of cache lines (32 bytes) in SrcBuffer

FetchSrc1   mov       eax, [esi]            ;"touch" first value in Source1
            add       esi, 0x20             ;inc Source1 pointer 32 Bytes (1 cache line)
            dec       ecx                   ;decrement loop counter
            jnz       FetchSrc1             ;continue until Source1 is "prefetched"

Reset       mov       ecx, SrcBufferSize    ;Reset the length of the array for Source2
            shr       ecx, 5                ; number of cache lines (32 bytes) in SrcBuffer

FetchSrc2   mov       edx, [ebx]            ;"touch" first value in Source2
            add       ebx, 0x20             ;inc Source2 pointer 32 Bytes (1 cache line)
            dec       ecx                   ;decrement loop counter
            jnz       FetchSrc2             ;continue until Source2 is "prefetched"

Setup2      mov       esi, Source1          ;pointer reset to first input array
            mov       ebx, Source2          ;pointer to second input array
            mov       ecx, SrcBufferSize    ;length of the output array
            shr       ecx, 3                ;number of quadwords in SrcBufferSize
```

```
MainLoop    movq        mm0, [esi]      ;load first operand into mm0 (8 bytes)
            movq        mm1, [ebx]      ;load second value into mm1 (8 bytes)
            movq        mm2, mm1        ;duplicate mm1 into mm2
            punpklbw    mm1, mm0        ;interleave low bytes  into mm1
            add         esi, 8          ;increment first array pointer by 8 bytes
            punpkhbw    mm2, mm0        ;interleave high bytes  into mm2
            add         ebx, 8          ;increment second array pointer by 8 bytes
            movq        [edi], mm1      ;store mm1 into SrcBuffer
            movq        [edi+8], mm2    ;store mm2 into SrcBuffer
            add         edi, 16         ;increment edi by 16 bytes for next result(s)
            dec         ecx             ;decrement loop counter
            jnz         MainLoop        ;repeat MainLoop until complete
```

Figure 13. Same Code as Figure 12, with an Initial Prefetch Stage

Adding the prefetch algorithms increases the number of instructions executed by 17%; however the overall number of core clocks required to complete the loop is decreased as a result of the improved average memory access times. With all the input values read into L2 cache, the main algorithm achieves page hit accesses on nearly all memory reads. The results are shown in Table 7 below.

	Instructions Executed	Core Clocks*	Relative Performance
Normal Code	6004	23503	1.0
with Prefetch	7010	19647	1.16

* Input Array Size = 4000

Table 8. Performance Improvement with Prefetch Stage Added

Thus we can see that, although adding to the total number of instructions that must be executed, a prefetch algorithm implemented to fill the L2 cache, can yield significant gains in overall code throughput. Knowing when to use a prefetch mechanism like the one above involves knowing whether or not the data exists in L2 or main memory; if there is a high probability the data is already in L2, then adding a prefetch mechanism will yield little or no improvement. Profiling the decoder with an emphasis on data location is necessary to succesfully apply techniques like the one above. These types of enhancements will continue to yield performance improvements with better algorithm development, compiler technology, and faster memory technologies.

With the coming of new memory technologies like Synchronous DRAM (SDRAM), pipelined page hit accesses can achieve 2-1-1-1 timings over today's EDO page-hit 3-2-2-2 accesses. This could provide up to a 2x performance improvement for code loops containing successive reads that are pipelined in the fashion illustrated above. Further MPEG-2 data analysis will provide a better estimate of how much this mechanism will improve overall MPEG-2 playback performance.

Future Processor/PCIset Architecture Enhancements

In the future, architectural enhancements to the Intel Architecture Platform will further enhance the ability to decode digital video of all types, including MPEG-2. Future processor and platform architectural enhancements will continue to improve the available bandwidth and efficiency of memory transfers.

Concurrent PCI, for example, a feature found in Intel's 430VX and 430HX PCIsets, improves PCI bus utilization significantly. The addition of features such as the Multi Transaction Timer (MTT) for "short burst" applications, Passive Release for interleaving PCI/ISA transfers, and improved write performance, all will help benefit MPEG-2 video decoding.

Dynamic Execution is a new processor architecture technology found in P6 family processors. Dynamic Execution is the combination of several advanced processing techniques: multiple branch prediction, data flow analysis, and speculative execution. These technologies allow P6 family processors to more intelligently process instruction. Multiple branch prediction allows the processor to "look ahead" down multiple execution paths. Data flow analysis allows the processor to identify and react to data dependencies in the execution path. Speculative execution allows the processor to execute instructions prior to the actual execution path being determined. The combination of these techniques makes P6 family processors particularly well suited to run multimedia applications, where highly repetitive operations are performed on large data sets (like a video stream).

Frame Buffer Write Analysis

Writes to the frame buffer consume 30% of the processor time it takes to do MPEG-2 movie playback and constitute the second major bottleneck that will be examined in this paper. This section looks closely at issues associated with frame buffer writes and outlines some of the possible areas for optimization and enhancement.

To display MPEG-2 video, a minimum bandwidth of ~15 MB/s on the PCI bus to the graphics controller is required. This figure comes from a simple calculation of the MPEG-2 format, using full CCIR601 resolution, YUV4:2:0 format, and full 30 frames per second display:

$$\frac{704 \times 480\,\text{pixels}}{\text{frame}} \times \frac{12\,\text{bits}}{\text{pixel}} \times \frac{1\,\text{byte}}{8\,\text{bits}} \times \frac{30\,\text{frames}}{\text{second}} = \frac{15{,}206{,}400\,\text{bytes}}{\text{second}} \cong 15.2\,\text{MB/sec}$$

Equation 1. MPEG-2 (YUV12) Display Transfer Rate

Many graphics controllers today support video pixel formats in frame buffer memory. Windows 95 with DirectDraw allows video data to be decoded into its native format and written directly into a dedicated block of frame buffer memory, called an *offscreen surface*. The graphics controller, while reading the frame buffer for screen displays, is able to read and properly mix the data between the primary surface, containing RGB pixels, and offscreen surfaces that may contain YUV pixels (or other video formats or RGB). The result is a seamless video display within a windowed graphics environment.

Many different video formats exist. MPEG-2 Main Profile Main Level (MP@ML) uses the Planar YUV12 format, or YUV4:2:0. The diagram below illustrates the YUV4:2:0 macroblock format for a MPEG-2 block of 16 pixels:

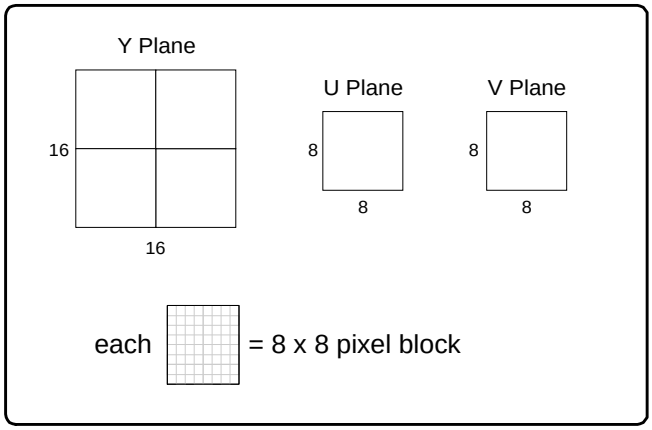

Figure 14. YUV 4:2:0 Macroblock Structure

In this format, the U and V planes (chrominance, or color) are subsampled 2:1 both horizontally and vertically. The Y plane (luminance, or lightness) is not subsampled. Hence, 384 Bytes are required for each 256 pixels. The result is a 12 bit per pixel (bpp) format.

YUV4:2:0 is called a planar format because the data is stored separately for each of the three planes Y, U, and V. The optimal MPEG-2 decoding architecture requires that the graphics controller and decoding software use the planar YUV12 format, to insure that no additional bus traffic or processor MIPS are wasted converting the data between different video formats. For example, many video-enabled graphics controllers today support byte-ordered ("packed") video formats like YUY2. Byte ordering is an efficient way to store video data in the frame buffer, because the planar data is interleaved, as shown in the following diagram. The DAC can easily read byte-ordered pixel data and perform the pixel reconstruction by storing the U and V values temporarily in a buffer.

BYTE3	BYTE2	BYTE1	BYTE0
Y1	U1	Y2	V1
Y3	U2	Y4	V2

Pixel 1 = Y1 + U1 + V1
Pixel 2 = Y2 + U1 + V1
Pixel 3 = Y3 + U2 + V2
Pixel 4 = Y4 + U2 + V2

Figure 15. YUY2 Byte-Ordered Format

Note from the diagram above, however, that YUY2 requires 4 Bytes for every 2 pixels, or 16 bpp. To decode MPEG-2 into Planar YUV12 format and then convert to YUY2 format does not require much in the way of MIPS to do the actual conversion. Much of it can be done "on the fly," when writing the data to the frame buffer. However, by converting from a 12 bpp format to a 16 bpp format, the amount of data transmitted over the PCI bus to the graphics controller is increased by 33%. The minimum frame buffer display transfer rate now becomes:

$$\frac{704 \times 480\,\text{pixels}}{\text{frame}} \times \frac{16\,\text{bits}}{\text{pixel}} \times \frac{1\,\text{byte}}{8\,\text{bits}} \times \frac{30\,\text{frames}}{\text{second}} = \frac{20{,}275{,}200\,\text{bytes}}{\text{second}} \cong 20.3\,\text{MB / sec}$$

Equation 2. YUY2 (16bpp) Display Transfer Rate

Storing the MPEG-2 data in a 16 bpp format offers no increase in quality, since the compressed video contains only YUV12 data. The "extra" pixels required for converting to 16 bpp are simply replicated from the YUV12 data. At a penalty of 33% additional data, with no increase in video quality, the value of supporting planar YUV12 in frame buffer memory is clear.

Graphics Controller Write Buffer Depth

Another enhancement possible for graphics controllers suitable for MPEG-2 can be made by providing deeper write posting buffers on the PCI interface. Typical graphics controllers with video support today provide write posting buffers capable of storing 4 DWORDS (16 Bytes). In the planar YUV12 format, 16 Bytes represents less than 11 pixels (ignoring the fact that the data is planar, and not interleaved).

Optimizing for MPEG-2 by increasing the depth of write posting buffers helps decrease the overall system bus utilization and avoid processor stalls. The question is, what is the desirable buffer depth? Dedicated registers are expensive in terms of silicon; adding buffers can directly impact the cost of the device.

As we discussed before, writes to the frame buffer are first posted in the core logic chip set and then written over the PCI bus. Hence, the depth of the chip sets write posting buffers will also have an impact on the efficiency of frame buffer writes.

Optimizing for Frame Buffer Writes

MPEG-2 decoders must optimize for frame buffer writes, or added buffer depth will not help reduce bus utilization. An analysis of the bus activity profiles showed that most software MPEG-1 decoders function in a highly serial fashion. That is, the software tends to decode a frame, write the data to the frame buffer, and repeat the process. In this scenario, the write posting buffers of both the graphics controller and the PCI bus quickly overflow. The resulting bus stalls then cause the processor to stall, and decode work is interrupted.

To avoid the problem of overflowing write buffers during frame buffer writes, it is desirable for the decoder to be architected in such a way that this activity is highly parallel. Specifically, the decoder should attempt to optimize for low bus utilization for frame buffer writes. If the frame buffer writes are periodic and of manageable size, the overall bus utilization required for the frame buffer write traffic is minimized. To illustrate this, the following data was collected by varying the size of frame buffer writes and the interval between writes and measuring the instantaneous transfer rate at the PCI interface for a given graphics controller.

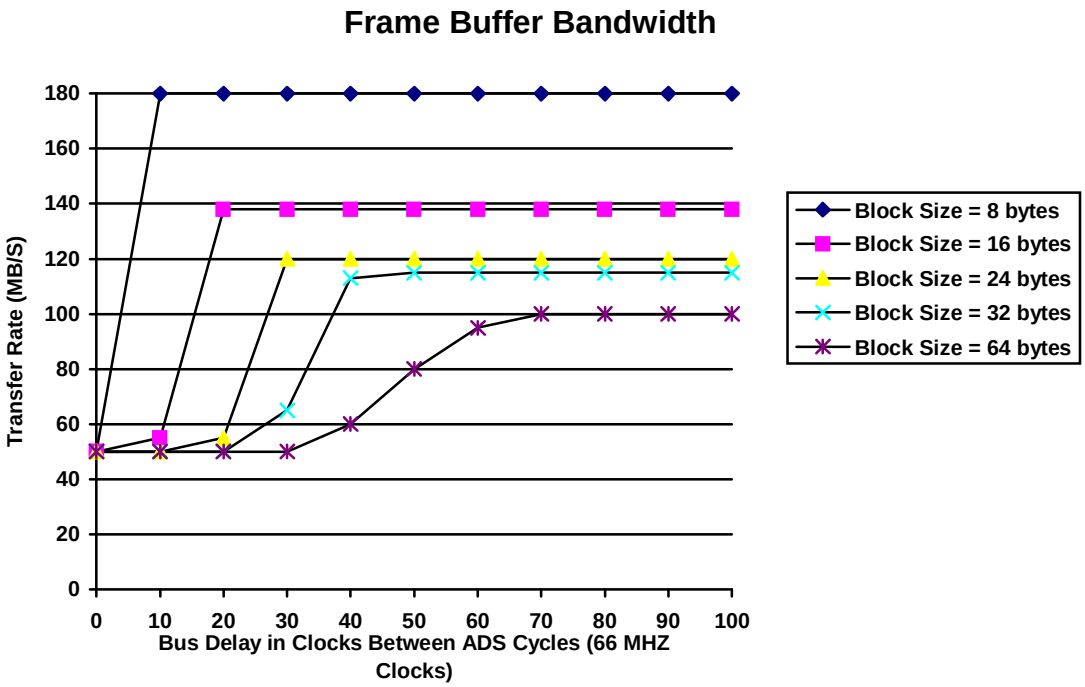

Figure 15. Frame Buffer Write Transfer Rate

Each of the curves above represent the write transfer rate at the host for a given block size. As the delay time between frame buffer writes increases, the write transfer rate increases. This is directly related to the ability of the graphics controller to accept writes from its write posting buffers and write them out to the frame buffer. Eventually, the curves flatten, regardless of the spacing between writes. The "knee" where the transfer rate flattens indicates where the write posting buffers in the PCI chip set overflow, causing a bottleneck higher up the data pipe. At this point, write performance is dictated by the depth of the write posting buffers in the chip set.

The different curves were then obtained by varying the block size of the writes. As seen above, smaller block sizes achieve higher instantaneous transfer rate (i.e., more efficient bandwidth transfer). This result is not exactly intuitive, but the optimal frame buffer write traffic for MPEG decoding is achieved by maintaining a constant, steady stream of writes with the block size optimized so as not to overflow the write posting buffers in the graphics controller. By optimizing in this manner, our calculations for an MPEG-2 data rate show that for a 15 MB/s display bandwidth requirement, the bus utilization can be reduced from 30% of the bus activity to 15% or lower.

Other Graphics Controller Enhancements

In addition to the hardware and software optimizations illustrated above, there are many other potential areas for improving the graphics controller for MPEG-2 playback. The PCI interface of many graphics controllers today is in the range of 50 MB/s. For MPEG-2, 100 MB/s is ideal, when targeting 15% bus utilization for 15 MB/s display bandwidth. The use of faster memory technologies for frame buffer memory will further improve bus transfer rates and overall MPEG-2 performance. Like EDO DRAM, SDRAM offers significant performance improvements at little or no price premium. Memory technologies specifically targeted for video and graphics, such as SGRAM, may offer even faster performance gains.

Another potential enhancement to the graphics controller for video acceleration is PCI bus mastering. PCI Bus mastering graphics controllers may further improve bus utilization on frame buffer writes by maximizing efficiency on transfers between main memory and frame buffer memory.

Faster bus interfaces for the graphics device could also result in a significant performance boost. For more information on the recently announced Accelerated Graphics Port standard see the following web site:

www.teleport.com/~agfxport

In the coming MPEG-2 System Design Guide, we will publish data which will further qualify and quantify the performance benefits of these and other enhancements for decoding MPEG-2 video.

MMX Technology Accelerates MPEG-2

As mentioned before, the SIMD instruction set added to Intel Architecture processors with MMX technology is well suited to assist in the acceleration of MPEG-2 video decoding.

If we take a closer look at the activities involved in decoding MPEG video, we notice that from an algorithm standpoint, the process consists of three major functions:

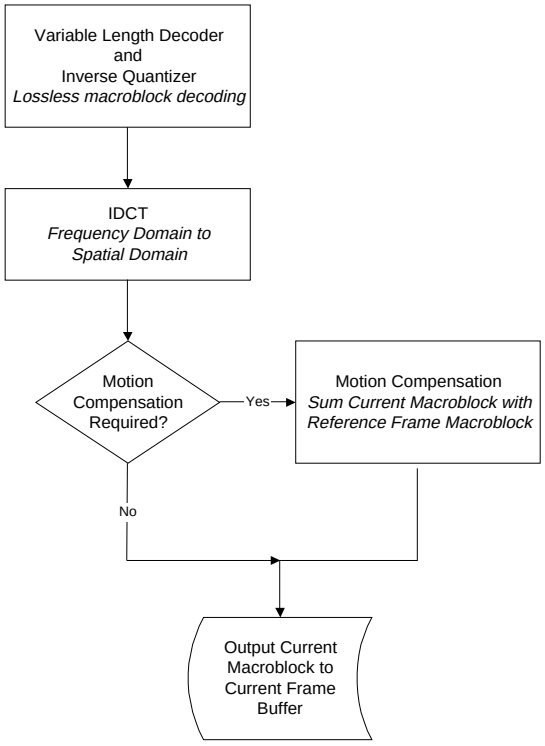

Figure 17. Functional Block Diagram of an MPEG Decoder

The following table shows the relative computational load for each of the above tasks for a software-based decoder using an MPEG-1 video stream at 1.15 Mbps:

Decoder Function*	Load
Bit Stream Header Parsing	--
VLC Decoding and Inverse Quantization	24%
Inverse DCT	28%
Motion Compensation	48%

* Note that Color Space Conversion in Hardware is assumed

Table 9. Computational Load of a Software-Based MPEG Decoder[1]

Bit Stream parsing consumes less than 1% and is therefore ignored. If we next look at what is required to accelerate the above algorithms, we notice that all are good candidates for optimization with MMX technology. More specifically, we can profile each of the algorithms into its basic operation(s) and speculate how the MMX Instruction set might be applied.

The raw image decompression algorithm(s), Huffman Decoding and Inverse Quantization, are primarily a parsing exercise. Fast compares can accelerate this stage, as can byte unpacking operations. The Inverse Discrete Cosine Transform, or IDCT, is the heart of the MPEG video decoder. It translates the frequency domain data of a particular macroblock back into the spatial (YUV format) domain. Since the IDCT is performed by a series of matrix multiplications (summations), the Multiply-Accumulate operations capable with MMX instructions may speed up an IDCT as much as 3.5x[2] over a similar decoder not utilizing MMX instructions.

Next, we look at the motion compensation portion of MPEG video decoding, Motion compensation is the process of decoding a macroblock based on motion vectors pointing to past or future *reference* macroblocks. The reference macroblock(s) are used to reconstruct the current macroblock by summing their pixel values. Maximum compression is achieved in this way since only the error, or difference vector, between the current macroblock and the reference macroblock(s) is coded. If the motion estimation in the encoder is good, then the difference vector produces all zeros for the macroblock DCT coefficients and the resulting average frame compression increases.

Recall here our earlier discussion, in the section on DRAM read performance optimization, on the working set size for MPEG-2 decoding. If we now look at the macroblock as a set size, we have a greater appreciation for the MPEG-2 data requirements. An MPEG-2 macroblock consists of 384 Bytes representing 256 pixels. At a per frame resolution of 720 x 480 pixels, a single frame contains 345,600 pixels, or 1,350 macroblocks. For raw data alone, this consumes ~518 KB of pixel data, twice as large as the standard 256 KB L2 cache. Because P- and B- macroblocks make reference to one or two I- or P-macroblocks, we can see that the memory requirements required to assemble a frame can be quite complex, especially for P- or B- frames.

Fortunately, MMX instructions lend themselves particularly well to motion compensation, which, like an IDCT, is a “Multiply-Accumulate” type of algorithm. When averaging or summing pixels, the packed registers used in MMX technology can be used to accelerate these functions[3] [4] .

Conclusions

In summary, from the exploration of system bottlenecks above and the discussion of MMX technology acceleration, our platform architecture recommendations for MPEG-2 video playback are:

Software Enhancements

- Accelerate MPEG-2 Decode with MMX Technology
- Interleave Decoding and Frame Buffer Write activities
  - Goal is an even, optimized frame buffer write stream
  - Don't overflow write posting buffers
- Optimize for Fast Data Reads
  - Improve L2 Hit Rates
  - Prefetch for Sequential Access (improve average DRAM read performance)

Hardware Enhancements

- CPU/PCIset
  - Platform MUST have an L2 Cache
  - Future chip sets should support deeper write buffers
  - SDRAM may help—performance vs. EDO being investigated
  - Future Technologies:
    - PCI Concurrency
    - Dynamic Execution in P6 Family processors
- Graphics Controller/Frame Buffer
  - Frame Buffer Support for Planar YUV12
  - Deep Write Posting Buffers (2 Cache Lines or better)
  - Target 100+ MB/s frame buffer Bandwidth @ PCI Interface
    - SDRAM, SGRAM may be necessary
  - Future Technologies:
    - PCI Bus Mastering
    - Accelerated Graphics Port

Balanced System Model Verification

As a final system cost/performance check, if we map the system requirements against the "balanced system model", as described earlier in the paper, we get the following:

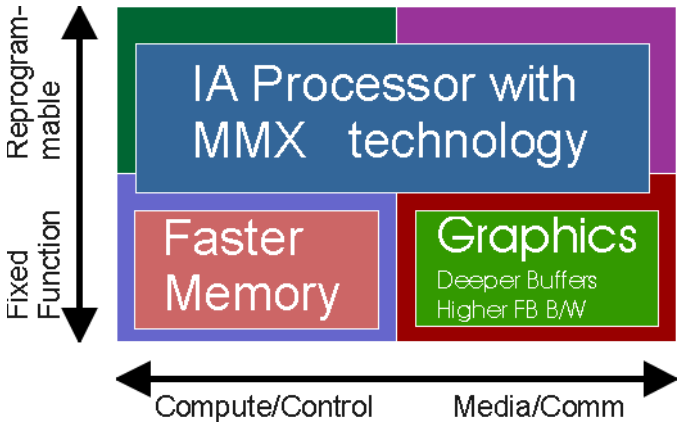

Figure 18. Balanced System for MPEG-2 Video Playback

If the model is correct in its assumptions, the suggested performance elements required to implement MPEG-2 movie playback, by adding appropriately to each quadrant, should also result in overall system performance improvement. Whether these features are the best and final recommendations that we make remains to be seen. More lab work is necessary before we get to the final results but it is likely that much of what we have learned thus far will directly apply to an eventual solution. Intel will publish this as a DVD/MPEG-2 Design Guide on the web, http://www.intel.com/.

Other References

A good, short MPEG backgrounder is available on the World Wide Web at:

http://www.optivision.com/compress/textonly/compress/technica/wpaps3.html

A more comprehensive MPEG FAQ is available on the World Wide Web at:

http://www.cs.tu~berlin.de/~phade/mpegfaq/mpe8912.html

For more information about MPEG, contact the OpenMPEG Consortium WWW page at:

http://www.openmpeg.org/

---

[1] "Image and Video Compression Standards: Algorithms and Architectures." Bhaskaran, Vasudev, and Konstantinides, Konstantinos. Kluwer Academic Publishers, London. 1995. Pp.181

[2] AP-528, "Using MMX Instructions in a Fast IDCT Algorithm for MPEG Decoding." Intel Corp., March 1996.

[3] AP-529, "Using MMX Instructions to Implement Optimized Motion Compensation for MPEG1 Video Playback." Intel Corp., March 1996.

[4] AP-527, "Using MMX Instructions to Get Bits from a Data Stream." Intel Corp., March 1996.