WHITE PAPER

AMD Alchemy™ Au1200™ Processor Media Acceleration Engine

by: Erik Schlanger

Advanced Micro Devices
9500 Arboretum Blvd, Suite 400
Austin, TX 78759

Abstract

The Media Acceleration Engine (MAE) is an on-chip hardware device that shoulders the work of video decoding, scaling, color space conversion, and filtering in the AMD Alchemy™ Au1200™ processor. By performing the most demanding video decoding tasks in hardware, the MAE frees the MIPS32™ core to handle user interface functions, audio processing, and other tasks. Conversely, because software running in the core passes variable length decoding information to the MAE, the MAE is endowed with the flexibility needed to accommodate multiple video codecs.

MAE processes are divided between front end and back end hardware. Each segment is designed to handle specific types of tasks.

- The MAE front end autonomously performs the most complex and intense video decoding tasks, including inverse quantization, inverse discrete cosine transformations, motion compensation, and Windows® Media Video 9 overlay smoothing.
- The MAE back end performs one-pass horizontal and vertical scaling, programmable color space processing, and filtering functions. When video decoding is not taking place, these back end functions are available for use by other data sources such as processing CMOS/CCD or NTSC/PAL data received from the on-chip Camera Interface Module.

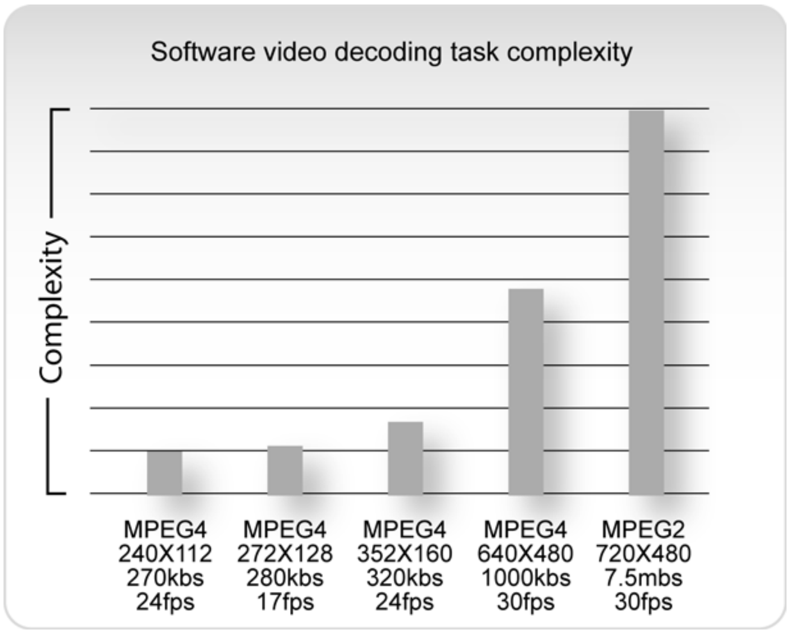
Introduction

The future of multimedia will be heavily influenced by its portability. The advent and subsequent popularity of portable media players initiated discussions among AMD Alchemy™ engineers about what seemed inevitable: the evolution of Personal Media Player (PMP) devices that reproduce high-quality full-size video.

Their initial considerations about enabling high-performance video capabilities in a system-on-a-chip centered on the demands of the video decoding process – typically performed in software – that converts video content from MPEG1, MPEG2, MPEG4, DIVX 3,4,5 data to RGB display data.

This intense decoding process taxes even powerful CPUs, often leaving little headroom available for other tasks. It became evident that low cost, low-power/high-quality core processors of interest for video processing would be unable to handle the work of full-size full-frame-rate video decoding in software. The processing demands for high-quality MPEG/WMV video content outstripped CPU capabilities long before anything resembling acceptable results were achieved.

Figure 1. Complexity of Software Video Decoding Demands on Processors



It became clear that the shortfall between workload and processor capability would best be addressed by separating the work of video decoding from other processing loads.

MAE Design Goals

For information about the PMP use model, see the System Architecture white paper included in this suite.

Because the engineering team wished to avoid an expensive and power-hungry multi-chip solution, they focused their design efforts on a hardware-based media accelerator that would operate within the architecture of a low-power/high-performance system-on-a-chip fine-tuned to the needs of PMP developers.

The MAE was forged under the influence of two essential and opposing factors: *keep bandwidth high* and *keep physical area low*. Careful adherence to these two factors resulted in a versatile hybrid software/hardware SOC design that consistently meets performance requirements, yet contains no “fat” that increases developers’ BOM costs. An additional and happy result of this hybrid design is that developers will not have to write or purchase video decoding software.

Because direct access to the vast library of existing content – particularly MPEG 2 – is essential to the appeal of the PMP use model, AMD Alchemy™ engineers decided that direct video decoding capability must be maintained for established formats. This goal has been fundamental to the project’s success, because the variety of existing video formats represents a set of standards by which the design team could measure the success of their design.

The Media Acceleration Engine fulfills its role as the hardware component in the Au1200 processor that handles the most complex portions of the video decoding process, including inverse quantization, inverse DCT, motion compensation, scaling, color space conversion, and filtering. The MAE was built specifically to decode MPEG (1,2,4), Div-X (3,4,5), WMV9, and H.263 video data. The MAE enables the Au1200™ processor to deliver full D1 video rendering at full frame-rates on portable devices without the need for transcoding.

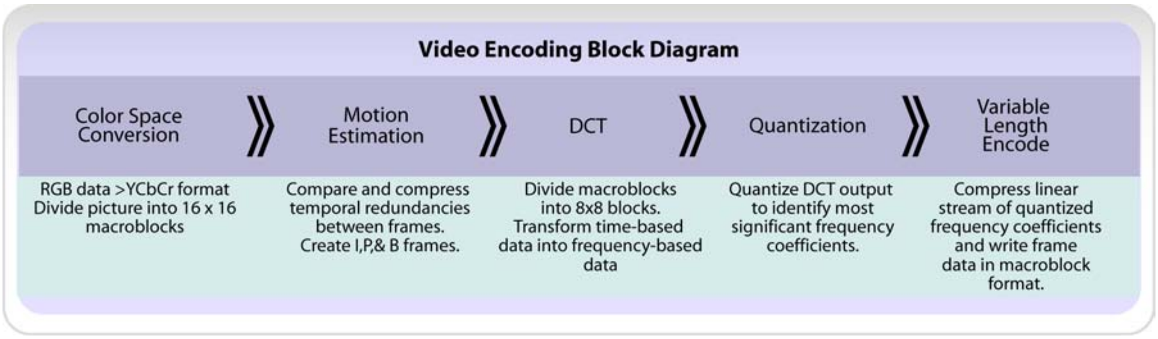
Overview of Video Compression & MAE Support

For ease of presentation this material is based on the video encoding *process. The MAE performs a reversal -* decoding *- of this process.*

An understanding of the functions and features of the Media Acceleration Engine depends upon some familiarity with video compression. Although a comprehensive recounting of video compression technologies is beyond the scope of this paper, the standards these technologies support are integral to the design and performance goals of the MAE. This overview illustrates how the design of the MAE supports each phase of block-based video compression technologies.

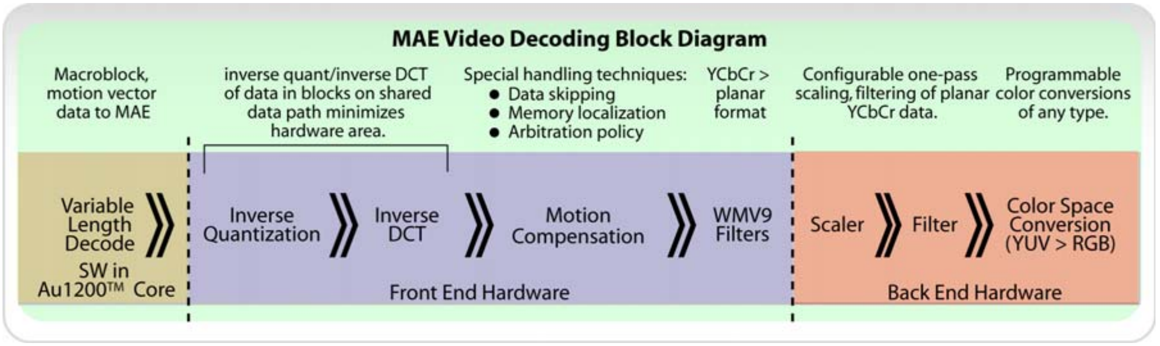
Video encoding relies heavily on data compression techniques that enable convenient digital storage of the massive amounts of information that characterizes video data. Figure 2 represents the basic processing steps that video data undergoes when it is encoded in a block-based format, such as MPEG 2.

Figure 2. Video Encoding Process Overview


Video Encoding Block Diagram
Color Space Conversion
Motion Estimation
DCT
Quantization
Variable Length Encode
RGB data >YCbCr format Divide picture into 16 x 16 macroblocks
Compare and compress temporal redundancies between frames. Create I,P,& B frames.
Divide macroblocks into 8x8 blocks. Transform time-based data into frequency-based data
Quantize DCT output to identify most significant frequency coefficients.
Compress linear stream of quantized frequency coefficients and write frame data in macroblock format.

Video *decoding* tasks on the Au1200™ processor are divided between software-based and hardware-based processing that fully supports all decoding tasks, as shown in Figure 3.

Figure 3. MAE Video Decoding Overview

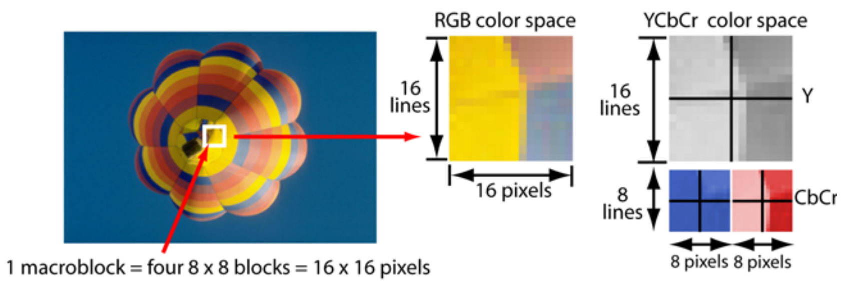

MAE Video Decoding Block Diagram
Macroblock, motion vector data to MAE
inverse quant/inverse DCT of data in blocks on shared data path minimizes hardware area.
Special handling techniques: ● Data skipping ● Memory localization ● Arbitration policy
YCbCr > planar format
Configurable one-pass scaling, filtering of planar YCbCr data.
Programmable color conversions of any type.
Variable Length Decode
SW in Au1200™ Core
Inverse Quantization
Inverse DCT
Motion Compensation
WMV9 Filters
Front End Hardware
Scaler
Filter
Color Space Conversion (YUV > RGB)
Back End Hardware

How Video Data is Packaged for Encoding

Figure 4 illustrates that two sizes of data “chunks” support different phases of block-based video encoding processes:

- ***Macroblocks*** are the standard chunk size by which a full frame of video data is processed when it is converted into YCbCr color space, and by which variable length encoding is carried out. Macroblocks comprise 16 lines x 16 pixel blocks.
- ***Blocks*** comprise 8 pixel by 8 line sets of Y, Cb, or Cr values within a macroblock. Figure 4 illustrates that there are four Y blocks, one Cb block, and one Cr block per macroblock. Macroblocks can also be formatted to contain two Cb and two Cr blocks. Block-sized data is used in motion estimation, the DCT, quantization, and zig-zag processing of run/amp pairs.

Figure 4. Macroblock & 8 x 8 Data Block Overview



MAE Decoding Support for Image Data

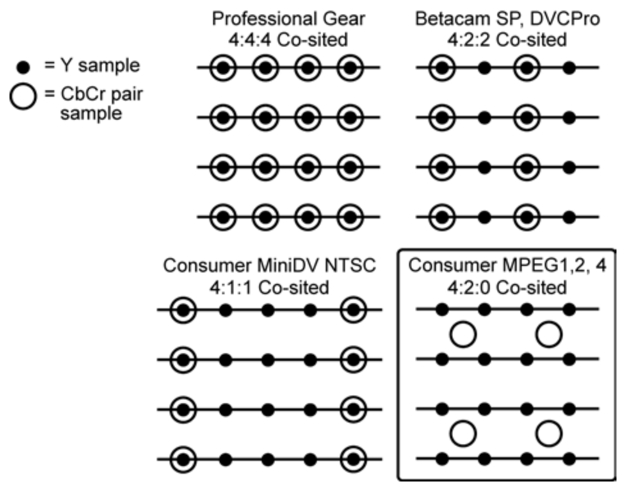
The MAE was built specifically to handle the decoding of video data for block-based formats. The Au1200™ processor maintains the ability to handle *multiple* block-based formats by performing variable length decoding in software, then passing macroblock data to the MAE along with motion vector (and other) data. From that point, the MAE autonomously processes image data in both macroblock and 8 x 8 block sizes in hardware that is specifically designed to handle each decoding step.

Color Space and Video Compression

The RGB (Red/Green/Blue) color space is the model upon which colors in computer displays are based, and is the model that is commonly used in the output of many CCD/CMOS sensors and video cameras. These three component RGB values combine interdependently to describe a color’s intensity, brightness, and hue. Even minor changes in RGB component values are easily discerned. Because of this, RGB color space does not lend itself well to video data compression techniques.

However, RGB values can be converted to YCbCr color space values in which the Y value represents luminance (brightness), and the CbCr value represents chrominance (color). YCbCr color space is popular in image/video processing because its characteristics lend themselves to data compression. Because our visual perception is less sensitive to color (chrominance) information than it is to brightness, it is possible to delete some chrominance (CbCr) data and retain acceptable image quality. This deletion of chrominance data is called *subsampling*. When data is actually lost (discarded) to achieve compression, then the scheme is termed “lossy”. Figure 5 illustrates that YCbCr formats vary in the amount of chrominance information that is deleted (lossiness) to achieve data compression. Various interpolation algorithms (filters) can help preserve the quality of images rendered from compressed YCbCr color space data.

Figure 5. YCbCr Color Space Subsampling Schemes


● = Y sample
○ = CbCr pair sample
Professional Gear 4:4:4 Co-sited
Betacam SP, DVCPro 4:2:2 Co-sited
Consumer MiniDV NTSC 4:1:1 Co-sited
Consumer MPEG1,2, 4 4:2:0 Co-sited

MAE Decoding Support for Color Space Conversion

Because the Au1200™ processor supports the most popular standard video formats, the MAE processes input formats of YCbCr 4:2:2 and YCbCr 4:2:0, and outputs aRGB8888, RGB888, RGB565, and RGB555 formats – although any color space conversion can be performed (for example, RGB>YCbCr). Refer to page 16 for more detailed information about color space conversion and the special features of the MAE “back end”.

Motion Estimation

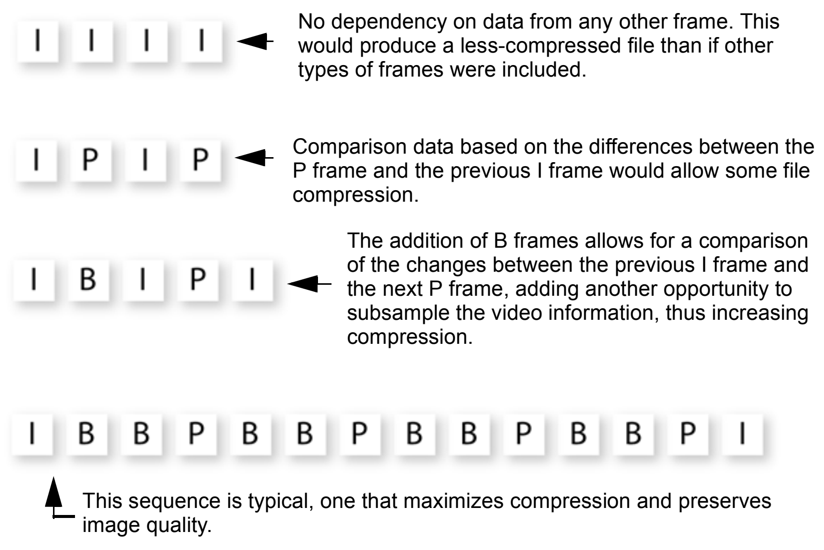
After conversion to YCbCr color space, groups of video frames are evaluated to determine the rate of change in data between frames. Motion estimation processes video data in block size (8 pixels x 8 lines). More information about this follows on page 9 but *generally*:

- Groups of frames having little change between frames offer more opportunity for data compression. Most of these frames are further compressed by temporal compression techniques.
- Groups of frames having great amounts of change between frames offer fewer opportunities for data compression. Some of these frames become “reference” frames that receive no further compression.

Frames within each group are typically classified in this way:

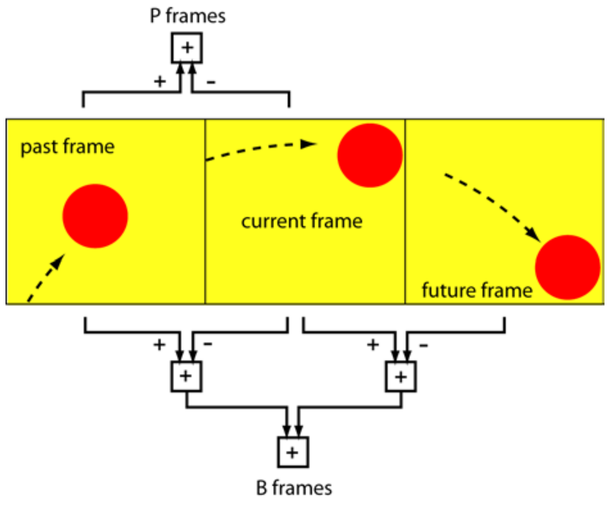
- I frames - these frames are not compressed, and are coded as stand-alone still images that become references for other frames.
- P&B frames - the data in these frames is further compressed by motion estimation algorithms applied to the similarities / differences between frames, as shown in Figure 6.
  - P frames are compared to the previous I frame.
  - B frames are compared to the previous I frame and the next P frame.

Figure 6. Example Sequences of Video Frame Encoding


I I I I
No dependency on data from any other frame. This would produce a less-compressed file than if other types of frames were included.
I P I P
Comparison data based on the differences between the P frame and the previous I frame would allow some file compression.
I B I P I
The addition of B frames allows for a comparison of the changes between the previous I frame and the next P frame, adding another opportunity to subsample the video information, thus increasing compression.
I B B P B B P B B P B B P I
This sequence is typical, one that maximizes compression and preserves image quality.

In P and B frames, as shown in Figure 7, the apparent movement of a red ball across a static yellow background represents a small change from frame to frame. *Motion estimation* allows that instead of writing full data (up to 900 kbytes) for each frame, only the data that describes the position of the ball needs to be written. This can be represented with two small vector values (x/y).

Figure 7. P and B Frame Comparisons for Motion Estimation

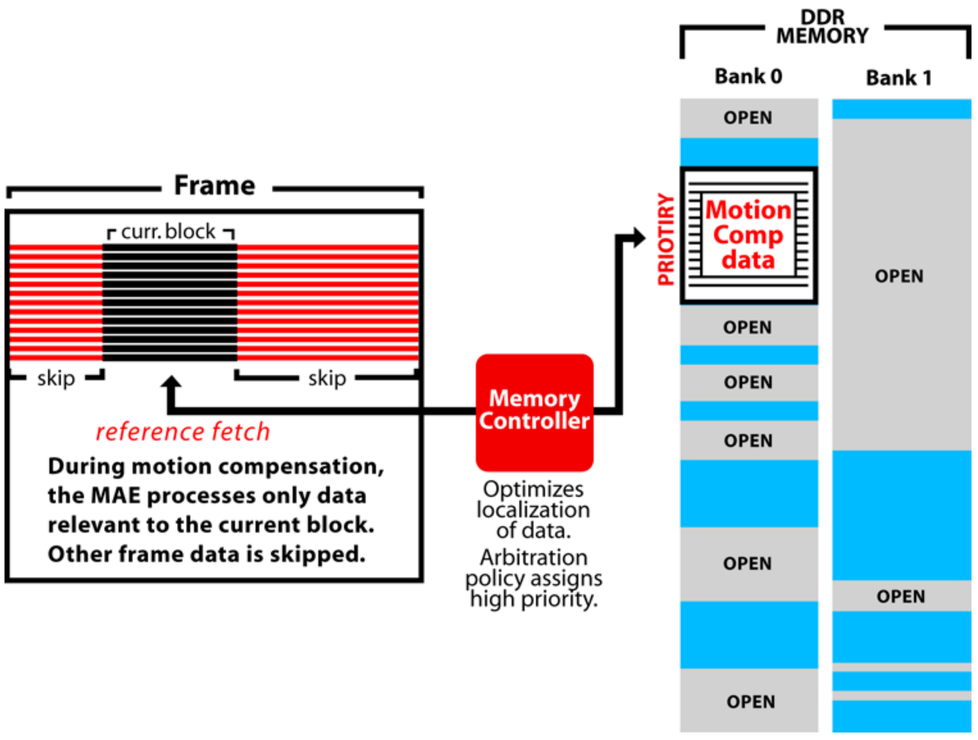


MAE Support for Motion Compensation

The Media Acceleration Engine provides a multiplicity of configurable filters that enable support for all MPEG and WMV9-based motion compensation processes having resolutions of 1, 1/2, or 1/4 pel.

AMD Alchemy™ engineers designed special techniques that optimize data handling for motion compensation. These techniques can and do work independently, but they often work together, having a synergistic effect on system performance. For example, when reference fetches in I or P frames occur, the MAE skips over sections of the reference frame that are outside the boundaries of the fetched data block – as shown in Figure 8. Consequently, less data is handled, and the system architecture's "open bank" policy ensures that fetched data is localized in memory in the most efficient form possible. After reference frame data is fetched, its significance to system operation immediately becomes critical. Therefore, an arbitration scheme developed over many hours of performance modeling assigns high priority to reference fetch data.

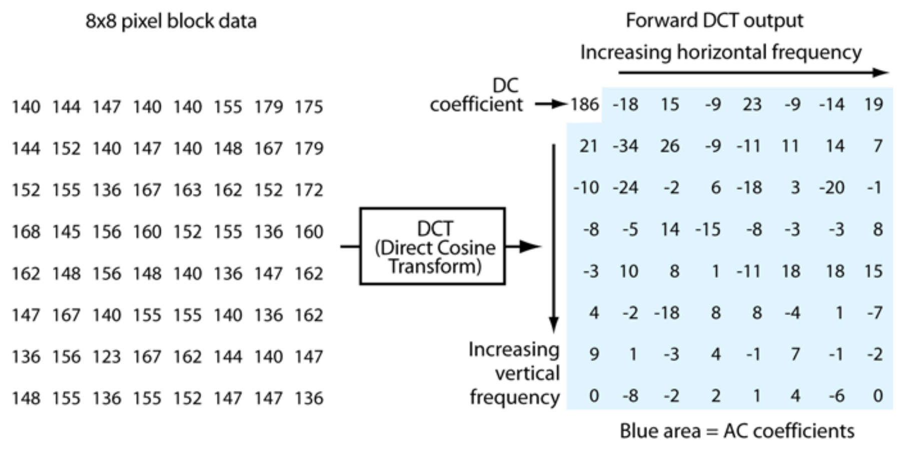
Figure 8. Data Skipping, Memory Localization, and Arbitration Policy


DDR MEMORY
Bank 0
Bank 1
OPEN
Motion Comp data
PRIOTIRY
Frame
curr. block
skip
skip
reference fetch
During motion compensation, the MAE processes only data relevant to the current block. Other frame data is skipped.
Memory Controller
Optimizes localization of data. Arbitration policy assigns high priority.

DCT (Discrete Cosine Transform) and Quantization

The DCT is applied to the values in each 8 x 8 block of pixel data. It converts time-based data to frequency-based data. The DCT in itself is not a compression scheme, but the 8 x 8 block of frequency coefficients it yields can be used to further compress video data in later steps.

Figure 9. The DCT Produces Frequency/Amplitude Data for Each Block

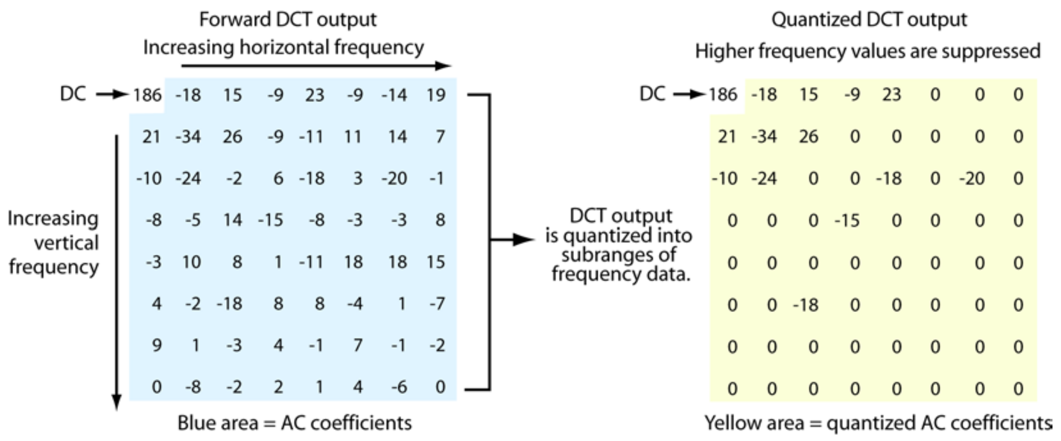

8x8 pixel block data
Forward DCT output
Increasing horizontal frequency
DC coefficient
DCT (Direct Cosine Transform)
Increasing vertical frequency
Blue area = AC coefficients

140	144	147	140	140	155	179	175
144	152	140	147	140	148	167	179
152	155	136	167	163	162	152	172
168	145	156	160	152	155	136	160
162	148	156	148	140	136	147	162
147	167	140	155	155	140	136	162
136	156	123	167	162	144	140	147
148	155	136	155	152	147	147	136

186	-18	15	-9	23	-9	-14	19
21	-34	26	-9	-11	11	14	7
-10	-24	-2	6	-18	3	-20	-1
-8	-5	14	-15	-8	-3	-3	8
-3	10	8	1	-11	18	18	15
4	-2	-18	8	8	-4	1	-7
9	1	-3	4	-1	7	-1	-2
0	-8	-2	2	1	4	-6	0

Quantization divides the values of the DCT output into discrete (non-overlapping) subranges (coefficients) of frequency data. A video frame's low-frequency data is typically critical to perceived video image quality. The process of quantization filters out less-critical image data by driving as many high-frequency DCT coefficient values to 0 (zero) as is possible. Generally, as a pixel's frequency value rises, the quantization process applies greater mathematical leverage to drive it to zero. Quantization is performed on all DCT data except the DC coefficient.

Figure 10. Quantization Divides DCT Output into Discrete Coefficients


Forward DCT output
Increasing horizontal frequency
DC
Increasing vertical frequency
Blue area = AC coefficients
DCT output is quantized into subranges of frequency data.
Quantized DCT output
Higher frequency values are suppressed
DC
Yellow area = quantized AC coefficients

186	-18	15	-9	23	-9	-14	19
21	-34	26	-9	-11	11	14	7
-10	-24	-2	6	-18	3	-20	-1
-8	-5	14	-15	-8	-3	-3	8
-3	10	8	1	-11	18	18	15
4	-2	-18	8	8	-4	1	-7
9	1	-3	4	-1	7	-1	-2
0	-8	-2	2	1	4	-6	0

186	-18	15	-9	23	0	0	0
21	-34	26	0	0	0	0	0
-10	-24	0	0	-18	0	-20	0
0	0	0	-15	0	0	0	0
0	0	0	0	0	0	0	0
0	0	-18	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Many quantization schemes exist, each designed to achieve different results. The quantized block of frequency coefficients is used as the basis for the final step in compression: *variable length encoding*.

MAE Design Support for Inverse Quantization and Inverse DCT

Inverse quantization, inverse DCT, and motion compensation are off-loaded from the core and are instead performed in MAE hardware that is optimized to handle the required workloads. Because the processing of inverse quantization and inverse DCT are similar and interdependent, a shared MAE data path was designed specifically for these processes. This significantly reduced on-chip space requirements; thereby holding BOM cost down. Both iQ and iDCT are supported by a configurable set of registers and muxed math modules that process inverse DCT and inverse quantization for MPEG 1, 2, & 4, and DivX 3, 4, & 5; and additionally inverse quantization (8x8, 8x4, 4x8, and 4x4) for WMV9.

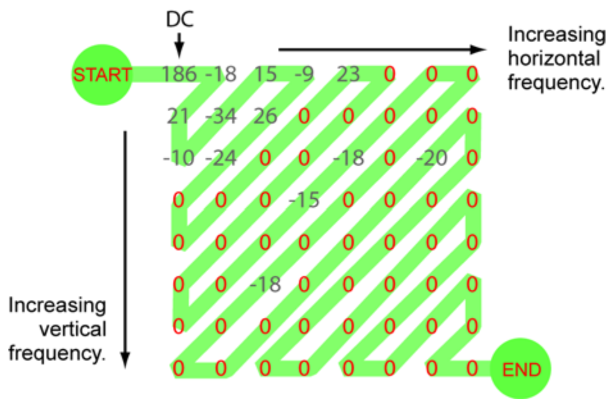
Output from iQ and iDCT operations is in 8 x 8 blocks of planar YCbCr data processed column-wise, in preparation for MAE back end processing. For more information about the MAE back end, see page 14.

Variable Length Encoding

Variable length encoding begins by scanning the quantized DCT output in a zig-zag pattern, as Figure 11 illustrates. The zig zag scan produces a linear stream of quantized coefficients arranged in order of increasing frequency. Note, as an example, that in Figure 11 the zig zag stream contains “runs” of consecutive zero-value coefficients. These are converted into “run/amp pair” values that describe the length of each run and the amplitude of the frequency that ended each run.

Run/amp pairs that occur frequently are identified with short codes. Run/amp pairs that occur infrequently are identified with longer codes. Thus, this variable length encoding achieves further data compression, because shorter codes are more frequently used to identify the greater amount of data.

Figure 11. Zig Zag Scanning



Au1200™ Processor Support for Variable Length Decoding

AMD Alchemy engineers observed that the variable length decoding (VLD) process is different for each of the popular standard formats, and that the nature of VLD is different from and less complex than other parts of the video decoding process.

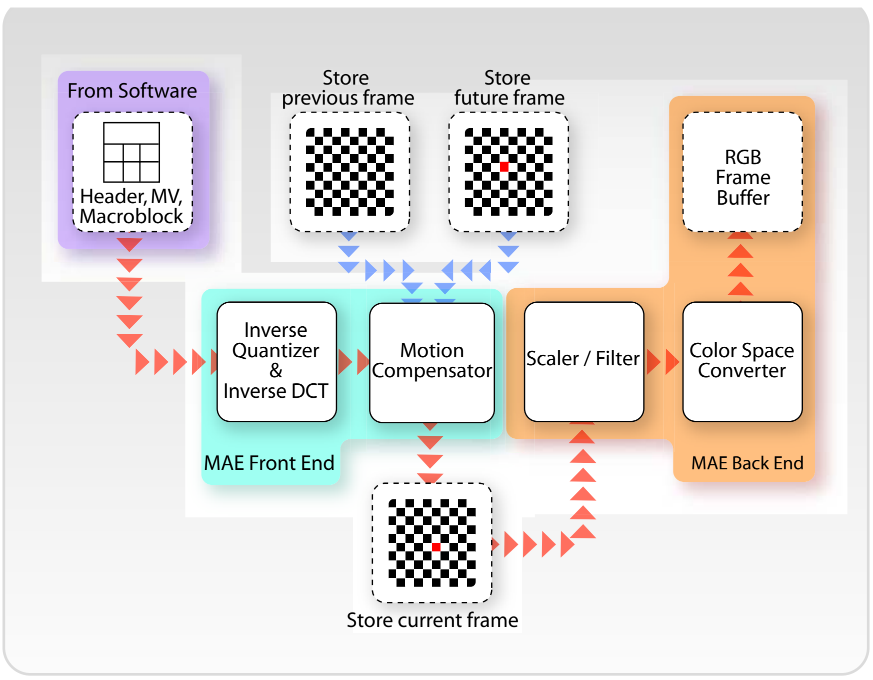
Capitalizing on these characteristic similarities and dissimilarities, they divided video decoding processes into software-based and hardware-based tasks. In the Au1200 processor, software-based VLD ensures the versatility that is necessary to handle the variety of VLD implementations among different video formats. Also, because it is by nature less complex than other uncompression processes, VLD can be accomplished with little load on the core.

The MAE Back End

MAE hardware is segmented into a *front end* and *back end*. Each performs groups of functions that are separately accessible:

- The front end hardware is specifically dedicated to video decoding: the quantization, inverse DCT, and motion compensation processes discussed in the preceding pages.
- The back end hardware is a unique environment that performs autonomous hardware scaling, filtering, and color space conversion. When video decoding is not taking place, the MAE can provide these functions to other sources of image data such as still pictures, or to NTSC/PAL image data received from the on-chip Camera Interface Module.

Figure 12. Data Flow for Media Acceleration Engine



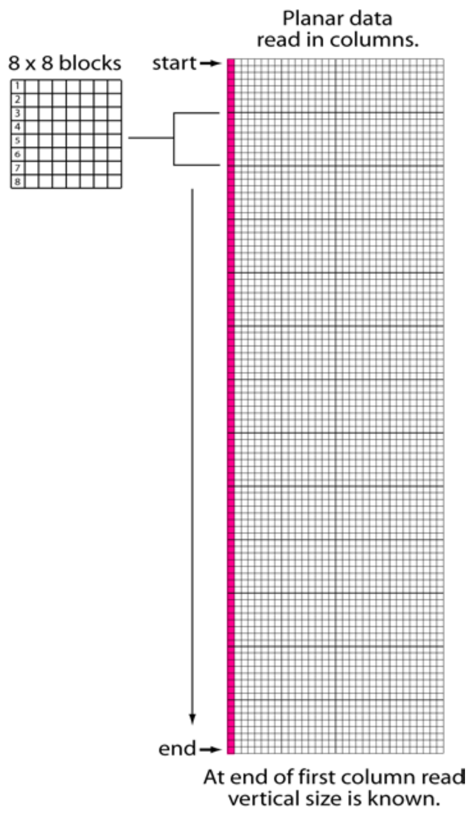
For example, when it is not decoding video data, the MAE back end can operate on planar YCbCr data received from the Camera Interface Module and perform Bayer pattern demosaic.

Scaling / Filtering

The MAE back end is a unique hardware component that is designed to process video data in a way that is distinctly different than the front end. The back end reads all frame data in columns. Figure 13 illustrates that at the end of the first column read, the vertical scale of the frame is known. Independent horizontal and vertical scaling are accomplished with a single pass.

Because it eliminates the round-trips to memory that are a typical step in display scaling, this data-handling technique helps minimizes use of bandwidth, and reduces overall processor size and cost by eliminating the need for on-chip buffers to handle scaling functions.

Figure 13. MAE Back End Processes Data in Columns


Planar data read in columns.
8 x 8 blocks
start
1
2
3
4
5
6
7
8
end
At end of first column read vertical size is known.

Scaling functions are in themselves scalable, supporting black & white operation with reduced memory requirements via pixel mirroring.

A four-tap filter optimized for 4x interpolation and 1/4 decimation is used for both directions. Each tap is augmented with 32 user-programmable coefficients that add a high degree of versatility to any filtering process.

Color Conversion

It is expected that the MAE back end will typically handle YCbCr > RGB conversion. However, color space conversion functions of the MAE are configurable through use of coefficient variables, and will support virtually any color space conversion.

New Thinking Solves Old Problems

In many technologies, new thinking about old problems results in breakthrough solutions. The MAE represents such a solution.

- Instead of transcoding video content down to meet the limitations of a power-hungry system, the MAE enables D1 native video decoding while using power frugally.
- Where DSP designs offer complexity and cumbersome architecture, the MAE delivers developers from complexity by supporting a simple programming model in a robust MIPS32™ environment.
- MAE design and function flows naturally, from well-established standard decoding requirements of the most popular block-based video formats.
- Its segmented design lends the speed of hardware based processing to alternative media sources, enabling more versatility in product function and differentiation.
- It maximizes processor functions by implementing unique data-handling techniques that maximize bandwidth.
- It contributes significantly to lower BOM costs because it minimizes physical impact on materials.

Document Revision History

Date	Revision	Changes
January 1, 2005	A	Initial version
January 5, 2005	B	Corrected the text in Figure 6 beginning “Comparison data...”.