

Tensilica White Paper

**Developing a High-Performance, Programmable
MPEG-4 Decoder by Adding a Programmable
SIMD Engine to a Configurable, Extensible
Microprocessor**

By Dennis Moolenaar

June, 2007

Digital video compression has become an important feature for a wide variety of products and designing those products to support multiple video standards can be challenging. Today, the main video compression standards are MPEG-1, MPEG -2, H.263, and MPEG-4. System designers face a major challenge when trying to create high-performance, flexible architectures for video coding standards. The problem is not just designing a solution, but creating a full-featured product in a reasonable amount of time. One design approach is to use hardware blocks, if they are available, but this is often not an option for complex, multi-featured products because of the large number of hardware blocks required to support numerous product features and ever-changing market requirements.

Another design approach employs programmable processors to provide the flexibility and fast time to market that is required by modern-day products. Unfortunately, standard programmable processors and cores are often incapable of handling real-time video decoding unless run at multi-GHz clock rates, which is quite impractical for portable devices. Designers usually must seek other solutions to reduce processor loads to levels that can be achieved by the typical processor-clock speeds used in consumer electronics.

This article shows that the desired video-decoding performance can be achieved by extending a configurable, extensible processor core's instructions with SIMD (single-instruction, multiple-data) instructions. Configurable, extensible processor cores are now available from multiple vendors and the advantage of using processor extensibility to accelerate video decoding is that SIMD instructions can also accelerate many other tasks that the processor might handle. This article uses the publicly available MoMuSys MPEG-4 video decoder (available at www.osi.org) as a reference, Tensilica's configurable and extensible Xtensa microprocessor core for the decoder's programmable processor, and the TIE (Tensilica Instruction Extension) language to add the SIMD instructions needed to boost an MPEG-4 simple profile decoder performance.

Using a reference-code implementation like the MoMuSys decoder as a base for a commercial product has one big disadvantage: reference code is generally written for understandability, not for efficiency. The MoMuSys reference code can handle any MPEG-4 Natural Video stream and even multiple video streams; whereas the simple profile is a smaller subset of MPEG-4 Natural Video coding. Support for all MPEG-4 video features incurs substantial overhead. However, this article demonstrates that these perceived reference-code deficiencies actually highlight the advantages of using a configurable, easily extensible, and verifiable processor core.

The MoMuSys MPEG-4 Video Decoder

The MPEG committee employed two software implementations of the standard to minimize ambiguities and to validate proposals made during the standardization process. Both of these verification models are downloadable from the OSI website. Tensilica selected the Mobile Multimedia Systems (MoMuSys) verification model for this MPEG-4 demonstration project. The MoMuSys MPEG-4 encoder and decoder are written in ANSI C and can be compiled with standard GNU compilers. The MoMuSys MPEG-4 encoder, set for no rate control and no skipped frames, generated the five QCIF (Quarter Common Intermediate Format) video streams shown in Table 1 that were used to validate the correctness of all MPEG-4 decoder optimizations.

Profiling the baseline MoMuSys decoder showed that integer multiplication consumed a large number of decoding cycles. Just including the Xtensa microprocessor's 16x16-bit hardware multiplier configuration option reduced the number of decoder execution cycles by 25%. This processor configuration serves as the base configuration for this article. Code simulation further established that the MoMuSys decoder uses floating-point arithmetic for the MPEG-4 inverse discrete cosine transform (iDCT) but a floating-point implementation is not actually required. An integer implementation of the iDCT runs much faster and still satisfies the minimum precision requirements defined by the IEEE 1180-1990 DCT standard. The integerized-iDCT version of the MoMuSys code serves as the reference for all performance comparisons in this article.

Video Clip	Number of Frames	Stream Size (Kbytes)	Average Bit Rate (@15 fps)
Miss America	149	66	52 Kbps
Suzie	149	129	104 Kbps
Foreman	400	724	217 Kbps
Car Phone	381	606	191 Kbps
Monsters Inc.	1438	1456	146 Kbps @ 18 fps

Table 1: MPEG-4 QCIF Video Test Clip Characteristics

Developing an MPEG-4 SIMD Engine

Creating processor extensions with the TIE language allows designers to use two basic code-optimization methods: reduce execution cycles by combining multiple operations into one TIE instruction or reduce execution cycles by operating on multiple data elements simultaneously (these are single-instruction, multiple-data or SIMD operations). These two methods can be combined. For an MPEG-4 decoder, the second approach offers the most promise because video decoding repetitively performs the same operations on large data blocks.

For example, consider motion compensation, which loads eight elements from the reference video frame and writes them into the current frame. SIMD TIE instructions can perform this task with just two instructions by using an 8-element wide load/store path, which boosts performance by 8x. All other portions of the decoding algorithm except bit-stream processing can be optimized in a similarly parallel way, suggesting that a tailored SIMD engine is suitable for most of the MPEG-4 decoder optimizations.

The resulting MPEG-4 SIMD engine operates on eight 16-bit or four 32-bit data elements at one time and executes the following instructions: arithmetic/logic instructions (addition, subtraction, arithmetic/logical shift, and logic operations), 128-bit load/store instructions, multiply/accumulate instructions, compare and conditional-move instructions, and a select instruction. The select instruction, shown in Figure 1, generates an output vector from two 64-bit source vectors and is used to rearrange related pixel information for faster SIMD processing. Code profiling indicated the main functions in the code that consumed the largest portions of the decoding cycles. Of these, the most highly parallel in nature were the MPEG-4 decoder routines that the SIMD engine could best optimize including the motion compensation, iDCT, variable-length decoding, and color-space conversion algorithms.

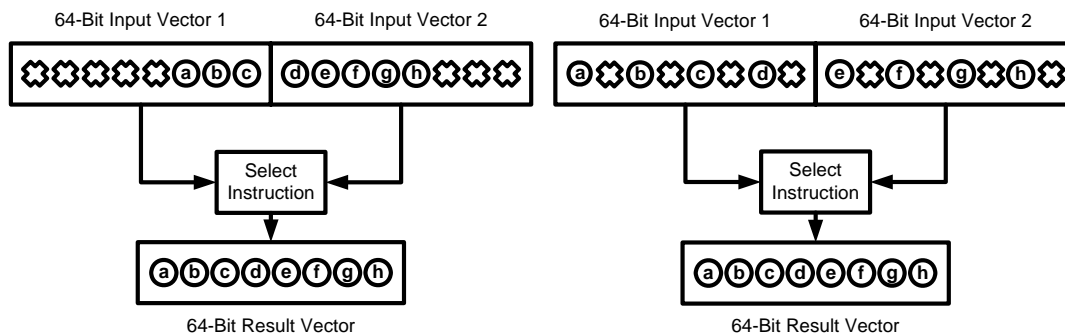


Figure 1: The Select instruction can combine pixel information from two vectors into one in several different ways (two ways are shown).

Optimizing Motion Compensation

Motion compensation exploits frame-to-frame image redundancy to achieve high video-compression ratios. The MPEG-4 encoder generates motion-estimation vectors on pixel blocks

using a pixel-search algorithm. The MPEG-4 decoder refines the resulting estimation vector to improve the appearance of the decoded picture using half-pel interpolation as shown in Figure 2. However, half-pel interpolation creates a performance challenge because every pixel value must be calculated from two pel values and a rounding factor (fac), resulting in many operations.

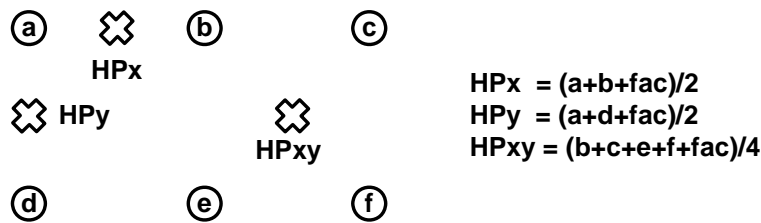


Figure 2: Half-pel (HP) motion-compensation interpolation from pels “a” through “f.”

Vertical half-pel computation is the simplest. First, the SIMD engine loads the data to interpolate an 8x8-pel block; a task that requires 9 SIMD loads. It then interpolates the data by adding the pel values from two adjacent rows, rounding, and dividing by using SIMD add and shift instructions. Horizontal half-pel motion compensation is slightly more complex because the pel data isn’t organized by columns initially. However, two SIMD-engine select instructions can reorganize the pel data and then compute the column interpolations. The original MoMuSys motion-compensation code required about 272 instructions executed in two loops. The SIMD-optimized code requires only 40 instructions executed in one loop—a 7x improvement. Although the MPEG-4 simple profile does not use quarter-pel motion compensation (also shown in Figure 2), the same programmable approach used for half-pel motion compensation can be used for quarter-pel interpolations in other MPEG-4 profiles.

Unrestricted Motion Compensation

Video decoders can use two methods to handle unrestricted (outside-the-frame) motion compensation: they can create a border image of pels around the reference frame or they can test the pel addresses during motion compensation to see if they fall outside the frame and then adjust the address if necessary. The advantage of the border-image approach is that no extra instructions are executed inside the critical loop of the decoder’s motion-compensation algorithm. The disadvantage of this approach is that extra memory transfers are needed to generate the border image. The address-adjusting approach does not add memory transfers but does add instructions and branches inside the motion-compensation loop.

The MoMuSys decoder implements both methods for handling unrestricted motion compensation. The first method involves copying the reference frame to a larger buffer and then creating the border around the frame. In principle, this work is sufficient for unrestricted motion compensation but the MoMuSys code also implements the second method. The advantage of using extensible processors and a programmable approach to building an MPEG-4 decoder is that they easily accommodate unrestricted motion-compensation methods and can incorporate the second motion-compensation method with no additional overhead.

Performance of Optimized Motion Compensation

Before summarizing the performance gains from these motion-compensation optimizations, we include one more optimization. In the original MoMuSys decoder, the AddImagel function consumes 3.24% of the cycles. This function adds the frame containing all iDCT results to the frame containing all motion-compensation results. Performing motion compensation at the block level and then adding the iDCT result-frame as a part of the motion-compensation function removes the need to add in the iDCT results separately. This optimization eliminates many memory transfers and reduces the required memory to handle the current frame to the size of six

iDCT blocks and one video frame instead of requiring two frame memories. Table 2 shows the substantial code acceleration achieved using the above techniques.

Video Clip	Original MPEG4 Motion Compensation Cycle Count	Optimized MPEG4 Motion Compensation Cycle Count	TIE Speedup
Miss America	1.620 G cycles	21.43 M cycles	75.6x
Suzie	1.660 G cycles	27.18 M cycles	61.1x
Foreman	4.571 G cycles	90.04 M cycles	50.8x
Car Phone	4.296 G cycles	79.35 M cycles	54.1x
Monsters Inc.	15.578 G cycles	194.96 M cycles	79.9x

TABLE 2: Effects of MPEG-4 motion compensation optimization

Inverse Discrete Cosine Transform (iDCT)

There are two groups of algorithmic implementations available for computing a two-dimensional (2-D) 8x8 iDCT: full 2-D iDCTs and algorithms based on the sequential application of a 1-D iDCT to the columns and then the rows of the 8x8 block. The MoMuSys MPEG-4 uses the latter approach, which is well suited to the SIMD engine. Executing eight 1-D iDCTs in parallel using the SIMD engine accelerates the 8x8 iDCT. The parallelized routine performs 1-D iDCTs on the columns directly. However, the image data isn't organized in memory properly to allow the SIMD engine to directly transform the rows so the data must be transposed after performing the column-oriented 1-D iDCT. Once the data has been transposed, the SIMD engine performs 1-D iDCTs on the row data, which is then transposed back to the original data layout to complete the full 2-D iDCT.

A full transposition of an 8x8 block using the SIMD select instruction requires 24 cycles so the overhead of the two transposition operations consumes 48 cycles. However, this overhead can be greatly reduced by transposing the data in a special transposition memory added to the processor using TIE. Table 3 shows the results of iDCT optimization.

Video Clip	Original Average iDCT Cycle Count	Optimized Average iDCT Cycle Count	TIE Speedup
Miss America	3220.86	345.24	9.4x
Suzie	3283	343.77	9.5x
Foreman	3414.7	342.22	10x
Car Phone	3356.83	342.62	9.8x
Monsters Inc.	3349.21	343.26	9.8x

TABLE 3: Results of MPEG-4 iDCT optimization

Bit-Stream Processing

Video decoders use a set of bit-stream functions to process video. All data is extracted from the video stream using one of these functions. Processor extensions can accelerate bit-stream processing functions by moving the video bit stream through a special shift register, added to the processor using TIE, that aids in bit-field manipulation and extraction. Note that this is not a SIMD operation.

Performing bit-stream processing functions with processor extensions also helps optimize variable-length decoding. The variable-length coding algorithm reduces the average number of bits needed to encode a code word by assigning a short bit string to the most frequently occurring code word and longer bit strings to code words that hardly ever occur. Variable-length coding allows the MPEG-4 encoder to reduce the number of bits needed to encode an 8x8 iDCT block. The code word represents the run, level, and sign of the iDCT coefficient (also called run-length

coding). The run values indicate the number of zero's that occur between two values. The decoder fills the iDCT block by extracting variable-length code words from the incoming bit stream and decoding them. The bit-stream processing hardware discussed above also extracts the variable-length codes from the video stream.

Additional MPEG-4 Decoder Optimizations

One portion of the MPEG-4 compression algorithm transforms the values of an 8x8 block into the frequency domain and then divides the resulting numbers by a quantization value. The video bit stream carries the quantization value and the decoder must perform the reverse transformation, called dequantization. The dequantization algorithm multiplies the decoded iDCT value by the quantization value to approximate the original DCT value. It's possible to quantize multiple iDCT values simultaneously and the MPEG-4 SIMD engine easily performs this task. Using the SIMD approach reduces the dequantization cycle count from 1647 cycles to 259 cycles per iDCT block for the Suzie test stream, a 6.4x improvement.

AC/DC prediction is only used for MPEG-4 Intra frames, which occur much less frequently than Inter frames. However, Intra frames require significantly more decoding cycles than Inter frames so it's very important to optimize this function because the clock rate required to decode an Intra frame sets a lower bound on the decoder's clock frequency. The decoder must be fast enough to decode any Intra frame in real time. AC/DC prediction reduces the number of bits required for encoding an Intra frame by estimating DC and/or AC values from the iDCT blocks. AC prediction, in particular, requires a large number of cycles to perform divisions and logarithmic functions inside a loop. These calculations can be moved outside the loop by rewriting the MoMuSys code. This optimization reduces the cycle count for AC/DC prediction from 2080 to 520 cycles.

The MoMuSys MPEG-4 decoder uses YCbCr color-space coding. The YCbCr representation usually must be transformed into a different color space such as RGB during the decoding operation. Transforming data from the YCbCr color space to RGB requires matrix multiplication. The transformed values for R, G, and B must also be shifted right by 16 bits to obtain the real RGB value. Color conversion is yet another critical decoding operation because every pixel undergoes matrix multiplication.

To accommodate the YCbCr-to-RGB color-space conversion requirements, the MPEG-4 SIMD engine multipliers are designed to handle 16x18-bit multiplications. The higher precision multipliers eliminate extra instructions that would otherwise be needed to deal with constants that are too big for traditional 16-bit, two's-complement multipliers. The SIMD engine performs a matrix multiplication for 8 pixels in parallel, producing an 8x performance improvement in the color-space conversion algorithm.

Table 4 shows overall MoMuSys MPEG-4 decoder optimization factors ranging from 28x to 40x for different video test streams. Consequently, this MPEG-4 decoder based on an extended processor core runs at a very low clock rate—much lower than for an unaugmented processor. The reason for the range of optimization factors is due to differences in the video streams. For example, this MPEG-4 decoder optimizes motion compensation more than dequantization or computing the iDCT so encoded streams with more iDCT blocks require more decoding cycles. Different test streams simply have different requirements. This characteristic of all video makes defining the worst-case bit stream problematic and it also demonstrates how important it is to use exactly the same test streams when comparing performance numbers for various video decoders.

Video Clip	Original MPEG4 Decoder Cycle Count	Optimized MPEG4 Decoder Cycle Count	Optimized Clock Frequency (15 frames/sec)	TIE Speedup
Miss America	3.126 G cycles	76.81 M cycles	7.7 MHz	40.1x
Suzie	3.389 G cycles	102.19 M cycles	10.3 MHz	33.2x
Foreman	10.045 G cycles	359.5 M cycles	13.5 MHz	27.9x
Car Phone	9.222 G cycles	308.7 M cycles	12.2 MHz	29.9x
Monsters Inc.	29.327 G cycles	822.8 M cycles	8.6 MHz	35.6x

TABLE 4: Overall optimized MPEG-4 video decoder results for QCIF video streams

Even at these performance levels, this software-based MPEG-4 decoder is not fully optimized. However, because this is a software-based decoder, it would be easy to refine the optimizations further should even better performance be required (for higher-resolution MPEG-4 implementations as an example).

The processor configuration used for this MPEG-4 decoder requires around 53K gates and the gate count for the programmable SIMD-engine and bit-stream-processing extensions is 67K gates. The resulting decoder is a flexible, programmable engine capable of decoding MPEG-4 video streams and other compressed-video streams with performance on par with hardware video decoders and a relatively low gate count. Even more important, the entire project (developing and tailoring a SIMD-enhanced processor using the MoMuSys reference code) consumed just seven man months, proving that this design approach can provide real time-to-market benefits to SOC developers.