MPEG-2 Video Decoding on the TMS320C6X DSP Architecture

Sundararajan Sriram, and Ching-Yu Hung
DSPS R&D Center, Texas Instruments, Dallas TX75265
{sriram,hung}@hc.ti.com

Abstract

This paper explores implementation of MPEG-2 decoding functions (bitstream parsing, IDCT, variable length decoding, motion compensation, dequantization) in software on the T.I. TMS320C6X architecture. We discuss cycle count estimates for these functions; our estimates are based on optimized, functionally accurate implementations in some cases, and on analysis of C implementations of the function in other cases. We describe how we arrive at these estimates in detail, and discuss how we were able to use automatic compilation effectively for certain functions. We also compare the C6x implementation to other MPEG-2 implementations that have been reported for general purpose CPUs that support a multimedia enhanced instruction set, such as Intel Pentium (MMX), SUN UltraSPARC (VIS), and HP PA (MAX).

1. Introduction

Over the next few years, we are going to see an explosive growth in the market for digital video devices, such as set-top-boxes, DVD players, DSS units etc. In all these products, video decompression is one of the most compute intensive, and memory and bandwidth intensive function. As a result there is great interest in obtaining the lowest cost solution for video decompression, the standard for which is MPEG-2. Current solutions for MPEG-2 are based principally on VLSI implementations because of the compute power required, except for certain control functions that may be implemented on a programmable microcontroller [1]. To make the implementation flexible and cost effective over a variety of products and product generations, however, there is now a great deal of interest in migrating functionality from application specific hardware into software running on a programmable CPU or DSP.

A number of manufacturers are offering "multimedia processors" that are claimed to be able to decode MPEG-2 coded video streams in real-time in software. Notable among these are the Trimedia processor from Philips, Mpact from Chromatics research, Multimedia signal processor from Samsung, and Mitsubishi's multimedia processor [2]. Most of these usually have hardware assists (in the form of peripherals) for one or more of the video decoding functions. There are also a number of general purpose CPU manufacturers that are offering "multimedia enhanced" versions of their CPUs for accelerating audio and video processing; the UltraSPARC processor enhanced with the "Visual Instruction Set" (VIS) from Sun [11], and the multimedia-enhanced MMX Pentium processors from Intel [8] are examples. Such CPUs will likely take over multimedia functions like A/V decoding/ encoding, modem, telephony functions, and network access functions on a PC/workstation platform, along with the general purpose computing they currently perform. The market for special purpose multimedia processors will be in low cost embedded applications such as set-top boxes, wireless terminals, digital TVs, and stand-alone entertainment devices such as DVD players.

The TI road map for high performance multimedia processors includes the newly introduced TMS320C6201 processor and its derivatives (C6x family). This processor is
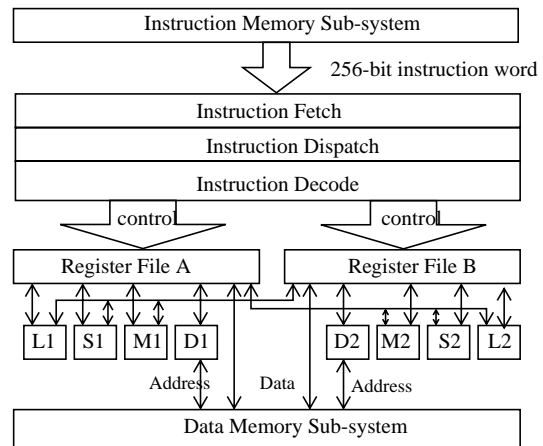

FIGURE 1. The TMS320C6x VLIW architecture

first in a series of announcements by major DSP manufacturers planning to come out with high performance general purpose DSPs using a VLIW architecture. The C6x processor is designed around eight functional units that are grouped into two identical sets of four units each (Figure 1). These functional units are the D unit for memory load/ store and add/subtract operations; the M unit for multiplication; the L unit for addition/subtraction, logical and comparison operations; and the S unit for shifts in addition

to add/subtract and logical operations. Each set of four functional units has its own register file, and a bypass is provided for accessing each half of the register by either sets of functional units. All eight functional units are controlled by a single 256-bit wide instruction word. Please refer to the C6x technical documentation [3] for details.

In this paper, we examine the various functions that are performed as a part of MPEG-2 decoding, and discuss implications for software implementation of these functions on the TI C6x architecture. By "MPEG-2" we will refer to main level, main profile, CCIR 601 format video, at 30 frames per second (fps). We present cycle count estimates for various functions implemented on the C6x, and discuss the system design issues for a set-top-box designed around this processor.

2. MPEG-2 video decoding overview

The block diagram for the functions involved in MPEG-2 decoding is shown in Fig. 2. The main functions involved are bitstream parsing, variable length decoding (VLD), inverse quantization and run length expansion, inverse discrete cosine transform (IDCT), and motion compensation (Mocom). We will only briefly define these functions in this paper; refer to [4][5] for details.

The memory system plays a very important part for any practical implementation of an MPEG video decoder, since the decoder core interfaces to between 12 and 16 Mbits of external SDRAM (Synchronous Dynamic Random Access Memory), depending on the decoder implementation, and the bandwidth to external DRAM can be as high as 100 MB/s.
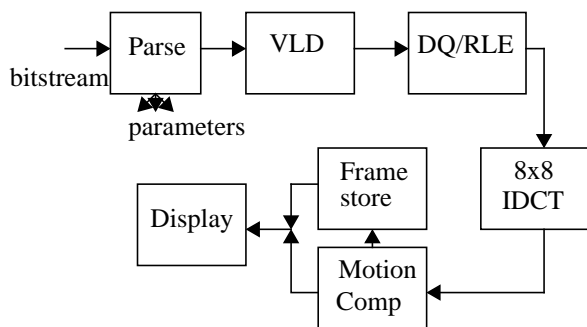

FIGURE 2. Simplified block diagram depicting the major functions in MPEG-2

3. Bitstream parsing

The MPEG-2 bitstream is serial, and has a well defined syntax as specified by the ISO/IEC 13818-2 standard [5]. The parsing function consists of identifying headers, reading parameters for the decoding process, and then actually reading out picture data such as variable length coded motion vectors and transform coefficients. This function involves bit-level operations (variable length shifts, masking a bit-field, comparison operations), as well as a significant amount of control and decision making. The latter arises owing to various mode decisions (picture type, macroblock type, motion compensation mode, VLC table to use etc.) that are made during the parsing phase. Even though parsing is best handled in software, processors with deep pipelines do not perform well for parsing because of frequent branching and condition evaluation. The predicated instructions in processors such as the C6x are helpful for short CASE statements, and conditionals that are not deeply nested. We estimate that 37 Million cycles/s will be required for video parsing on the C6x. This estimate is based on analysis of the C implementation of this operation, using worst case paths through the code, and profiling the "mpeg2play" program on a Sun workstation for determining trip counts on loops. This estimate assumes (pessimistically) that four motion vectors need to be computed in every macroblock.

Bandwidth to local and external memory is negligible for parsing, because the only accesses to memory are for storing values for various decoding parameters that are pulled off of the bitstream. Reading the bitstream from external DRAM is of course required, but we lump that bandwidth with the VLD operation below, because much of the bitstream represents the variable length coded transform coefficients anyway.

4. Variable length decode (VLD)

Variable length encoding of symbols is performed using a Huffman type encoding (MPEG-2 VL codes are not true Huffman codes). Motion vector information as well as run-level coded transform coefficients are variable length encoded [4]. Decoding operation for the coded run-level pairs is the most significant VLD operation, so we focus on decoding run-level pairs only. A run-level pair (r, l) represents a string of zero DCT coefficients of length r, followed by a non-zero coefficient with value l (more on this in Section 5). The decoding process involves matching the input bits to the correct VL code, and then advancing the bitstream by the length of the code. There are a number of ways of implementing the VLD function [10}. The easiest method is to use a table lookup; input bits address a ROM containing the decoded symbol, and the length of the VL code. The ROM is addressed by the maximum size of the VL code. Thus, for the AC transform coefficients, 17 bits at the head of the bitstream would address a ROM, which would then output the length of the code, and the run-level value corresponding to it, at which point the bitstream is advanced by the code length just determined.

A single table in a ROM of size 2^{17} for codes that could have lengths as large as 17 is clearly expensive. Such tables are also wasteful, because the shorter codes
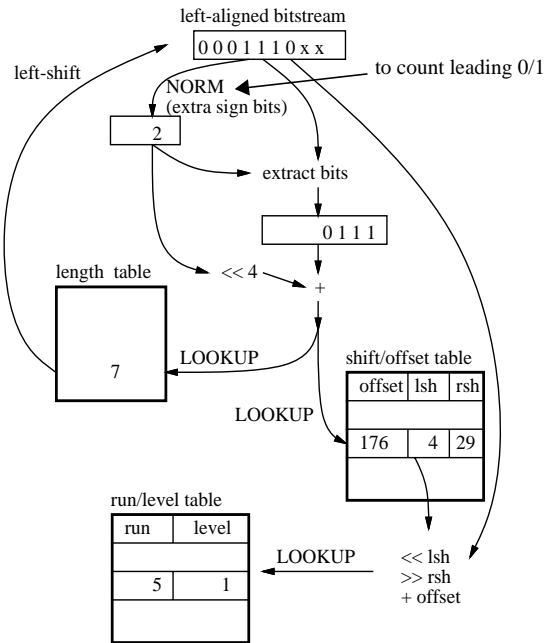
FIGURE 3.   Variable length decoding

have many repeated entries in the table. E.g. a code of length 4 would have to be repeated 2^{13} times within the table. In the case of hardware, such tables are implemented in PLAs or standard cells, and hence can be optimized significantly (only about 2000 gates for all the MPEG-2 VLC tables).

There are several different ways to decode the VL symbols in software [10]. One approach uses multiple-pass lookups. First we look up the first few bits of the bitstream in a look-up table. If the code lies within those few bits (which is very likely by the manner in which VLC tables are constructed) then the decoding process ends and the bitstream is advanced by the appropriate length. If the code does not lie within those bits, another table is pointed to based upon the first few bits. The second level table does the same for the successive few bits. This is commonly referred to as the tree method.

Although this method saves memory, it does so at the expense of processor cycles. An alternative is to bound the first few bits of the code-word according to the unique structure of a particular VLC table. The "mpeg2play" program implements decoding by determining the magnitude of the first 16 bits of the bitstream. Depending on the range in which this number lies, one of 8 different lookup tables is chosen. Finally, a masking operation and a translation is done to generate the address for the lookup within the chosen table. This yields the decoded run-level pair as well as the length of the code; the decoder advances the bitstream by this length.

Yet another way to decode the VLC is to partition the VLC tables, and then make use of the length of successive leading ones and zeros in the code, to pick which partition to use. This method is also highly dependent on the structure of the MPEG Video VLC tables. Fortunately, MPEG Video specifies two VLC tables and does not allow a customized table (like JPEG), enabling such an optimization.

The fact that the length of a code is known only after is has been decoded poses a serious problem for deeply pipelined processors that have a significant memory access latency, since the subsequent code cannot be decoded unless the length information of the current code is known. The C6x for example has a memory access latency of 4 cycles. Since the next code cannot be decoded before the current one, these cycles are essentially unusable, unless we overlap the VLD function with another function such as inverse quantization or IDCT.

The bounding approach leads to data-dependent number of cycles for decoding a symbol. It varies from 14 cycles for length 3 codes to 29 cycles for length 17 codes (the maximum possible code length for the AC coefficients is 17). Escape codes are ignored in this analysis, as are the DC coefficients, since these contribute significantly less to the VLD computation load than do the AC coefficients. The leading ones/zeros approach leads to a lower and more predictable cycle count. It takes 10 cycles to decode a symbol, which consists of an AC coefficient plus the length of zero run preceding the nonzero. Figure 3 shows the lookup and calculation process using an example. A fixed number of lookup operations, 3, are performed for each VLC symbol.

The VLD function needs to be done only for non-zero coefficients in each macroblock (MB). Thus the number of cycles required for VLD depends on the input data. On average, it is observed that 10 coefficients for inter-macroblocks, and less than 15 coefficients for intra-macroblocks are non-zero. However, a design based on these average numbers still has to deal with bitstreams containing a sequence of "bad" MBs with mostly non-zero transform coefficients. The amount of memory allocated for buffering during such peaks determines whether such peaks can be handled successfully. Both the worst case (all DCT coefficients coded) and a conservative average case (25% DCT coefficients coded) are reported.

5. Dequantization / Run-length expansion / Un-zigzag

The VLD outputs are run-level pairs in a zigzag scan order [4]. The next step is to expand the zero runs, quantize the level values and write the result in a row major scan order. Instead of explicitly performing the run length expansion, we propose to simply zero out an 8x8 block in memory, and write out the scaled levels into the correct address within the 8x8 block. The address can be looked up in a table indexed by the position of the run-level pair

in the zigzag scan order. This index is determined simply by accumulating the run values, and adding one to the sum for each run-level pair. This same index can be used to get the quantization (Q) matrix entry, since the Q matrix is also extracted from the bitstream in a zigzag scanned fashion, and it need not be un-zigzagged before being stored.

Writing 64 zeros to the 8x8 block consumes 15 MCyc/s at 30 fps, if this operation is performed by the CPU. This operation, however, can equally well be off-loaded to a DMA device external to the CPU. Assuming such a device, the DQ/RLE/unzigzag step takes 5 cycles for 2 pixels if performed in a separate loop. Again, only the nonzero DCT coefficients need to be processed, so the processing load for the VLD operation is data dependent.

It is in fact possible to merge the DQ/RLE/unzigzag loop into the VLD loop without incurring additional cycles. In other words, the overall VLD/DQ/RLE/unzigzag operation takes 10 cycles for each nonzero DCT coefficient. Such a merged implementation is possible because of the many free slots in the sequential VLD-only computation. Processing load for the merged VLD/DQ loop is 156 Mcyc/sec for the worst case when all the DCT coefficients are non-zero, and 39 Mcyc/sec for the average case.

6. IDCT

The 2-D 8x8 IDCT was implemented with the row-column method of using 16 instances of 1-D 8-point IDCT. The 8-point IDCT was carried out using the even-odd decomposition algorithm [12]. Figure 4 shows the signal flow graph of 8-point IDCT, wherein the cosine factor ci denotes $\cos(\pi i/16)$.

During our scheduling of the algorithm for the C6x architecture, we found that there is more pressure in the addition (S & L) units than in the multiplication (M) units. We used the following "trick" to reduce the number of additions. Each 8-point IDCT needs to properly round down all 8 outputs; simple truncation will not meet the IEEE precision requirements. To that end a half unit needs to be added to all 8 outputs before the right shifts. We utilize the structure of the signal flowgraph to add this half unit, instead, to the DC term $X(0)$ right after multiplying with $c4$, and have the half unit propagate to all outputs during the normal computation. Seven additions per 8-point IDCT are thus saved.

This implementation has been verified to pass both the IEEE precision requirement for IDCT as well as the dynamic range requirement in the Technical Corrigendum 2 of MPEG-2 Video [5].

The C6x implementation of IDCT has 12-cycle loop for the first dimension, and 13-cycle loop for the second dimension. Saturation is required by MPEG-2 Video for the second dimension. It takes 1249 cycles to compute 6 8x8 blocks, leading to 51 Mcycles/sec load.
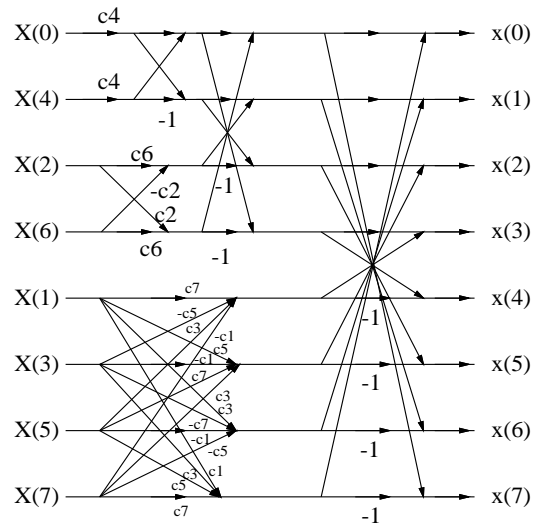

FIGURE 4.   Even/odd decomposition algorithm for IDCT

7. Motion compensation

The motion compensation operation consists of forming a prediction macroblock and adding it to the error block from the IDCT unit. The prediction is formed by using reference frame information pointed to by motion vectors specified for each intra-coded MB. Each motion vector points to a (potentially interpolated) MB in the reference frame. Maximum computation occurs when an MB requires both vertical and horizontal interpolation for each direction of prediction.

There are two ways in which the motion compensation algorithm may be implemented. One is to fetch pixels from the on-chip memory one pixel at a time using byte loads. This has the advantage of simplicity, at the expense of bandwidth to on-chip memory. The final program, compiled from C, consumes 6 pixels/cycle. The bandwidth to memory is 112 MW/s (out of the available 400 MW/s for a 200MHz C6x).

An alternative to fetching reference pixels a byte at a time is to use word loads to fetch four pixels of the reference MB at a time. This approach has the advantage of lower bandwidth to on-chip memory (28 MW/s for B pictures). Doing this, however, means the pixels have to be unpacked into half words, since the averaging operations need to be performed using 16 bit operations to maintain precision. The C6x shifter (S) units are likely to bottleneck seriously due to the packing, unpacking and shifts involved.

We decided to proceed with hand-optimized assembly coding for the byte-load approach. Additional arithmetic manipulations are employed to reduce the number of operations. The original expression of

avg_2_ref = ((a+b+c+d+2)>>2 + (e+f+g+h+2)>>2 + 1)>>1

was changed to

avg_2_ref = ((a+b+c+d+2) & ~3 +(e+f+g+h+2) + 4)>>3
= (((a+b+c+d+2) & ~3) +e+f+g+h+6) >> 3.

Two reference blocks, each 17 x 17 for luminance component and 9 x 9 for each chrominance component, are assumed to have been brought in by a DMA mechanism before the computation starts.

The benchmark code achieves 4 cycles per output point. The looping overhead is about 14%. The overall processing load for the worst case motion compensation processing comes out to be 71 Mcycles/sec. This load is for B frames that are bidirectionally predicted, with both horizontal and vertical interpolation. In reality an MPEG-2 image sequence has a mix of I, P and B frames, where the I frames require no motion compensation, and the P frames require only one direction of prediction (and hence take roughly half the CPU load as compared to a B frame).

8. Benchmark results

Table 1 summarizes cycle counts (in Megacycles per second) for the main MPEG-2 functions as implemented on the C6x. We also compare against published benchmark results for the Pentium (with MMX), HP PA (with MAX), and UltraSPARC (with VIS). Functions, such as bitstream parsing, for which we do not have benchmark numbers available are left blank. In some cases the VLD & DQ operations are merged, and so we provide a single number for the combined function. We refrain from comparing against platforms such as the Philips Trimedia, and Chromatics Mpact since these processors provide hardware acceleration for certain MPEG functions, e.g. VLD.

We have normalized all the published benchmarks to the same picture size (720x480 pels, at 30fps), and to the same number of non-zero DCT coefficients and I/P/B frame mix for average cycle count calculations. The 25% non-zero DCT coefficient assumption is somewhat pessimistic for real bitstreams; this is deliberately chosen to leave headroom for particularly "bad" bit-streams that may contain many more non-zero coefficients. The I/P/B mix of 7%/67%/26% is also similarly pessimistic.

9. Conclusion

We discussed the cycle count estimates for key functions in the MPEG-2 decoding algorithm, as implemented on the TMS320C6x architecture. Our methodology employed different strategies to obtain these numbers, based upon the effort required to obtain these numbers and the required accuracy of the cycle counts. The strategy ranged from rough estimates based on a C implementation (for the parsing function) to detailed functionally correct hand optimized assembly implementation of the function (e.g. for the VLD). Finally, we compared the benchmark numbers obtained for the C6x with published MPEG-2 benchmarks for "multimedia enhanced" general purpose CPUs.

The comparison results show that the C6x, with its flexible 8-way VLIW architecture, provides improved performance over general purpose CPUs that include multimedia acceleration. Note that these general purpose CPUs are several times more expensive in terms of dollar cost than the C6x, which is targeted toward embedded applications.

10. References

[1] D. Hocevar, S. Sriram, C-Y. Hung, "Performance modeling for system design: an MPEG A/V decoder example," ISCAS, June 1998.

[2] Proceedings of Hot Chips 8 Symposium, 1996, Palo Alto, CA.

[3] Texas Instruments, TMS320C62X/C67X CPU and Instruction Set Reference Guide, TI literature no. SPRU189C, March 1998.

[4] V. Bhaskaran, and K. Konstantinides, "Image and Video Compression Standards: Algorithms and Architectures," Kluwer International Series in Engineering and Comp. Sci., 408, 1997.

[5] ISO/IEC 13818-2:1996, MPEG-2 Video Standard Document.

[6] R. B. Lee, *et. al.*, "Real-time software MPEG video decoder on multimedia-enhanced PA 7100LC processor," Hewlett-Packard Journal April 1995.

[7] D. Ishii, *et. al.*, "Parallel variable length decoding with inverse quantization for software MPEG-2 decoders," Proceedings of 1997 IEEE Workshop on Signal Processing Systems (SiPS), 1997.

[8] Intel, "Using MMX instructions in a fast IDCT algorithm for MPEG decoding," Intel Application Note AP-528.

[9] Intel, "Using MMX Instructions to Implement Optimized Motion Compensation for MPEG1 Video Playback," Intel Application Note AP-529.

[10] C. Fogg, "Survey of software and hardware VLC architectures," SPIE Vol. 2186, Image and Video Compression, 1994.

[11] C.-G. Zhou, L. Kohn, D. Rice, I. Kabir, A. Jabbi, and X.-P. Hu, "MPEG video decoding with the UltraSPARC visual instruction set," Compcon, Spring 1995.

[12] C.-Y. Hung and P. Landman, "A compact IDCT design for MPEG video decoding," Proceedings, 1997 IEEE Workshop on Signal Processing Systems (SiPS 97), November 1997.fi

Table 1. MPEG2 decode benchmark summary (in Mcyc/s)

	Parse	IDCT	VLD[a]	DQ[a]	M. Comp[b]
TMS320C6x	37	51	39		62
Pentium MMX [8][9]	NA	59	33		72
HP MAX [6]	NA	97	41	13	72
UltraSPARC VIS [11]	NA	52	48	14	68

a. Assumes 25% DCT coefficients are non-zero; one number for both operations indicates the CPU load for a combined implementation

b. Assumes an average of 20 B frames, 8 P frames and 2 I frames out of every 30 frames.

NA ⇒ Data Not Available