REAL-TIME IMPLEMENTATION OF H.263+ USING TI TMS320C6201 DIGITAL SIGNAL PROCESSOR

K.-T. Shih, C.-Y. Tsai, H.-M. Hang
Department of Electronics Engineering
National Chiao Tung University
Hsinchu, Taiwan, R.O.C.
cytsai.ee90g@nctu.edu.tw, hmhang@mail.nctu.edu.tw

ABSTRACT

In this paper, we use a digital signal processor (DSP) to implement a real-time H.263+ codec. We use fast algorithms to reduce the codec computational complexity. Furthermore, the C programs are modified to take advantages of the DSP architecture and its C compiler features to reduce the on-chip memory and to increase the processing speed. In addition, a simple but effective rate-control algorithm is implemented to maintain the target bit rate. We can encode about 20 QCIF frames/second using one TI DSP. And the average decoding speed is about 26 QCIF frames/second.

1. INTRODUCTION

With the growing popularity of multimedia demand, video transmission over wireless network becomes very desirable in the near future. Because of the limited bandwidth of wireless channel, video signals have to be highly compressed. ITU-T H.263+ [1] is a specification for video compression targeting at very low bit-rate applications. Our codec is developed based on the H.263+ simulation software provided by Telenor Research and University of British Columbia [2].

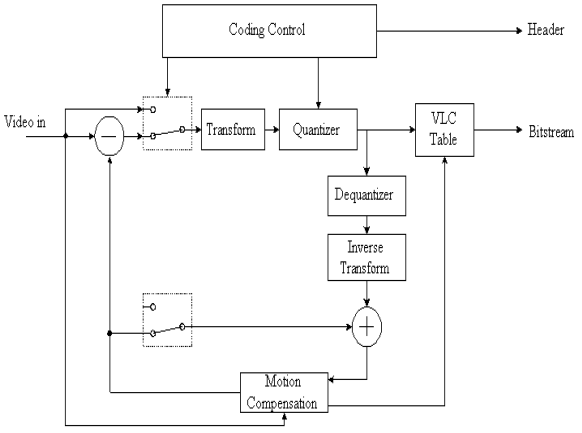


Figure 1. Block diagram of the basic H.263+ encoder [3].

Figure 1 shows the basic H.263+ encoder. The key elements are motion-compensated prediction, discrete cosine transformation (DCT), quantization, and variable-length coding (VLC). Motion-compensated prediction is used to remove temporal redundancy. The purpose of DCT and quantization is to reduce spatial redundancy. The VLC technique reduces syntax redundancy. All together, H.263 needs a very low bit rate, 64K bits/s or less, to transmit videophone type pictures. Because of high computational requirement of motion-compensated prediction and DCT, we use fast motion search algorithms and fast discrete cosine transforms to achieve real-time implementation.

The TI TMS320C62xx fixed-point DSP has a rather good performance. Its instruction cycle time is 5 ns (200 MHz clock). It adopts the advanced VelociTI very long instruction word (VLIW) architecture that enables sustained throughput of up to eight instructions in parallel and thus allows the processor running much faster. Therefore, the maximum computation power is 1600 million instructions per second (MIPS).

This paper is organized as follows. Section 2 describes the fast algorithms we use, diamond search, DIF DCT. Rate control algorithm is described in Section 3. In Section 4, the DSP implementation techniques and results are presented. Finally, a summary is given in the last section.

2. FAST ALGORITHM

2.1 Diamond Search

The full search algorithm for motion estimation examines all search points inside the search area. Therefore, the amount of its computation is proportion to the size of the search area. Although it finds the best possible match, it requires a very large computational power. Hence, many fast algorithms are proposed to reduce computation at the price of slightly performance loss. The basic principle of these fast algorithms is dividing the search process into a few sequential steps and choosing the next search direction according to the current search result [6].

The diamond search algorithm [7] starts with zero-vector candidate. Then, it moves to the most promising search point and does another search after the current step is completed. This procedure is repeated until it cannot move further and the local optimum is reached. The diamond search assumes that the matching function is monotonic along any direction away from the optimal point. But in reality the monotonic matching function assumption is sometimes invalid and thus the diamond search algorithm is suboptimal [6].

The procedure of the diamond search is described below.

Step 1: Compute the sum-of-absolute-difference (SAD) between the current macroblock and the same location macroblock in the previous reconstructed frame, called SAD0. This value is the prediction error when the current macroblock is

predicted using the zero-vector. Then, we set the current best vector (0,0) to be the search center.

Step 2: Four search points are chosen to center around the current best vector in a shape of diamond as shown in Figure 2. Their locations are (-1,0), (0,1), (1,0), (0,-1), respectively. Then, we compute the SAD of every search point and compare them to SAD0. If SAD0 is the minimum, the center represents the best motion vector, stop; otherwise, continue.

Step 3: Move the search center to the best vector that has the minimum SAD computed in Step 2 and replace SAD0 with this SAD value. Then, go to Step 2.

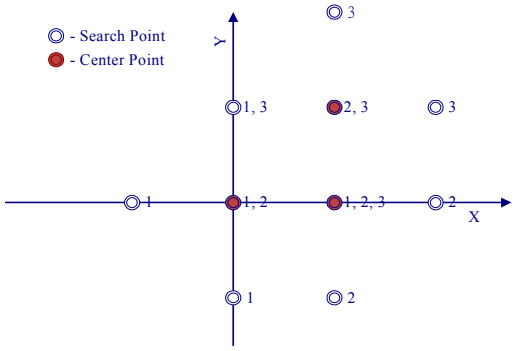


Figure 2. An example of diamond search procedure [11]

Figure 2 shows an example of the diamond search. Each stage contains four search points and a central point. They are all labeled by the same number. Three stages are presented in Figure 2. The central point moves to the right in the second stage. Then, it moves to the top in the third stage.

2.2 Decimation-In-Frequency (DIF) DCT

The original DCT algorithm in the software uses the sparse matrix factorizations. It requires 26 real additions and 16 real multiplications for an 8-point DCT. The amount of computation is not large, but this process is non-recursive and needs a complicated index mapping [8]. These two disadvantages leads to a serious delay in DSP implementation. Therefore, we need another fast DCT algorithm. The DIF DCT algorithm requires about the same number of additions and multiplications comparable to the original algorithm. But its process is recursive and needs only a moderate index mapping. So it fits better to the chosen DSP.

The key concept of the DIF DCT algorithm is to divide an N-point DCT to two N/2-point DCTs by rearranging of the input samples in the frequency domain. This concept results in a desirable recursive modularity of a fast decomposition. In our case, an 8-point DCT is decomposed into two 4-point DCTs. This method divides a DCT process into two parts with equal computing load when implemented on the DSP. Therefore, the DSP compiler can optimize the for-loop with ease. Moreover, this algorithm also can save three-fourths of the DCT code size. A 2-D 8-point DCT is obtained by computing two 1-D 8-point DCT along horizontal and vertical axes. The forward and inverse DCT used are specified in [8].

Because our DSP arithmetic unit uses fixed-point operations, floating-point computations are very inefficient when we implement DCT on this DSP. Therefore, we convert this floating-point algorithm to a fixed-point one. We multiply C_k by 2^{14}. Then, we round these values to their nearest integers. We use the new C_k to compute DCT and then divide the final DCT coefficients by 2^{14}. The multiplication and division of 2^{14} are realized simply by left-shift and right-shift by 14 bits on the DSP. It increases nearly no additional load. However, this fixed-point algorithm has the accuracy problem.

H.263+ Annex A specifies the desirable inverse transform accuracy [1]. We use it to examine the fixed-point DIF DCT, the floating-point DIF DCT, and the fixed-point original DCT. (We modify the original DCT into the fixed-point DCT in a similar manner as described in the above.) The results are shown in Tables 1 to 3. Annex A specifies that the overall mean error should not exceed 0.0015 in magnitude and the overall mean square error should not exceed 0.02. The floating-point DIF DCT satisfies this specification, but the fixed-point DIF DCT and the fixed-point DCT cannot meet the specification due to rounding errors in the fixed-point computing process. The errors are accumulated and propagated to the next frames in the interframe coding. Therefore, frames become blurred if we encode a number of P-frames (inter-coded frames).

Data Range	Overall Mean Absolute Error	Overall Mean Square Error
L=256,H=255	0.861547	1.589669
L=5,H=5	0.854330	1.545261
L=300,H=300	0.734516	1.354213

Table 1. The accuracy of the fixed-point DIF DCT

Data Range	Overall Mean Absolute Error	Overall Mean Square Error
L=256,H=255	0.000014	0.000014
L=5,H=5	0	0
L=300,H=300	0.000011	0.000011

Table 2. The accuracy of the floating-point DIF DCT

Data Range	Overall Mean Absolute Error	Overall Mean Square Error
L=256,H=255	1.262053	3.570425
L=5,H=5	1.222489	3.271673
L=300,H=300	1.076280	3.047527

Table 3. The accuracy of the fixed-point DCT

Figure 4 shows this situation. Figure 4 depicts the 100-th P-frame of “Salesman” encoded using different DCTs. The floating-point DCT is used in (a), while the fixed-point DIF DCT is used in (b). Clearly, the salesman’s right hand and the box blur more significantly in (b). The errors are generated in the I-frame stage and then are diffused by motion vectors into the P-frames. Hence, the moving part has the strong blurred effect. In order to avoid this effect, we force our encoder to perform one INTRA-frame coding for every ten P-frames. Consequently, the DCT errors are not accumulated large enough to distort the video quality. There is another solution stated in the Annex W [5]. The fixed-point DCT specified in Annex W has a better accuracy, but it, on the other hand, increases computations significantly. In practice, to eliminate transmission error propagation in particularly wireless environment, frequent frame refresh or intra-frame coding is necessary. Intra-coding strategy is thus adopted.

(a)

(b)

Figure 3. The 100-th P-frame of "Salesman" encoded by using (a) floating-point DCT and (b) fixed-point DIF DCT.

3. RATE CONTROL

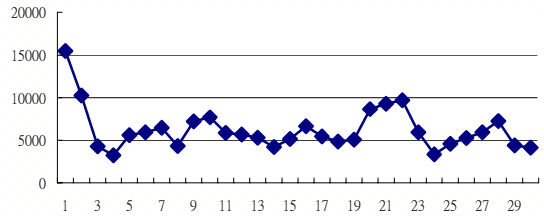
To maintain a constant output bitrate, a rate control algorithm must be used in our encoder. There are two major considerations: the delay produced by bits accumulated in the encoder buffer and the bit allocation issue, which affect the coded picture quality. In addition, including a rate control into our encoder implementation requires a significant amount of additional computation and memory (for program and data). Hence, we choose a less complicated rate control algorithm described in TMN5 [12].

At the beginning, we have a target frame rate f_{target} and a target bitrate R . Consequently, we can calculate the target number of bits per picture B. When finishing encoding one frame, we obtain $\Delta_1 B$, which is the difference of bits spent on the coded picture and the target bits B at frame level. If the available bits per frame (B) are distributed uniformly over all macroblocks, we know the target number of bits per macroblock. Thus, we can calculte $\Delta_2 B$, which is the difference of coded bits spent on a macroblock and its target bits at macroblock level. Essentially, we use these two values, $\Delta_1 B$ and $\Delta_2 B$ to adjust the quantization parameters to allocate bits based on the following formula [12]:

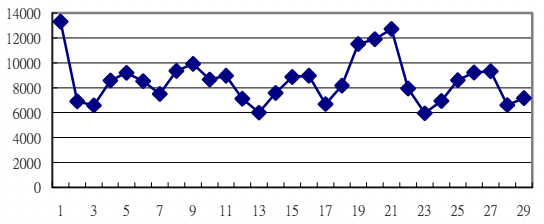
$$QP_{new} = QP_{i-1}\left(1 + \frac{\Delta_1 B}{2B} + \frac{12\,\Delta_2 B}{R}\right),$$

where QP_{i-1} is the quantization stepsize for the previous frame and QP_{new} is the quantization stepsize for the current macroblock to be coded. Moreover, we need to take care of the delay due to buffering and maintain the buffer fullness as near constant as we can. Therefore, if the buffer fullness exceeds a threshold, the encoder will skip the next frame until it goes down to the acceptable region.

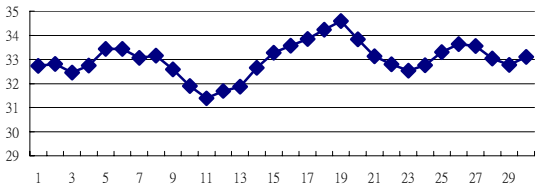
Figure 4 shows the simulation results using the aforementioned (TMN5) rate control algorithm. In this experiment, the target bit rate is 53kbps, and the buffer threshold is 8kbps for skipping frames. Because one intraframe is enforced for every ten frames, the bit rate is somewhat higher at multiples of the tenth frame. Also, the PSNR values are lower for these frames since intraframe coding requires more bits. Although the frame coding mode (inter/intra) is changed regularly, the output bit rate and the buffer level remain nearly constant. In general, the rate control mechanism works quite well.



(a)



(b)



(c)

Figure 4. QCIF foreman sequence encoded with TMN5 rate control (a) bits used per frame, (b) encoder buffer fullness, and (c) PSNR per frame.

4. DSP IMPLEMENTATION

We implement our system on the Blue Wave Systems PCI/C6600 applications board. It provides a software support library for data transfer between the host PC and the DSP. This library is called "Generic Host Interface Library" (GenrHL) [10]. It includes two types of data transfer: "message system" and "mailbox interrupt". The former one provides efficient bulk of data transfer between host PC and DSP. The latter one provides interrupt between host PC and DSP and is useful for synchronization. Besides, This board has two TI TMS320C6201 fixed-point processors on it. Each DSP has 64kbytes internal program memory, 64kbytes internal data memory, and 16Mbytes external SDRAM. We will describe our implementation in detail in the next subsection.

4.1 Optimization for DSP Architecture

- ***Intrinsic operator:*** The C6000 compiler provides intrinsics, special functions that map directly to inlined C62x instructions to optimize C codes. All instructions that are not easily expressed in C codes are supported as intrinsics [13][15]. For example, we can use the intrinsic operator "_abs" to calculate the saturated absolute value.
- ***Wider memory access for smaller data widths:*** In order to maximize data throughput, it is often desirable to use a

single load or store instruction to access multiple data values consecutively located in memory. For example, C6x have instructions with associated intrinsics, such as "_add2()", "_mpyhl()", "_mpylh()", etc, that operate on 16-bit data stored in the high and low parts of a 32-bit register. When operating on a stream of 16-bit data, we can use word accesses to read two 16-bit values at a time, and then use another C6x intrinsics to operate on the data [13].

- ***Memory management:*** TI TMS20C6201 DSP has only totally 128kbytes internal memory. Therefore, memory management becomes very important. In program memory management, we delete unused codes and re-write some functions to reduce the program code size. Furthermore, we use the compiler options to optimize the execution speed. In data memory management, we put all dynamic allocated memory sections into the external SDRAM and put frequently used data in the internal data memory.

4.2 Implementation Results

We replace the motion vector search algorithm and the DCT algorithm with the fast algorithms described before. In addition, we optimize the program using the optimization techniques described in the previous section. The final results are shown in Table 4. Our code size is about 46kbytes without rate control and 58kbytes with it.

	Original	Optimized
I- frame coding	67.48MIPS	2.89MIPS
P-frame coding	229.52MIPS	6.44MIPS

Table 4: Encoding speed comparison between the original source codes and the optimized codes.

On the other hand, the decoder is easy to implement on DSP as compared to the encoder. Hence, we do not describe in details on its implementation. We simply give the final results here. The average decoding speed is about 26 QCIF frames per second, and the code size is about 27kbytes.

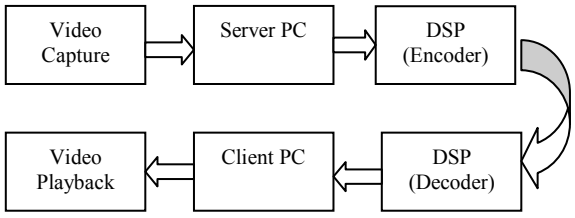


Figure 5. System block diagram of the pipeline-like structure.

4.3 Multitasking System

Multitasking programming for operating system means the computation power of CPU is not dedicated to a single application but can be distributed to multiple tasks simultaneously [14]. In our system design, each block in Figure 5 is an individual thread. Therefore, the system has a pipeline-like structure. Furthermore, each interface between different blocks requires a synchronization object to control the data flow. This server-client based system can achieve about 14 QCIF frames per second encoding and decoding speed.

5. SUMMARY

In this paper, we implement the basic H.263+ encoder using the TI TMS320C6201. To reduce the computation load, we bring in the diamond search algorithm and the fixed-point DIF DCT algorithm. However, the fixed-point DIF DCT contains the rounding errors, which affect the video quality. One simple and practical approach to reduce this effect is forcing one intra frame per ten or so inter-coded frames. In addition, we modify our C codes to take the advantages of the TI DSP architecture and the compiler's features. At the end, we can encode about 20 QCIF frames/second using one TI DSP. And the average decoding speed is about 26 QCIF frames/second.

6. REFERENCES

[1] ITU-T Study Group 16, *Video Coding for Low Bit Rate Communication*, 1998.

[2] Telenor research official ftp site: ftp://bonde.nta.no/pub/tmn/software.

[3] C. Côté, B.Erol, M.Gallant, and F.Kossentini "H.263+: Video Coding at Low Bit Rates". *IEEE Trans. Circuit Syst. Video Technol*., vol. 8, no. 7, Nov 1998, pp. 849-866.

[4] Texas Instruments "TMS320C6x Technical Brief." *Texas Instruments*, 1999.

[5] ITU-T Study Group 16, *H.263 Draft Annex U, V, and W*, Feb. 2000.

[6] H.-M. Hang and J. W. Woods, *Handbook of Visual Communications*, Academic Press, 1995

[7] ITU-T Study Group 16, *Video Codec Test Model, Near-Term, Version 8* (TMN8), 1997.

[8] K. R. Rao and R. Yip, *Discrete Cosine Transform- Algorithms, Advantages, Applications*, Academic Press, 1990.

[9] Blue Wave Systems, *Blue Wave Systems PCI/C6600 Applications Board Technical Reference Manual*, 1999

[10] Blue Wave Systems, *Blue Wave Systems Host and MPC860 C GenrHL User Guide for C6x Boards*, 1999

[11] M. L. Woo, *Real-Time Implementation of H.263+ Using TI TMS320C62xx*, MS Thesis, Institute of Electronics, National Chiao Tung University, June 2000.

[12] ITU-T Study Group 15, *Video Codec Test Model, TMN5*, Jan. 1995.

[13] Texas Instruments "TMS320C6000 Programmer's Guide" *Texas Instruments*, 1999

[14] J. R. Woo, *DSP-based Real-Time H.263 Encoding /Decoding and Transportation for Video Conferencing*, MS Thesis, Institute of Electronics, National Chiao Tung University, June 2000.

[15] Texas Instruments, *C Implementation of the TMS320C62xx Intrinsic Operators*. Texas Instruments, Application Report, Literature Number: SPRA616, Dec. 1999.