# Software-Only Real-Time MPEG-2 Video Encoding on The C62x VLIW Processor*

**H. Lee, K. Nguyen-Phi, H. M. Alnuweiri, F. Kossentini**

Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, B. C., Canada V6T 1Z4
{henryl,knguyen}@ece.ubc.ca

## ABSTRACT

Due to their high computational demand, MPEG-2 video coding solutions have been based mainly on custom hardware (ASIC) systems. Such systems lack the flexibility and adaptability of software-based solutions. Achieving real-time MPEG-2 video encoding in software remains to be a major challenge. A typical MPEG-2 encoder performs 20 to 30 GOPS (Giga operations per second), which exceeds the capabilities of the most advanced contemporary processors.

We have developed and tested a highly optimized, low complexity, high-quality MPEG-2 video encoder software based on Texas Instruments' fixed-point TMS320C6201 VLIW processor. The current version of the encoder can handle SIF (or 320x240 resolution) video format at 40 frame per second, i.e. faster than real-time. For CCIR 601 (or 720x480 resolution) at our algorithm encodes 15 frames per second. Our real-time MPEG-2 encoder has been implemented and tested on the C62x Evaluation Model (EVM) board from TI. Our encoder software was optimized at the assembly language level to maximize the attainable ILP of the C62x VLIW architecture. Currently, the real-time encoder can handle both I and P pictures in CIF video format, and I pictures only in CCIR video format. We believe that this is a big leap forward towards achieving real-time video encoding in software only on a VLIW DSP. Our next work will be the implementation of a full-featured MPEG-2 encoder at MP@ML (CCIR in real-time with I, P, and B frames) using a quad-chip C6x configuration.

* This research was supported by a grant from the National Sciences and Engineering Research Council (NSERC) of Canada and by a grant Rogers Communications.

# I. Introduction

With the growing deployment and commercialization of multimedia applications, comes the demand for higher performance to be offered to such applications at the lowest possible cost. Additionally, multimedia processing platforms must become more flexible and easily re-programmable to keep up with changing standards and application domains. Currently, video signal processing is the dominating task in terms of computational demands as well as amount of information bandwidth, unless compression technology is used. The video coding applications have led to the development of video coding standards for video compression such as ISO/IEC MPEG-1 and MPEG-2. The required processing rate for video compression ranges from 100 mega operations per second (MOPS) to more than one tera operations per second (TOPS). For instance, real-time MPEG-2 video decoding for National Television Systems Committee (NTSC) resolution requires more than 400 MOPS, while MPEG-2 video requires about 30 giga operations per second (GOPS) [1]. Such requirements are beyond the capabilities of contemporary microprocessors and have been satisfied mainly through the use of ASIC technology only. However, this trend is changing with the advent of new fundamental algorithmic enhancements to video encoding [--reference Dr. Kossentini's work on MPEG-2 --], and with the development of media-enhanced processors from various vendors.

Current solutions for real-time MPEG-2 video encoding and decoding are based principally on VLSI implementations such as the custom hardware (ASIC) systems. However, in order to make implementations flexible and cost-effective, migrating functionality from application specific hardware into software running on a programmable general purpose processor or DPS have started to gain great deal of interest. Recently developed microprocessors and programmable DSP chips offer powerful processing capabilities to do real-time video compression/decompression.

This paper reports a highly optimized, low complexity, high-quality MPEG-2 video encoder software that runs in real-time or near real-time on Texas Instruments' fixed-point TMS320C6201 VLIW (very long instruction word) processor. The current version of the encoder can handle SIF video format (320x240 resolution) at 40 frame per second, i.e. faster than real-time. For CCIR-601 format (or 720x480 resolution) our algorithm encodes 15 frames per second. Our real-time MPEG-2 encoder has been implemented and tested on the C62x Evaluation Model (EVM) board from TI. Our encoder software was optimized at the assembly language level to maximize the attainable instruction-level parallelism (ILP) of the C62x VLIW architecture. Currently, the real-time encoder can handle both I and P pictures in CIF video format, and I pictures only in CCIR-601 video format. We believe that this is a big leap forward towards achieving real-time video encoding in software only on a VLIW DSP.

# II. A Real-Time MPEG-2 Video Encoder

The objective of MPEG-2 video coding is to provide high quality and multi-channel compressed video signals transmission over limited-capacity broadcasting infrastructure such as Cable/HFC and ATM networks. Specifically, MPEG-2 was given the mandate of providing a video quality no less than NTSC/PAL and up to CCIR 601 quality with a target bit rates in the range 2 to 10 Mbit/s [2]. The MPEG-2 coding standard only specifies the syntax for encoded bit-stream, and many of the complex encoding decisions are to provide maximum flexibility for implementing compliant video codecs. This flexibility has posed quite a challenge for video coding engineers to design and determine the trade-offs between coding performance (high compression with acceptable quality) and implementation complexity of various MPEG-2 video coding schemes.

In general, video sequences contain a significant amount of statistical and subjective redundancy within and between frames, i.e. video sequences usually contain statistical redundancies in both the temporal and spatial domains. Consequently, two major types of compression schemes have been proposed in the past to reduce the temporal and spatial redundancies: inter-frame coding and intra-frame coding. Intra-frame compression reduces spatial redundancies within a video frame by employing transform coding as in still image coding. This type of compression is ideal when the difference between two consecutive video frames is very large. Intra-coded pictures are also referred to as I-pictures in MPEG-2. Inter-frame compression takes advantage of temporal redundancies by coding the difference

between a predicted frame and the current video frame instead of encoding the current frame itself. A forward predicted inter-frame is also referred to as a P-picture while a bidirectionally predicted inter-frame is referred to as a B-picture. The combination of intra and inter-frame coding is called hybrid coding.

### *A. The MPEG-2 Video Coding Standard*

The MPEG-2 video bit stream consists of six hierarchical layers starting at the block layer (composed of 8x8 pels) followed by the macroblock, slice, picture, group of pictures (GOP), and the sequence layers as shown in Figure 2. The GOP level is where the video sequence determines the ratio of I, P and B frames, and where the most relevant work on temporal picture structure is carried out. In a typical GOP size of 12 interlaced video frames, the first frame (I-picture) is intra coded, and the following 11 frames are inter coded using alternatively forward motion compensation (P-pictures) and bi-directional motion compensation (B-pictures) as shown in Figure 3. MPEG-2 video supports different standardized picture formats such as CCIR 601 (720x480 resolution) and CIF (352x240 resolution), etc. Each input picture from the video sequence is divided into marcoblocks, and every macroblock (MB) consists of a 16x16 block (or four 8x8 blocks) of luminance (Y) component and two 8x8 blocks of chrominance (C*b* and C*r*) components. Figure 4 shows the MPEG-2 picture structure at CCIR 601 resolution.

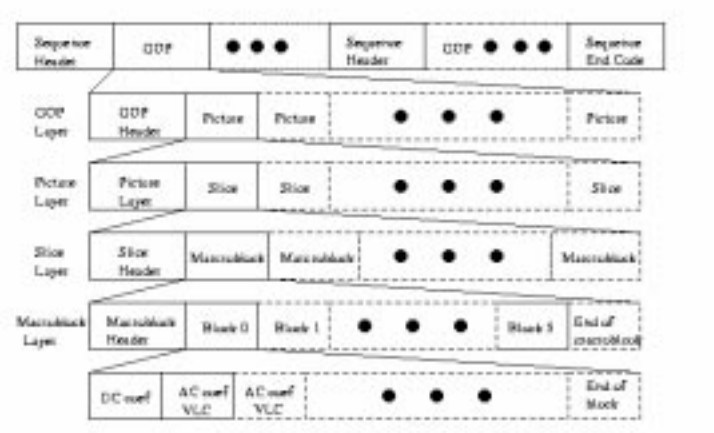


Figure 2: Video sequences layer

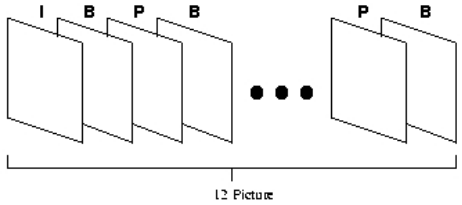


Figure 3: GOP of 12 frames

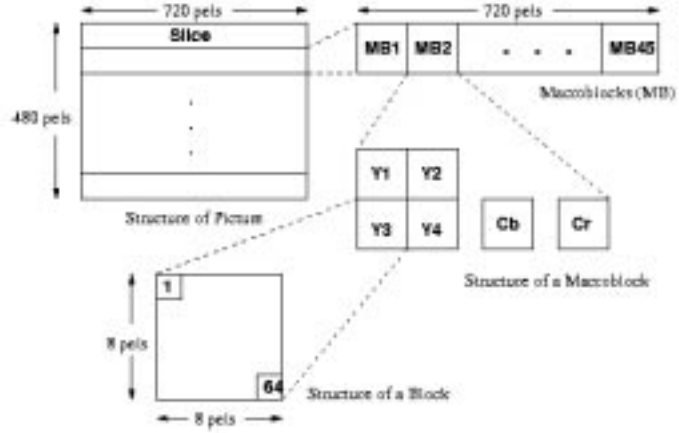


Figure 4: MPEG-2 picture structure at CCIR 601 resolution

### *B. Baseline MPEG-2 Video Encoder*

The basic MPEG-2 algorithm has features similar to those used in ITU-T H.261 and MPEG-1 video coding standards. There are six key algorithmic components in MPEG-2 video encoding: motion-estimation and motion-compensation (ME/MC), mode selection (MS), transform coding (DCT/IDCT),

quantization (Q/IQ), variable length coding (VLC), and rate control (RC). Figure 5 depicts a block diagram of the MPEG-2 baseline video encoder. Essentially, motion estimation searches for the best motion vector with reference to the pervious frame and performs motion-compensation prediction to reduce the temporal redundancies. Then transform coding (using discrete cosine transform (DCT) algorithm) encodes the motion-compensated prediction of the difference frame to reduce spatial redundancies. Following transform coding, the resulting coefficients are quantized then the quantized DCT coefficients, the macroblock (MB) coding mode, the quantization step, the frame/field motion vectors, and the residuals are finally variable-length encoded.

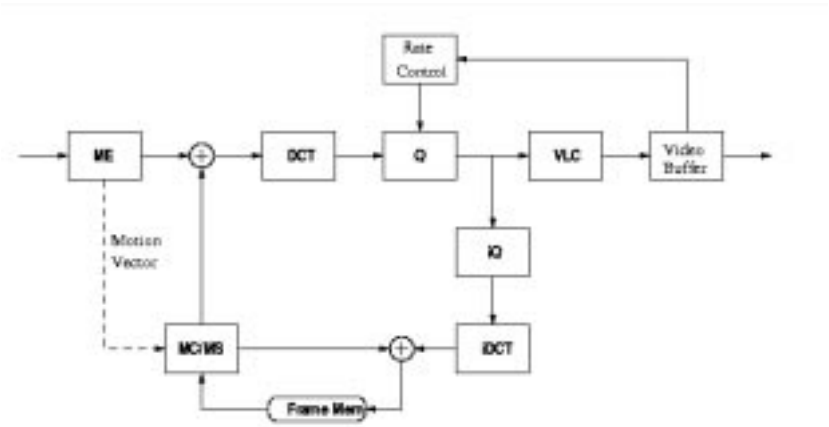


Figure 5: MPEG-2 video encoder

### *C. Proposed Software Solution for Real-Time MPEG-2 Video Encoder at UBC*

The starting point of a software-only real-time MPEG-2 video encoder on the C62x VLIW processor, is the development of efficient code that can be efficiently compiled on the C6x. Extensive work has been carried out on software MPEG-2 video encoder to improve and to implement an efficient real-time MPEG-2 video coding algorithm Over the past two years, the Signal Processing and Multimedia Group at the University of British Columbia, has carried out extensive work towards developing software-only MPEG-2 compliant video encoders. We have mainly targeted general-purpose processor platforms such Intel's Pentium II and III, and VLIW DSPs such as Texas Instrument's C6x. The computational complexity reduction in our MPEG-2 software implementation is achieved by employing several advanced techniques which are briefly describe below, further detail description of the algorithms can be found in [3], [4], and [5]. The macroblock layer has been the major focus of our optimization since that is where coding gains are obtained and where most of the computational load resides.

*1)* ***Motion estimation (ME)*** is considered be one of the most time consuming tasks to perform in video encoding. It involves searching for the best match between the current block and candidate blocks in a confined search windows of the previous encoded frame. The location of the best match block is the estimated motion vector and the motion vector is computed using the most widely known matching function -- sum of absolute differences (SAD). The main objective of ME is finding an optimal motion vector in a limited search window using the minimal number of SAD computations. Our encoder employs the following techniques.

*a) Optimum Fast Block Matching Strategy*: The fast ME algorithm breaks up the search process into a few sequential steps, and each step is based on searching the candidate points located on diamond-shape contours. There are two version of the matching strategy: fixed-center diamond search, and floating-center diamond search. Experiments have shown that the number of SAD operations is reduced by 50% and the actual number of computations is reduced by a factor ranging from 100:1 to 250:1 compared to the full-search ME [4].

*b) Hierarchical Block Matching Strategy*: The search algorithm reduces the computation by utilizing lower resolution of the current and reference frames. This is done by downsampling both the current and reference frames by a factor of two in both directions.

*c) Additional Stopping Criterion*: The number of computations associated with motion estimation is reduced, as the algorithm terminate even earlier by using a prediction method that detects zero macroblocks.

*d) The Optimum Block Matching Function*: our encoder uses a standard procedure called partial distortion technique to reduce the computations associated with SAD operation. This technique abandons the current SAD computation if it already exceeds the minimum known SAD.

*e) Robust Motion Vector Prediction*: A reliability measure of the predicted motion vector (PMV) has been incorporated to assess the PMV before using it as a center for motion search. If the PMV was unreliable, then the fast diamond search will be abandoned and a full search will commence in a lower resolution image. By using different MV predictors and the reliability measure, the PSNR of fast motion sequences is improved significantly while the computation demands are also reduced slightly [6].

2) ***Rate-Distortion Optimized Mode Selection***: Our encoder obtains improved rate-distortion (RD) performance by employing a rate-distortion based macroblock mode selection. The coding mode is chosen to minimize the RD Lagrangian to improve performance An efficient rate-distortion optimized mode search algorithm is used to attain a close-to-optimal performance while maintaining the number of computations low by exploiting the statistical redundancies in the video sequence [4].

3) ***Combining Discrete Cosine Transform (DCT) and Quantization***: For the hybrid macroblock coding modes, both the DCT and quantization need to be performed. By merging the DCT and quantization into a single operation, known as quantized-DCT or QDCT, the number of computation can be substantially reduced. QDCT is performed only during inter-frame encoding leading to an order of magnitude reduction in computations. Early Prediction techniques have been also incorporated into the QDCT algorithm to further reduce the computation [7]. The integer implementation of QDCT is especially efficient to run on the C62x platform since it uses fixed-point arithmetic.

### *D. Analysis of the proposed MPEG-2 video encoder*

Timing analysis is critical to identifying the most computational intensive components of the MPEG-2 encoder which need substantial optimization. Table 1 shows the relative weights of the most compute-intensive MPEG-2 encoding functions. Clearly, motion-estimation algorithm is still the most computation demanding.

We have identified the set of components that are computationally intensive but which potentially can be optimized efficiently on the C62 platform by exploiting the instruction-level parallelism inherent in the algorithm. The main components optimized for the C6x encoder implementation include the SAD calculation, DCT/IDCT, Quantization, QDCT, mean square error (MSE) calculation, variable length encoding (VLC).

| **Function** | **Computational Load** |
|---|---|
| Quantization | 4.2% |
| DCT | 5% |
| IDCT | 5% |
| ME (SAD Computation) | 50.1% |
| Motion Compensation | 4% |
| MSE | 1% |
| VLC | 10% |

Table 1: Relative weights of the most compute-intensive components of an MPEG-2 encoder

## III. Implementation Issues for the TMS320C6201 DSP

Texas Instrument's TMS320C6201 VLIW DSP is representative of a new class of DSPs that exploit explicit instruction-level parallelism (ILP) and advanced compiler techniques to provide thousands of MIPS to potential applications

### *A. Description of the C62x DSP*

The C6x series DSP consists of three main components; VLIW core, peripherals, and memory. The VLIW core consists of two symmetrical data paths with four different function units and access to 16 multiported 32-bit register each. Direct communication between both data path is provided through the cross path. The four function units of each data path are divided into a logic/arithmetic/compare unit, a shift/arithmetic/branch unit, a 16-bit multiplier unit, and a data address calculation unit. For all the arithmetic calculations, saturation arithmetic is supported. Figures 7 shows the block diagram of C6201Core. The memory architecture of the C6201 consists of a 64-kbyte data memory and a direct-mapped 64-kbyte instruction cache. The instruction cache may alternatively be configured as an on-chip program memory containing a valid address space.

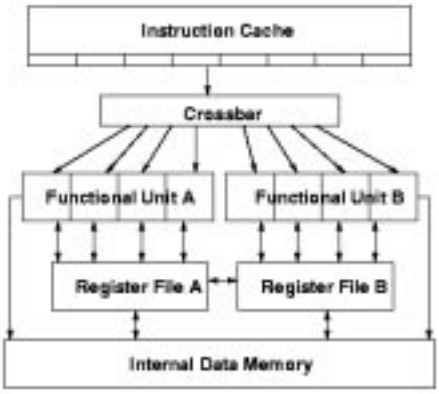


Figure 7: Block diagram of the C6201Core

The C62 Evaluation Model (EVM) board by Texas Instrument has been used for our preliminary software implementation platform. The board has a 200 MHz C62 fixed-point DSP processor with access to two 4-Mbytes of DRAM, and 256K of SBSRAM. The communication between the Host PC and the DSP can either use DMA or Host Port Interface (HPI). More details can be found in [8].

### *B. Implementation Bottlenecks*

*1)* ***Memory Issues***: The internal on-chip bus, external memory interface (EMIF), and peripherals often represent the crucial bottlenecks in a real-time system. Access to vital external programs and data must pass through these critical components on their way to the VLIW core for processing. In fact, external memory access delays form the greatest performance bottleneck to our MPEG-2 video encoder on the C62 platform. One major problem was that the MPEG-2 video encoder program is much larger than the 64-kbyte on-chip memory of the C62x, which means that the main program must be run out of an external memory. This can cause significant slowdown of the program. Internal data memory can transfer data 20 to 30 times faster than the external memory can. Therefore, we have treated the internal program memory as a memory cache and attempted to optimize the cache hit-ratio to enhance performance. Given that, video processing does not preserve data locality, moving program data to internal DSP memory is not likely to result in significant speedup gains. We use DMA transfers for that purpose.

*2)* ***Compiler Optimization Issues***: Even though TI provides an optimizing compiler with different levels of code optimization, the typical code performance is improved only by 30% - 40% on average. Much more aggressive optimization can be obtained by hand-optimized assembly code. For many of the video coding components, assembly-level optimization performs three to twenty times faster than the code generated by the optimizing compiler. We discuss this further in the following sections

*3)* ***Instruction Latency***: For most of the C62x arithmetic operations, instruction latency is one cycle. Multiplication instructions require two cycles. However, for load/store and branch instructions have a latency of five and six cycles, respectively [9].

*4)* ***Memory Bank Conflict***: The memory architecture of C62 consists of 4 memory banks each of which is half-word size (or 16-bytes) wide. Simultaneous memory access to the same bank can cause a memory bank conflict which stalls a load/store instruction for one cycle [9].

# IV. VLIW Implementation of the MPEG-2 Video Encoder

This presents platform-specific optimizations for the computationally intensive components of MPEG-2 video coding mentioned in the previous section. The purpose of these optimizations is to take advantage of the VLIW architecture of the C62x and to overcome some inherent memory-access and I/O bottlenecks so as to enable real-time video encoding on the C62x processor. In the following, we present the main performance gains achieved by our optimization techniques for the major components of the encoder.

## *A. Optimizing SAD computations for Motion Estimation*

During inter-frame coding in MPEG-2, motion vectors are found by performing a number of SAD operations on 16x16 blocks. For every SAD calculation, there are two unsigned load byte instructions, one subtraction, one absolute operation, and one addition which consumes a total of five out of the eight available functional units (the two multiplier functional units are not use in SAD calculations). The performance of the inner loop of the SAD computations is constrained mainly by the number of load/store functional units in the C62x processor. Because only two load operations can be performed per cycle, the SAD computation can compute the absolute difference of these two values in one step, producing one result per cycle at most. Two approaches can be used to optimize the SAD computation.

*1)* ***First Approach***: This approach unrolls the 16x16 two-dimension loop into a one-dimension loop with 16 iterations, each consisting of 16 SAD computations. This loop unrolling reduces the number of loop-end branch instructions from 64 to 16. Because each branch instruction has a latency of 6 cycles, reducing the number of branch instruction has significant impact on speedup. The structure of the SAD program is as follows. Before entering the inner loop, eight execution packets are preloaded to increase the pipeline efficiency and eliminate load-instruction latency (which is 5 cycles). The inner loop contains 16 execution packets representing a row of 16 SAD operations, with each execution packet consisting of at least two load instruction, one subtraction, one absolute instruction, and one addition which are executed concurrently in different functional units. The $11^{th}$ execution packet also includes a "delayed" branch instruction that actually takes effect after the $16^{th}$ execution packet is completed.

The next step of optimizing the SAD computations involves resolving any possible memory-bank conflicts which can degrade performance. Indeed, the parallel loading of two contiguous unsigned data bytes from memory can result in memory bank conflicts 50% of the time. Figure 8 shows how our program avoids this conflict by loading every other unsigned data byte during a parallel-load execution packet. Overall, our SAD function implementation runs approximately 4 times faster than the version obtained form the optimizing C compiler of the C62x. Specifically computing the SAD function for a 16x16 block requires only 270 cycles, as compared to 1220 cycles if the C compiler is used.

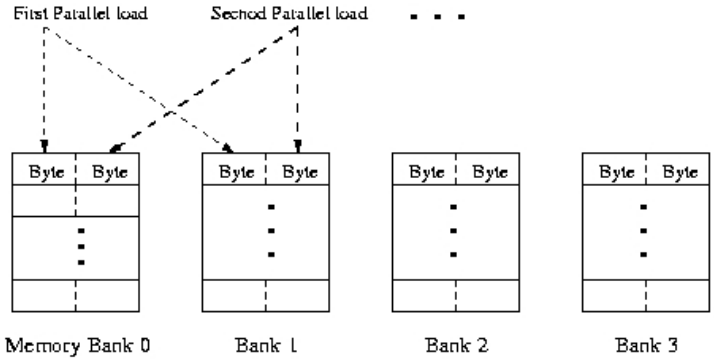


Figure 8: Avoiding memory bank conflicts by loading every second data from different memory banks

*2)* ***Second Approach***: Further optimization is possible by applying the subword parallelism feature of the C62 architecture. This can result in producing four SAD results per three execution cycles (i.e. about 1.333 results per cycle) compared to the one result per cycle attained by the first approach. Instead of loading the two unsigned data bytes in each cycle, Two data words, containing four unsigned byte data, are loaded. The operation of the improved SAD computation technique is depicted in Figure 9. After the two data words have been loaded, the half-word subtraction function (SUB2) is used to compute the difference between A2 and B2 to obtain r2, and the difference between A4 and B4 to obtain r4. Then both data words are right-shifted by 8 bits and a similar SUB2 function is applied to obtain r1 and r3. Now, both r1 and r2 use the EXTU function to extract the 8-bit result of the difference data; while r3 and r4 will be ANDed with 0x000Fh to clear the upper 24 unrelated bits. Four absolute computations are performed concurrently on the results in (r1, r2, r3, r4), then the results are added up to obtain the SAD for four unsigned data bytes. This implementation requires two load instructions, two half-word subtraction, two unsigned right shift, two unsigned extract, two AND instruction, four absolute instruction, and four additions. Therefore, the total of number operations require 18 instructions which can be packed into three execution packets, as compared to four execution packet in the first approach. Hence, the resulting speedup.

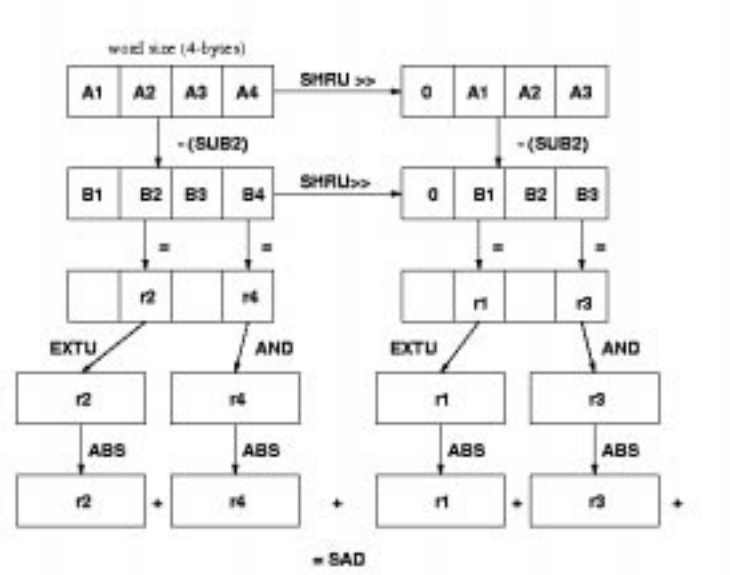


Figure 9: Flowgraph of the SAD computation in VLIW implementation

### *B. Quantization*

After the transform coding is performed, quantization of the DCT coefficient will be processed. The quantizer we use is set to a fixed scale of 10 bits with all of the quantizer scale coefficient calculated at the beginning. We use this quantization function for the intra-coding stage only. The main quantization step performed is of the following form

$$Q[i] = \{(x[i] + c1[i]) * c2[i]\} >> 16$$

where x is a DCT coefficient, c1 is a quantizer scale, and c2 is a computed scale. The main computation here consists of quantizing 64 DCT coefficients for an 8x8 intra-coded block. Although the computation required for the quantization of DCT coefficient is straight forward, the C6x compiler optimization could not fully exploit the available instruction-level parallelism. In our optimization, the main constraint each execution packet allows only two load/store instructions, while the above quantization step requires three loads and one store. Under this constraint we are able to achieve 2 results per cycle in the inner loop. Loop initialization, loop unrolling, and delayed branching have been all used in the optimization. The main inner loop consists of 16 execution packets each consisting of two load/store operations. However, most of the execution packets are not fully utilized since only one addition, one multiplication, and one shift are performed within every 2 cycles on average. Overall, our optimized quantization function runs

on the C62 approximately 20 times faster than the optimized C-compiler version. Specifically, quantizing an 8x8 DCT block requires 128 cycles by our method, as opposed to 2505 cycles generated by the C-compiler.

### *C. Quantized Discrete Cosine Transform (QDCT)*

As mentioned before, the QDCT algorithm combines both DCT and quantization into a single computation and reduces the overall computational load substantially. Unlike SAD and quantization calculations, QDCT's high computational cost is dominated by its complexity. The two-dimensional QDCT computation must be unrolled to an 8x1 vector such that column QDCT is preformed first then row QDCT is performed second. In QDCT, load and store instructions are no longer the dominant constrains for optimization. However, the order of load and store operations still has an important impact on the overall performance of the optimization. Two methods have been implemented to determine which order of loads and stores provides a better optimization level of the QDCT. The first, and simplest, method is to do a parallel load of 8 data at the beginning of each iteration, then transformed results are stored at end of each iteration. The advantage of this method is that the code linearity is preserved and register usage becomes more flexible. The drawback is that the resources are poorly utilized when intermediate result, such as those from multiplications (2-cycle latency) and load operations (5-cycle latency), are not available. The result of computing a column of QDCT requires 24 instruction cycles in the inner loop, and eight iterations are required for completing the column QDCT calculation for a total of 192 instruction cycles. In the second method, the order of load/store instruction is sequential executed which mean one load and one store operation per instruction execution. This method results in much better functional unit utilization than the first methods. The main limitation, however, is increased register usage caused by the use of preloaded quantized coefficients and the storage of intermediate results. Using the second method, a QDCT column computation requires 14-instruction cycle in the inner loop, and nine iterations for completing the column QDCT calculation for a total of 126 instruction cycles. Therefore, the second method has a 50% performance gain over the first method. The two methods of load/store orderings are depicted in Figures 10 and 11. Finally, the performance gain of using optimized assembly is almost 20 times faster as than the C62x C-compiler implementation of QDCT.

```
 LDH     Data1     ;parallel load
||LDH    Data2     ;of two data

 LDH     Data3
||LDH    Data4
        .
        .
        .
 STH     Data1     ;parallel store
||STH    Data2     ;of two data

 STH     Data3
||STH    Data4
```

Figure 10: Parallel execution of loads at the beginning & stores at the end

```
 LDH     Data1     ;parallel load
||STH    Data1     ;store data

 LDH     Data2
||STH    Data2
        .
        .
        .
 LDH     Data7
||STH    Data7

 LDH     Data8
||STH    Data8
```

Figure 11: Sequential execution of load/store pairs

### *D. Variable Length Coding (VLC)*

For the SAD computation, quantization, and QDCT, the performance of the optimization is independent of the context of the data, and all of these algorithms have tight inner-loops that are good candidates for optimization on an ILP architecture. The picture is quite different for the VLC algorithm whose structure is more inherently sequential with strong data dependencies. The performance of the VLC algorithm is highly dependent on the quantized DCT coefficients; the more zeros the quantized 8x8 blocks have, the more efficient the algorithm runs. The zig-zag scan of the DCT coefficients in an 8x8 block causes further inefficiencies in performance of the program code. We have, therefore, employed a method for hard-

coding the zig-zag scan pattern and generated a table of zero run-length codes for use with Huffman coding later on. The performance of our optimized code is 3 to 4 times faster the code generated by the C-compiler. This has been one of hardest algorithms to optimize for the C62x. For example, in the intra-coding process, VLC accounts for almost 50% of the computational load after the other components have been optimized.

### *E. Mean Square Error (MSE)*

Mean square error (MSE) computation is similar to the SAD computation, except that MSE calculate square of the difference between two data elements. For the MSE calculation, multiplication is used instead of the absolute function used in the SAD computation. The hand-optimized assembly code for MSE is very similar to that of the SAD except for the use of an additional multiplication (with a 2-cycle latency) per execution packet, and this results in a total of 271 instruction cycles for computing the MSE for a 16x16 luminance block. For the two 8x8 chrominance blocks, 95 instruction cycles are required for each block. The performance of the assembly level optimization on the MSE on a macroblock is 12 times faster than the optimized C version.

## V. Overall Performance of MPEG-2 video Encoder on the C62x

There are currently two versions of the MPEG-2 video encoder being implementing on the C62x EVM board. The first is an I-frame only version of the MPEG-2 video encoder which does Intra-coding at SIF resolution. This version currently runs at above real-time rate of 40 frames per second. In this setup, the MPEG-2 video encoder directly captures video frames from a camera at SIF resolution, and encodes the video in real-time (or faster). An updated I-frame version of MPEG-2 video encoder can encode 15 frames per second at CCIR-601 resolution.

The second version of the MPEG-2 video encoder is an IPP-version which does both intra- and inter-coding and that significantly improves the compression ratio. The IPP-version of the encoder is expected to encode 15 frames per second at CIF resolution. Currently we are working on developing a full featured, software-only, real-time MPEG-2 video encoder possibly on a dual or quad C62x processor platform.

## VI. Reference


[1] Yuen-Wen Lee, Faouzi Kossentini, Rabab Ward and Mark Smith, *"Towards MPEG4: An Improved H.263-based video coder,"* in Signal Processing Image Communication, Oct. 1997.

[2] Thomas Sikora, *"MPEG Digital Video-Coding Standards,"* in IEEE Signal Processing Magazine, Sept 1997.

[3] Yuen-Wen Lee, Faouzi Kossentini, Rabab Ward, *"Efficient MPEG-2 Encoding of Interlaced Video,"* in Signal Processing Image Communication, Oct. 1997.

[4] Alen Docef, Faouzi Kossentini, Rabab Ward, *"Towards a Software Solution for Real-time MPEG-2 Encoding of Interlaced Video,"* in UBC Abstract draft.

[5] J.L.Mitchell, W.B.Pennebaker, C.E. Fogg, D.J.LeGall, *"MPEG Video Compression Standard,"* Chapman & Hall 1996.

[6] Ismaeil Ismaeil, Alen Docef, Faouzi Kossentini, and Rabab Ward, *"Motion Estimation Using Long Term Motion Vector Prediction,"* in UBC Abstract draft.

[7] Khanh Nguyen-Phi, Alen Docef, and Faouzi Kossentini, *"Quantized Discrete Cosine Transform: A combination of DCT and scalar quantization,"* in ICASSP, 1999. Submitted.

[8] *"TMS320C6000 Technical Brief,"* SPRU197D, Texas Instruments Press, Feb. 1999.

[9] *"Optimizing C compiler User's Guide,"* SPRU187, Texas Instruments Press, Feb. 1999.