Computational*RAM implementation of MPEG-2 for real-time encoding

T. M. Le, W. M. Snelgrove*, and S. Panchanathan

*Department of Electronics, Carleton University
Visual Computing and Communications Laboratory
Department of Electrical and Computer Engineering, University of Ottawa
Ottawa, Ontario, Canada, K1N 6N5
URL: http://marge.genie.uottawa.ca

ABSTRACT

In this paper, a Computational Random Access Memory (C*RAM) implementation of MPEG-2 video compression standard is presented. This implementation has the advantage of processing image/video data in parallel and directly in the frame buffers. Therefore, savings in execution time and I/O bandwidth due to massively parallel on-chip computation and reduction in the data transfer among chips is achieved. As a result, MPEG-2 video encoding can be realized in real-time on a programmable 64Mb DRAM-based C*RAM.

Key words: MPEG, SIMD architecture, parallel processing, logic in memory.

1. INTRODUCTION

Real-time implementation of multimedia applications such as digital camera, video-on-demand, interactive TV, etc., have started a trend for complexity reductions by both simplifying the algorithms and designing application-specific architectures. A video sequence (typically of size 576 x 720 pixels/frame at 30 frames per second) involves processing in both spatial and temporal dimensions to remove the redundancies for compression. In the MPEG encoders, *Discrete Cosine Transform* (DCT) is employed to exploit the spatial correlation (intraframe coding), while *Motion Estimation / Compensation* (ME/C) is used to remove the temporal redundancies (interframe coding). We note that ME/C is a computation intensive process and requires extra storage for storing the reference frames. The encoded information is further compressed using a lossless coding technique such as *Variable-Length Coding* (VLC). A large computation and I/O bandwidth for implementing the MPEG-2[1] compression algorithm in real-time poses a challenging problem.

A significant number of implementations for reducing the complexity using parallel and pipelined approaches have been proposed in the literature. These include reduced complexity algorithms[2-4] and high-performance architectures[5-9] for implementing the DCT, ME/C, and VLC algorithms. From the perspective of architectures, the above mentioned functions have been mostly addressed either in the form of a general *Digital Video Processor* (DVP) or dedicated *Application Specific Integrated Circuit* (ASIC) implementations. The key limitation in these implementations are the high processor-memory bandwidth in the DVP, and the inter-chip bandwidth in the multi-chip ASIC.

In this paper, a single chip, massively parallel implementation of MPEG-2 based on a *Computational Random Access Memory* (C*RAM)[10,11] is presented. C*RAM is a SIMD-memory hybrid architecture and has 2 functions, namely, storage and computation. In the storage mode, C*RAM functions as a RAM or frame buffer. In the computing mode, C*RAM can be programmed to execute a variety of algorithms, including image and video compression[12-14]. The principle advantage of C*RAM over other approaches is the capability of processing image and video data in parallel and directly in the frame buffers. Hence, this technique results in savings in execution time and I/O bandwidth due to massively parallel on-chip computation and reduction in the data transfer among chips.

The current C*RAM is a 64Mb DRAM consisting of an array of 8192 *processing elements* (PE's). Its structure has been enhanced from its predecessors to accommodate large frame sizes and reference frames in MPEG-2, as well as to facilitate on-chip computation of functions such as DCT, ME/C, and VLC. As a result, MPEG-2 *main profile / main-level* (MP/ML) video encoding can be realized in real-time. The continuing advance in VLSI memory technology is expected to result in the fabrication of 256Mb[15] C*RAM chips by the end of this decade.

The rest of the paper is organized as follows: An overview of C*RAM is provided in section 2, followed by a brief review of the MPEG-2 video compression standard in section 3. The C*RAM implementation of MP/ML MPEG-2 is provided in section 4, followed by the performance analysis in section 5. The conclusions are presented in section 6, followed by the references.

2. OVERVIEW OF C*RAM

C*RAM is a SIMD-memory hybrid architecture, where PE's are attached to the memory array at the sense-amplifiers. Primarily designed with the goal of augmenting conventional computer RAM, the C*RAM concept makes use of the large on-chip bandwidth to perform massively parallel computations. There are two major functions in a C*RAM: storage and computation. When functioning as a storage device, C*RAM is read or written as part of the host processor address space. When functioning as a computing device, all PE's execute (on their own local memory) in parallel.

2.1 C*RAM Architecture

A number of C*RAM versions[10,11] have been designed based on both DRAM and SRAM technologies. In this paper, a model of 64Mb DRAM-based C*RAM is implemented. There are 8192 single-bit PE's attached to the memory at the sense-amplifiers (Fig.1), which can be programmed to execute in both single-bit and multiples of 8-bit parallel operations. As a result, the terms 1-bit, 8-bit, 16-bit, etc. *computing units* (CU) are used hereafter. Data shifting and neighbourhood communications are possible via left-right interconnections. The maximum clock rate of the 64Mb C*RAM is 25Mhz (40ns).

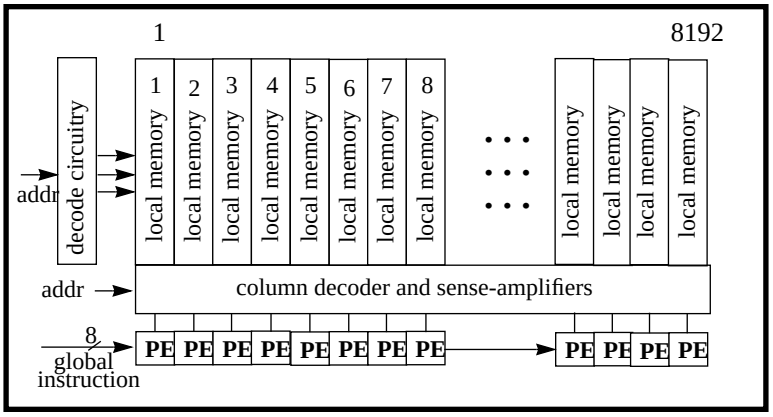

Figure 1. C*RAM architecture

2.2 Processing Element Model

The PE model used in this C*RAM is an enhanced version of the baseline PE presented in[10]. PE's features are summarized as follows. A PE (Fig. 2) normally has three 1-bit registers: X, Y, and M; and a 1-bit arithmetic-logic unit (ALU) which functions as an 8-to-1 multiplexer. Inputs to the single-bit ALU can be contents of registers X, Y and the output Z. Output of the ALU can be one bit of the 8-bit global instruction sent from the off-chip controller. After each computation, the result is written back into one of the registers.

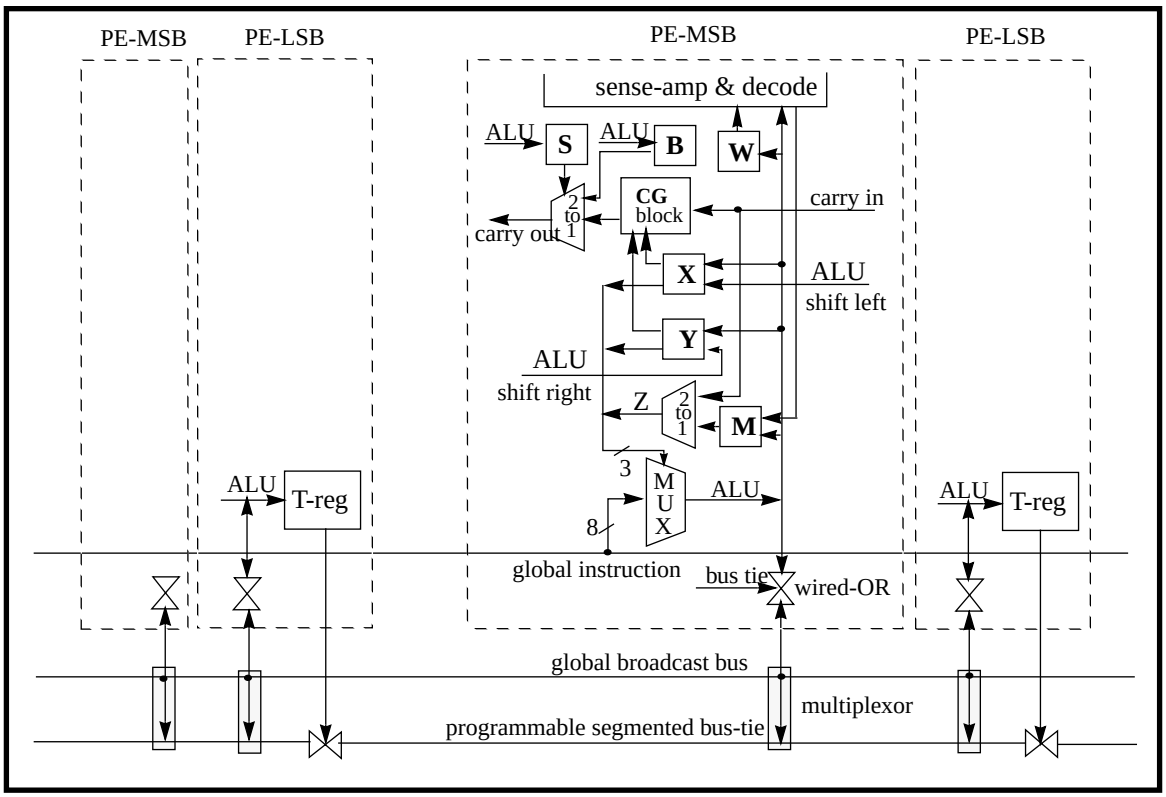

Figure 2. PE model and programmable segment bus-tie

In addition to the X, Y, M registers and the ALU, a Write-enable (W) register is added to allow for conditional operations. W register functions as a mask, blocking the non-participating PE's in various operations. In other words, the W register is set only when there is a "Write" to the local memory of the corresponding PE, and reset otherwise. We note that different masks may be stored in the local memory for different PE operations. These masks enable all the PE's in the C*RAM to participate in the global operations without modifying the masked memory contents.

Multiples of 8-bit PE's can be grouped to form a CU. A bus-tie register T is placed at every 8^{th} PE at the LSB position. The value stored in this register is used to disable the transmission gate connecting the *programmable segment bus-tie* (PSB). To perform the wired-OR logic of all the multi-bit CU's in parallel (bottom of Fig. 2), the values in the T registers are loaded to the corresponding transmission gates, followed by the wired-OR instruction. All ALU outputs (sharing the same segmented bus) of the same CU are ORed, and the result is fed back to the corresponding CU. The PSB circuit is extensively used in zero-value checking and bit-extension operations. In addition, a global broadcast bus is designed to broadcast a 1-bit constant value from the controller to all the PE's. A multiplexor is used to select between the PSB and the global bus.

A ripple-carry circuit is also implemented by adding the carry generator (CG) block. Recall that for addition (or subtraction), two operations are required: carry (or borrow) and sum (or difference) generations. Inputs to the CG block are the *carry-in* from the right-neighbour PE, and 2 other operands stored in registers X and Y. We note that the value of carry-in is set to 0 for addition and 1 for subtraction. The *carry-out* will propagate to the left-neighbor PE if the *programmable word boundary* (PWB) is not reached. Otherwise, the carry-out is stopped. The PWB is implemented by the Select register (S) and By-pass register (B), placed at every 8^{th} PE at the MSB position. We now present an overview of MPEG-2 video compression standard.

3. OVERVIEW OF MPEG-2 VIDEO COMPRESSION STANDARD

In the MPEG-2 video compression standard, a block-based ME/C is employed to remove the interframe correlation and DCT for the reduction of the intraframe correlation. Here, *a group of pictures* (GOP) approach is used instead of frame by frame coding. A GOP is typically a combination of one or two intra-coded frame (I), some predictive-coded frames (P), and the rest of bidirectional predictive-coded frames (B). The I frames are also used as a reference for P frames. The block diagram of MPEG-video encoder is shown in Fig. 3.

The I-frames are coded using 2-D DCT on 8x8 blocks. The resulting coefficients are then quantized using a pre-defined quantization table. The quantized DC and AC coefficients are coded using different techniques. The DC coefficients are differentially coded, while the AC coefficients are zig-zag scanned and run-length coded into *symbol-1* of {RUN, SIZE} pair, and *symbol-2* of {AMPLITUDE}. The resulting differences in DC and symbols for AC coefficients are variable-length coded and sent to the formatter for codeword packaging. In order to reconstruct the I- and P- frames, the quantized DC and AC coefficients are inverse-quantized, and inverse DCT is performed.

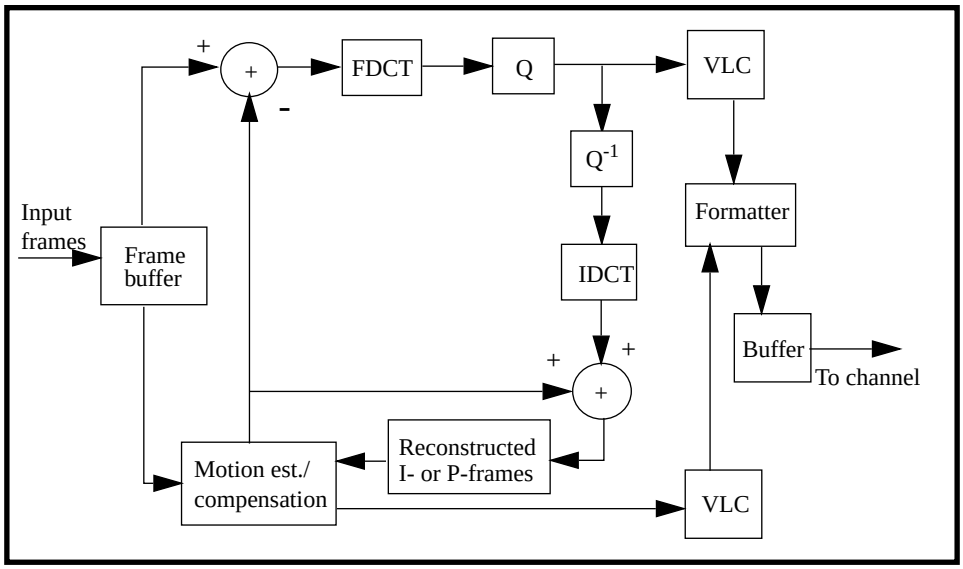

Figure 3. Block diagram of MPEG-2 video encoder.

The P- and B-frames are partitioned into 16x16 *macro-blocks*. We note that macro-blocks in a P-frame are motion-compensation predicted using the previously reconstructed I- (or P-) frame; whereas, those in B-frames are motion-compensation interpolated from the previously reconstructed I- (or P-) and P-frames. First, the current macro-block is searched in the corresponding search area (SA) of the reconstructed I- (or P-) frame, and the motion vector of a macro-block which results in the least distortion is then variable length coded. Similar to variable length codewords for DC and AC coefficients, the variable length codewords for motion vectors are also sent to the formatter for codeword packaging. The motion compensated difference frame (DFD) is partitioned into 8x8 blocks which then undergoes a 2-D DCT. The resulting DC and AC coefficients of the DFD are quantized similar to the I-frames.

4. C*RAM IMPLEMENTATION

4.1 C*RAM Configuration and Data Arrangement

We recall from section 2.1 that the 8192 1-bit PE's can be arranged into CU's of 1-bit or multiples of 8-bit. In our implementation, the PE's are grouped into 1024 CU's, each of 8-bit. These CU's are capable of performing: zero checking (using the PSB), ripple-carry addition (and subtraction), multiplication (by shifting and addition), and parallel search. These basic operations are subsequently used in macros such as: DCT computation, coefficient quantization, ME/C, differential coding, run-length coding, and VLC.

The MP/ML MPEG-2 specifications require processing of 576 x 720 pixels/frame at 30 frames per second (fps). Each frame is divided into 72 x 90 blocks (of size 8 x 8). In order to maximize C*RAM performance while maintaining simple control instructions, the frame is equally partitioned into 9 parts where each part occupies 90 CU's (Fig. 4). Therefore, each CU

accommodates a column of 8 blocks. Since the frame data is now partioned into separate portions, padding of partial frame data above and below the partion boundaries are required for the ME/C process.

4.2 Implementation of DCT and Coefficient Quantization

Consider the following 2-D DCT operation, where C is the DCT matrix:

$$V = C^t UC \tag{1}$$

In Eq.(1), DCT is obtained by multiplying U with C, and then multiplying the resulting matrix with C^t. The 2-D DCT matrix computation is often performed by row-column decomposition, where the input is read row by row. The output of the first DCT is UC. The transposer memory executes matrix transposition to yield C^tU^t. We note that matrix transposition can be performed by addressing the appropriate coefficients at the controller, without physically transposing the coefficients. Similarly, the output of the second DCT gives C^tU^tC, which is the transpose of V. The output of the 2-D DCT is then written column by column.

The classic DCT computation may involve $2n^3$ multiply-add operations[16]. In our implementation, the fast DCT algorithm by Chen *et al.*[2] has been used. For each n x n block, the number of multiplications and additions are $2*8*(n\log_2 n - 1.5n + 4)$ and $2*8*(1.5n\log_2 n - 1.5n + 2)$, respectively. For n=8, the number of multiplications and additions are 256 and 416, respectively.

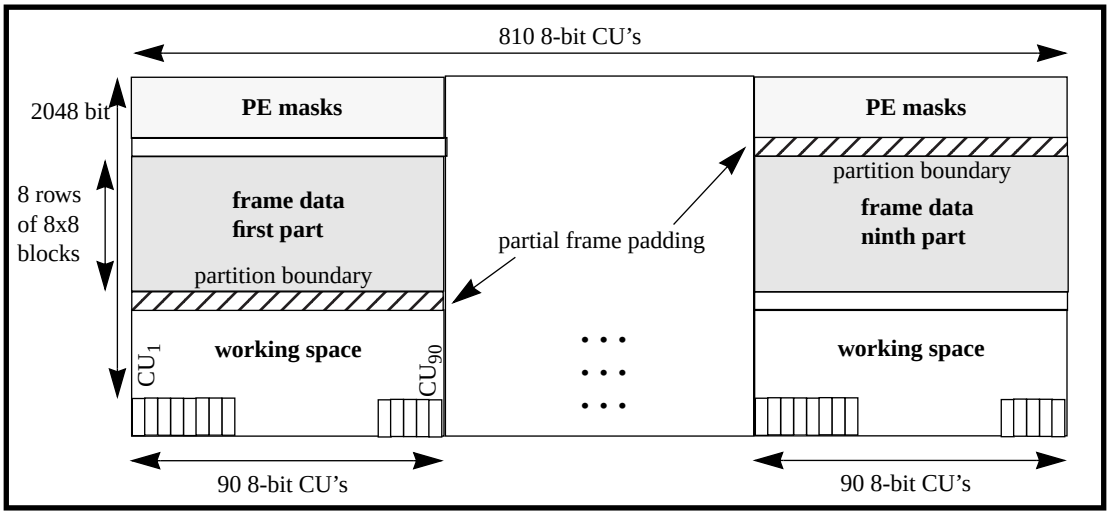

Figure 4. Frame data arrangement in the 64Mb C*RAM

DCT computation involves a sequence of multiplications and additions. Since the addition in C*RAM was already mentioned in section 2.2, only the multiplication is described here. We note that since the DCT coefficients (multipliers) are the same for all blocks (at a particular row address) and known in advance, they can be broadcasted from the controller in sequences of '0' and '1'. If the multiplier bit is a '1', multiplication is performed in parallel by loading the corresponding pixel at a CU, and shifting by a number of positions. If the multiplier bit is a '0', no action is required. The final results are obtained by adding all

the partial terms. The resulting coefficients are next quantized. Quantization is a division process where each coefficient is divided by different (or the same) number depending on the zig-zag scanned position of the coefficients. Division is performed by multiplying the coefficients with the inverse of a number, also broadcasted from the C*RAM controller.

4.3 Implementation of Motion Estimation / Compensation

The temporal redundancy of a sequence is often reduced by ME/C. In this technique, the macro-block is first searched within the pre-determined SA of the previously reconstructed I- (or P-) frame. We note that the best match macro-block results in the least distortion. The corresponding motion vector is then variable-length coded. Next, the difference between the best match macro-block and the candidate macro block is coded similar to those in the I-frame.

In order to avoid transfer of frame data, reference frames should be retained in the C*RAM until they are no longer referred. A typical GOP, where N (the distance between I-frames) is 9 and M (the distance between 2-P frames) is 3, is shown below:

$$I_1 \Rightarrow B_1 \Rightarrow B_2 \Rightarrow P_1 \Rightarrow B_3 \Rightarrow B_4 \Rightarrow P_2 \Rightarrow B_5 \Rightarrow B_6 \Rightarrow I_2$$

For proper processing, this GOP is to be reordered as follows:

$$I_1 \Rightarrow P_1 \Rightarrow B_1 \Rightarrow B_2 \Rightarrow P_2 \Rightarrow B_3 \Rightarrow B_4 \Rightarrow I_2 \Rightarrow B_5 \Rightarrow B_6$$

In this arrangement, I_1 is encoded first. Then P_1 is encoded based on reference frame I_1. Next B_1 is encoded based on reference frames I_1 and P_1. Similarly, B_2 is encoded based on reference frames I_1 and P_1. It is crucial that reconstructed frames of I_1 and P_1 remain in C*RAM until the encoding of B_2 is complete. After B_2 is encoded, I_1 is no longer required. It's hence replaced by P_2. P_2 is encoded based on reference frame P_1, while P_1 and P_2 are used as reference frames for encoding B_3 and B_4, and so on. We note that no more than 4 frames are required to be in the C*RAM at any instant in time.

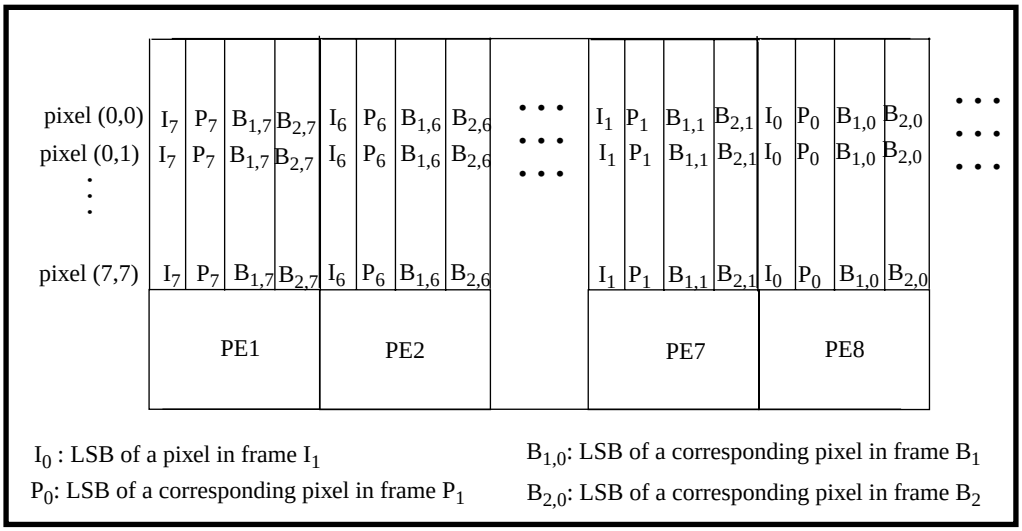

Figure 5. 8x8 block arrangement in a 8-bit CU

Additional savings in data shifting can be achieved if pixels at the same spatial location belonging to a sub-group of I-, P-, and B-frames are placed in the same CU as shown in Fig. 5. Since each CU sees all pixels of a column of 8 blocks, vertical shifts are performed by addressing the appropriate rows and hence, data shifting is avoided. When all pixels of a block are not in the same 8-bit CU, horizontal shifts via left-right interconnections are required.

The *mean absolute error* (MAE) is chosen as the distortion measure in determining the best match macro block. This distortion calculation involves pixel-wide subtraction, absolute operation on the difference, and accumulation of the absolute differences. For search ranges of $\pm h$ and $\pm v$, the *total number of distortion calculations* (D) for every macro-block is 16*16*(2h+1)*(2v+1). We note that the *full-search* (FS) method involves motion estimation of areas which might not be used for final motion vector determination.

The *3-step hierarchical search block matching* (TSS) algorithm[4], on the other hand, searches for the best motion vector in a coarse-to-fine manner. It has been claimed that the TSS has a speedup of up to 9 compared to the FS. In the TSS algorithm, 9 centers (corresponding to the 9 candidate locations) out of (2h+1)*(2v+1) are first selected and the search operations are performed. The center corresponding to the least distortion is selected and other centers are eliminated. The new SA corresponding to the selected center is formed by reducing the distance between the candidate locations by half. The procedure is repeated until the final motion vector is determined.

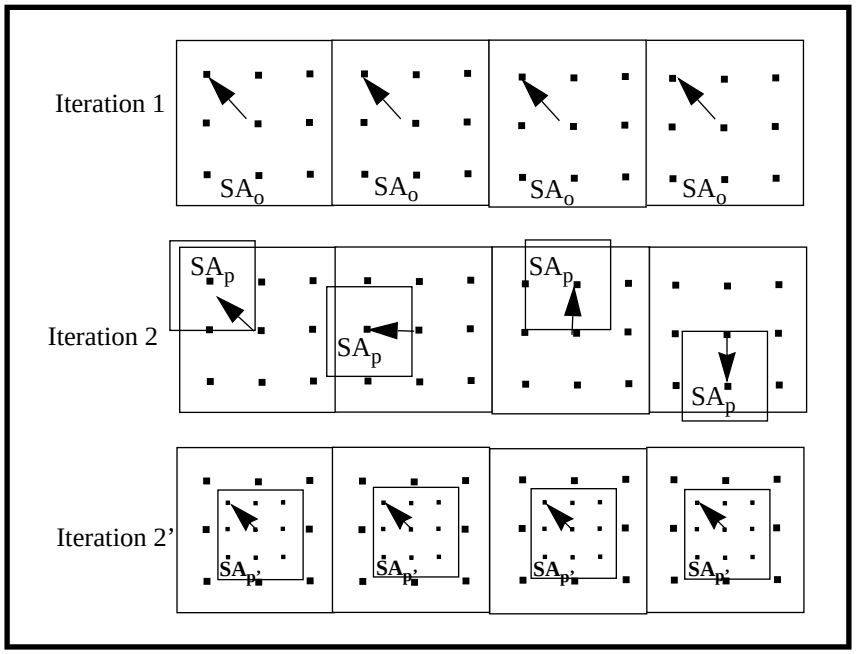

Figure 6. Reallocation of the new SA_p for SIMD-based operation

Compared to the FS method which can be efficiently mapped onto the SIMD architecture, the TSS algorithm employs all the SIMD operations only in the first iteration (Fig. 6). In the first iteration, all 9 candidate locations in SA_o participate in the

search operation. Therefore, the controller will check, one candidate location at a time, all SA_o's within a row in parallel. In the second iteration, each SA_o has already been narrowed down to a differently-centered SA_p (as shown in Iteration 2 of Fig. 6) which requires separate instructions. In order to map the TSS algorithm onto the SIMD architecture, the following procedure is used:

- Step 1: If the new SA_p is small, it should be reallocated to the center of the original SA_o where they can be searched using the SIMD instructions (shown as Iteration 2' in Fig. 6).
- Step 2: If the new SA_p is large, the amount of data shifting due to the above reallocation is not justified for the SIMD computation. Therefore, non-SIMD computation is used until the SA_p becomes small after which step 1 is applied.

In our implementation, the search range of (-16,+15) pixels in each direction has been chosen. Since we started with a reasonably small SA, step 1 is used throughout the ME/C process.

4.4 Implementation of VLC

In the VLC process, the quantized DC and AC coefficients, and motion vectors are losslessly compressed. The run-length coder represents consecutive zeros by their run lengths thus reducing the number of samples, while the VLC assigns shorter codewords to more probable source symbols so that the average number of bits for each source symbol is reduced. Implementation of the VLC process can be described in the following 3 stages:

4.4.1 Differential Coding of Quantized DC Coefficients

In Fig. 4, within a row of 90 CU's, the first quantized DC (in CU_1) is variable-length coded directly. Subsequent DC coefficients are differentially coded. This is performed by first shifting DC_i onto CU_{i+1}, then subtracting DC_i from DC_{i+1}. The DC coefficients at CU_{91} to CU_{180}, from CU_{181} to CU_{270}, etc., are similarly coded. The differences in DC values are variable-length coded as described in section 4.4.3.

4.4.2 Run-Length Coding of Zig-Zag Scanned, Quantized AC Coefficients

The next stage is to determine the runs of zeros from the zig-zag scanned AC coefficients. Recall that an advantage of C*RAM architecture is that the zig-zag scanned sequence and matrix transposition can be performed by manipulating the addresses (of coefficients). Hence, a significant amount of data shifting can be avoided.

Among the CU's, the number and order of zeros and non-zeros are not necessarily the same. However, in an SIMD architecture, at every instance, all the CU's have to execute the same instruction. The following algorithm describes the run-length coding process on a SIMD architecture:

- Step 1: Check for zeros using the PSB. This zero-checking operation results in group A with zero-value CU's, and group B with non-zero CU's.

- Step 2: The W registers of group A are masked out, and all the RUN counters and non-zero values are stored. Then, all the RUN counters are reset. The masking operation ensures that only those of group B are affected by the subsequent operations.
- Step 3: The W registers of group B are masked out, and all the RUN counters are incremented. Similar to the masking operation in step 2, only the RUN counters of group A are incremented, while those of group B are not affected.

We note that the RUN counters can be represented using 6-bit integers since there are 63 AC coefficients in each block, whereas the after-DCT coefficients can be represented using integers of maximum 11 bits.

4.4.3 Variable-Length Coding for Run-Length Coded Coefficients and Motion Vectors

This stage involves conversions of differential DC values, RUN counters and non-zero AC coefficients, and motion vectors to variable-length codeword numbers. Since the VLC process of differential DC coefficients and motion vectors are similar to those of RUN counter and non-zero AC coefficient pairs, they are not discussed here.

Recall from the previous section that the RUN counters and non-zero coefficients can be represented by 6 and 11 bits, respectively. The 2 MSB's of the RUN counter represent the number of 16-runs of zeros. Therefore, these 2 bits are checked and coded first. The remaining 4 bits of the RUN counter and 11 bits of non-zero coefficient are grouped together and are subjected to a search operation. A table of 15-bit search patterns along with the corresponding variable length codeword numbers is available at the controller. We note that many of the 15-bit search patterns are statistically improbable. Hence, they do not contribute to any delay in the coding time. The remaining 15-bit search patterns are progressively loaded onto the C*RAM and compared, in parallel, with those in the CU's. If a match is found, the corresponding variable-length codeword number is sent to the formatter, where a VLC lookup table is assumed available, for codeword packaging.

5. PERFOMANCE ANALYSIS

The analysis is based on the MP/ML MPEG-2 specifications which require processing of 576 x 720 pixels/frame at 30 fps. Table 1 shows the execution times in milliseconds (ms) for DCT, ME/C, and VLC functions. It can be seen that ME/C is the bottle neck of the encoding process. This is largely due to the amount of data shifting and the intensive computations involved.

Table 1. Execution times of DCT, ME/C, and VLC (ms)

t_{DCT}	$t_{ME/C}$	$t_{VLC\ (mv)*}$
2.7	14.8	4.0 (4.8)

* The numbers in brackets indicate the VLC with motion vectors encoding.

The execution times for the I-, P-, and B-frames are as follows:

$$t_{I\text{-frame}} = 2*t_{DCT} + t_{VLC}$$
$$t_{P\text{-frame}} = t_{ME/C} + t_{VLC(mv)} + 2*t_{DCT}$$
$$t_{B\text{-frame}} = 2*t_{ME/C} + t_{VLC(mv)} + t_{DCT}$$

We note here that P-frame coding requires only forward (or backward) prediction, and hence, only one ME/C operation is performed. However, B-frame encoding involves bi-directional interpolation, and thus, requires 2 ME/C operations. We also note that the I- (or P-) frame performs DCT both on the frame data (or DFD) and for the reconstructed frame, therefore, requires 2 DCT operations.

Table 2. Execution times for I-, P-, and B-frames (ms)

I-frame	P-frame	B-frame
9.4	25.1	37.2

Table 3 shows the average frame execution time with different GOP characteristics. We recall from section 4.3 that N is the distance between 2 I-frames (or size of GOP), and M is the distance between two P-frames. GOP's of N= 9, 12, 15, 18, and 21; and M = 1, and 3 were investigated.

Table 3. Average frame execution time for different GOP sizes (ms)

	N=9	N=12	N=15	N=18	N=21
M=1	23.4	23.8	24.0	24.2	24.4
M=3	31.4	31.9	32.1	32.2	32.4

It can be seen from the simulation results in Table 3 that C*RAM is capable of implementing MPEG-2 video compression algorithm for a wide range of GOP types and sizes in real-time.

6. CONCLUSIONS

In this paper, we have presented a C*RAM implementation of the MPEG-2 algorithm for compression of MP/ML quality video. C*RAM has the advantages of processing image/video data in parallel and directly in the frame buffers. Therefore, savings in execution time and I/O bandwidth due to massively parallel on-chip computation and the reduction in data transfer among chips is achieved. As a result, the DCT, ME/C, and VLC functions required by the video encoder loop have successfully been implemented in real-time on a programmable 64Mb DRAM-based C*RAM.

ACKNOWLEDGEMENT

The authors wish to thank Ousama Hage and Peter Nyasulu for their useful comments.

REFERENCES

[1] ISO/IEC 13818-2, Draft International Standard: Generic Coding of Moving Picture and Associated Audio, March 1994.

[2] W. H. Chen, C. H. Smith, and S. C. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Transform", *IEEE Trans. on Commun.*, vol. COM-25, pp.1004-1009, Sep. 1977.

[3] B. G. Lee, "A New Algorithm to Compute Discrete Cosine Transform", *IEEE Trans. on Acoustic, Speech, and Signal Processing*, vol. ASSP-32, no.6, pp.1243-1245, Dec. 1984.

[4] H. M. Jong, L. G. Chen, and T. D. Chiueh, "Parallel Architectures for 3-Step Hierarchical Search Block-Matching Algorithm", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 4, No. 4, Aug. 1994.

[5] M. L. Liou, "Visual Telephony as an ISDN Application", *IEEE Communications Magazine*, vol 28(2), pp.30-38, Feb. 1990

[6] M. T. Sun, T. C. Chen, and A. M. Gottleib, "VLSI Implementation of a 16x16 Discrete Cosine Transform", *IEEE Trans. on Circuits and Systems*, vol. CAS-36, no.4, pp.610-617, Apr. 1989.

[7] L. De Vos and M. Schobinger, "VLSI Architecture for a Flexible Block Matching Processor", *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 5, no.5, pp.417-428, Oct. 1995.

[8] S. M. Lei and M. T. Sun, "An Entropy Coding System for Digital HDTV Application", *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 1, no. 1, pp.147-155, Mar. 1996.

[9] G.L. Chen, J.S. Pan, and J.L. Wang, "Video Encoder Architecture for MPEG-2 Real Time Encoding", *IEEE Trans. on Consumer Electronics*, vol.42, no.3, pp.290-299, Aug. 1996

[10] D. G. Elliott, M. Snelgrove, and M. Stumm, "Computational-RAM: A Memory-SIMD Hybrid and Its Applications to DSP", *IEEE Custom Integrated Circuits Conference*, pp.30.6.1-30.6.4, Boston, May 1992.

[11] C. Cojocaru, *Computational-RAM: Implementation and Bit-Parallel Architecture*, Master's dissertation, Department of Electronics, Carleton University, Dec. 1994.

[12] Thinh M. Le, S. Panchanathan, and W. M. Snelgrove, "Computational-RAM Implementation of Vector Quantization for Image Compression", *Proceedings of the IEEE Workshop on Visual Signal Processing and Communications '94*, pp.157-162, Sep. 1994.

[13] Thinh M. Le and S. Panchanathan, "Computational-RAM Implementation of an Adaptive Vector Quantization Algorithm for Video Compression", *IEEE Trans. on Consumer Electronics*, vol.41, no.3, pp.738-747, Aug. 1995.

[14] Thinh M. Le, S. Panchanathan, and W. M. Snelgrove, "Computational-RAM Implementation of Mean-Average Scalable Vector Quantization for Real-Time Progressive Image Transmission", *Proceedings of the 1996 Canadian Conference on Electrical and Computer Engineering*, vol.1, pp.442-445, May 1996.

[15] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: the Hardware / Software Interface*, pp.22, 1994.

[16] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, 1993.