KEY COMPONENTS IN THE DESIGN OF IMAGE AND VIDEO COMPRESSION ICs

Konstantinos Konstantinides

Hewlett-Packard Laboratories
Computer Peripherals Laboratory
Imaging Peripherals Department

Abstract

There is a new generation of digital signal processors for image and video compression and decompression. Regardless of their complexity, most of the image and video compression architectures share three major components: an accelerator for computing two-dimensional DCTs and IDCTs, a motion estimator, and a variable length coder and decoder. This paper presents a general overview of some of the most common designs and architectures for the hardware implementation of these key components in image compression ICs.

---

† This paper will be presented at the 1994 DSPx Exposition and Symposium, San Francisco, CA, June 1994.

1. Introduction

Recently, a new type of a digital signal processor (DSP) has emerged: the image and video compression processor. Image compression is one of the core technologies in video teleconferencing, digital television, and a variety of CD-ROM based multimedia applications. The establishment of industry accepted standards in image, video, and audio compression, and the opportunity to attract new markets allow now IC manufacturers to invest in the design and development of this new generation of DSPs.

There are three main standards in image and video compression:

- JPEG (Joint Photographic Experts Group) for the compression of still images [1].
- Px64 (also known as CCITT Recommendation H.261) for video tele-conferencing [2].
- MPEG (Moving Picture Experts Group) for motion video and audio for use in computer systems (MPEG-1 [3]) and broadcast applications (MPEG-2 [4]).

All of these compression standards have a common set of core functions, namely a spatial-to-frequency domain transformation, quantization of frequency-domain signals, and entropy-coding of the quantized data. For compression of motion sequences, additional processing in the temporal domain, such as motion compensation, is also employed. The main processing

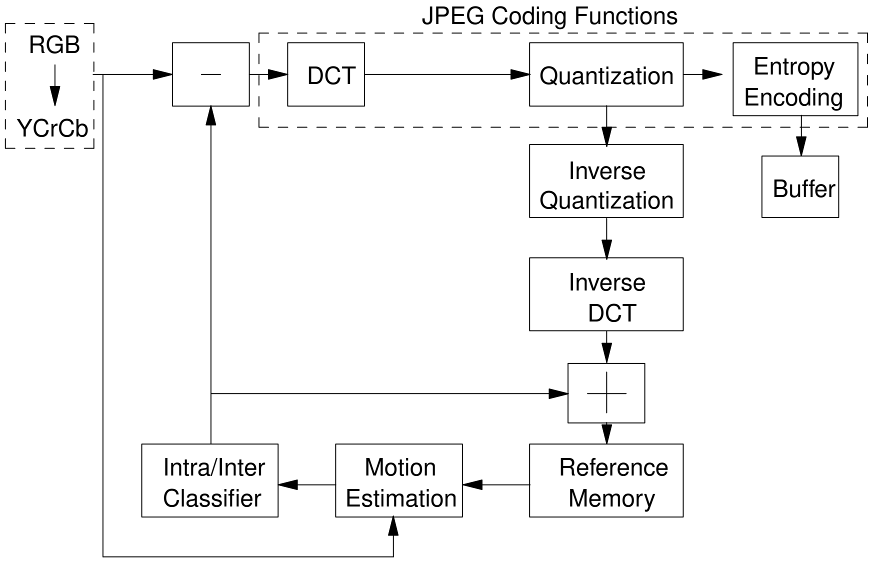

Fig. 1 : Processing flow in JPEG, MPEG, and Px64 coding schemes.

flow for JPEG, MPEG, and Px64 is depicted in Fig. 1 [5]. Spatial-to-frequency domain transformation is performed via an 8x8 2-D DCT (Discrete Cosine Transform) on the spatial domain data. After quantization, the quantized DCT coefficients are further compressed using a lossless compression scheme, such as Huffman coding. For motion sequences, higher compression can be achieved using motion compensation. Frame-to-frame changes are determined and only the differences among frames are encoded and transmitted.

Conventional DSPs cannot meet the computational requirements for real-time motion video compression and decompression. Hence, a new generation of image compression ICs has been developed. Most of the image compression ICs use a RISC (Reduced Instruction-Set Computer) or DSP core, for general purpose computing, DMA ports for fast I/O, and special assist units for the three most compute-intensive operations: the DCT, motion estimation, and variable length coding. In this paper we will present a general overview of the designs used in image compression ICs for the computation of these core functions.

In Section 2 we describe the implementation of a 2-D DCT based on distributed arithmetic. In Section 3 we present a systolic architecture for motion estimation, and in Section 4 we present a hardware implementation for variable-length coding and decoding.

2. Implementation of the 2-D DCT

Let x denote an NxN block of data. Then, the 2-D DCT of x is defined as

$$X = C^t \; x \; C \quad , \tag{1}$$

where C is a coefficient matrix defined as

$$c_{ij} = \sqrt{\frac{2}{N}} \cos\left[\frac{(2i-1)(j-1)\pi}{2N}\right], \; i = 1, 2, \ldots, N \;, \quad j = 2, 3, \ldots, N \;, \tag{2}$$

$c_{ij} = \dfrac{1}{\sqrt{N}}$ for j=1, and C^t is the transpose of C [6].

Because of the wide use of the 2-D DCT in many image processing applications, there is extensive literature on its hardware implementation [7]. There are two general classes of implementations: those using conventional matrix analysis and those that use number theoretic or polynomial transforms (i.e., the Winograd transform). Polynomial transforms require the least number of multiplications. However, they require complex control and are not easy to implement. Techniques based on conventional matrix analysis are the most popular since they have a regular structure.

From (1), the 2-D DCT can be rewritten in terms of two 1-D DCT transforms, as follows:

$$y = x^t C \;, \qquad X = y^t C \quad . \tag{3}$$

From (3), an NxN 2-D DCT can be computed by a) performing N N-point 1-D DCTs of the rows, b) transposing the output matrix (y), and c) performing another N N-point 1-D DCTs. This row-column transformation allows a 2-D DCT to be computed using fast algorithms and hardware developed for 1-D DCTs.

Hardware implementations of the 1-D DCT range in complexity from simple multiplier-free designs to multi-processor systolic arrays. In choosing the implementation for an image compression IC one has to also take into account that the DCT is only a small part of the computational pipeline in an image compression algorithm. Hence, smaller, but slower, designs may be preferable over faster, but larger ones. Recent image compression ICs use two main techniques for the evaluation of the DCT: a) a multiplier-free processor array based on distributed arithmetic or b) a four-processor multiply-accumulate array.

2.1 8x8 DCT Using Distributed Arithmetic

Distributed arithmetic for the design of DSP circuitry was first introduced by Peled and Liu in the early 70's [8]. In those days, the level of circuit integration was very low and an 8x8 parallel multiplier would require multiple ICs. The main idea of distributed arithmetic is that in digital signal processing (i.e., in a digital filter) most multiplications are performed with known constants. Hence, intermediate product results can be pre-computed and stored in memory. By using bit-serial arithmetic, these operations can be computed using look-up tables, additions, and shifts.

Advances in VLSI now allow multiple parallel multipliers on a single chip. However, an image compression IC has to perform numerous other operations besides the DCT. As will be seen, distributed arithmetic still provides a very good alternative for the hardware implementation of the DCT.

From (3),

$$y(k,l) = x(k)^t \ c(l) = \sum_{m=1}^{N} c(m,l) \ x(k,m) \ , \tag{4}$$

where $x(k)^t$ is the k-th row vector of x and $c(l)$ is the l-th column vector of C. Let

$$x(k,m) = -\, x(k,m)^0 + \sum_{j=1}^{B-1} x(k,m)^j \ 2^{-j} \ \ , \tag{5}$$

denote the two's-complement binary representation of $x(k,m)$ using B-bits of precision. $x(k,m)^j$ is the j-th bit of $x(k,m)$ and it can be either 0 or 1. We also assume, without loss of generality, that the decimal point is immediately after the most significant bit, hence, $x(k,m)$ can only take values between -1 and 1. Substituting (5) into (4), yields

$$y(k,l) = -\, F_{k,l}(c(l), x(k)^0) \qquad + \sum_{j=1}^{B-1} F_{k,l}(c(l),\ x(k)^j) \ 2^{-j} \ , \tag{6}$$

where

$$F_{k,l}(c(l), x(k)^j) = \sum_{m=1}^{N} c(m,l) \ x(k,m)^j \ , \ \ j = 0, 1, \ldots, B-1. \tag{7}$$

From (4)-(7), we can make the following observations: a) For each vector $x(k)$, we can concurrently generate N values of $y(k,l)$ ($y(k,1)$ to $y(k,N)$). b) All $F_{k,l}$ are functions of $c(l)$ and bit-patterns of the input data. Hence, they can be precomputed for all 2^N possible bit-patterns and stored in memory. c) Evaluation of $y(k,l)$ requires only adds, shifts, and table look-ups. Using bit-serial arithmetic for the input data, a $y(k,l)$ value can be computed in B cycles.

Fig. 2 shows a block diagram for the implementation of an 8-point DCT using distributed arithmetic. The design uses eight ROM accumulation units (RACs) for the concurrent computation of eight $y(k,l)$ vectors. Input is done bit-serially through the input delay line. After B cycles (B can range from 8 to 16), each RAC unit loads (in parallel bit format) its output to the output delay-line, and results can be shifted-out bit-serially again.

In practice, additional schemes may be used to reduce the size of the ROM tables or speed-up the computations in the RAC array [6]. Furthermore, distributed arithmetic can also be used

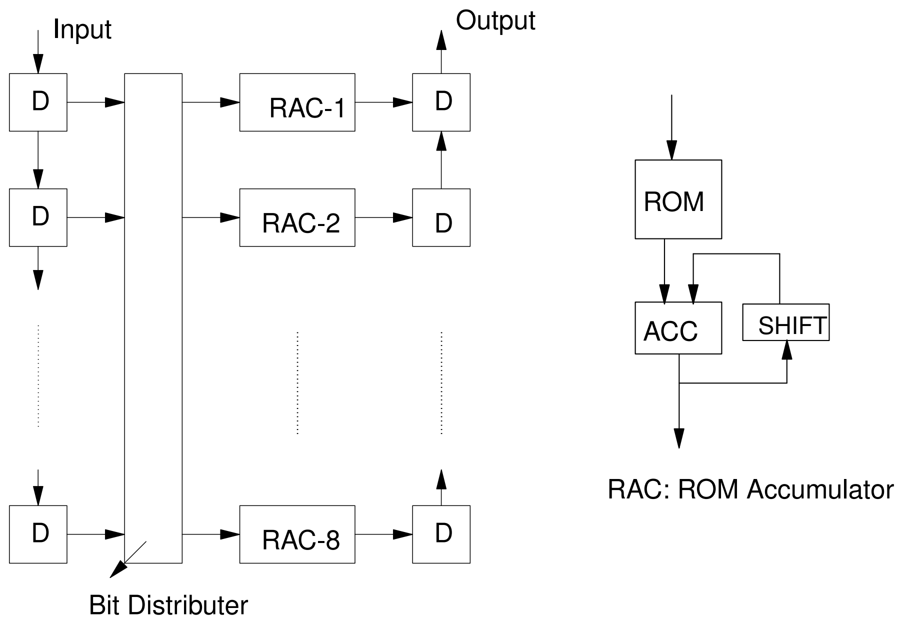

Fig. 2: Block diagram of an 8x8 DCT processor using distributed arithmetic.

on butterfly-based DCT algorithms [7]. An example of an image codec that uses a distributed implementation of the DCT is presented by Aono et al. in [9].

2.2 8x8 DCT Using Parallel Multipliers

A number of image compression ICs incorporate multiple on-chip multiply-accumulate units (usually four) [5]. These are used for computing the DCT, data format conversions, quantization, and other operations. The number of on-chip multiplier-accumulate units is heavily dependent on the technology and chip-size constraints. However, four-level parallelism is efficient in many algorithms used in data compression, and especially for the fast implementation of the 8-point DCT.

Let $u(k, m) = x(k, m) + x(k, N - m + 1)$ and $v(k, m) = x(k, m) - x(k, N - m + 1)$, then

$$y(k, l) = u(k)^t c(l), \quad l = 1, 3, \ldots, N - 1 \tag{8a}$$
$$y(k, l) = v(k)^t c(l), \quad l = 2, 4, \ldots, N \tag{8b}$$

Equation (8) represents the first stage of a decimation in time N-point DCT. From (8), after some preprocessing of the input data (evaluating u and v), an 8-point DCT can be computed using two 4-point DCTs. By definition of the 4-point DCT, even brute-force evaluation does not require more than four multiply-accumulate operations per output sample. Hence, by executing these four operations in parallel, an output sample can be available in each cycle. In practice of course, one can also use fast DCT algorithms that require fewer multiply-accumulate operations.

3. Implementation of Motion Estimators

In block-based motion compensated prediction, a frame is divided into blocks. For each block (called the *reference* block) we search the previous frame for the block that best matches the reference block. This process is commonly referred to as motion estimation. The search is done in a rectangular area (called *search window*) around the position of the reference block. Typical search windows are 16x16 pixels wider than the reference window (commonly denoted as a [-8,7] search).

Let $x(k, l)$ denote a pixel from the reference block and $y(k, l)$ denote a pixel from the search block. Under the Mean Absolute Error Criterion, the distortion between two MxN blocks is defined as

$$D(i, j) = \sum_{m=1}^{M} \sum_{n=1}^{N} \left| x(m, n) - y(m+i, n+j) \right| , \tag{9}$$

where (i, j) denotes the position of y (or candidate) block relative to the reference block. From (9), for 8x8 blocks, evaluating $D(i, j)$ requires 64x2 loads, 64x3 math operations (one subtract, one absolute value, and one add), and one store, for a total of 321 operations per block. A **Y** (Luminance) CIF (352x288) frame has 396 16x16 blocks. Hence, at 30 frames per second, a full [-8,7] motion estimation search (256 total distortion values) requires 396 x 321 x 256 x 30 = 976 million operations per second (MOPS). These numbers do not even include the operations to find the minimum among all the distortion values.

Because of the enormous computation required for motion estimation, all video compression ICs use special (internal or external) motion estimation units. Since (9) has no multiply operations, a common approach is to design a systolic array, where each element of the systolic array evaluates the operation $a + |b - c|$. For example, Fig. 3 shows a block diagram of a motion estimator IC (STI3220) provided by SGS-Thomson [10].

The design has an 128-element systolic array plus a special unit for computing the minimum distortion. Each element of the systolic array has memory to compute two distortion values. For efficient I/O, this IC has four input ports: three for the search window and one for the reference block.

However, the number of elements in the array can be significantly reduced if one takes into account that each pixel of the search window is used in computing distortion values in multiple sub-windows. For example, for a 16x16 reference block, and a [-8,7] search window, each pixel of the search window is used in the computation of 16 distortion values [11]. Yang et al. [11] showed that in this case only 16 processors are needed for 100% processor utilization. Fig. 4 shows a block diagram of the motion estimator design proposed by Yang et al. The x input provides the reference block, while the $y1$ and $y2$ inputs provide pixels from the search window. For example, at time 16, $x = x(1, 0)$, $y1 = y(1, 0)$ and $y2 = y(0, 16)$. PE-0 computes $x(1, 0) - y(1, 0)$, PE-2 computes $x(0, 15) - y(0, 16)$, and PE-15 computes $x(0, 1) - y(0. 16)$. All data input is completed in 16x16+15 cycles. The data input operation is followed by the comparison operation, where a tree-based comparator computes the minimum distortion value. The processor design (PE-0 to PE-15) is the same with the one shown in Fig. 3 for the SGS-Thomson IC.

This architecture provides pel-level motion-vector accuracy. For subpel-level accuracy, multiple units can be cascaded and operate in parallel [12].

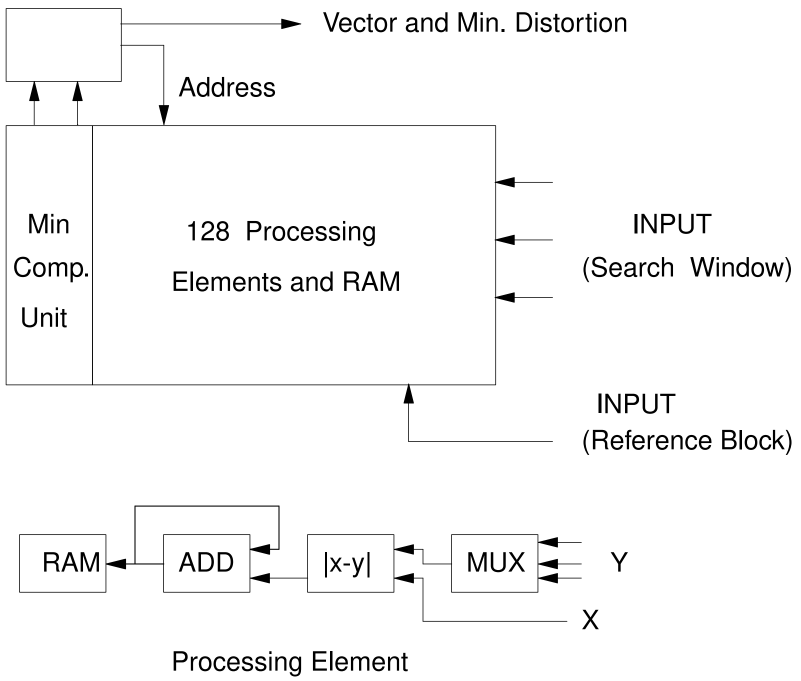

Fig. 3: Block diagram of the SGS-Thomson motion estimator IC.

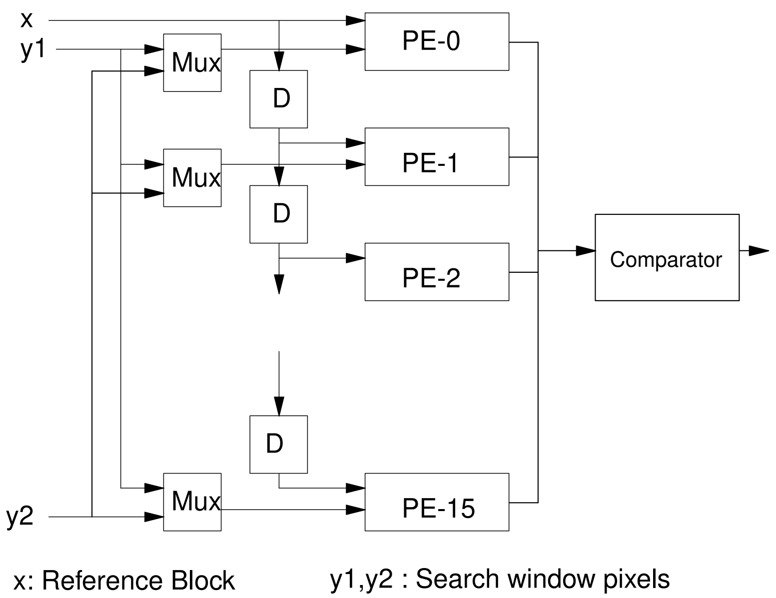

Fig. 4: Block diagram of a 16-processor motion estimator.

4. Hardware for Variable Length Coding

Variable length coding (VLC) and run length encoding (RLE) are used in the image compression standards for the lossless compression of the quantized DCT coefficients. Variable length coders (like Huffman) assign short codewords to input symbols of high probability.

Coding is usually done via look-up tables that map an input source symbol to a codeword. Decoding can be done by tracing bit-serially a decoding tree. Fig. 5 shows a simple coding

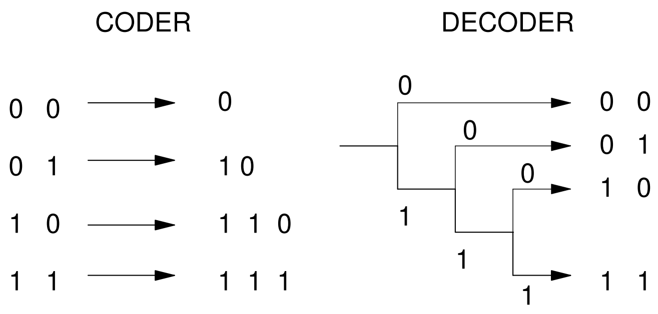

Fig. 5: Example for VLC coding table and decoding tree.

table and the corresponding decoding tree for 2-bit data symbols.

4.1 The Entropy Coder

The task of the VLC encoder is to map the input data into codewords of variable-length, concatenate them together, and segment them into 16-bit output words. Recently, Lei et al. [13] proposed a hardware implementation that uses parallel operations to perform these operations

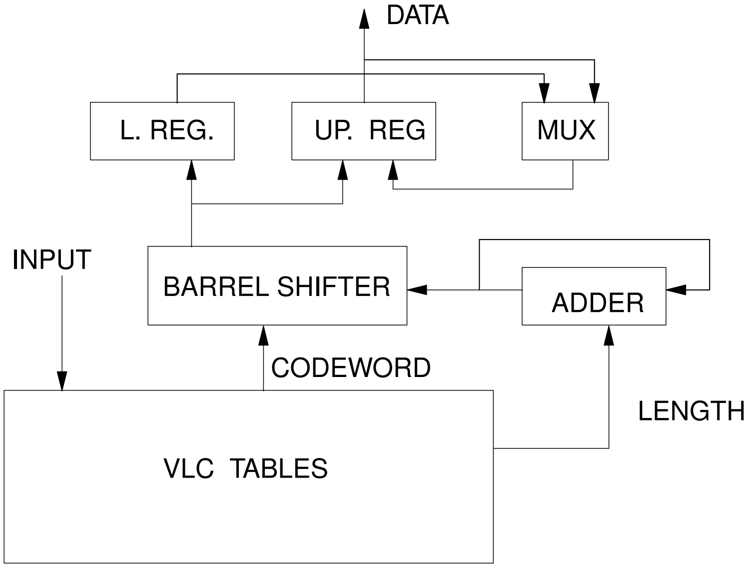

Fig. 6: Block diagram of a VLC encoder.

in one cycle. Fig. 6 shows a simplified block diagram of their design.

The VLC tables map input data into variable-length codewords. Codewords are concatenated in the upper register into 16-bit words using a shift-and-or operation. A four-bit adder keeps track of the length of the concatenated codeword and controls the shifts of the barrel-shifter. When the value of the accumulator is bigger than 15, the upper register outputs its content and the lower register is shifted to the upper register.

Fig. 7 shows an example of the operation of the VLC coder for a given codebook and the

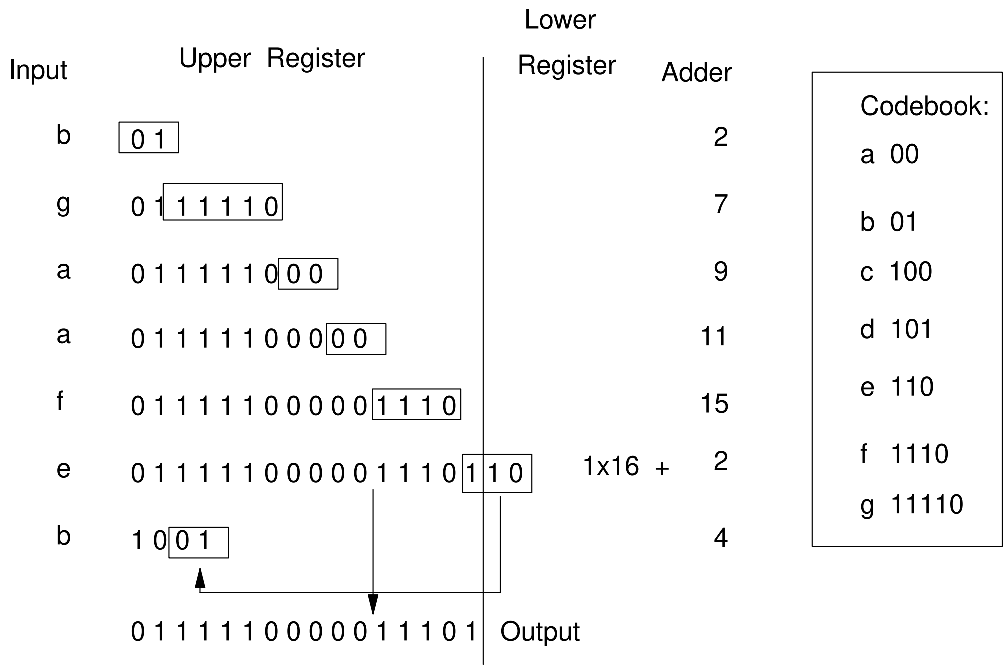

Fig. 7: Example of operation of the VLC encoder.

sequence: "b", "g", ..., "b". After the first input, "b", the adder is incremented by two, the length of the corresponding codeword output (01). The next output from the VLC table, 11110, is shifted by two bits and is appended to the prior codeword. The adder is incremented by five. After the input symbol "e", the adder overflows. The carry-out bit of the adder triggers an output of the data stored in the upper register and the content of the lower register (10) is loaded into the upper register. Then, operation continues as before.

4.2 Design of a VLC Decoder

A parallel VLC decoder is very similar to the encoder. As shown in Fig. 8, it consists again of VLC tables, a barrel shifter, two data registers (upper and lower), and a 4-bit adder. The adder keeps now track of the accumulated decoded length and shifts appropriately the combined output of the two registers. When a codeword is matched, the VLC table outputs the decoded symbol and its length. When the 4-bit adder overflows, the content of the lower register is transferred to the upper register and a new 16-bit segment is input to the lower register. This architecture allows us to decode one codeword per cycle, regardless of its length. Fig. 9 shows an example of the operation of the VLC decoder for the same codebook and encoded sequence we described in Fig. 7. The rectangle shows the position of the barrel-shifter. Starting from top, a search for a match of the upper register with a codeword in the codebook yields a match with "01" which corresponds to "b". The adder is incremented by two, the "window" of the barrel shifter is shifted by two positions (the length of "b"), and a new search yields a match with "g". The procedure is repeated until the adder overflows (after "e"). Then the lower register is shifted to the upper register and a new input is loaded into the lower register.

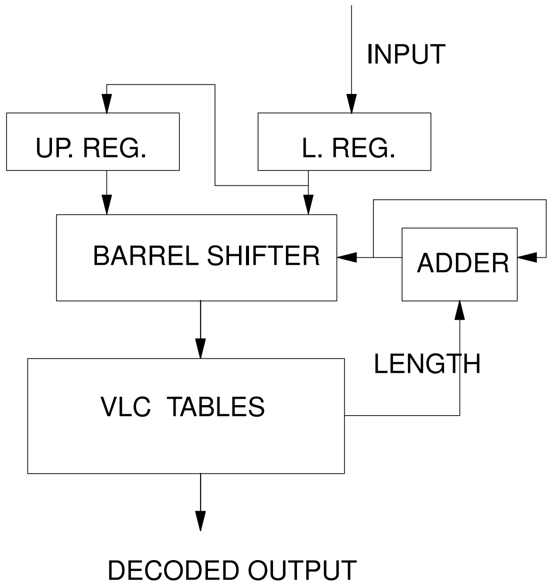

Fig. 8: Block diagram of a VLC decoder.

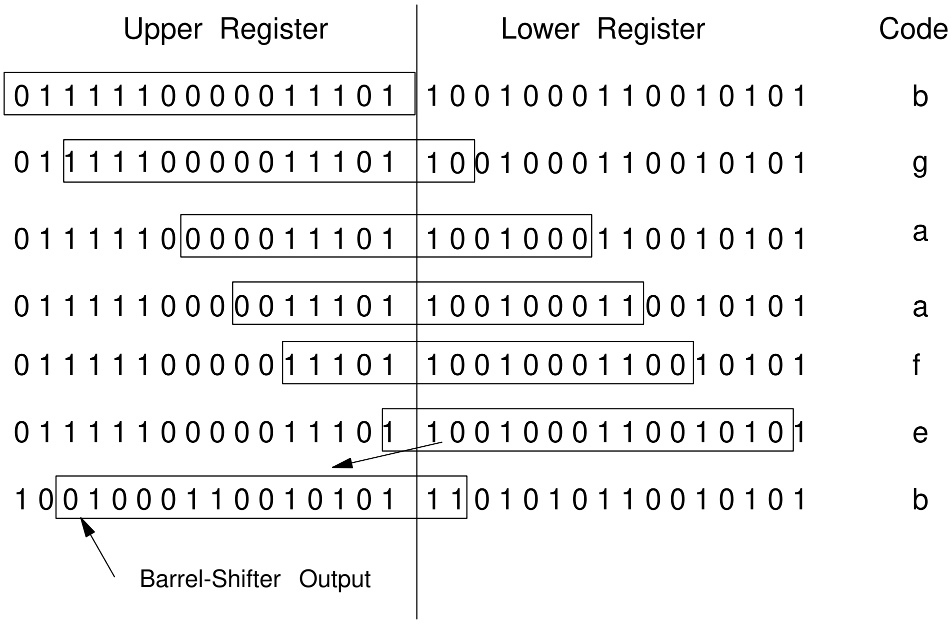

Fig. 9: Example of operation of the VLC decoder.

5. Summary and Conclusions

Compression standards like JPEG, Px64, and MPEG share a set of common and computationally intensive functions, namely, the 2-D DCT, motion estimation, and variable length coding. For the 2-D DCT, most video processors use row column transformations, and either a multiplier-free architecture based on distributed arithmetic, or a four-processor multiplier array. The computational requirements of motion estimation can only be met by systolic-type architectures. For an [-8, 7] search, a 16-processor array achieves 100% utilization of its processors and provides computational power for real-time motion estimation. Finally, for variable length coding, a simple architecture with VLC tables, a barrel shifter, two main registers, and a 4-bit adder can be used for VLC coding and decoding at a rate of one codeword per cycle, regardless of the length of the codewords.

REFERENCES

1. JPEG-1 DIS Working Group 10, Draft International Standard, DIS 10918-1, CCITT Rec. T.81, Jan. 2, 1992.
2. Video Codec for Audiovisual Services at p x 64 Kbits/s. CCITT Recommendation H.261, CDM XV-R 37-E, International Telegraph and Telephone Consultive Committee (CCITT), Aug. 1990.
3. MPEG-1 CD, Working Group 11, Committee Draft ISO/IEC 11172, Intern. Standards Organization, IPSJ, Tokyo, Dec. 6, 1991.
4. MPEG 2, *Generic Coding of Moving Pictures and Associated Audio,* ISO/IEC 13818-1,2,3.
5. K. Konstantinides and V. Bhaskaran, "Monolithic architectures for image processing and compression," IEEE Trans. on CG&A, Nov. 1992, pp. 75-86.
6. M. T. Sun, L. Wu and M.L. Liou, "A concurrent architecture for VLSI implementation of discrete cosine transform," IEEE Trans. on Circuits and Systems, Vol. CAS-34, no. 8, Aug. 87, pp. 992-994.
7. S. Wolter, D. Birreck, and R. Laur, "Classification for 2D-DCTs and a new architecture with distributed arithmetic," 1991 Intern. Symp. on Circuits and Systems, June 1991, Singapore, pp. 2204-7.
8. A. Peled and B. Liu, "A new hardware realization of digital filter," IEEE Trans. on ASSP, Vol. ASSP-22, No. 6, Dec. 74, pp. 456-462.
9. K. Aono et al., "A video digital signal processor with a vector-pipeline architecture," IEEE J. Solid State Circuits, Vol. 27, No. 12, Dec. 92, pp. 1886-1894.
10. O. Colavin, A. Artieri, J-F. Naviner, and R. Pacalet, "A dedicated circuit for real time motion estimation," Proc. Euro-ASIC '91, pp. 96-99, Paris, France, 1991.
11. K-M Yang, M-T Sun, and L. Wu, "A family of VLSI designs for the motion compensation block-matching algorithm," IEEE Trans. on Circuits and Systems, Vol. 36, No. 10, Oct. 89, pp. 1317-1325.
12. I. Tamitani et al., "An encoder decoder chip set for the MPEG video standard," Proc. IEEE ICASSP-92, pp. V-661-664, 1992.
13. S-M. Lei et al., "VLSI implementation of an entropy coder and decoder for advanced TV applications," 1990 IEEE Symp. on Circuits and Systems, Vol. 4, pp. 3030-3, 1990.