

AN ABSTRACT OF THE THESIS OF


Haowei Zhang for the degree of Master of Science in
Electrical and Computer Engineering presented on January 14, 1997.

Title:

A High-Performance, Low-Power and Memory-Efficient VLD for MPEG Applications.

Redacted for Privacy

Abstract approved: _____


Mario E. Magana

An extremely important area that has enabled or will enable many of the digital video services and applications such as VideoCD, DVD, DVC, HDTV, video conferencing, and DSS is digital video compression. The great success of digital video compression is mainly because of two factors. The state of the art in very large scale integrated circuit (VLSI) and a considerable body of knowledge accumulated over the last several decades in applying video compression algorithms such as discrete cosine transform (DCT), motion estimation (ME), motion compensation (MC) and entropy coding techniques. The MPEG (Moving Pictures Expert Group) standard reflects the second factor. In this thesis, MPEG standards are discussed thoroughly and interpreted, and a VLSI chip implementation (CMOS 0.35 μ technology and 3 layer metal) of a variable length decoder (VLD) for MPEG applications is developed. The VLD developed here achieves high performance by using a parallel and pipeline architecture. Furthermore, MPEG bitstream patterns are carefully analyzed in order to drastically improve VLD memory efficiency. Finally, a special clock scheme is applied to reduce the chip's power consumption.

A High-Performance, Low-Power and Memory-Efficient VLD for MPEG
Applications

by

Haowei Zhang

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed January 14, 1997
Commencement June 1998

Master of Science thesis of Haowei Zhang presented on January 14, 1997

APPROVED:

Redacted for Privacy

Major Professor, representing Electrical and Computer Engineering

Redacted for Privacy

Head of Department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Haowei Zhang, Author

TABLE OF CONTENTS

	<u>Page</u>
1 INTRODUCTION	1
1.1 Motivation	1
1.2 Outline of the thesis	3
2 STILL IMAGE COMPRESSION	5
2.1 Introduction	5
2.1.1 Color Spaces	5
2.1.2 Human Visual System	7
2.1.3 DCT and Fast DCT	8
2.2 Image Compression Systems	11
2.2.1 Introduction to JPEG Standard	11
2.2.2 Quantization	12
2.2.3 Entropy Coding	15
2.3 Operation Modes	17
3 LOSSY VIDEO COMPRESSION	22
3.1 Introduction	22
3.2 Motion Estimation	22
3.3 Motion Compensation	30
4 MPEG STANDARDS	36
4.1 Background	36
4.2 Digital Video Formats	39
4.3 Temporal processing	41

TABLE OF CONTENTS (Continued)

	<u>Page</u>
4.4 Encoder and Decoder	43
4.5 Bit Stream Structure	45
4.6 Macroblock Coding	48
4.7 Scalable Bit Streams	50
5 VLSI ARCHITECTURE FOR VIDEO COMPRESSION	53
5.1 Overview	53
5.2 Architectures for Motion Estimation	55
5.3 Mapping Algorithms to VLSI	62
5.4 Low Power Design	68
6 VLSI IMPLEMENTATION OF VARIABLE LENGTH DECODER	73
6.1 Overview	73
6.2 System Model	73
6.3 VLSI Architectures	76
6.4 VLSI Implementation	80
7 CONCLUSIONS	88
BIBLIOGRAPHY	90
APPENDICES	94
APPENDIX A VLD Layout	95

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
2.1 JPEG Encoder Decoder	19
2.2 Entropy Coding	20
2.3 Zig-zag scan of the AC Coefficients	20
2.4 Three-level Hierarchical Coder	21
3.1 Temporal Correlation Between Pictures	23
3.2 Block Matching Motion Estimation	25
3.3 Hierarchical Motion Estimation	29
3.4 Half-Pel accurate MV estimation	31
3.5 Field Image	32
3.6 Dual Prime for ME	35
4.1 I B P frame in a Video Sequence	41
4.2 Bidirectional Motion Compensation	42
4.3 MPEG Encoder (P Pictures)	44
4.4 MPEG Decoder	45
4.5 Bitstream Layer of MPEG-1	46
4.6 I Frame Macroblock Coding	49
4.7 B Frame Macroblock Coding	50
4.8 MPEG-2 Profiles	51
5.1 Data-flow Design	57
5.2 Architecture for ME using Linear Array	57
5.3 Architecture 2 for ME using Linear Array	58
5.4 Architecture for ME using 2-D Linear Array	60
5.5 New Architecture for ME	62
5.6 Task Distribution and Function Distribution	64
5.7 Hybrid Architecture	65

LIST OF FIGURES (Continued)

<u>Figure</u>		<u>Page</u>
5.8	Chip Package Interface	67
5.9	Voltage Swing Reduction Circuit	72
6.1	Simple Huffman Coding example	74
6.2	Block Layer Pseudocode	75
6.3	System Model Pseudocode	82
6.4	PLA-based constant-output rate VLD	83
6.5	Barrel Shifter and Control Logic	84
6.6	Architecture of VLD	85
6.7	Architecture of VLD (Contd)	86
6.8	VLD Pipeline Clock	86
6.9	Verilog Code for Start Code Length Decoder	87

LIST OF TABLES

<u>Table</u>		<u>Page</u>
4.1	MPEG-2 Profiles Levels	52
5.1	Basic Data Flow for ME	59
6.1	Variable Length Codes for Motion-code	79

A HIGH-PERFORMANCE, LOW-POWER AND MEMORY-EFFICIENT VLD FOR MPEG APPLICATIONS

1. INTRODUCTION

1.1. Motivation

Now that we are in the midst of a major revolution in human computer interface, the multimedia interface is becoming the combination of computer, television and telephone. The coming of the information super-highway is indeed a big merge of communications, computation and consumer electronics. The core technology of this transition is digital video which will be the bulk of the traffic in future ATM networks. Also it is the key to many applications such as interactive TV, video-on-demand, video conferencing, HDTV, video phone, set-top box, videoCD and DVD, just to name a few.

The current two important topics in digital video communications are representation and transmission. The efficient digital representations of video signals (also called source coding) has been the subject of considerable research over the past twenty years.

Compression is the most critical aspect of digital video, because uncompressed video is far too huge to transmit, store, and manipulate. For example, a 640*480 color video will require transmit rate at 27.6MB/s, and a 2-hour full motion video will need 99,360 MB storage space which is certainly not practical.

This field is sufficiently mature that several standards are now available. These include the ITU-T H.261 standard for teleconferencing and the ISO/IEC MPEG family of standards. Right now these standards are all based on the same general architecture, namely motion-compensated temporal coding coupled with block discrete cosine transform spatial coding. The standards and their implementations will be discussed thoroughly in the following chapters.

Although a great deal of work remains to be done to approach fundamental limits of coding performance, these standard coding algorithms will form the basis for many services in the coming years. In particular, the telecommunication and semiconductor industries are working on these standards. However, standardization is a time-consuming process and development goes on.

The intraframe coding of video signals is very similar to the still image compression standard developed by the Joint Photographic Experts Group and the coding standard is accordingly called JPEG. JPEG is based on a division of the image into blocks of 8x8 pixels. Each block is transformed with discrete cosine transform (DCT). After the transform, the DCT coefficients are quantized and each frequency is weighted according to its importance to the human visual system (HVS). Finally, the coefficients are zig-zag scanned, run length coded and the data stream is sent to an entropy coder such as Huffman encoder.

The above schemes are the core part of video coding standards such as H.261 and MPEG-1 and MPEG-2. For low bit rate applications like video conference and videotelephony the Recommendation H.261 of the International Consultative Committee for Telephone and Telegraph (CCITT) was internationally adopted in 1991. The Moving Pictures Expert Group of the International Organization for Standardization (ISO/IEC) has been developing a similar but extended and improved scheme. MPEG-1 is intended for the storage of video (For example, CD-ROM and

VideoCD) and associated audio at low to medium bit rates and was standardized in 1992. MPEG-2 is intended for the transmission of video and associated audio at high bit rates, such that it can also be used for digital high definition television (HDTV). It can also be used for the broadcasting of standard digital television.

The Subband related coding techniques are promising especially for HDTV because they have an inherent multiresolution structure, so the normal digital TV can extract the low-resolution video from the video stream without affecting the high resolution quality. Nevertheless, compared to DCT related techniques, subband coding has a short history for standardization. However, it may play an important role in the forcoming MPEG-4 video coding standard which is for very low bit rate special-purpose applications such as wireless video transmission.

1.2. Outline of the thesis

This thesis discusses digital video compression algorithms (MPEG) and their VLSI implementations. Moreover, a VLSI implementation (CMOS 0.35μ technology and 3 layer metal) of a variable length decoder (VLD) for MPEG applications is developed. The VLD achieves high performance by using parallel and pipeline architecture. MPEG bitstream patterns have been carefully analysed to drastically improve VLD memory efficiency. In addition, a special clock scheme is applied to reduce the power consumption.

Chapter 2 contains overview of different techniques for still image compression. JPEG standard has been extensively studied. Several important fast DCT methods are presented. Huffman coding as one of the VLC methods has been explained. Rate Distortion function is used to evaluate the performance.

Chapter 3 describes the fundamentals of lossy video compression, emphasis on motion estimation (ME) and motion compensation (MC).

Chapter 4 introduces the very important digital video coding standards: MPEG-1 and MPEG-2. Both field and frame based coding are introduced.

Chapter 5 describes the VLSI architectures of video compression. First the different architectures are compared, then several ways of designing ME are presented. The practical issues involved are also studied, including low power design techniques and interconnection analysis.

In Chapter 6, a high-performance, low-power and memory-efficient VLD has been designed for MPEG applications.

2. STILL IMAGE COMPRESSION

2.1. Introduction

At eight bits/sample, the image has sufficient precision to be considered a gray scale or continuous-tone image. For binary images (e.g. fax image), there is only one bit per sample and are usually images of text or line drawings. They only have two tones, black and white. A process called *digital halftoning* creates the effect of continuous tones in a binary image by alternating between closely spaced black and white samples.

JPEG works best on "continuous tone" images; images with many sudden jumps in color values will not compress well. There are a lot of parameters to the JPEG compression process. By adjusting the parameters, one can trade off compressed image size against reconstructed image quality over a wide range.

JPEG has defined a "baseline" capability which must be present in all the JPEG modes of operation which use the DCT. It also defines optional extensions for progressive and hierarchical coding. There is also a separate lossless compression mode which typically gives about 2:1 compression where the lossy one gives about 10 to 20:1 compression or even higher compression depending on different applications.

2.1.1. Color Spaces

Red, green and blue (RGB) is one example of a color representation requiring three independent values to describe the colors. Each of the values can be varied

independently, and we can therefore create a three-dimensional space with R, G, and B as independent coordinates. Colors are represented as points in this space.

RGB may not always be the most convenient way of color representation. One particularly important color space for digital image compression is called Luminance-chrominance representation where one component is the luminance (Provides a grayscale version of the image) and the other two components provide the extra information that converts the grayscale image to color image.

The reason for doing this is that one can afford to lose a lot more information in the chrominance components than you can in the luminance component, because the human eye is not as sensitive to high-frequency color information as it is to high-frequency luminance. So the first step in JPEG is the color space transformation.

If the values of the three colors R, G and B are expressed by a relative scale from 0 to 1, then the luminance(Y) of any color can be calculated from the following weighted sum:

$$Y = 0.3R + 0.6G + 0.1B.$$

The scaling is chosen such that the luminance is also expressed by a relative scale from 0 to 1 and weights reflect the contributions of the individual primaries to the total luminance.

The term *chrominance* is defined as the difference between a color and a reference white at the same luminance. The chrominance information can therefore be expressed by a set of color differences, V and U, where V and U are defined by:

$$V = R - Y; U = B - Y.$$

These color differences are zero whenever R=G=B, as this condition produces gray, which has no chrominance. The V component controls colors ranging from red

$V > 0$ to blue-green $V < 0$, whereas the U component controls ranging from blue $U > 0$ to yellow $U < 0$. Together with the luminance, these chrominance coordinates make up the color coordinate system known as YUV.

Another color coordinate system, YCbCr, was used extensively in the development of the JPEG standard. This system is closely related to YUV where the U and V are scaled and zero-shifted to produce the variables Cb and Cr, respectively:

$$Cb = (U/2) + 0.5Cr = (V/1.6) + 0.5$$

This will ensure that Cb and Cr are always in the range 0 to 1 so they can be multiplied by 255.

A number of other color spaces, including the YIQ can also be related to these spaces by simple linear transformation.

2.1.2. Human Visual System

The frequency response of the human eye is more sensitive to low frequency than to high frequency. For vertical patterns, it is similar to the response for horizontal patterns. On the diagonals, however, the response is significantly reduced. Furthermore, the human eyes are much less sensitive to the high frequency of the color information which is a very important property from the standpoint of data compression [30].

Down-sampling is the typical technique to compress the color image. The luminance component is left at full resolution, while the color components are usually reduced 2:1 horizontally and either 2:1 or 1:1 (no change) vertically. This step immediately reduces the data volume by one-half or one-third, while having almost no impact on perceived quality. (Obviously this would not be true if we tried it in RGB color space).

However, simple down-sampling methods such as discard every other samples may introduces “aliasing artifacts“, therefore the method of averaging together groups of pixels in the processing of down-sampling. This is equivalent to lowpass filtering.

2.1.3. DCT and Fast DCT

The discrete cosine transform was first applied to image compression in Ahmed, Natarajan, and Rao’s pioneering work, in which they showed that this particular transform was very close to the KLH (Karhunen-Loeve-Hotelling) transform, a transform that produces uncorrelated coefficients [44].

We have seen that the human visual system response is very dependent on spatial frequency. If we could somehow decompose the image into a set of waveforms, each with a particular spatial frequency, we might be able to separate the image structure the eye can see from the structure that is imperceptible.

Group the pixel values for each component into 8x8 blocks. Transform each 8x8 block through a discrete cosine transform (DCT) with following equations

$$F(u, v) = \frac{C(u)}{2} \frac{C(v)}{2} \sum_{x=0}^7 \sum_{y=0}^7 f(x, y) \cos\left(\frac{(2x+1)u\pi}{16}\right) \cos\left(\frac{(2y+1)v\pi}{16}\right)$$

and the Inverse DCT (IDCT):

$$f(x, y) = \sum_{u=0}^7 \sum_{v=0}^7 \frac{C(u)}{2} \frac{C(v)}{2} F(u, v) \cos\left(\frac{(2x+1)u\pi}{16}\right) \cos\left(\frac{(2y+1)v\pi}{16}\right)$$

where $C(u), C(v) = \frac{1}{\sqrt{2}}$ for $u, v = 0$ and $C(u), C(v) = 1$ otherwise.

We can think the DCT as a harmonic analyzer. This is a relative of the Fourier transform and likewise gives a frequency map, with 8x8 components. Each 8x8 block of source image samples is effectively a 64 point discrete signal which is a function of the two spatial dimensions x and y . The DCT takes such a signal as

its input and decomposes it into 64 unique 2D “spatial frequencies” which comprise the input signals “spectrum”. The output is the set of 64 basis-signal amplitudes whose values are uniquely determined by the particular 64 -point input signal.

Thus we now have numbers representing the average value in each block and successively higher-frequency changes within the block. The motivation for doing this is that we can now throw away high-frequency information without affecting low-frequency information. (The DCT transform itself is reversible except for roundoff error.)

The most straightforward way to implement the DCT is to follow the theoretical equations. When we do this, we get an upper limit of 64 multiplications and 56 additions for each 1-D 8-point DCT. Or, 1024 multiplications and 896 additions for 8x8 block.

If the DCT really required this many operations to compute, JPEG would have chosen a different algorithm. In fact, there are many fast DCT techniques take advantage of the symmetries in the DCT equations. And there exists a close connection between DCT and DFT. For example, It has been shown that the N-point DCT can be expressed in terms of the real and imaginary parts of an N-point DFT and rotations of the DFT outputs. It has also been shown that the first N coefficients of a 2N- point DFT with appropriate symmetry of input values can be used to compute an N-point DCT. This will also lead to a very efficient scaled DCT structure. This can be shown as follows:

Let $W_K = \exp(-j2\pi/K)$, then the K-point DFT is given by:

$$F(u) = \sum_{x=0}^{K-1} s(x)W_K^{ux}$$

We then define the appropriate symmetry of the input value by using the mirror of the input, that is, extend the N -point sequence $s(x)$, $x = 0, \dots, N - 1$ by defining another N points with symmetry about the point $(2N-1)/2$,

$$s(x) = s(2N - x - 1), x = N, \dots, 2N - 1$$

then the $2N$ -point DFT becomes

$$F(u) = \sum_{x=0}^{N-1} s(x)W_K^{ux} + \sum_{x=N}^{2N-1} s(2N - x - 1)W_K^{ux}.$$

If we define the new index $k = 2N - x - 1$ (Note that $W_K^{2N} = 1$), this becomes

$$F(u) = \sum_{x=0}^{N-1} s(x)W_K^{ux} + \sum_{k=0}^{N-1} s(k)W_K^{-u(k+1)}.$$

Replacing the index k by x and multiplying by $(1/2)W_K^{u/2}$, yields

$$(1/2)F(u)W_K^{u/2} = \sum_{x=0}^{N-1} s(x) \cos((2x + 1)u\pi/2N),$$

which proves that multiplying a complex scaling factor by the first eight DFT coefficients (of the 16-point DFT), gives the 8-point DCT coefficients.

Take one step further and we can show that the DCT coefficients can be obtained by a simple scaling of the real part of the DFT coefficients, that is,

$$(1/2)Re(F(u))sec(\pi u/2N) = \sum_{x=0}^{N-1} s(x) \cos((2x + 1)u\pi/2N)$$

Because of the orthogonal nature of the DCT transform, the IDCT has the identical computational complexity of the DCT.

2.2. Image Compression Systems

2.2.1. Introduction to JPEG Standard

We now study a generic transform based image coding system, normally speaking, a coding system which exploits the intra-frame correlation composed of several major subsystems [29].

Figure 2.1 represents the core computation pipeline employed in all the popular lossy image and video compression algorithms. In the encoder the DCT transforms each 8x8 block into a set of DCT coefficients. There is no compression at this step, in fact, there is expansion because each DCT coefficient needs 12 bits to represent it. The “lossy” comes from the next step, namely, quantization. This process is an irreversible process and thus we will encounter information loss. After quantization, the nonzero DCT coefficients are further compressed using an entropy coder. In most applications, the entropy coder combines a run-length coder with a Huffman coder. This process is lossless. In the following section, JPEG will be given as an example to illustrate the various steps performed during compression and decompression:

For the color RGB image, one approach would be handle each color component separately. However, as we mentioned before, there is significant correlation between the color components. Thus, the approach is to transform the RGB image into another component such YCbCr. There is very little correlation among the components in this representation, furthermore, since most of the spatial information is in the luminance (Y) component, we lose little information if we subsample the Cb and Cr components by a factor of two in both dimensions. Therefore we have reduced the image size by half before it goes into the encoder.

For more efficient processing, images with multiple components can be interleaved. A data unit is defined as the smallest logical unit (LU) of source data that can be processed by JPEG. For lossy JPEG, this is a single 8x8 block of data. A minimum coded unit (MCU) is formed by interleaving the Y, Cb and Cr data units.

The interleaved data come to the encoder in 8x8 blocks, then the pixel values are changed to frequency domain by 2-D DCT. Which can be any one of the fast DCT algorithm. Then the coefficients are quantized and entropy coded.

2.2.2. Quantization

The quantizer is the main source of compression. To do this, an 8x8 quantization matrix is required. It can be designed based on human perception and psycho visual experiments or based on rate-distortion theory and bit-rate control. First we give a quick review of rate-distortion theory. However, before going into the details, a review of information theory is necessary.

We define the information content of a message, x , as I_x in:

$$I_x = \log \frac{1}{P_x},$$

where the P_x is the probability of occurrence of the event (message) x . Note that if we use base 2 for the logarithm, the unit of information is called the *bit of information*.

Entropy is defined as the average information per message. To calculate the entropy, we take the various information contents associated with the messages and weight each by the fraction of time we can expect that particular message to occur. This fraction is the probability of the message. Thus, given n messages, x_1 through x_n , the entropy is defined by

$$H = \sum_{i=1}^n P_{xi} \log\left(\frac{1}{P_{xi}}\right).$$

The spatial redundancy of an image can be modeled by a two-dimensional covariance function. A typical isotropic (non-separable) one looks like

$$Cov_X(i, j) = \sigma^2 e^{-\alpha \sqrt{i^2 + j^2}},$$

where σ^2 is the variance of the image and i and j refer to the distance from the reference pixel about which the covariance function is defined. It can be seen from the above equation that the covariance function decays rapidly beyond $i, j > 8$. This is also one of the reasons that 8x8 block is chosen.

Using quantization, with some loss of information, much higher compression ratios are possible. However, there is a tradeoff between distortion (D) and the bit rate (R) of the compressed bit stream. For a given D , the rate-distortion function $R(D)$ is defined as the minimum possible rate R necessary to achieve average distortion D or less. Note that $R(D)$ is independent of the particular compression method and depends only on the underlying stochastic model for the input images and the distortion measure [30].

For the isotropic covariance function above with $\sigma^2 = 1$, the 2-D power spectral density of an image X has the form

$$S_x(f_x, f_y) = \frac{f_0}{2\pi(f^2 + f_0^2)^{\frac{3}{2}}} = S_0(f),$$

where $f = \sqrt{f_x^2 + f_y^2}$. It has been shown by Berger [43] that the optimal encoder-decoder tandem yields distortion $D(\theta)$ and a minimum rate $R(\theta)$ given by

$$D(\theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \min(\theta, S_x(f_x, f_y)) df_x df_y$$

$$R(\theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \max\left(0, \frac{1}{2} \log_2 \frac{S_x(f_x, f_y)}{\theta}\right) df_x df_y$$

The $R(D)$ function has several practical uses, for example, one can measure D and the corresponding rate R and compare them against the theoretical limit $R(D)$, it can even be used to optimize the design of functional blocks within the encoder and decoder. For instance, the design of quantization tables.

The techniques for the design of quantization tables can be divided into two main classes: (1) those based on human perception and psycho visual experiments, and (2) those that are based on rate-distortion theory and bit rate control. The first method defines the elements of the quantization matrix from visibility thresholds for the DCT coefficients so that the human eyes will not detect them.

The tables give very good perceptual quality for most natural scenes. Moreover, one can tradeoff between image quality and data compression by uniformly scaling the original quantization table by a *quality factor*, for example, halving them will get nearly indistinguishable quality. This technique will become very handy when we try to design a video compression system.

One recent actively researched area is rate control techniques based on rate distortion functions. This can yield better compression result.

The idea behind this kind of rate control is to allocate more bits to the coefficients with large variances. Thus it also known as *Bit-Allocation* [23]. From rate-distortion theory, the optimum bit allocation is given by

$$b_{i,j} = r + \frac{1}{2} \log_2 \frac{\sigma_{i,j}^2}{[\prod_{i,j} \sigma_{i,j}^2]^{\frac{1}{64}}},$$

where $b_{i,j}$ is the bits allocated to each DCT coefficients and

$$\mu_{i,j} = \frac{1}{B} \sum_{k=1}^B y_k[i, j]$$

where B is the number of 8x8 blocks in the image,

$$\sigma_{i,j}^2 = \frac{1}{B} \sum_{k=1}^B [y_k[i, j] - \mu_{i,j}]^2$$

and r is the desired average compressed bit rate.

$$r = \frac{1}{64} \sum_{i=1}^8 \sum_{j=1}^8 b_{ij}.$$

Therefore we get the quantization matrix:

$$Q[i, j] = \frac{2046}{2^{b_{ij}}}.$$

Note that we use 2046 because the AC coefficients (Any component of the DCT output except the (0,0) component) range of eight bits pixel is from -1023 to 1023.

2.2.3. Entropy Coding

The coefficients after quantization can be further compressed using entropy coder as in Figure 2.2. This normally includes a Run-Length coder and a Variable Length Coder (VLC) which can be a Huffman coder or Arithmetic Coder. The baseline JPEG implementation uses Huffman coding only. Now we describe the operation of the entropy coder in JPEG.

Huffman code construction procedure the following steps:

1. Order the symbols according to their probabilities. The frequency of occurrence of each symbol must be known a priori. In practice, the frequency of occurrence can be estimated from a training set of data that is representative of the data to be compressed in a lossless manner. If, say, the alphabet is composed of N distinct symbols s_1, s_2, \dots, s_N and the probabilities of occurrence are p_1, p_2, \dots, p_N , then the symbols are rearranged so that $p_1 \geq p_2 \geq p_3 \dots \geq p_N$.

2. A contraction process is applied to the two symbols with the smallest probabilities. This can be viewed as the construction of a binary tree, since at each

step we are merging two symbols. At the end of this recursion process, all the symbols s_1, s_2, \dots, s_N will be leaf nodes of this tree.

3. The codeword for each symbol s_i is obtained by traversing the binary tree from its root to the leaf node.

As we have seen, the Huffman encoding process is relatively straightforward. Fixed-length input symbols are mapped into variable-length codewords. Hence the name VLC. One of the desirable features of a Huffman code is that it is a *prefix-condition* code which makes it *uniquely decodable*. Since there are no fixed-sized boundaries between codewords, if some of the bits are incorrectly received, the error will propagate such that all the data is lost. So, special markers are designated to indicate the start or end of a compressed stream packet. Huffman decoding has several implementations, such as Bit-Serial decoding and Lookup-Table-Based Decoding.

There are separately coding methods for DC coefficients (The (0,0) component of the DCT output) and AC coefficients because of their different statistics. Due to the high correlation of DC values among adjacent blocks, JPEG uses differential coding for the DC coefficients. That is, instead of coding each DC, the difference between the DC coefficients of two blocks ($DC_i - DC_{i-1}$) are coded. Actually, the residual is not directly Huffman coded, instead, it is expressed as a pair of symbols: the *category* and the *magnitude*. The first symbol represents the number of bits needed to encode the magnitude. Only this value is Huffman coded. This notion of using a category table is a form of context modeling and simplifies the Huffman coder. Without categorization of the difference, we would require a very large Huffman table.

For 8 bit-per-pixel data, DC differentials can take values in the range [-2,047, 2,047]. This range is divided into 12 size categories. Thus, after a table lookup, each

DC differential can be described by the pair (size, amplitude), where *size* defines the number of bits required to represent the amplitude, and *amplitude* is simply the amplitude of the differential.

AC coefficients are processed in a different manner. They are scanned in Zig-Zag order (Figure 2.3) which allows for a more efficient operation of the run-length coder.

A run-length coder yields the *value* of the next nonzero AC coefficient and a *run*. This structure is motivated by the fact that there are lots of zeros among AC coefficients after quantization. Therefore, each nonzero AC coefficient can be described by the pair (run/size, amplitude). Similar to the case for DC coefficients, the value of run/size is Huffman coded instead of amplitude. For example, an AC coefficient with value 3 is preceded by 3 zeros. Hence, the coefficient is represented by (3/2,3), If the Huffman code of 3/2 is 111110111 then the code will look like 11111011111. Another important symbol is end of block (EOB) which denotes that the rest of the AC coefficients are all zeros. This way we can greatly reduce the code table size.

2.3. Operation Modes

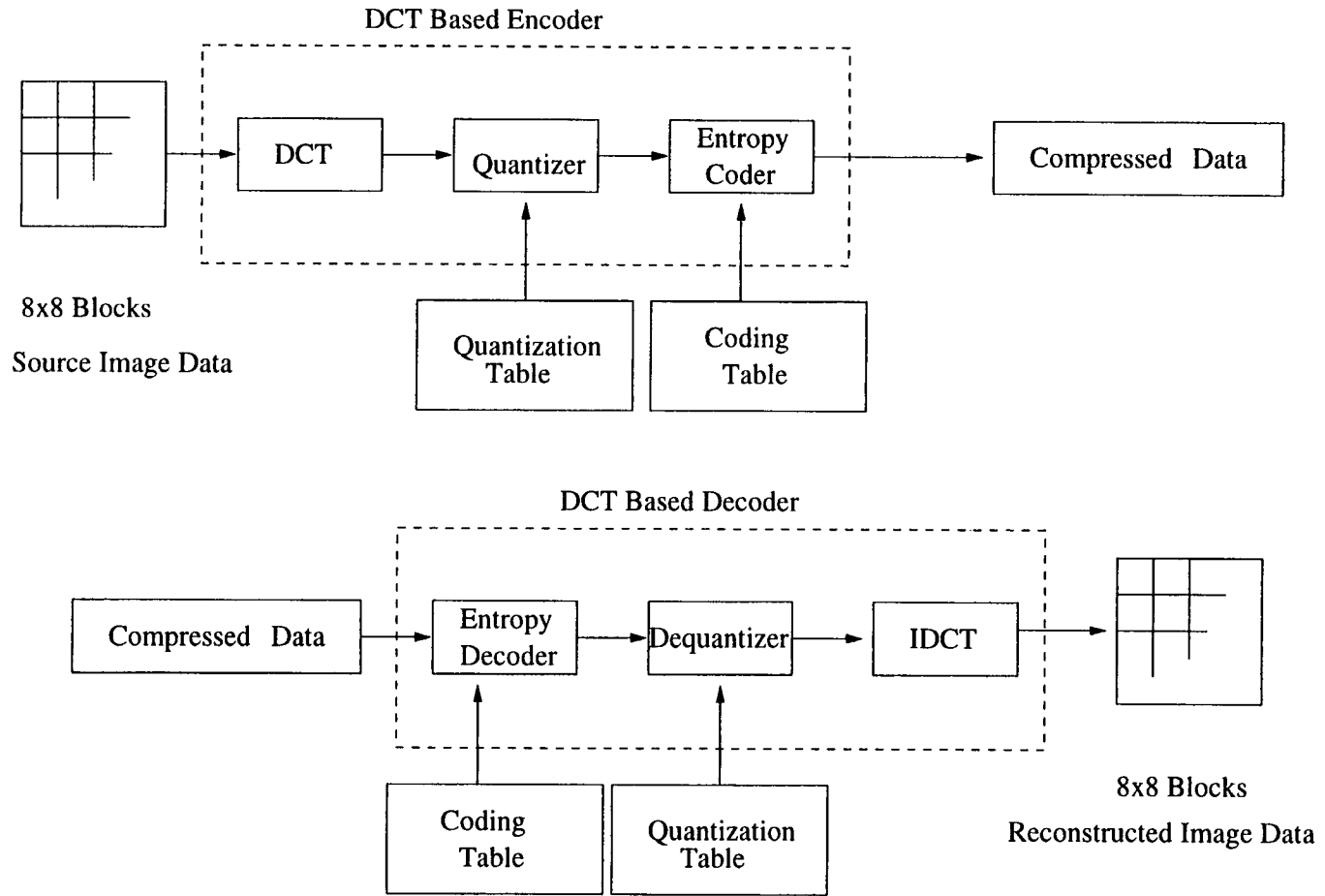
The JPEG standard specifies four modes of operation: Sequential DCT-based, left to right, top to bottom; progressive DCT-based, coarse to sharp resolution; lossless, exact reconstruction; hierarchical, can decode lower resolution without decoding full resolution image first (Figure 2.4).

Under the lossless mode, a predictive coder followed by either a Huffman or an arithmetic coder is used instead of a DCT-based thus get rid of the quantization step where the information is lost [36].

In the progressive mode, coding is performed in *multiple scans*. There are two procedures that are allowed for progressive coding: *spectral selection* where DCT coefficients are grouped into “spectral” bands of related spatial frequencies and the lower-frequency bands are sent first, and *successive approximation* which the information is first sent with lower precision and then refined in later scans. Figure 2.4 is a detail graph of another operation mode, namely, hierarchical coding. This mode is very useful when there are multiresolution requirement. This will also lead us to next chapter when we introduce the multiresolution techniques using wavelets.

From this three-level hierarchical coder we can get some ideas of the coding procedure. First, the source image data S is subsampled twice S_2 and S_4 . The reconstructed pictures have different resolution, R_4 is constructed from S_4 . R_2 is constructed from the encoded difference of S_2 and its upsampled version.

FIGURE 2.1. JPEG Encoder Decoder



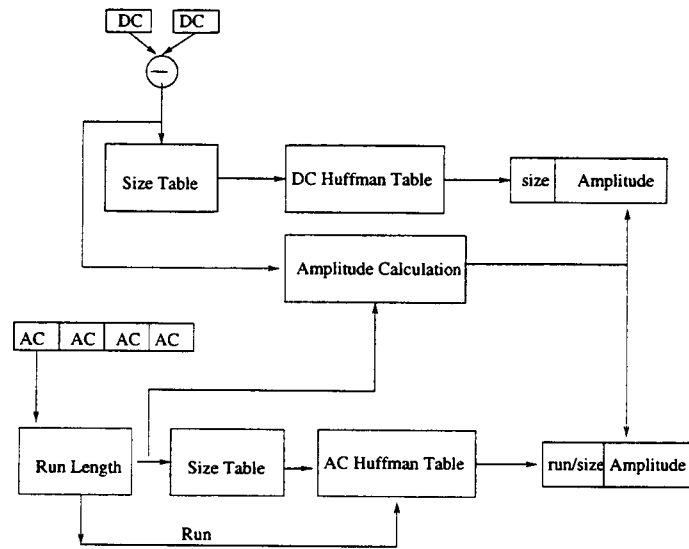


FIGURE 2.2. Entropy Coding

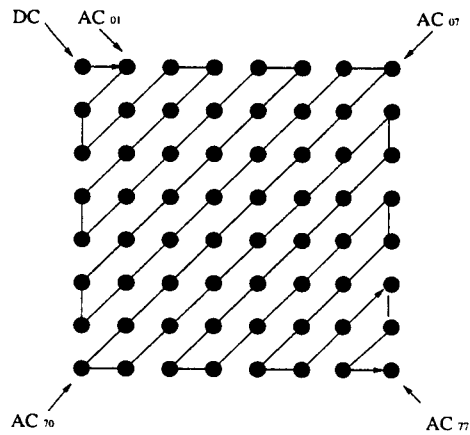


FIGURE 2.3. Zig-zag scan of the AC Coefficients

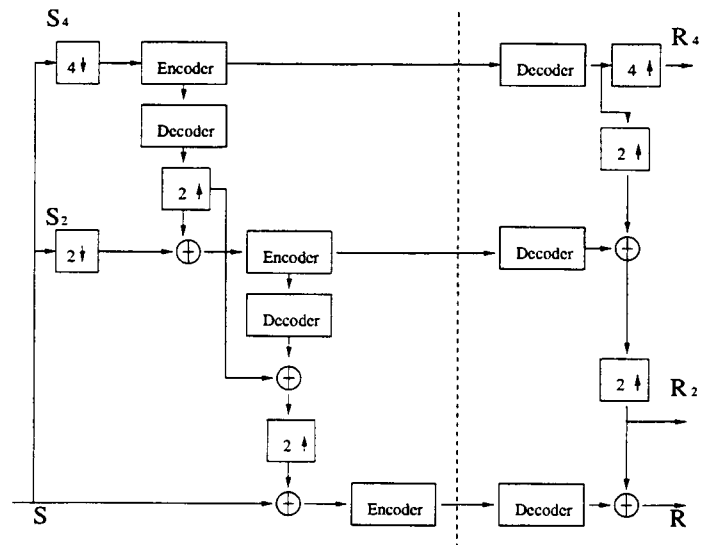


FIGURE 2.4. Three-level Hierarchical Coder

3. LOSSY VIDEO COMPRESSION

3.1. Introduction

Video can be regarded as still images with another dimension: time. Video is merely still images displayed at a speed of 30 frames per second. This is not only true in the digital world but also applies to the analog world, TV is broadcast at 30 frame or 60 fields per second whereas movies are showed at 24 frames per second. Most of the video compression is lossy because it is not possible to achieve very high compression ratio by employing lossless compression methods. Nor is it necessary because human eyes cannot tell the loss if the compression is done gracefully. Although most of the still image compression methods are used in video compression, or *intra-frame coding*, many efforts have been made in to explore the redundancy along the temporal domain to remove the correlation between the frames, or, *inter-frame coding*. Among all the techniques, motion estimation (ME) is by far the most widely used and implemented methods for reducing the temporal redundancy [25].

3.2. Motion Estimation

Consider a picture sequence of a singer singing on stage, we will find that two kinds of correlation exist, one is the spatial correlation among the pixels composed of a singer. Whereas the other is the temporal correlation, the same object (the singer) appears in many frames and the background is almost fixed. Thus, the successive frames of a video sequence are analyzed to estimate the motion (displacement) vectors of moving pixels or blocks of pixels. Many different types of algorithms such

as Blocking-Matching Algorithms (BMA), Pel-Recursive Algorithms (PRA), Phase Correlation, etc., have been developed and implemented in video encoders. We will focus on BMA since it is the more widely used in real-time video encoders. And it is also used in MPEG standard [31].

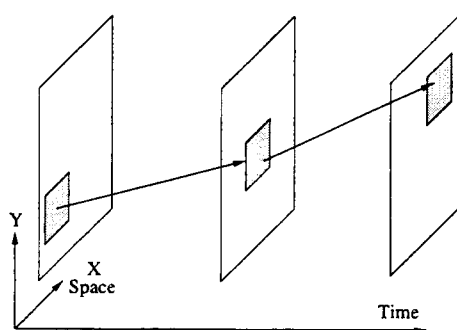


FIGURE 3.1. Temporal Correlation Between Pictures

In BMA, each video frame is partitioned into blocks. The motion vectors are extracted on a block by block basis. Two frames are used in motion estimation: a *reference frame* and a *current frame* (Figure 3.1). For each block in the current frame, it is searched in a search-window in the previous frame for a best matched block based on a match-criterion. If we define a frame as a current frame at time t , we define a reference frame at past time $t - n$ for *forward motion estimation*, and future time $t + k$ for *backward motion estimation*. If we define the location of current macroblock as (x, y) , then the best matching block with MV (u, v) is located at $(x + u, y + v)$. However, we make the assumption that all the pixels in the macroblock go through the same common displacement. In BMA, the motion vectors have to be transmitted to the decoder for reconstruction of the image. These are transmission overheads compared to the simple pel-by-pel predictive coding. The amount of the overhead is dependent on the block-size and the search-window size. Larger block-sizes result in fewer blocks in a frame, and thus, less transmission overhead. However, since the algorithms assume that the displacement is constant

within the block, larger block-sizes will degrade the performance. In most video coding standards, the size of the block is 16x16.

The most used matching criterion is the MAD or Mean Absolute Difference, or

$$MAD(i, j) = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} |C(x+k, y+l) - R(x+k+i, y+l+j)|,$$

where $C(x+k, y+l)$ is the pixel value of the current block, $R(x+k+i, y+l+j)$ is the reference block and $-p \leq i \leq p$. The best matching block $R(x+i, y+j)$ is the one with the minimized MAD. It can be shown that MAD almost performs as well as other much more complicated cost functions such as MSE (mean squared error). Therefore, for the rest of the thesis, the best match for ME means minimum MAD is considered.

In order to be able to track large motion, it is desirable to search in a large search-window. However, a larger search-window results in more search positions in the window and the required computation increases rapidly. Also, the transmission overhead is increased since more bits are needed to represent the motion vector. In videoconference applications, for example, H.261 standard a search-window that does not exceed +/- 15 is defined. However, to track a car chasing scene we may need a larger search-window.

There are several strategies to search for the best matched blocks. The most straightforward and brute-force method is the full-search (FS) which searches all the $(2p+1)^2$ locations in the search window. For each position, MAD is calculated and the smallest generates the motion vector as illustrates in Figure 3.2.

The FS method will guarantee the optimum solution since all possible positions of search-window are exhaustively searched. However, it also requires very large amount of computations and data-accesses. If we ignore the operation of load-

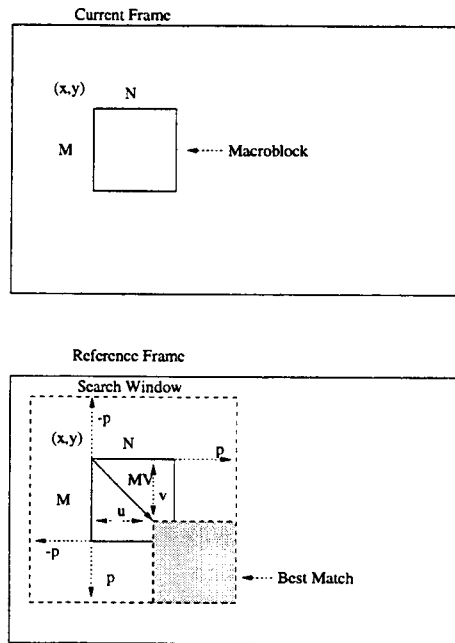


FIGURE 3.2. Block Matching Motion Estimation

ing and storing of the current and reference blocks, then for each $N \times M$ macroblock we need to do $3(2p+1)^2MN$ operations, where 3 means a subtraction, an absolute-value calculation, and one addition. For a typical TV broadcast with resolution 720×480 and 30 frames/s, and if we take $N = M = 16, p = 15$ the total operation need will be $3(31)^2 720 \times 480 \times 30$ or 28.89 GOPS (Giga operations per second). It should be noted that the amount of computation is independent of the block-size and is a quadratic function of the search-range. So for higher search range (which is necessary for high quality sports broadcast) the required number of operations is even higher. If we consider the memory access, the required I/O bandwidth is very difficult to achieve [43].

In order to reduce the amount of required computations and data accesses, many fast search methods have been developed. These algorithms are suboptimum in the sense that the search strategies do not guarantee that we will find the

minimum MAD value. However, the complexity is significantly reduced either by decreasing the number of search locations or by computing fewer pixel difference per location. In the following we will give some examples of these methods.

The three-step search (TSS) is a fast ME method that decreases the number of search areas. Instead of searching all the locations inside the search-window we only search the major points in the search window. First we compute the minimum MAD from the locations of $(0, 0), (0, d_1), (0, -d_1), (d_1, -d_1), (d_1, 0), (d_1, d_1), (-d_1, d_1), (-d_1, 0), (-d_1, -d_1)$ where d_1 is given by $d_1 = 2^k - 1$ where $k = \log_2 p - 1$ and p is the search range.

After we find the best match among these locations (for TSS where $k = 3$ there are nine locations), we continue this process using $d_2 = \frac{d_1}{2}$ until the we find the minimum. This greatly reduces the computation requirements. For instance, with $p = 7$ only 25 locations need to be calculated instead of the 225 locations if we use full search. Comparing the implementation complexity of this method with that of full search, for the same resolution TV (720x480, 30 frames/s) and $p = 15$, yields $3 \times 720 \times 480 \times 30 \times (8[\log_2 p] + 1)$ or about 1 GOPS.

Similar to TSS, PHODS (Parallel Hierarchical One-dimensional Search) also uses the idea of reducing the search locations. However, in PHODS the search is done independently along the two dimensions, in this case, the MAD calculations are parallelizable and it has a regular data flow, since the search locations are always along the x and y axis. However, PHODS and TSS all assume that the MAD increases monotonically as the search area moves away from the best-matched location. If this assumption fails, search for a global minimum may get trapped into a local minimum, in other words, the search may be misguided by local minima at the beginning steps.

As we mentioned before, there exists another method to reduce the computation requirement of ME, by using reduced pixel numbers. Since block matching implies that all pixels within the macroblock have the same motion, an estimate of the motion can also be obtained with fewer pixels. One straightforward way is by pixel subsampling. However, this decimation has to be done very carefully such that the ME accuracy will not be greatly affected. For example, for each macroblock, one fourth of the pixels can be used to calculate MAD, by alternating the pixel patterns and associating a different pattern for each neighboring search location, all the pixels within the block can be covered thus minimizing the possibility of not considering one-pixel-wide horizontal, vertical and diagonal lines.

The two fast ME methods we discussed before are such that one reduces the search locations and the other reduces the pixel numbers. How about combining them together? Their combination will lead naturally to Hierarchical Motion Estimation (HME) which combines two features together and is widely used in almost all the video compression algorithms. In the HME, the matching of blocks is performed in two or three steps with decreasing block-size and increasing search resolutions. Lowpass filtering is applied to improve the reliability. Subsampling is used to reduce the required computation load. In the following example we give in depth introduction to a 3 level hierarchical motion estimation.

First, several low-resolution versions of the current picture and the reference picture is formed by lowpass filtering and subsampling. In Figure 3.3 we show two low levels (namely, level 1 and level 2) of the original resolution (level 0). This step applies to both the reference picture and the current picture.

The ME starts from the lowest resolution, in our case, level 2. Because the picture size is small so the 4x4 macroblock is used in ME instead of 16x16. The scaled versions of search parameter can also be used, that is we use $\frac{p}{4}$ instead of

p . We can use any of the search methods mentioned before. However, because of the smaller search area and smaller macroblock, full search method can be used to increase the accuracy. If we assume that the search origin in level 0 is (x, y) then it will be $(\frac{x}{4}, \frac{y}{4})$ at level 2. The complexity of ME at level 2 will be:

$$\frac{\text{picture size}}{\text{Macroblock size}} \times (2 \times p + 1)^2 \times (\text{Macroblock size}) \times 3,$$

where picture size will be 180x120 and p is 4, and the result is 157.46 MOPS. Assume the MV we found at this step is (u_2, v_2) , then we go up one level and do the ME at level 1. Because the (u_2, v_2) is the MV we found at the lower resolution, we need to refine the search. Therefore we do not have to search in a very big area, instead the search is centered around the (u_2, v_2) . Thus, a search region $[-1, 1]$ is enough. As before, we use full search since only 9 locations are needed to compute. We need to compute the origin $(x/2 + 2u_2, y/2 + 2v_2)$ again because in this level we are using macroblocks of size 8x8. Repeating the complexity analysis, we can get 69.98 MOPS, assuming the minimized MAD is found at (u_1, v_1) .

At level 0 or the reference picture level, again we search 9 locations around origin $(x + 2u_1, y + 2v_1)$ with macroblock size of 16x16, the computation complexity is now 279.9 MOPS. At this level the final MV will be generated.

If we add the computation requirement at each level together, we will get the total complexity of 507.38 MOPS, which is greatly reduced from the 28.89 GOPS needed for full-search, and almost half of the TSS search of 1 GOPS. Therefore, this is by far the most popular ME algorithm [28] (Figure 3.3).

By subsampling the reference picture will also created some problems since inaccurate MV may be generated for regions containing small objects. Because the search starts from the coarse version of the reference picture, regions containing small objects maybe completely eliminated and thus failed to be traced. If there are

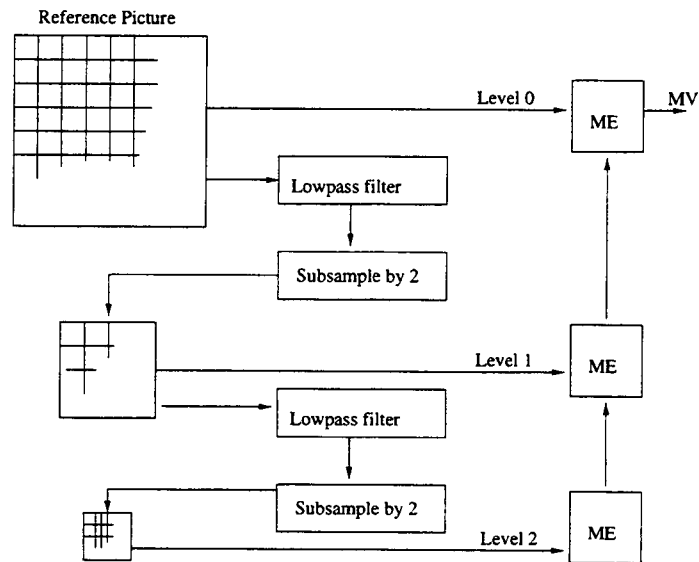


FIGURE 3.3. Hierarchical Motion Estimation

lots of small objects moving then the situation gets worse. Moreover, the subsampled version of the picture has to be stored for use in ME, thus increasing the system memory requirement. On the other hand, the creation of low-resolution version (especially the lowpass filtering) will reduce noise.

Because the true motion between frames is unrelated to the sampling grid, if the displacement estimates are obtained at a finer resolution, a better prediction can also be obtained. The ME we talked about before is all restricted to integer pixel grids. This means that we can have half-pixel or even quarter-pixel accuracy motion vector.

Half-pixel accuracy MV can be easily found by interpolating the current and reference pictures by a factor of two and then using any of the ME methods described in the previous section. However, if both interpolated pictures have to be stored for computation, excessive memory has to be provided. To avoid this, we can follow these steps for the half-pixel ME.

In the first step, a MV with integer-pel accuracy using any of the ME methods is obtained. The resulting integer-pel motion vector (u, v) which yields the minimum MAD is shown in the figure.

In the following steps, we refine the results of the first step to obtain the motion vector with the desired half-pel accuracy. As shows in Figure 3.4, eight new MAD are calculated at the new half-pel locations around the origin. The macroblock is 16x16 and the top-left corner of each block being at the locations marked circle in the figure. For each of these eight new search blocks, we know only those pixel values with coordinates matching our original integer-pel grid. The remaining values on the half-pel grid can be estimated using simple interpolation techniques. The MAD is computed by comparing each of these eight search blocks with the original macroblock. Therefore, the position where the MAD is minimum is the location for the half-pel MV. If we define this location as (k, l) then the refinement MV will be $(u + k/2, v + l/2)$.

The quarter-pel ME can be done in the same manner. One of the computation intensive steps of the above procedure is the interpolation step. There are some fast interpolation methods discussed in the literature [43], thus, we are not going to discuss them in greater detail. We will come back to this topic when we are discuss the VLSI implementation of the ME.

3.3. Motion Compensation

The next step after the step of ME where the MV is found, is to use this information to reduce the interframe redundancy. The output of the motion estimator, the coordinates (u, v) which define the relative motion of a block from one frame to another, are used to define the motion compensation (MC) as the process of com-

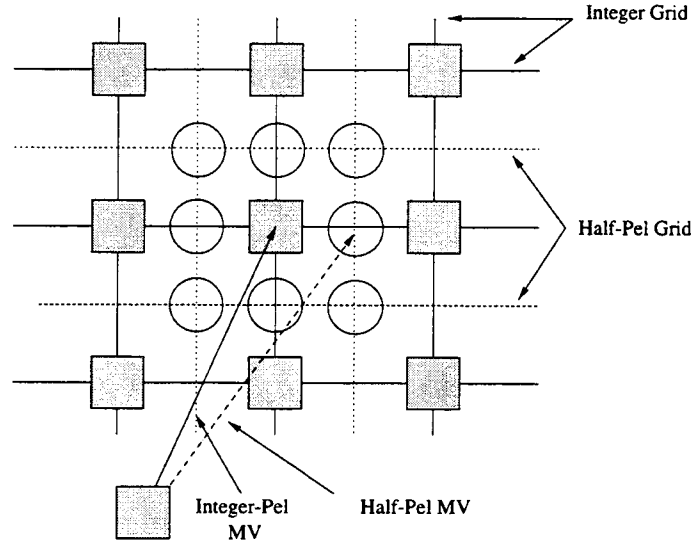


FIGURE 3.4. Half-Pel accurate MV estimation

putting changes among frames by establishing correspondence between frames. In other words, as the process of compensating for the displacement of moving objects from one frame to another. The prediction error as :

$$e(x, y, t) = I(x, y, t) - I(x - u, y - v, t - 1),$$

where $I(x, y, t)$ is the pixel value at spatial location (x, y) in frame t and $I(x - u, y - v, t - 1)$ is the *corresponding* pixel value at spatial location $(x - u, y - v)$ in frame $t - 1$.

Note that the above temporal prediction definition is very similar to the differential coding scheme DPCM. The difference is that we form the temporal prediction using temporally adjacent samples, which are determined through the process of motion estimation, so it gives a better prediction. Notice the MC is performed in both the encoder and the decoder but ME is needed only in the encoder.

We can now start to put everything together and build a hybrid coding system using the methods we mentioned so far. Before that, we need to discuss the

concepts of *frame* and *field*. In current TV industry the video is *interlaced*, which means that each frame is comprised of fields, a top field and a bottom field (or odd and even field, Figure 3.5). Scanlines from the two fields are interleaved. Note that, spatially adjacent scanlines are not temporally adjacent. For example, at 30 fps, the adjacent scanlines are $\frac{1}{60}$ -th of a second apart.

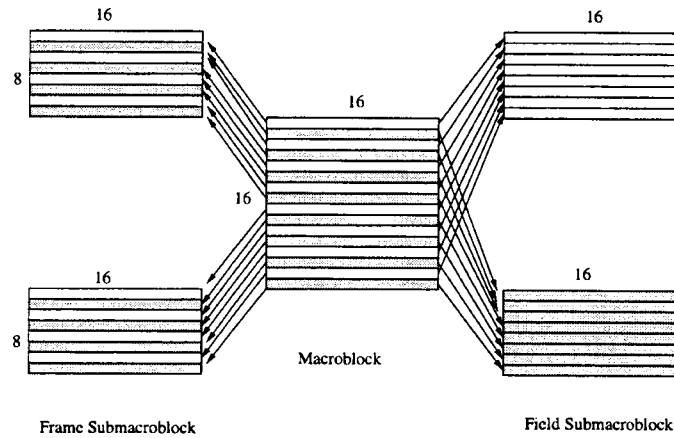


FIGURE 3.5. Field Image

In order to exploit the presence in a macroblock of lines from two temporally separated fields, two fundamental modes of MC are defined: the *frame-MC* and the *field-MC* modes. The frame-MC mode does not discriminate between the two fields inside a macroblock, it compensates a macroblock with another macroblock containing lines from both fields. In the field-MC mode, the lines from each field in a macroblock are compensated with a block containing only lines from the field of the same parity. For the frame-MC mode two block sizes are allowed, with dimensions of 16x16 and 16x8, respectively. In the latter case, a macroblock is separated into two horizontal halves. In the field-MC mode a macroblock is separated into two fields. From only one field with a size of 16x8.

A MV can point either to a field in another reference frame or to a field in the current frame. For example, the first field can be predicted either from

the top or the bottom fields in a reference frame, or the field can be predicted either from the bottom field of another frame or from the top field of the current frame. The vertical range for the integer MV is halved, since one field-line shift is equivalent to a two-line shift in the picture. The half-pixel positions for the field-MC mode are obtained as follows. A first intra-field spatial interpolation operation places the interpolated pixels where the lines of the other field lie. Since these positions correspond to integer-line locations of the picture, a second intra-field interpolation set up is performed that places the interpolated pixels at half-line positions of the picture, which is equivalent to quarter-line positions inside the field. MV for the field-MC mode are defined with respect to these quarter-line field positions. The half-pel search thus consists of updating the field integer motion vector in two successive steps. The effect is that the frame-mode and field-mode MV have the same resolution relative to the picture.

Another MC mode that will be discussed in the later chapter is the *Dual-prime* motion compensation in which only one MV is encoded (in its full format) in the bitstream together with a small differential MV. In the case of field pictures two MVs are then derived from this information. These are used to form predictions from two reference fields (top and bottom) which are averaged from the final prediction. In the case of frame pictures this process is repeated for the two fields so that a total of four field predictions are made.

In order to form a MV for the opposite parity, the existing MV is scaled to reflect the different temporal distance between the fields. A correction is made to the vertical component to reflect the vertical shift between the lines of top field and bottom field and then a small differential MV is added. This process is illustrated in Figure 3.6, which shows the situation for a frame picture.

Therefore, rather than sending four vectors, a single vector is sent with the four required vectors derived using a constant velocity model, and then a small accurately tuning parameter allows an independent adjustment of +/- 1 pel vertically in two of the vectors. Thus we can achieve low delay and less overhead. Assuming that m_{11} is the MV predicted from reference picture top field to the predicted picture top field, then we have the following relationship:

$$m_{22} \approx m_{11} + \delta$$

$$m_{21} \approx 1/2m_{11} + \delta$$

$$m_{12} \approx 3/2m_{11} + \delta$$

where δ can take the values in -1, 0 or 1.

Similar to ME, there is also frame DCT and field DCT and this can also be selected based on the analysis before coding. So if we have adaptive DCT and adaptive MC, then we have the major building blocks for adaptive frame/field (AFF) [4].

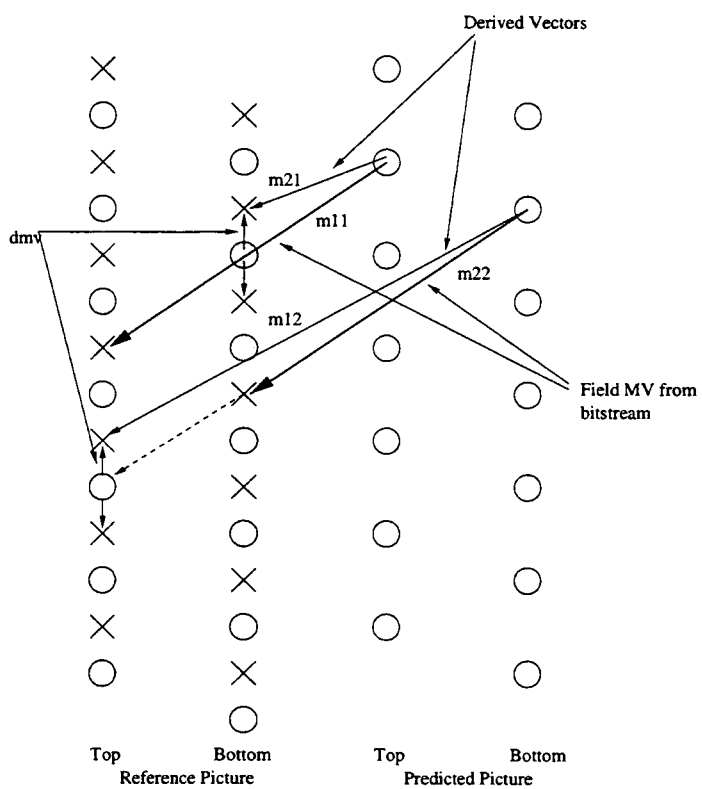


FIGURE 3.6. Dual Prime for ME

4. MPEG STANDARDS

4.1. Background

The MPEG committee was established by ISO in 1988 with the mission to develop standards for the coded representation of moving pictures and associated audio information on digital storage media. Over the next few years, participation amassed from international technical experts in the areas of Video, Audio, and Systems, reaching over 200 participants by 1992. By the end of the third year (1990), a syntax emerged, which when applied to code SIF (standard input format) video and compact disc audio samples rates at a combined coded bit rate of 1.5 Mbit/sec, approximated the perceptual quality of consumer video tape (VHS). Formally known as ISO standard 11172, or MPEG-1 which is mainly used in CD-ROM and VideoCD applications [20].

After demonstrations proved that the syntax was generic enough to be applied to bit rates and sample rates far higher than the original primary target application, a second phase (MPEG-2) [21] was initiated within the committee to define a syntax for efficient representation of broadcast video. Efficient representation of interlaced (broadcast) video signals was more challenging than the progressive (non-interlaced) signals coded by MPEG-1. Similarly, MPEG-1 audio was capable of only directly representing two channels of sound. MPEG-2 introduced a scheme to correlate multichannel discrete surround sound audio. Need for a third phase (MPEG-3) was anticipated in 1991 for High Definition Television, although it was later discovered by late 1992 and 1993 that the MPEG-2 syntax simply scaled with the bit rate could phase out MPEG-3 [37]. In 1994, this effort led to the ISO

standard 13818. MPEG-4 was launched in late 1992 to explore the requirements of a more diverse set of applications, while finding a more efficient means of coding low bit rate/low sample rate video and audio signals, such as wireless transmission of video and audio signals. MPEG-4 is expected to be defined by 1998.

From the beginning, MPEG has been embraced by industry, in fact, it is the driving force behind this effort. In 1991 C-Cube Microsystem made the world's first MPEG-1 decoder, then the world's first MPEG-1 encoder in 1993 and the world's first MPEG-2 encoder in 1994. Other companies then joined the band wagon, e.g., AT&T, IBM, IIT, LSI, NEC, SGS-Thomson and TI, etc.. MPEG applications such as VideoCD player (MPEG-1 decoder) and Direct TV set top box (MPEG-2 decoder) have become the most successful consumer electronics ever. Moreover, MPEG is an exclusive syntax of the United States Grand Alliance HDTV specification, the European Digital Video Broadcasting (DVB) and Sony/Toshiba Digital Video Disk (DVD).

The MPEG standards are published in four parts: systems, video, audio, and conformance testing [8].

Part 1—Systems: The first part of the MPEG standard has two primary purposes: 1) a syntax for transporting packets of audio and video bit streams over digital channels and storage mediums (DSM), 2) a syntax for synchronizing video and audio streams.

Part 2—Video: describes syntax (header and bit stream elements) and semantics (algorithms telling what to do with the bits). Video breaks the image sequence into a series of nested layers, each containing a finer granularity of sample clusters (sequence, picture, slice, macroblock, block, sample/coefficient). At each layer, algorithms are made available which can be used in combination to achieve efficient compression. The syntax also provides a number of different means for

assisting decoders in synchronization, random access, buffer regulation, and error recovery. The highest layer, sequence, defines the frame rate and picture pixel dimensions for the encoded image sequence.

Part 3—Audio: describes syntax and semantics for three classes of compression methods. Known as Layers I, II, and III, the classes trade increased syntax and coding complexity for improved coding efficiency at lower bit rates. The Layer II is the industrial favorite, applied almost exclusively in satellite broadcasting (Hughes DSS) and compact disc video (White Book). Layer I has similarities in terms of complexity, efficiency, and syntax to the Sony MiniDisc and the Philips Digital Compact Cassette (DCC). Layer III has found a home in ISDN, satellite, and Internet audio applications. The sweet spots for the three layers are 384 kbit/sec (DCC), 224 kbit/sec (CD Video, DSS), and 128 Kbits/sec (ISDN/Internet), respectively.

Part 4—Conformance: defines the meaning of MPEG conformance for all three parts (Systems, Video, and Audio), and provides two sets of test guidelines for determining compliance in bit streams and decoders. MPEG does not directly address encoder compliance.

Part 5—Software Simulation: Contains an example ANSI C language software encoder and compliant decoder for video and audio. An example system encoder/decoder is also provided which can multiplex and demultiplex separate video and audio elementary streams contained in computer data files.

As of March 1995, the MPEG-2 volume consists of a total of 9 parts under ISO/IEC 13818. Part 2 was jointly developed with the ITU-T, where it is known as recommendation H.262. The four additional parts are listed below:

Part 6—Digital Storage Medium Command and Control (DSM-CC): provides a syntax for controlling VCR-style playback and random-access of bit streams

encoded onto digital storage mediums such as compact disc. Playback commands include Still frame, Fast Forward, Advance, Goto.

Part 7—Non-Backwards Compatible Audio (NBC): addresses the need for a new syntax to efficiently -correlate discrete multichannel surround sound audio. By contrast, MPEG-2 audio (13818-3) attempts to code the surround channels as an ancillary data to the MPEG-1 backwards-compatible Left and Right channels. This allows existing MPEG-1 decoders to parse and decode only the two primary channels while ignoring the side channels (parse to /dev/null). This is analogous to the Base Layer concept in MPEG-2 Scalable video. NBC candidates include non-compatible syntaxes such as Dolby AC-3. Final document is not expected until 1996.

Part 8—10-bit video extension. Introduced in late 1994, this extension to the video part (13818-2) describes the syntax and semantics to coded representation of video with 10-bits of sample precision. The primary application is studio video (distribution, editing, archiving). This part has not been finished.

Part 9—is the real time interface.

All MPEG standards are generic, that is, application independent, they do not specify the operations of the encoder, instead, they specify the syntax of the coded bit stream and decoding process [15] [3].

4.2. Digital Video Formats

Both MPEG-1 and MPEG-2 video syntax can be applied at a wide range of bit rates and sample rates. The MPEG-1 that most people are familiar with has parameters of 30 SIF pictures (See following table) per second and a bit rate less than 1.86 megabits/s. Also known as “Constrained Parameters Bitstreams”. This

popular interoperability point is promoted by Compact Disc Video (White Book) [7].

Video Format	Frame Size (Y)	Frame Rate	Size
HDTV	1920*1250	50 Hz	1.9 Gb/s
CCIR 601	720*576 (PAL)	25 Hz	166 Mb/s
	720*480 (NTSC)	30 Hz	166 Mb/s
SIF	360*240	30 Hz	31 Mb/s
QSIF	180*120	30 Hz	8 Mb/s
QQSIF	90*60	30 Hz	2 Mb/s

Although MPEG bit-stream syntax allows for picture sizes of up to 4095*4095 pixels with a bitrates up to 100 Mb/s, many of the applications have been optimized for SIF (MPEG-1) and CCIR601 (MPEG-2 main level, main profile). A color video source has three color components, Y, Cb and Cr. One of the most useful formats is called 4:2:0 subsampling format where the resolution of the chroma components is half of the luminance resolution in both the horizontal and vertical dimensions. This is the only format for MPEG-1. Multiplex order within macroblock is YYCbCr, and targets main stream television and consumer entertainment. MPEG-2 also supports the 4:2:2 and 4:4:4 color subsampling formats. In the 4:2:2 format, the chrominance components have the same vertical resolution as the luminance component, but the horizontal resolution is halved. Multiplex order is YYYYCbCrCbCr, mainly used in studio production environments, professional editing equipment, distribution and servers. In the 4:4:4 format, all components have identical horizontal and vertical resolutions. Multiplexed order is YYYYCbCrCbCrCbCrCbCr and used in computer graphics application [14].

4.3. Temporal processing

One of the main applications of MPEG is in consumer and computer electronics. Thus it requires the random access of the video data, such as forward, rewind, still, stop and index. Because of the conflicting requirements of random access and highly efficient compression, three main picture types are defined, namely, I, B, P, as in Figure 4.1.

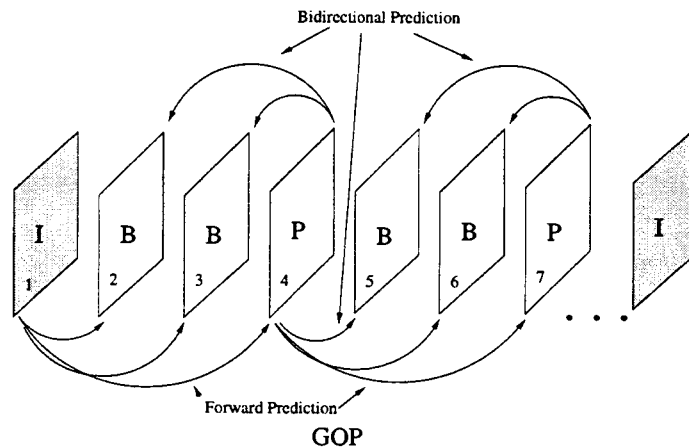


FIGURE 4.1. I B P frame in a Video Sequence

Intra coded pictures (I pictures) are coded without reference to other pictures. They provide access points to the coded sequence where decoding can begin, but are coded with only moderate compression. Predictive coded pictures (P pictures) are coded more efficiently using motion compensated prediction from a past intra or predictive coded picture and are generally used as a compensated prediction from a past intra or predictive coded pictures and are generally used as a reference for further prediction coded pictures (B pictures) provide the highest degree of compression but require both past and future reference pictures for motion compensation (Figure 4.2). B pictures are never used as references for prediction. B pictures are disliked by lots of people because the computational complexity, bandwidth, end-

to-end delay and picture buffer size all get much larger. We will discuss how to generate the B pictures in the following sections. (In MPEG-1 there is another type of picture defined as D picture which is coded like I picture but only with the DC coefficients of DCT output. D pictures are not used in MPEG-2)

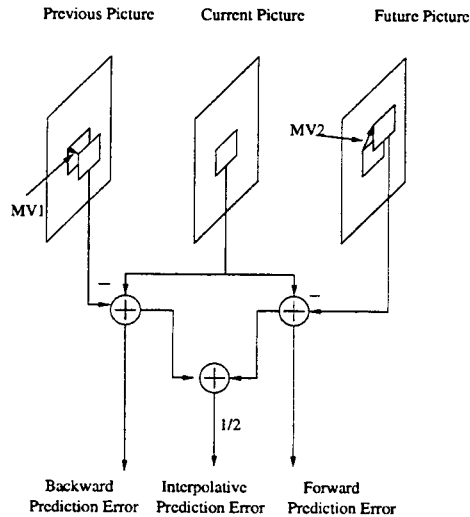


FIGURE 4.2. Bidirectional Motion Compensation

The organization of the three picture types in a sequence is very flexible, it usually goes like:

IBBPBBPBBPBBIBBPBBPB...,

where there are 12 frames from I to I. This is based on a random access requirement that is needed as a starting point at least once every 0.4 seconds or so. The ratio of P's to B's is based on experience.

The order of the coded frames in the bitstream, also called coded order, is the order in which a decoder reconstructs them. The order of the reconstructed frames at the output of the decoding process, also called the display order, is not always the same as the coded order.

When the sequence contains no coded B pictures, the coded order is the same as the display order. When B pictures are present, then the frame reordering

is performed according the following rules: If the current frame in coded order is a I or P, then the output frame is the frame reconstructed from the previous I or P. If the current frame is a B, then the output frame is the frame reconstructed from that B.

One example is Figure 4.1, which has two coded B frames between successive coded P frames and I frames. Frames '1I' is used for prediction of frame '4P'. '1I' and '4P' are both used for prediction of frames '2B' and '3B'. Therefore the order of coded frames in the coded sequence shall be '1I', '4P', '2B', '3B'. However, the decoder shall display them in the order of '1I', '2B', '3B', '4P'.

There is no special requirement to use either P pictures or B pictures in the MPEG bit stream. However, B pictures require significantly fewer bits than either I or P pictures. Increasing the number of B between I and P may not lead to better compression due to a drop off in temporal correlation as the distance between B and I or P increases.

4.4. Encoder and Decoder

The MPEG standard does not define an encoding process. Figure 4.3 shows the basic functions that need to to be executed by a typical MPEG encoder [26].

Preprocessing

This process may include color conversion, format translation, prefiltering, and subsampling. None of these operations is specified in the standards.

ME and MC

Figure 4.2 shows the bidirectional ME and MC. After the preprocessing, the encoder selects the coding type for the input picture, namely, I, B or P. The decision is made differently for each macroblock for every different types of picture. The I

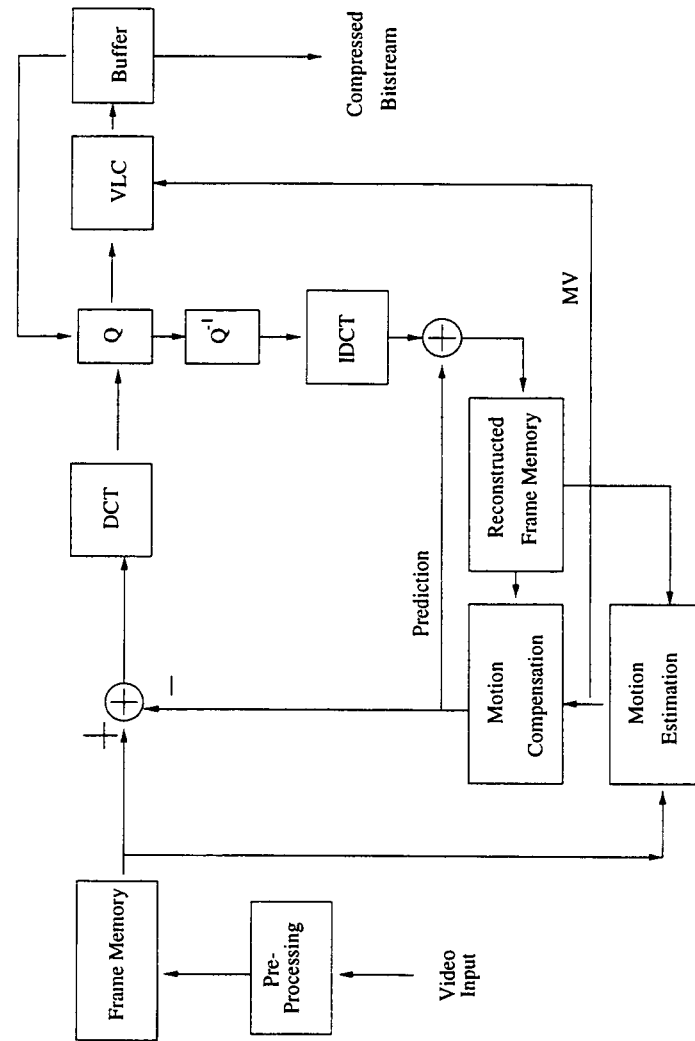


FIGURE 4.3. MPEG Encoder (P Pictures)

frame is coded like a still image. As P or B frame., the encoder does not code the picture directly. Instead it codes the prediction errors. Figure 4.3 shows a P coding. B frame can be coded very similar with more complicated ME methods.

As we noticed from the Figure 4.4, the decoder is very similar to the feedback loop of the encoder.

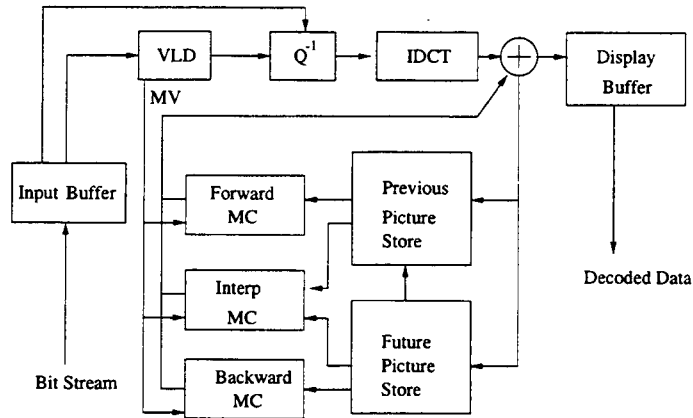


FIGURE 4.4. MPEG Decoder

4.5. Bit Stream Structure

The MPEG-1 syntax has a hierarchical representation with six layers (Figure 4.5: sequence, GOP, picture, slice, macroblock and a block layer.

Sequence a sequence of pictures. Information about frame size, pixel aspect ratio, quantization matrices, etc.

GOP or Group of Pictures, a set of pictures bracketed by I-frames. Permits random access.

Picture a single frame. Defines I, P, or B frame.

Slice a set of blocks within a frame. Header allows resynchronizations after errors

Macroblock a set of four luma blocks and two chroma blocks corresponding to the same spatial region of the picture.

Block a single 8x8 block for DCT coding.

The concept of the Group of Pictures layer does not exist in MPEG-2. It is an optional header useful only for establishing a SMPTE time code or for indicating

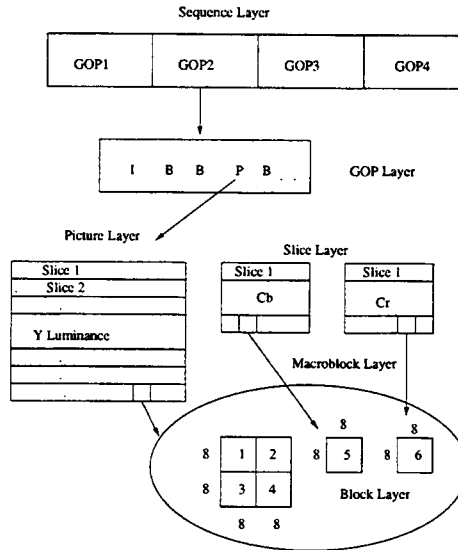


FIGURE 4.5. Bitstream Layer of MPEG-1

that certain B pictures at the beginning of an edited sequence comprise a broken link. This occurs when the current B picture requires prediction from a forward reference frame that has been removed from the bitstream by an editing process. In MPEG-1, the Group of Pictures header is mandatory, and must follow a sequence header.

Progressive frames are a logical choice for video material which originated from film, where all pixels are integrated or captured at the same time instant. Most electronic cameras today capture pictures in two separate stages: a top field consisting of all odd lines of the picture are nearly captured in the time instant, followed by a bottom field of all even lines. Frame pictures provide the option of coding each macroblock locally as either field or frame. An encoder may choose field pictures to save memory storage or reduce the end-to-end encoder-decoder delay by one field period.

To aid implementations which break the decoding process into parallel operations along horizontal strips within the same picture, MPEG-2 introduced a general semantic mandatory requirement that all macroblock rows must start and end with at least one slice. Since a slice commences with a start code, it can be identified by inexpensively parsing through the bitstream along byte boundaries. Before, an implementation might have had to parse all the variable length tokens between each slice (thereby completing a significant stage of decoding process in advance) to know the exact position of each macroblock within the bitstream. In MPEG-1, it was possible to code a picture with only a single slice. Naturally, the mandatory slice per macroblock row are restrictions also facilitates error recovery.

Motion vectors are now always represented along a half-pel grid for MPEG-2, A intrinsic half-pel accuracy can encourage use by encoders for the significant coding gain which half-pel interpolation offers.

In both MPEG-1 and MPEG-2, the dynamic range of motion vectors is specified on a picture basis. A set of pictures corresponding to a rapid motion scene may need a motion vector range of up to ± 64 integer pixels. A slower moving interval of pictures may need only a ± 16 range. Due to the syntax by which motion vectors are signaled in a bitstream, pictures with little motion would suffer unnecessary bit overhead in describing motion vectors in a coordinate system established for a much wider range. It later became practice in industry to have a greater horizontal search range than vertical, since motion tends to be more prominent across the screen than up or down. Secondly, a decoder has a limited frame buffer size in which to store both the current picture under decoding and the set of pictures (forward, backward) used for prediction by subsequent pictures. A decoder can write over the pixels of the oldest reference picture as soon as it no longer is needed by subsequent pictures for prediction. A restricted vertical motion vector

range creates a sliding window, which starts at the top of the reference picture and moves down as the macroblocks in the current picture are decoded in raster order. The moment a strip of pixels passes outside this window, they have ended their life in the MPEG decoding loop. As a result of all this, MPEG-2 created separate horizontal and vertical range specifiers, greater restrictions are also placed on the maximum vertical range than on the horizontal range. In Main Level frame pictures, in range $[-128,+127.5]$ vertically, and $[-1024,+1023.5]$ horizontally. In field pictures, the vertical range is restricted to $[-64,+63.5]$.

Also, *macroblock stuffing* or the “0000 0001 111” is a stuffing code and can be inserted into the bit stream wherever the encoder detects the possibility of a buffer underflow. Because the difficulty to implement in VLSI and the reduction of bit efficiency (This stream is ignored by the decoder). Stuffing is not permitted in MPEG-2.

4.6. Macroblock Coding

For the above discussion we assume that all the macroblocks with each frame are coded the same way. Actually, in a real application, even within a single I, B or picture, macroblocks can be coded differently. The decision trees for coding macroblocks are shown in Figure 4.6 and Figure 4.7.

I pictures are coded without MC, nevertheless, MPEG syntax allows each macroblock to be coded using a different effective quantization matrix. For the AC coefficients of DCT, the quantization step depends not only on the value of the corresponding element in the quantization matrix but also on the scale factor which is also referred to as MQQUANT. Thus, the macroblocks in I pictures can be coded using a new MQQUANT or using the old MQQUANT (Figure 4.6). After this stage

the DC coefficients are DPCM coded similar to JPEG. However, for AC coefficients, only the most frequent run/amplitude values are coded the same as JPEG. For the rest values are coded by an *escape* code followed by separate Huffman codes for the run-length and the amplitude.

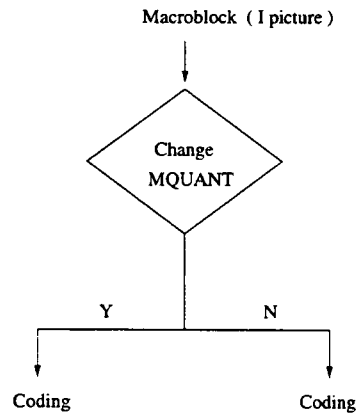


FIGURE 4.6. I Frame Macroblock Coding

For P pictures the encoder has more coding choices for each macroblock due to MC. First, decide if there is substantial motion between the two frames, if not, the MV can be set to zero. For example, if the background seldom changes between the frames, then there will not be bits to allocate to MV. If there exists a high level of temporal activity, in other words, the ME may fail (If the best match macroblock moved out of search window), the energy of prediction error may be greater than the macroblock; in this case, the decisions have to be made if the macroblock will be coded in intra-mode or inter-mode.

If the ME is very accurate and we almost have found the perfect match (Accurate MV), then after MC, the prediction error may be so small that the following quantization step will make every coefficient prediction error to zero. Thus, a macroblock like this can be skipped (without coding) to reduce the bits count to code the picture. Sometimes, the whole macroblock may not have to be quantized to

zero, instead, one or more blocks of the six blocks are all zero, in this case, a six-bit pattern referred to as the *coded block pattern* will indicate to the decoder which of the six blocks within the macroblock have been coded.

B pictures are very similar to the P pictures except that the MC mode, namely, forward, backward, or interpolated MC mode, has to be decided before all the above decisions can be made (Figure 4.7).

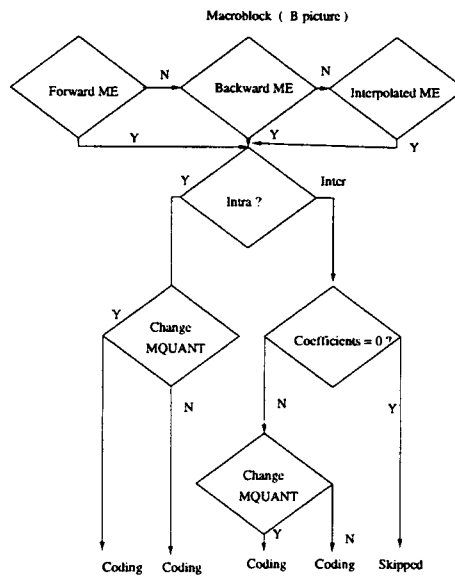


FIGURE 4.7. B Frame Macroblock Coding

4.7. Scalable Bit Streams

A new feature of MPEG-2 is bit stream scalability, which allows for a layered representation of the coded bit stream. The syntax allows four basic modes of bit stream scalability:

Data Partitioning The bit stream is split into two layers, one layer for critical header information (Such as headers and MV). This is intended for use in applications that can allocate two channels for a single bit stream.

SNR Scalability Support video transmission at multiple quality levels. All the layers have the same spatial resolution but different video qualities. This mode provides for better resilience to transmission errors. .

Spatial Scalability Support different spatial resolution transmission. Very similar to hierarchical JPEG coding mode where subsampled version can send first with enhancement follows..

Temporal Scalability All layers have the same frame size and chrominance formats but different frame rates.

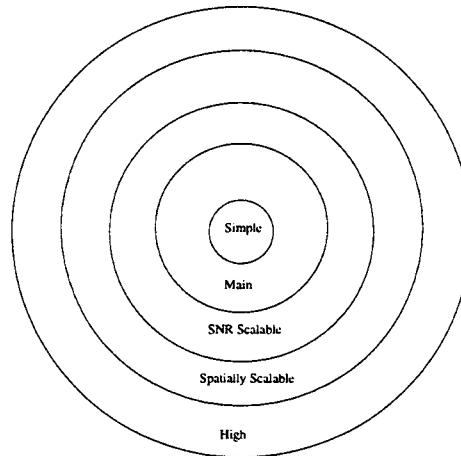


FIGURE 4.8. MPEG-2 Profiles

Considering the practicality of implementing the full syntax of the MPEG-2 specification, a limited number of subsets of the syntax are stipulated by means of “profiles” and “level”. A profiles is a defined subset of the entire bitstream syntax and a level is a defined set of constraints imposed on parameters in the bitstream (Figure 4.8).

Level	Profile				
	Simple	Main	SNR Scalable	Spatially Scalable	High
High (1920*1152)		X			X
High (1440*1152)		X		X	X
Main	X	X	X		X
Low (352*288)		X	X		

TABLE 4.1. MPEG-2 Profiles Levels

5. VLSI ARCHITECTURE FOR VIDEO COMPRESSION

5.1. Overview

The past few years have seen an explosive progress on digital video processing thanks to the rapidly improved VLSI technology. In this chapter, a VLSI implementation of the video compression algorithms discussed in previous part of this thesis (chapters 2, 3 and 4) is developed. Several methods of mapping algorithms to architectures are carefully evaluated and a real VLSI chip capable of real-time MPEG encoding and decoding is built and tested successfully.

The required silicon area for VLSI implementation of algorithms is related to the resources such as the number of logic gates, memory capacity, and the communication bandwidth for I/Os as well as between the modules. The number of operation becomes smaller when more of the priori knowledge about the specific algorithm is incorporated into the computation scheme. In addition to the operation part, we have to consider the memory and the interconnection between all modules. The memory requirements are influenced by the multiple access to original image data and intermediate results. According to the above consideration, the characteristics of video compression algorithms related to the hardware expense have been identified as computational rates, access rate and memory capacity. The high performance requirements of video coding schemes need special multiprocessor architectures with extensive parallel processing and pipelining. Mapping of video coding schemes onto multiprocessor systems can be based on data distribution as well as task distribution, or, dedicated (function specific).

In the case of data distribution, parallel processing is possible by assigning to each processor a subsection of an image. Thereby the image segments can be almost independently processed. The particular advantage of this method is that only one type of PE (Processing Element) is needed. For multiprocessor systems with identical PEs SIMD (single instruction multiple data stream) oriented control seems to be appropriate. SIMD systems have a common control unit for all PEs which offers an overall small silicon area for control. However, the disadvantages are caused by the data access and the operative part of the processor. This problem can be partly solved by assigning a local memory to each processor. By storing in the local memory all source data and intermediate results for processing of one macro block the video bus access rate can be restricted to the order of the source rate. This requires to store the search area for the block matching in the local memory.

The other method consists of mapping the functional blocks of video compression to different dedicated processors. This approach can be interpreted as pipelining at the macroblock level. A realization according to the task distribution can be optimized by adapting each processor architecture to the specific algorithm features such as algebraic theorems, linearity and symmetry. For video compression applications, this dedicated approach is mainly applied to the implementation of computation intensive tasks, namely ME and DCT/IDCT. The main disadvantage of dedicated architecture is the lack of flexibility.

Advantages of both the task distribution and data distribution approach can be combined by an hierarchical multiprocessor system architecture. On the first level of hierarchy data distribution is exploited for parallelization. Identical processors are connected by a bus system for distribution of video data and inter-processor communication. Each processor is assigned a local memory which contains one or a few macroblocks of image data. The second level of hierarchy is formed

within the processors. Each processor contains function specific modules and one flexible programmable control module for data dependent processing and module synchronization. Programmable architectures provide a significant higher flexibility compared to dedicated approaches, since, for example, modifications of the envisaged applications require software changes instead of a more cost intensive hardware redesign. Generally, this increased flexibility leads to a decreased architectural efficiency. This can be compensated by a combination of dedicated modules (typically for ME or DCT) and programmable modules for the remaining tasks of the hybrid coding scheme, like quantization, coder control, etc.. The design efforts and design errors will also be reduced by regular and modular architecture. Furthermore, the building block concept simplifies the design by use of high level synthesis tools [35].

5.2. Architectures for Motion Estimation

In this section we are going to discuss the implementation of core algorithms, or, Motion Estimation in VLSI. The different architectures will be discussed in detail and the tradeoffs will also be studied. Even the most powerful processors now can not adequately address the processing requirements of the ME. If we measure the complexity by the number of required RISC-like operations, for example, $r1 = a + b$, where a and b are data stored in external memory and $r1$ is the register, then the total operation will be 3 since we also count the two memory access. If we take H.261 encoder as an example, ME contribution more than half of the operations.

The inner loop operation, or, the accumulation of the Absolute Difference (AD) is very computational intensive. For NTSC resolution, the full search will need almost 50 GOPs (assume each $s = s + |a - b|$ takes 5 operations). Even after fast search method, such as hierarchical ME, this is still too mighty a job (Not

counting the half-pel ME!). For such high throughput, parallel architectures have to be explored.

There are ways to determine the feasibility of the architecture, first, the computation load, that is to say, the implementation has to be capable of the work. Second is the I/O bandwidth, data rate can not be too high. Third, the size, or, the number of gates, of the implementation can not be too large, last, latency can not be too long, in other words, the response time has to be fast enough to make real time ME possible. According to these standards, we will discuss the following architectures [28] [43]:

By studying the algorithm carefully we can find out that BMA has a very regular data-flow. Also the data in the search-window are used repeatedly in the computations of displaced block differences at different search-positions. These features can be exploited to result in very efficient and practical architectures such as linear array.

The key to utilize the regularity of the data flow, the data are piped to multiple PEs and used repeatedly so that memory access can be reduced. Without loss of generality, let us consider ME of one macroblock (16x16) and search range of -8 to +7 pixels (mostly used in videophone application). Figure 5.1 shows the pixel coordinates.

Because the pixels in the search-window are used repeatedly in the calculations, by using suitable control to broadcast these data to the required PEs, it relieves the burden of repeated accesses of the same data from the search-window for the multiple search-positions.

In Reference to Figure 5.2, more detailed description of the operation is given as follows:

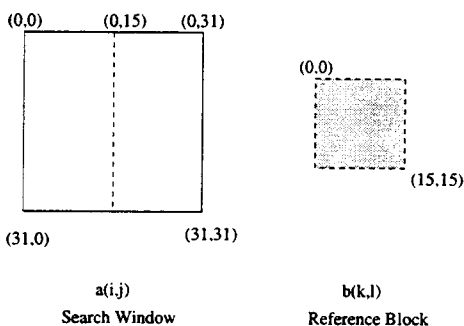


FIGURE 5.1. Data-flow Design

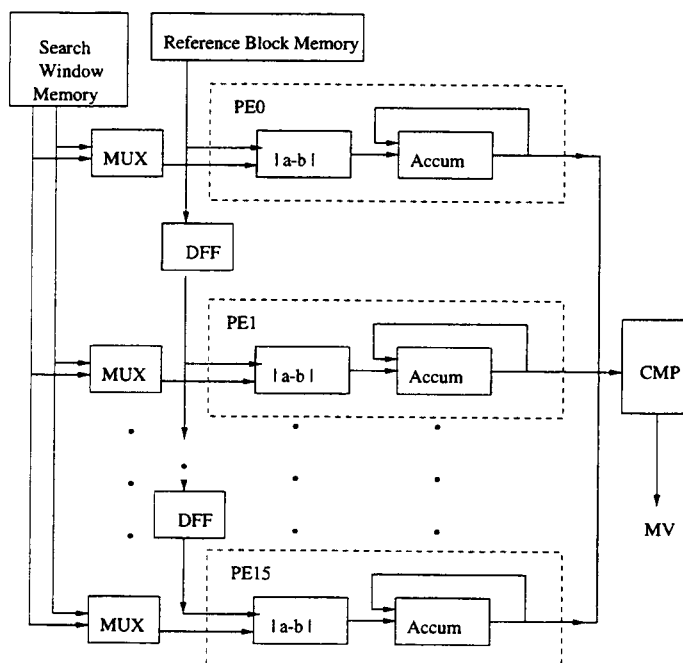


FIGURE 5.2. Architecture for ME using Linear Array

The following table shows the detail pixel data flow for the parallel computation of the first 16 distortion values. The pixel data are processed row by row. Reference block pixels move among PEs through a series of delays. At cycle 0, $a(0,0)$ and $b(0,0)$ are available to PE0, AD is performed in PE0, at cycle 1, $a(0,0)$ in DFF is used for PE2, at the same time, $b(0,1)$ is available to PE0. At cycle 15, pixel $a(0,0)$ reached the PE16 and from this time on the PEs will be fully utilized.

From cycle 16, all the input ports are active. Port $b1$ received the second row of the search and $b2$ continue the first row. This procedure is repeated until all the differences are completed then the MV is decided.

The reason two ports are used to search window memory is that the search window is bigger than the reference block memory, thus the I/O bandwidth is larger. Also the serial data can be inputed and thus reduce the pin count.

The basic idea of using the search-window data repeatedly can also be applied to a linear array architecture using the reference block data repeatedly. Figure 5.3 shows the architecture [28].

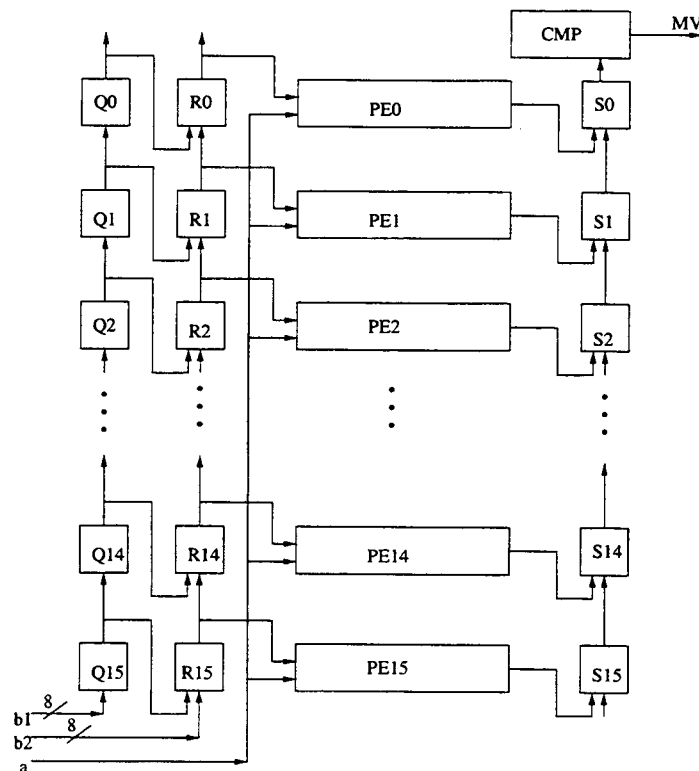


FIGURE 5.3. Architecture 2 for ME using Linear Array

Cycle Time	Input Data			Processor Inputs		
	a	b1	b2	PE0	PE1	PE15
0	a(0,0)	b(0,0)		a(0,0),b(0,0)		
1	a(0,1)	b(0,1)		a(0,1),b(0,1)	a(0,0),b(0,1)	
2	a(0,2)	b(0,2)		a(0,2),b(0,2)	a(0,1),b(0,2)	
.
.
.
14	a(0,14)	b(0,14)		a(0,14),b(0,14)	a(0,13),b(0,14)	
15	a(0,15)	b(0,15)		a(0,15),b(0,15)	a(0,14),b(0,15)	a(0,0),b(0,15)
16	a(1,0)	b(1,0)	b(0,16)	a(1,0),b(1,0)	a(0,15),b(0,16)	a(0,0),b(0,16)
.
.
.
31	a(1,15)	b(1,15)	b(0,31)	a(1,15),b(1,15)	a(1,14),b(1,15)	a(1,0),b(1,15)
.
.
.
256			b(15,16)		a(15,15),b(15,16)	a(15,1),b(15,16)
.
.
.
270			b(15,30)			a(15,15),b(15,30)

TABLE 5.1. Basic Data Flow for ME

In figure, $Q0 - Q15$ are latches while $R0 - R15, S0 - S15$ are latches with multiplexors to select one of the two inputs. In the very beginning 16 cycles, 16 data from the search window are shifted into the Q-latches. Then the data is parallel loaded into the R-latches and provided to the PEs concurrently. During the next 15 cycles, the contents of the R-latches are shifted up one at each clock cycle and b2 is shifted into the R15. The operation continues and we can show that the computation is correct. (The operations are very similar to the above table).

The above two methods are 1-D linear arrays each need $M \times N$ clock cycles to complete just one row of search positions, this is not enough for the higher sample-rate or larger search window applications. Thus we can take the idea of 1-D linear Array and extended into 2-D arrays which shows in the Figure 5.4:

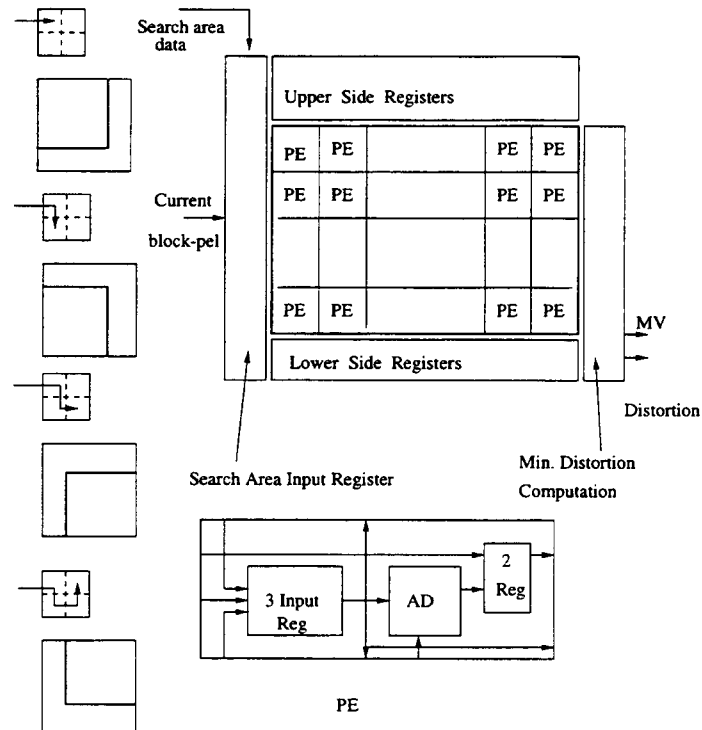


FIGURE 5.4. Architecture for ME using 2-D Linear Array

The reference block data are sent to all the PEs at each cycle, the sequence of the reference block and the search location are also illustrated in the figure. Each PE can accept a search-window pixel from its left, upper, or lower PE. At the beginning of the operation, each column of PEs and upper side-registers contain a column of search-window data, when the first reference block pixel comes in, note that the number of PEs equals to search locations, or, $(2p+1)^2$ and all the absolute difference contain $a(0,0)$ are computed, that is, $a(0,0) - b(0,0)$, $a(0,0) - b(1,0)$, $a(0,0) - b(1,1)$ etc are computed. After N cycles, the result will come out and another column of search window data are loaded.

In order to support real time MPEG-2 IPB AFF encoding of CCIR601 picture, especially when the search range is large and with half-pel accuracy, all the above architectures' performance is not enough. Although we can extend the architecture by extend the number of PEs, this is not the best approach for high data rate applications. In Figure 5.5 we are going to develop a new architecture with very high computation ability. The tradeoff is that the I/O bandwidth will get large too. However, with the advance of VLSI technology, this problem can now be overcome. Also, smart memory access can also take advantage of the regularity of the data flow and as a result reduce the I/O bandwidth requirement.

We assume that there is enough memory to hold the reference image data and part of the search window data. Thus we form an architecture to provide continuous data-flows for various block-size MEs by using a fixed-size computation unit, that is, an 8x8 block. The search can be 8x8, 16x8, 8x16 and 16x16 to cover all the cases (Since we are dealing with high resolution pictures, even at the lowest resolution ME level, 8x8 blocks are needed instead of the 4x4 block). The 8x8 blocks absolute difference is computed and the result is added together, first all the values of the same row are added together and then the 8 rows result can be added

together (Figure 5.5). The adder implementation can be any fast adder scheme such as: binary adder trees (3 level) or carry save adder (CSA), carry lookahead adder (CLA) or carry select adder (CSEA) just to name a few [24] [22].

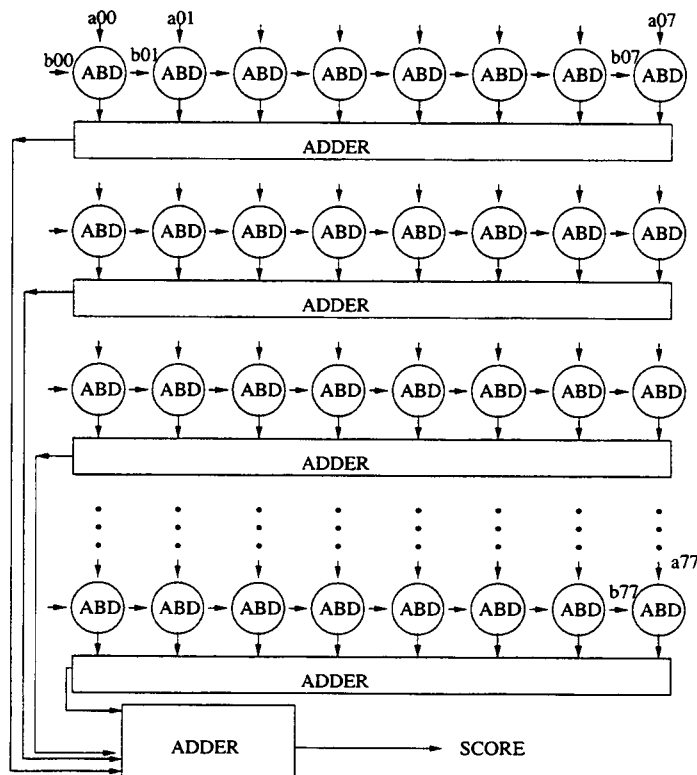


FIGURE 5.5. New Architecture for ME

5.3. Mapping Algorithms to VLSI

The required silicon area is related to the required resources such as logic gates, memory, and the interconnect between the modules. The amount of the logic depends on the concurrency of operations [35]:

$$N_{con,op} = R_s \times N_{op,pe} \times T_{op},$$

where R_s is the source rate in pels per time unit, $N_{op,pel}$ is the number of operations per pel and T_{op} is average time for performing an operation. This depends on the video format.

The required interconnects between the operative part and the memory depends highly on the access rate. If we store the the video data and intermediate data (such as ME computation data, difference data between frames) in one big external memory (for example, DRAM). Then the number of parallel bus lines for connecting the operative part and the memory becomes approximately

$$N_{bus} = R_s \times N_{acc,pel} \times T_{acc},$$

where $N_{acc,pel}$ specifies the mean number of accesses per pel and T_{acc} the memory access time. Thus the number of bus lines will become too large to be practical. Taking into consideration that the access rate is mainly influenced by multiple accesses of image source data and intermediate results, the access rate to an external memory be essentially reduced by assigning a local memory to the operative part. The memory size depends on the specific access structure of the coding algorithm. Concurrency of operations can be achieved by architectures with parallel processing or pipelining. This goes back to the different architectures such as task distribution (See Figure 5.6).

A cascade of processors will be derived which can easily operate in pipeline by placing between the processors a memory of appropriate size. However, by parallel operation of processors where a subsection of the image is assigned to each processor, the image segments can be processed separately. A local memory is assigned to each processor for the image data and intermediate data. This is also shown in Figure 5.6.

We can develop the above ideas one step further and thus lead us to the hybrid architecture showed in the Figure 5.7.

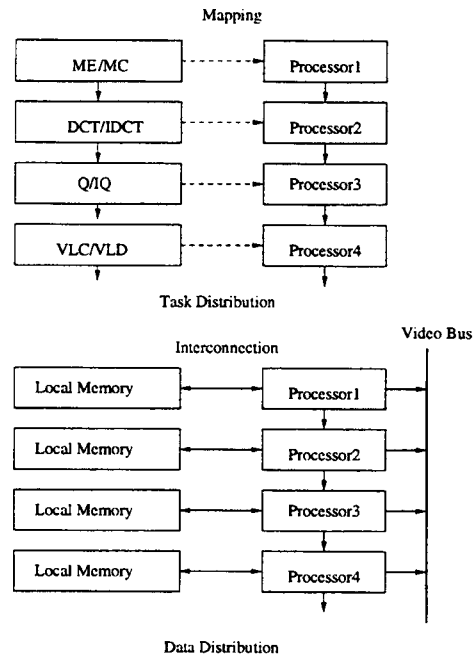


FIGURE 5.6. Task Distribution and Function Distribution

We need to define the efficiency of the architecture. First, from the practical point of view, because the silicon area is directly related to the cost. This leads to the well known AT -product for architecture assessment, that is,

$$E = \frac{1}{A_{si} \times T_p},$$

or in other words: $A_{si} = \alpha_T \times R_T$, where the throughput R_T is inversely proportional to the processing time for one sample (T_p). This shows that area A_{si} increases (parallel implementation) will increase the through-put. Since we have to include the rapidly advanced semiconductor technology, if we assume a linear scaling of voltages and doping concentration, in the “constant voltage” model of scaling of MOS transistors the gate delay scales as $\frac{\lambda}{\lambda_0}^2$, plus some practical, such as, interconnection delay, power requirement constraints on speed and the memory parts, etc.. As a result, we can get

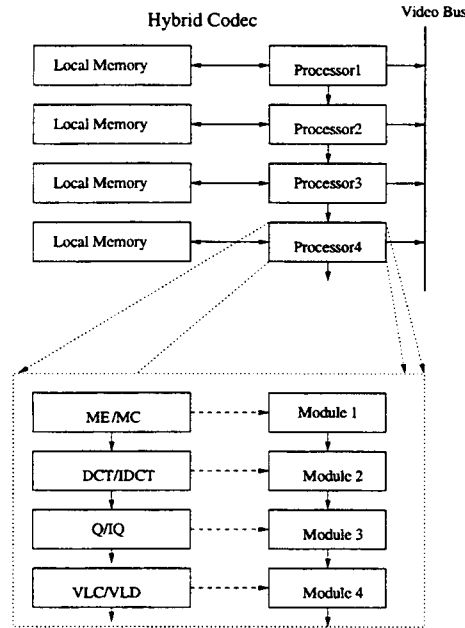


FIGURE 5.7. Hybrid Architecture

$$A_{si} = \alpha_{M,0} \times C_M \times \frac{\lambda^2}{\lambda_0} + \alpha_{T,0} \times R_T \times \frac{\lambda^{3.6}}{\lambda_0},$$

where C_M is the memory capacity.

In summary of what we have discussed, to improve the performance, there exists several ways; however, it also depends on what the major goal we want to achieve is. If we want to improve the through-put, we can:

- 1) Increase the clock frequency. This can be done by intensive pipelining.
- 2) Increase the on-chip memory, reduces the external memory access time.
- 3) Parallel data paths, exploits data distribution.
- 4) Increase the width of bus.
- 5) Dedicated modules such as ME and VLC/VLD, or floating point unit.

For the above 5 different methods, there are also existing tradeoffs. For example, the maximum clock frequency is limited by the maximum delay between two successive register stages. The intensive pipelining also introduces longer latency

of the circuit which may not be acceptable. Large memory on chip will influence the yield, parallel data paths will increase the silicon area thus increasing the cost where dedicated modules will limit flexibility.

The silicon technologies and the design methodologies play a very important role in video compression, especially with the intensive use of sub-micron technologies associated with fast on chip clock frequencies and huge numbers of transistors on the same substrate affects traditional methods of designing chips. This will not only change the hardware design process but also the use CAD tools [12].

The clocks that drive IC's are increasingly fast (100MHz is very common now), and CMOS devices scale down into the submicron region (0.35 μm process is the standard process). Therefore some important parts of each digital submicron chip have to be considered to be working in the analog mode rather than digital. As a result, lots of effects used to be neglected by digital designers, such as power management, interconnections and packaging have to be carefully considered.

The interconnections can be modeled by RC circuits, this method is still used to derived the clock trees to manage the clock skew. Because of the scaling, resistance plays a more and more important role. For example, fringing fields and the contributions of neighboring lines must also be included in the calculation. However, when the wire lengths increase, their inductance starts to play a major role. If the signal propagates from a source to a load and back in more than one-tenth of the time it takes the signal to rise, then transmission-line effects need to be taken into consideration. These effects include reflections, overshoot, undershoot, and crosstalk. Electromagnetic effects have to be analyzed in a first step but it might be necessary to use another circuit analysis to gain better insight into circuit behavior such as electromagnetic characteristics. However, the completed analysis of a whole chip using 3-D electromagnetic methods is not feasible.

A common method used to reduce the effects of interconnection is MCM, or Multichip Modules Chip, which puts two or more chips on the same die and packaged together. Because the delay between one chip and another on a board is one magnitude larger than on-chip delays, thus MCM can greatly improve electrical performance by drastically shortening interconnection lengths. Thus the corresponding signal reduces “time of flight” and results in having faster circuit switching speeds. Degrading noise and crosstalk are also minimized. A lowering of line capacitance also makes it possible to reduce power consumption.

Simultaneous switching noise, or, ground bouncing is another problem introduced by the inductance. The output driver has to be carefully designed so as to avoid false triggering, double clocking, or missing clock pulse. An in-depth analysis of the chip-package interface (Figure 5.8) is required to ensure the functionality of high-speed chips.

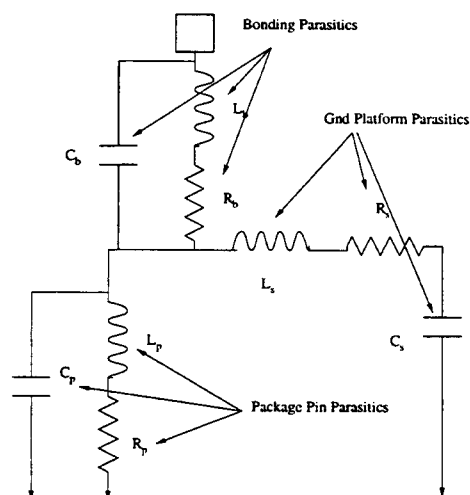


FIGURE 5.8. Chip Package Interface

From the above discussion we can find out that some important parts of each digital chip have to be considered to be working in analog mode rather than strictly digital. This is true for I/Os and is also true for the timing and clocking blocks in a

system. The entire floor plan has to be analyzed in the context of noise immunity, parasitics, and also propagation and reflection in the buses and main communication lines.

5.4. Low Power Design

Equally important as high speed, low power requirement has become one of the major design issues for VLSI. Especially the actual demand for portable consumer products implies that power consumption must be controlled at the same time that complex user interfaces and multimedia applications such as video compression are driving up computational complexity [1] [2].

In analog circuits, a desired SNR must be maintained, while for digital CMOS circuit, although the peak power consumption is very important for low-power design, the more critical element is the time averaged power consumption which is directly proportional to the battery weight and volume required to operate circuits for a given amount of time. There are four sources of power dissipation in digital CMOS circuits, that is,

$$P_{avg} = P_{dynamic} + P_{short_circuit} + P_{leakage} + P_{static},$$

where

$$P_{dynamic} = \alpha_{0 \rightarrow 1} \times C_L \times V \times V_{dd} \times f_{clk}.$$

Charging and discharging parasitic and loading capacitors, $P_{dynamic}$ represents the switching component of power, where C_L is the load capacitance, f_{clk} is the clock frequency and $\alpha_{0 \rightarrow 1}$ is the node transition activity factor, or the number of times the node makes a power consuming transition in one clock period, V is the voltage swing which in most cases equal to V_{dd} or the supply voltage. However,

there may be some cases when the voltage swing is less than supply voltage, such as pass-transistor implementation.

$$P_{short_circuit} = I_{sc} \times V_{dd}.$$

Due to short circuit current I_{sc} which arises when both the NMOS and PMOS transistors are simultaneously active, $P_{short_circuit}$ is also called P_{direct_path}

$$P_{leakage} = I_{leakage} \times V_{dd}.$$

The leakage current, $I_{leakage}$ comes from reverse bias diode currents and sub-threshold effects. This mostly depends on the process technology.

$$P_{static} = I_{static} \times V_{dd}.$$

This will rise if the logic used in the circuit has current source such as bias circuitry or pseudo-NMOS logic families.

Among these four components, the dynamic power will dominate and will contribute to more than 90 percent of the total power consumption thus making it the primary target for power reduction. In the following section, we focus on how to reduce the dynamic power.

From the above equations, the dynamic power arises when energy is drawn from the power supply to charge the parasitic capacitors C_L ,

$$C_L = C_{gate} + C_{diffusion} + C_{interconnect}.$$

First and foremost, from the equation of dynamic power we can tell that the supply voltage has the greatest impact on the power. The dynamic power is proportional to the square of the V_{dd} . However, this comes with a cost: the scaled down of the V_{dd} will also reduce the circuit speed and thus reduce the throughput.

As a result, the power-delay product is a better indicator of the performance. Thus the design objective becomes that of reducing power consumption while maintaining the overall system throughput through appropriate architectural, advanced process technology, careful sizing of transistors and the in-depth analysis of signal transition activity.

It is found that a simple first-order derivation adequately predicts the experimentally determined dependence and is given by:

$$T_d = \frac{C_L \times V_{dd}}{I} = \frac{C_L \times V_{dd}}{\frac{\mu C_{ox}}{2} \times (W/L)(V_{dd} - V_t)^2}$$

From this equation we can find out that to compensate for the increased delays at low voltages, parallel and pipelined architectures can be used. Computation can be done in n parallel functions, each of them operating n times slower. The circuit operates at a lower rate thus reduce the frequency and as a result to reduce the power. The downside is that the circuit overhead for the control and communication task grows for parallel approaches and latency gets longer for pipelining. Another approach is to modify the threshold voltage of the devices. Reducing the threshold voltage allows the supply voltage to be scaled down. However, the tradeoff is lack of noise margins and the increase in subthreshold currents. This will in turn result in significant static power dissipation.

Since CMOS circuits do not dissipate power if they are not switching, a major focus of low power design is to reduce the switching activity to the minimal level required to perform the computation. One method is to use gated clock to power down the module that is not active when the other module is working. Another important factor is $\alpha_{0 \rightarrow 1}$, or transition activity. Factors affecting α are as follows: type of logic function, type of logic style (Static logic or dynamic logic), signal statistics, circuit topology (the manner in which logic gates are interconnected) and

inter-signal correlation (correlation exist between values of a temporal sequence of data, since switching should decrease if the data is highly correlated). For example, video and audio signals are highly correlated. Another example is the memory access address, when one reads from or write to DRAM, for fast page mode, the column address signals switching one bit at a time if we are access one section of the memory.

At layout level, the place and route should be optimized such that signals that have high switching activity (such as clocks) should be assigned short wires and signals with lower switching activities can be allowed progressively longer wires, therefore, new CAD tools need to develop to reduce the overall capacitance that is switched.

At circuit level, transistors need to be carefully sized. It is important to equalize all delay paths so that a single critical path does not unnecessarily limit the performance of the entire circuit. Minimum size should be used wherever possible to reduce the parasitic capacitances without degrade the speed. Chandrakasan etal, [2] showed that if the total load capacitance is not dominated by the interconnect, the minimum sized devices resulted in the optimized sizing.

Another approach to reduce power is to reduce the voltage swing on the output nodes. This will be very effective on the internal buses. For example, NMOS is used so the output will be $V_{dd} - V_t$ instead of full swing V_t . The following figure shows a simplified schematic of such a gate, used in the FIFO memory cells of Berkeley low-power cell library. This circuit uses a precharged scheme and the transistor M3 is used to clip the voltage of the bit-line to $V_{dd} - V_t$. The transistors M1 and M4 are used to precharge the internal node which is the input of the inverter to V_{dd} . During evaluation (CLK = "1"), if V_{in} is high, the bit-line will start to discharge. Because the bit-line is heavily loaded, the capacitance ratio of the bit-

line to the internal node is very large, once the bit-line has dropped roughly 200 mV to sufficiently turn on M3, the internal node quickly drops to the potential of the bit-line, providing signal amplification thus also increase the speed (Figure 5.9).

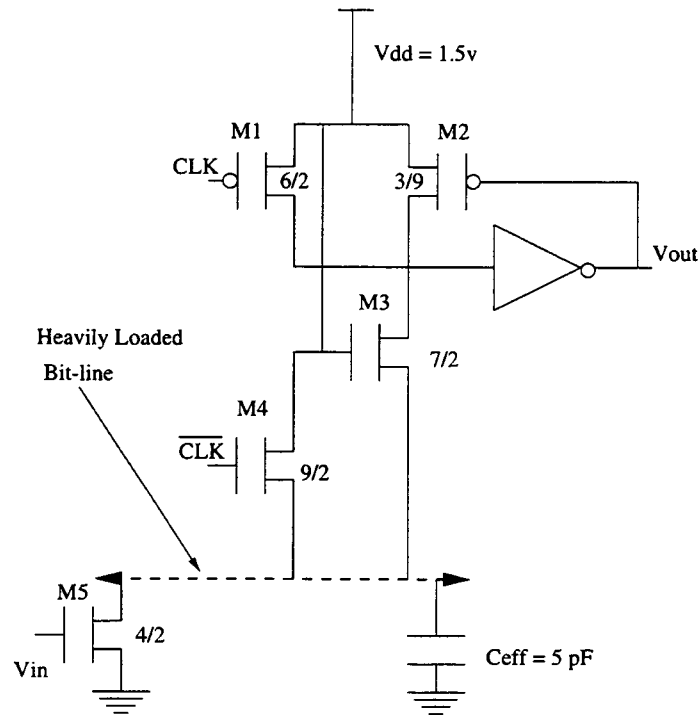


FIGURE 5.9. Voltage Swing Reduction Circuit

6. VLSI IMPLEMENTATION OF VARIABLE LENGTH DECODER

6.1. Overview

As we have discussed in Chapter 2, the Variable Length Coder (VLC) can achieve very good compression efficiency by combining RLC and Huffman coding together. Because of its lossless nature, it is also called entropy coding. The VLC is the last stage in the encoding pipeline of JPEG and MPEG (Chapter 2, Chapter 3 and Chapter 4). As a result, Variable Length Decoding (VLD) is the first stage in the decoding pipeline of MPEG applications. Since there is no explicit codeword boundary in the variable length coded bit stream, the VLD is more difficult to implement than the VLC. The VLD must decide the codeword length to extract the code and align the following bit stream for the next decoding, therefore, it is a recursive data dependent procedure for which the decoding speed is limited. In this chapter, a high-performance, low-power and memory-efficient VLD has been designed for MPEG applications. It has higher performance by using parallel and pipeline architectures, furthermore, with careful analysis of MPEG bitstream pattern, efficient on-chip memory usage has been achieved. At the same time, low-power design techniques (Chapter 5) have been implemented to reduce the power consumption of this highly active module.

6.2. System Model

First, a simple Huffman coding example is given in Figure 6.1.

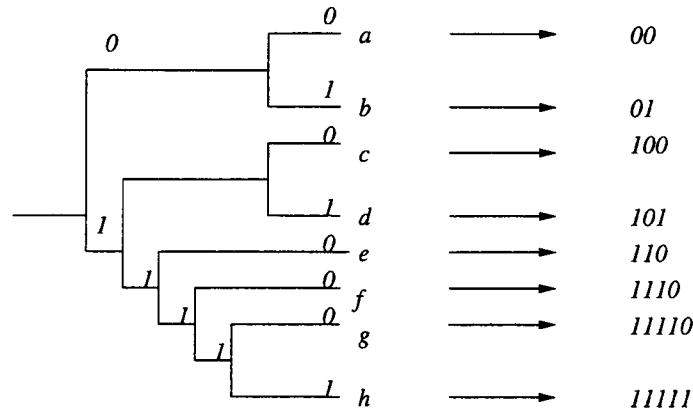
*Decoding Tree**Encoding Table*

FIGURE 6.1. Simple Huffman Coding example

The VLD used in MPEG application is much more complicated, it has to be able to cope with many MPEG audio and video bitstream corner cases. In order to achieve high-performance, the VLD we designed can automatically decode MPEG bitstreams up to Macroblock layer level (Chapter 4). Figure 6.3 is the system model of VLD in C-like pseudocode which shows the main function block of VLD in MPEG application. Notice that only the video decoding procedure is showed.

In Reference to Figure 6.2, during MPEG block layer video decoding, the DC coefficients are differentially coded in a way similar to the method employed in JPEG, as we discussed in Chapter 2. That is, each differential value is coded using a (size, level) pair. Then the size is Huffman coded and defines the number of bits used by the level. As in JPEG, the AC coefficients are scanned in zig-zag order and coded using run-length and level information. However, the coding details in MPEG are different from JPEG. MPEG does not use any size information, instead it provides Huffman codes for the most frequent run/level values. Run/level values

```

block(i) {
  if ( pattern_code[i] ) {
    if ( macroblock_intra ) {
      if ( i < 4 ) {
        dct_dc_size_luminance
        if ( dct_dc_size_lumainance != 0 )
          dct_dc_differential
      } else {
        dct_dc_size_chrominance
        if ( dct_dc_size_chrominance != 0 )
          dct_dc_differential
      }
    } else {
      Frist AC coefficient
    }
    while ( nextbits() != EOB )
      Rest AC coefficients
    EOB
  }
}

```

FIGURE 6.2. Block Layer Pseudocode

that are not listed in the table are coded by an escape code (Figure 6.3), followed by separate Huffman codes for the run-length and the level.

A more detailed description of the operation of Figure 6.3 is given in previous chapters (Motion estimation and motion compensation in Chapter 3 and MPEG standards in Chapter 4). In the following sections, the implementation will be given in detail.

6.3. VLSI Architectures

Traditionally, VLD is implemented through a tree searching algorithm as the input bits are received serially (k bits at a time). This is known as constant-input-rate VLD. The decoding process can be considered as traversing down a path of Huffman tree from the root, the route determined by the input encoded bitstream. Figure 6.1 is a simple example where $k = 1$.

We can implement the whole decoder as a pipelined decoding tree, where one pipeline stage corresponds to one level of the tree and can be implemented by a read-only memory (ROM). This method is still quite popular when the data rate is not too fast. However, for MPEG-2 audio/video decoder, other parallel architectures have to be used because of the vast amount of data that has to be processed.

Due to the variable-length property of the VLC code, the input and output speed of the decoder can not both be kept constant, therefore if we have variable-length input words, the constant-output-rate VLD can be achieved. Sun and Lei [38] used a programmable logic array (PLA) based constant-output-rate parallel architecture.

As Figure 6.4 shows, the output rate is fixed, one codeword every cycle. The barrel shifter stores a 32-bit window of the input data, where the PLA takes 16-bit as its input to decode one codeword. Since 16-bit is the maximal codeword length, it is guaranteed that at least one codeword can be decoded. The output of the PLA also includes the actual length of the decoded codeword. The length is accumulated and used to memorize the new starting position within the 32-bit window in order to get the 16-bit for decoding. The carry bit of the accumulator is used to issue a data fetch command and get the next word from the input buffer. The mapping

in PLA is done in parallel and one codeword is decoded each cycle hence the name constant-output-rate parallel architecture.

However, Sun and Lei [38] claimed that because of the recursive operations of VLD, pipeline can not be implemented. After careful study of MPEG bit patterns, we have implemented a new VLD architecture which can take the advantage of both pipeline and parallel structures at the same time.

A block diagram of new VLD is shown in the following Figures 6.5, 6.6 and 6.7.

Since the input to VLD is not constant, a first-in-first-out buffer (FIFO) of 32 words deep is added for VLD. The decoder receives the input bitstream from DRAM through this FIFO. A request for new data is generated when the FIFO is half empty. The control register *bitcount* is a pointer to the next bit to be read in the delay register. A 5 bit adder (Figure 6.6) is used to update the bit pointer as codeword are decoded.

Because the VLD can handle the macroblock layer decoding, an on-chip ROM has to maintain the tables for MacroBlock Address (MBA), Macroblock type I, P, B and D pictures, Coded Block Pattern (CBP), DC and AC coefficients for both luma and chroma components and Motion Vector code (MV) for both MPEG-1 and MPEG-2 (Chapter 4).

If we do not consider the 23-bit start code, the MPEG codeword is from 2 to 16 bits (without sign bit). The VLD table is mandatory to match all codewords in the codebook, but without pre-processing for the incoming bitstream, the look-up memory size could reach 65536 words by direct mapping all codewords in the codebook. For each MPEG decoding table, different ROM has to be used, the result means low speed, wasted memory resource and large die size.

Previous papers, proposed various algorithms for preprocessing of input bit stream of VLD [11] [40] [19] [6]. These algorithms utilize the clustering scheme to partition the codewords into groups by the same bit patterns or a fixed amount of bits. In other words, the VLD is simplified into two-step processing. First, the bit pattern in a codeword is recognized or a fixed number of bits is processed in the bit stream, then they are used as the first reference for the memory searching. After the bit pattern or fixed amount of bits is removed, the remaining bits in the codeword are used for the second step searching by accessing the smaller size memory. However, sometimes extra information is needed in order to determine if a codeword has ended. In what follows we will describe a different approach which can accomplish faster decoding with efficient memory usage. High throughput is achieved by pipelining and memory saving by exploring the special characteristics of MPEG code tables.

Table 6.1 is part of a variable length code for motion-code for MPEG-2 application.

As shows in Figure 6.6, the barrel shifter is used to extract 5 LSBs of the previous code and the next 23 bits of data from the bitstream. The 23 bits are used to determine the length of the code currently being decoded. The bit pointer is then updated with the length, so it points to the next symbol in the following cycle.

In the next pipeline stage, as the bit pointer has been updated, the barrel shifter produces 5 LSBs of the code currently being decoded. The 5 LSBs, along with the length from the previous cycle and the command are used to index into the VLD table stored in the decoder ROM. The output of the ROM is multiplexed with the escape code values to generate an 18-bit data. The 6 MSBs of the 18 bits are

Variable length code	motion-code[r][s][t]
0000 0011 001	-16
0000 0011 011	-15
0000 0011 101	-14
0000 0011 111	-13
0000 0100 001	-12
0000 0100 011	-11
0000 0100 11	-10
0000 0101 01	-9
0000 0101 11	-8
0000 0111	-7
0000 1001	-6
0000 1011	-5
0000 111	-4
0001 1	-3
...	...

TABLE 6.1. Variable Length Codes for Motion-code

set to the run value and the 12 LSBs are set to the level value. Only positive values of level are stored in ROM and if the decoded level is negative, a two's complement adder is used to generate the signed level. Thus, the ROM size can be greatly reduced.

6.4. VLSI Implementation

VLD is designed to run at a clock speed of 81MHz and a gated clock has been used to power down when the VLD module is idling. Furthermore, after careful analysis, under the condition that we can meet the data rate requirement of MPEG-2, the VLD pipeline is designed to run at half speed which is 40.5 MHz as shows in Figure 6.8 thus greatly reducing the power consumption. This is verified through the equation (Chapter 5):

$$P_{dynamic} = \alpha_{0 \rightarrow 1} \times C_L \times V \times V_{dd} \times f_{clk}.$$

We can draw the conclusion that if we reduce the f by half, the $P_{dynamic}$ will also cut in half.

In Chapter 5 we discussed the importance of the design methodology. In this VLD design, at algorithm level, C language is used to model the system, then the hardware is implemented according to the specifications of architecture using hardware description language Verilog. Part of the start code length decode Verilog file is given in Figure 6.9.

The Verilog code shown here implements a counter. Other types of finite state machine (FSM) can be implemented easily by following this example. Such codes are then remodeled into more structured descriptions, known as register transfer level (RTL) descriptions. Then the language is mapped to schematics using CAD tools such as Synthesis. After this step, circuit layout is generated, place and route

is done to connect the module. Along these steps the simulations and validations have to be carried out carefully. The interconnections have also been modeled and simulated carefully. VLD is also capable of decoding audio bitstream including MPEG-1, MPEG-2 (both Layer I and Layer II) and Dolby AC-3 5.1 bitstream. However, only the video decoding unit is discussed in this chapter. The VLD is fabricated in triple layer metal 0.35 μ CMOS technology, $V_{dd} = 3.3v$ with a size of 1598 $\mu \times$ 2897 μ . Appendix A is the actual layout of VLD.

```

Macroblock(){
  while( nextbits() = '0000 0001 00')
    macroblock_escape
  macroblock_address_increment
  if(macroblock_motion_forward or macroblock_motion_backward){
    if.picture_structure == '11' {
      if(frame_pred_frame_dct == 0)
        frame_motion_type
      }
    else {
      field_motion_type
    }
  }
  if(( picture_structure == "Frame picture" ) &&
    (frame_pred_frame_dct == 0) &&
    (macroblock_intra or macroblock_pattern)) {
    dct_type
  }

  if(macroblock_quant)
    quantiser_scale_code
  if(macroblock_motion_forward or (macroblock_intra && concealment_MV))
    MV(0)
  if(macroblock_motion_backward)
    MV(1)
  if(macroblock_intra && concealment_MV)
    marker_bit
  if(macroblock_pattern) {
    if(chroma_format == 4:2:2)
      coded_block_pattern_1
    else
      coded_block_pattern_0
  } // chroma_format == 4:4:4 is not supported
  for ( i = 0; i < block_count; i++) {
    block(i)
  }
}

```

FIGURE 6.3. System Model Pseudocode

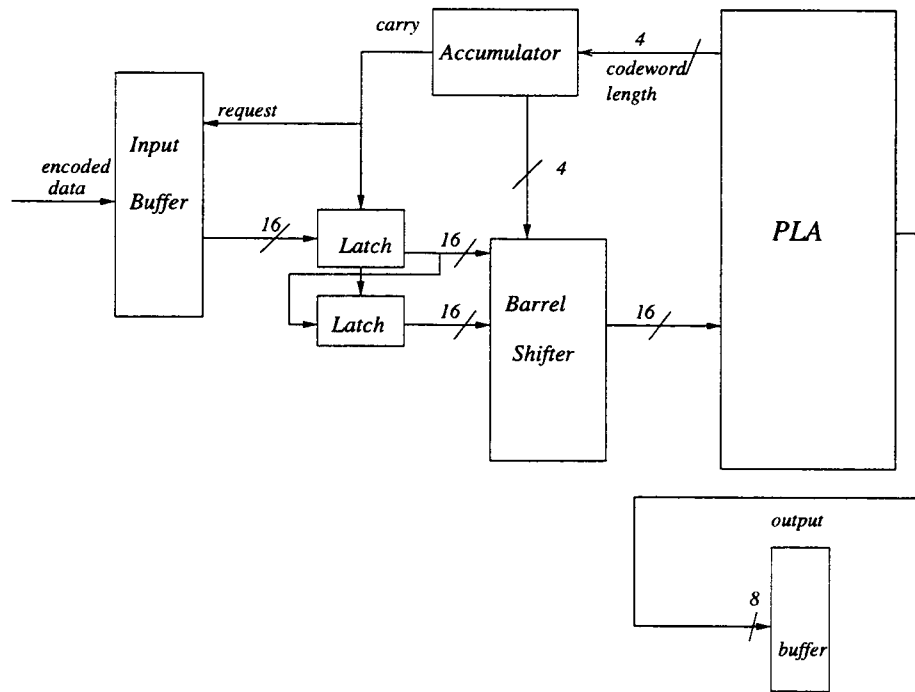


FIGURE 6.4. PLA-based constant-output rate VLD

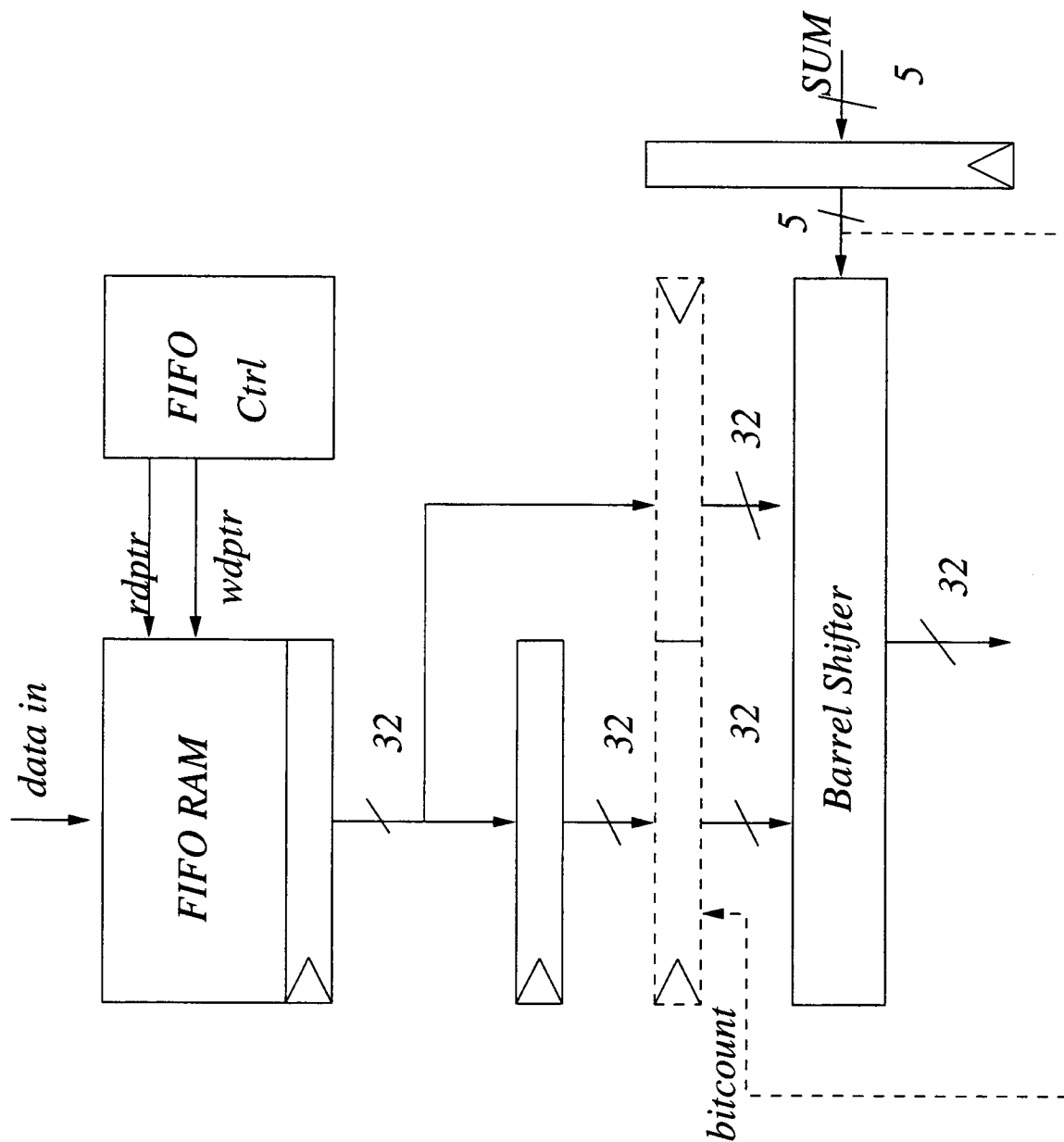


FIGURE 6.5. Barrel Shifter and Control Logic

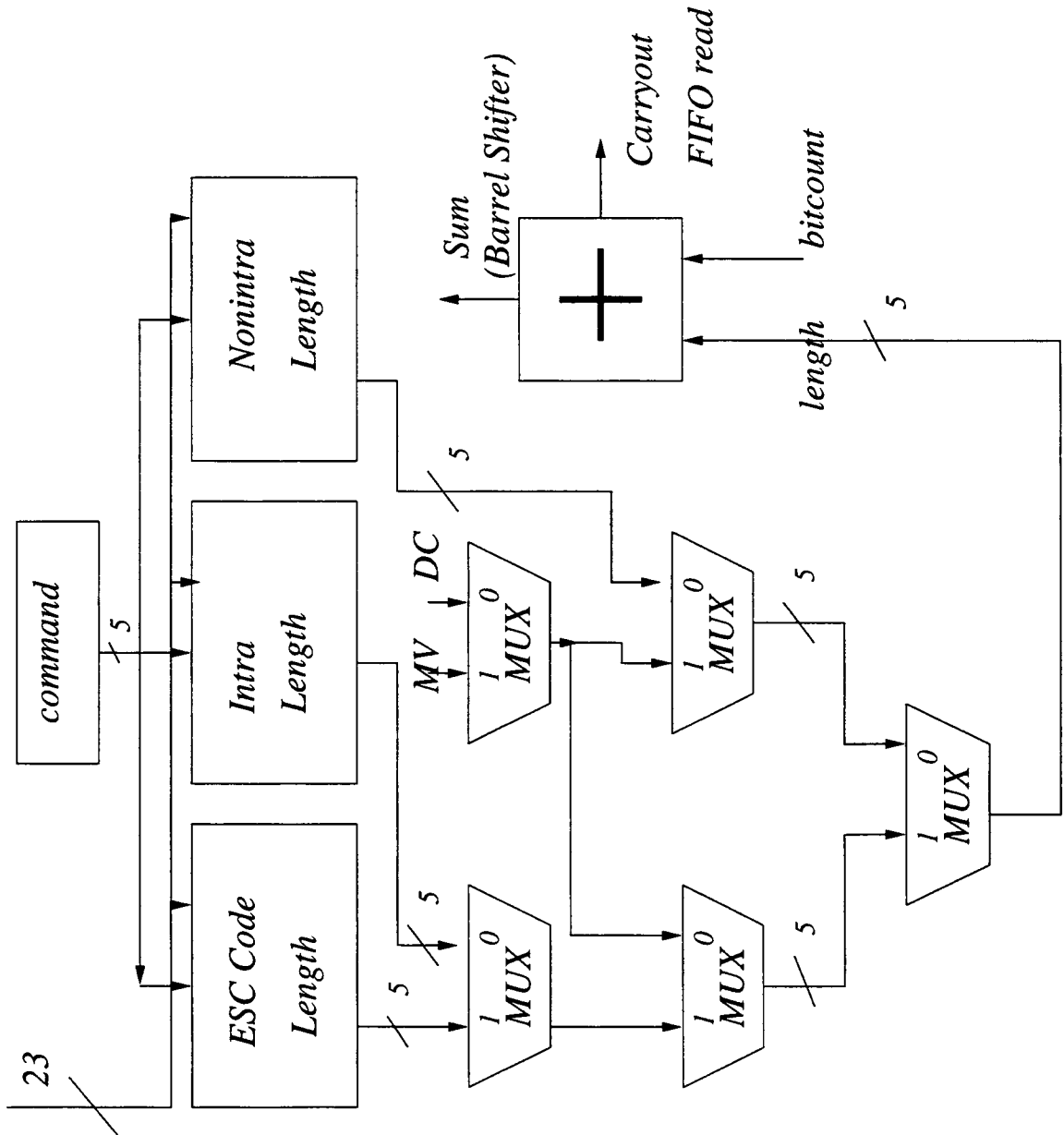


FIGURE 6.6. Architecture of VLD

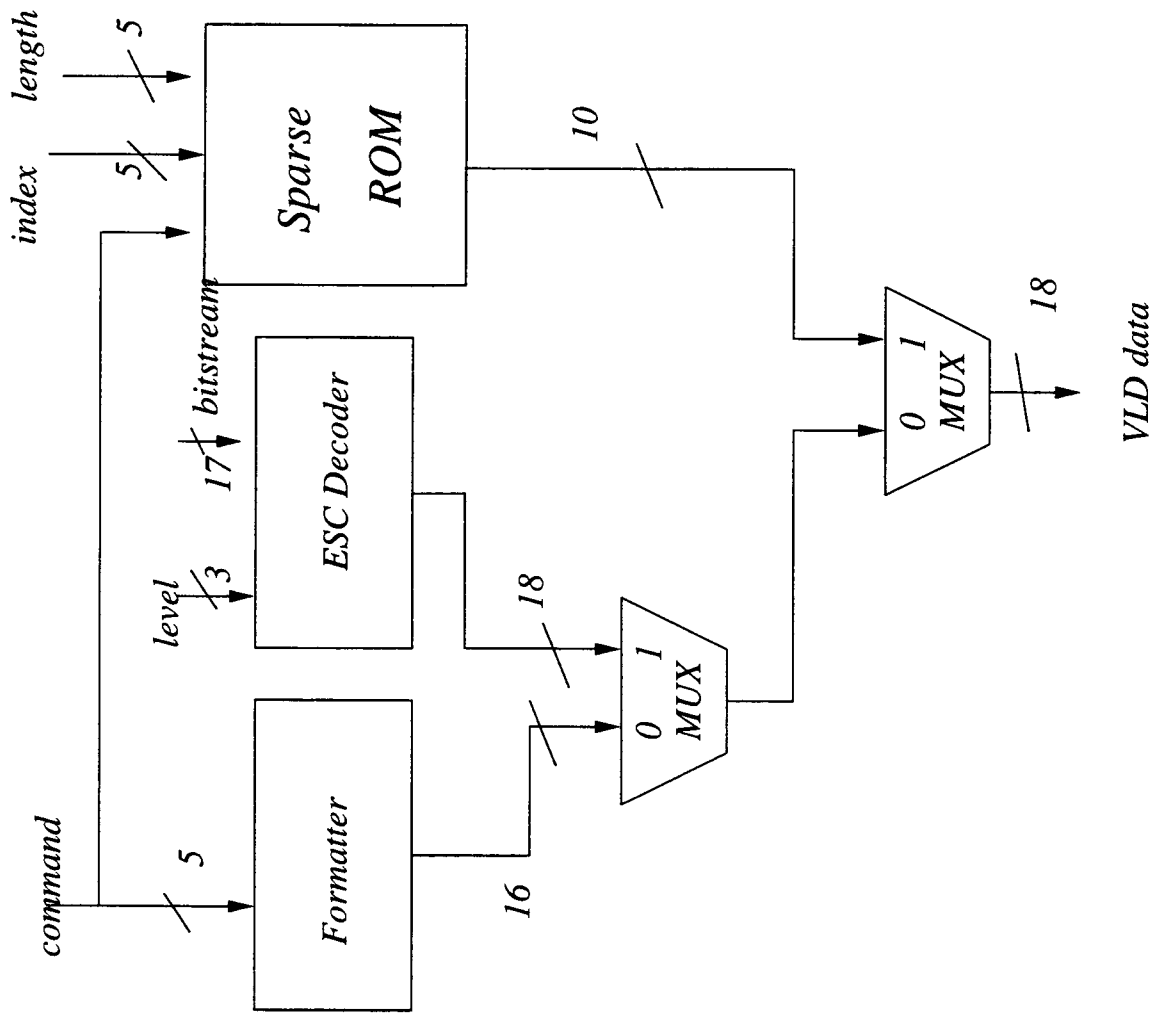


FIGURE 6.7. Architecture of VLD (Contd)

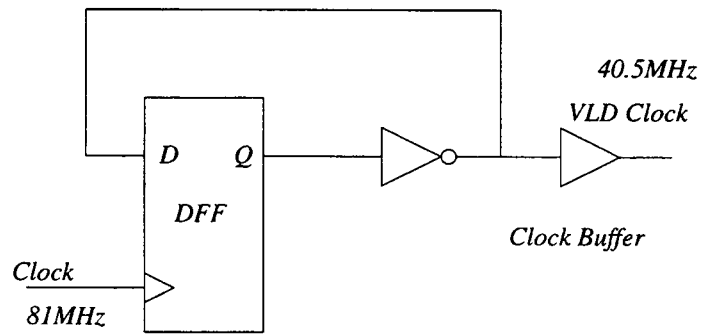


FIGURE 6.8. VLD Pipeline Clock

```

// Start code length decode (Counter)
function [2:0] nsstate;

input  [2:0] sstate;
input  [15:0] strm;
input          enb, video;

reg    [2:0] value;

begin
if(video) begin
case (sstate) // synopsys parallel_case full_case
3'b000:
begin
if(enb && strm 1+ 16'0)      value = 3'b010;
else if(enb && strm == 16'h0) value = 3'b001;
else                        value = 3'b000;
end
3'b001:
begin
if(strm[15:8] == 8'h0)      value = 3'b011;
else if(strm[15:8] == 8'h1) value = 3'b100;
else                        value = 3'b010;
end
3'b010:
begin
if(strm[15:0] == 16'h0)    value = 3'b001;
else                       value = 3'b010;
end
3'b011:
begin
if(strm[15:0] == 16b'0)    value = 3'b011;
else                       value = 3'b100;
end
default value = 3'b000;
endcase
nsstate =value;
end end
endfunction

```

FIGURE 6.9. Verilog Code for Start Code Length Decoder

7. CONCLUSIONS

Cost effective video compression solutions obtained through the combination of digital video and compression technology are becoming the main driving force behind a large number of applications. VLSI implementation of video compression has become the core technology of consumer electronics, computer and communication, such as VideoCD players, DVD, DVC, Add-in Cards for Game, content encoding equipment, video editing systems, video conferencing, Direct Broadcast Satellite (DBS), Multichannel Multipoint Distribution Service (MMDS), Satellite Uplinks, cable set-top box and telephone distribution systems.

In this thesis, MPEG standards are discussed thoroughly and interpreted, and a VLSI implementation (CMOS 0.35μ technology and 3 layer metal) of a variable length decoder (VLD) for MPEG applications is developed. The VLD achieves high performance by using a parallel and pipeline architecture. MPEG bitstream patterns have been carefully analysed to drastically improve VLD memory efficiency. In addition, a special clock scheme is applied to reduce the power consumption.

Future Directions

Although the MPEG-1 and MPEG-2 have been widely accepted by industry, there are still a lot of problems yet to be solved. Because the encoder standards only define the syntax of the bit-stream, it is still very important to continue research to find out new algorithms such as advanced preprocessing methods, wide search range ME and rate control. Audio is also a very important part. The system layer which we do not deal with in this thesis is very critical for the transport of the bit stream. The video and audio streams synchronization problem and rate control are some of the very active research areas. Video compression is truly the core technology which will

enable revolutionary development in the electronics industry. For example, wireless video transmission has been made possible because of low power VLSI, compression technology and communication. Furthermore, there are still many other aspects of digital video need to be explored, such as the modulation technique, error-correction techniques and conditional access (security). The application is only limited by our imagination.

BIBLIOGRAPHY

7.1. REFERENCES

- [1] A. Chandrakasan, A. Burstein, and R.W. Brodersen. “**A low-power chipset for a portable multimedia I/O terminal.**” in *IEEE journal of Solid State Circuits*, 29(12):1415-1428, Dec. 1994.
- [2] A. Chandrakasan and R.W. Brodersen. “**Minimizing Power Consumption in digital CMOS Circuits**” in *Proceedings of the IEEE*, April 1995, 498-523.
- [3] A. Puri, “**Video Coding Using the MPEG-1 Compression Standard,**” *ISSID* ., May, 1992, invited paper.
- [4] A. Puri, R. Aravind and Barry Haskell, “**Adaptive Frame/Field Motion Compensated Video Coding,**” *Signal Processing: Image Communications* 5, 1993, pp 39-58.
- [5] A. V. Oppenheim and R. W. Schaffer, “**Discrete-Time Signal Processing**”, Prentice-Hall, 1989.
- [6] Belle W. Y. Wei and Teresa H. Meng, “**A parallel decoder of programmable Huffman codes**”, *IEEE Trans. on Circuits and Systems for Video Technology*, April 1995. pp 175-178.
- [7] Brain Evans et, al “**Understanding Digital TV**” IEEE Press, 1995.
- [8] Chad Fogg, “**MPEG-2 FAQ**”, 1995.
- [9] “**CLM4500 Consumer MPEG Video Encoder User’s Manual**” C-Cube Microsystems, Technical Publications Department, Sept. 1995.
- [10] “**VideoRISC-3 Processor Hardware User’s Manual**” C-Cube Microsystems, Technical Publications Department, Feb. 1996.
- [11] Cheng-Teh Hsieh and Seung P. Kim “**A Concurrent Memory-efficient VLC Decoder for MPEG Applications**” in *IEEE Transactions on Consumer Electronics* August 1996, pp 439-445.
- [12] Daniel J. Mlynek and Jacques Kowalczyk, “**VLSI for Digital Television**” in *Proceedings of the IEEE* July 1995, pp 1055-1070.

- [13] D. Galbi et al. “**An MPEG-1 audio/video decoder with run-length compressed antialiased video overlays**” in *Proceedings IEEE International Solid State Circuits Conference*, Feb. 1995, pp 286-287.
- [14] Dimitris Anastassiou, “**Digital Television**” in *Proceedings of the IEEE*, Apr. 1994, pp 510-519.
- [15] D. Le Gall. “**MPEG: A video compression standard for multimedia applications**” in *Communications of the ACM*, 34(4), April 1991.
- [16] D. Donoho, “**De-noising by Soft Thresholding**,” *IEEE Trans. Inf. Th.* 41, 1995, pp 613-627.
- [17] H. B. Bakoglu, “**Circuits, Interconnections, and Packaging for VLSI**”, Addison-Wesley, 1990.
- [18] H. B. Bakoglu, “**Digital Coding of Waveforms, Principles and Applications to Speech and Video**”, Prentice-Hall, Englewood Cliffs, New Jersey, 1984.
- [19] Horng-Dar Lin and David G. Messerschmitt, “**Designing a High-Throughput VLC Decoder Part II - Parallel Decoding Methods**”, *IEEE Trans. on Circuits and Systems for Video Technology*, June 1992. pp 197-205.
- [20] ISO /IEC JTC 1/SC 29, “**Coded Representation of Picture, Audio and Multimedia/Hypermedia Information**”, Dec. 1991.
- [21] ISO /IEC 13818-1, “**Generic Coding of Moving Pictures and Associated Audio.**”, Nov. 1994.
- [22] Israel Koren, “**Computer Arithmetic Algorithms**, Prentice Hall, 1993.
- [23] J.W. Woods and S.D. O’Neil. “**Subband coding of images**” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Oct. 1986. 1278-1288.
- [24] John Hennessy and David Patterson “**Computer Architecture - A quantitative Approach**” Morgan Kaufmann Publishers, 1990.
- [25] Kou-Hu Tzou, “**Video Coding Techniques: An Overview**” in *VLSI Implementations for Image Communications*, Elsevier Science Publishers, 1993, pp 1-47.
- [26] Li Hua et al, “**Experimental Investigation on MPEG-2 Based Video coding at 22 Mbps**” in *IEEE Transactions on Consumer Electronics* Aug. 1995 pp 615-619.

- [27] Michael L. Hilton, Bjorn D. Jawerth and Ayan Sengupta, "**Compressing Still and Moving Images with Wavelets**" in *Preprint*, Apr. 1994.
- [28] Ming-Ting Sun, "**Algorithms and VLSI Architectures for Motion Estimation**" in *VLSI Implementations for Image Communications*, Elsevier Science Publishers, 1993, pp 251-282.
- [29] M. V. Wickerhauser, "**High-resolution Still Picture Compression**, *Preprint*.
- [30] Mark Nelson, "**The Data Compression Book**", M&T Books, 1992.
- [31] N. Ohta. "**Packet Video - Modeling and Signal Processing**" Artech House, 1994.
- [32] O. Rioul and M. Vetterli, "**Wavelets and Signal Processing**," *IEEE Signal Process.* Oct. 1991.
- [33] Olivier Rioul, "**On the choice of Wavelet Filters for Still Image Compression**," *Comm. IEEE .*, 1993, pp 551-553.
- [34] P. Pirsch, "**VLSI Implementation Strategies**" in *VLSI Implementations for Image Communications*, Elsevier Science Publishers, 1993, pp 49-68.
- [35] P. Pirsch et al, "**VLSI Architectures for Video Compression - A Survey**" in *Proceedings of the IEEE* February 1995, pp 220-246.
- [36] Peter J. Burt and Edward H. Adelson, "**The Laplacian Pyramid as a Compact Image Code**," *IEEE Transactions on Communications* April 1983, pp 532-540.
- [37] Robert J. Siracusa, etc., "**Flexible and Robust Packet Transport for Digital HDTV**," *IEEE J. Selected Areas Commun.*, Vol 11, Jan. 1993, 88-98.
- [38] Shaw-Ming Lei and Ming-Ting Sun, "**An entropy coding system for digital HDTV applications**", *IEEE Trans. on Circuits and Systems for Video Technology*, March 1991. pp 147-155.
- [39] S. G. Mallat. "**A Theory for multiresolution signal decomposition: the wavelet representation.**" *IEEE Transactions on Pattern Analysis and Machine Intelligence*, July 1989, pp 674-693.
- [40] Shih-Fu and David G. Messerschmitt, "**Designing a High-Throughput VLC Decoder Part I - Concurrent VLSI Architectures**", *IEEE*

Trans. on Circuits and Systems for Video Technology, June 1992. pp 187-197.

- [41] Teresa H. Meng et, al “**Portable Video-on-Demand in Wireless Communication**” in *Proceedings of the IEEE*, April 1995, 659-680.
- [42] Ulrich Golze etc., “**VLSI Chip Design with the Hardware Description Language VERILOG**”, Springer-Verlag Berlin Heidelberg, 1996.
- [43] V. Bhaskaran and K. Konstantinides, “**Image and Video Compression Standards**”, Kluwer Academic Publishers, 1995.
- [44] W. B. Pennebaker and J .L. Mitchell “**JPEG Still Image Data Compression Standard** Van Nostrand Reinhold, New York, 1993
- [45] Ya-Qin Zhang and Sohail Zafar, “**Motion-Compensated Wavelet Transform Coding for Color Video Compression**”, *IEEE Trans. on Circuits and Systems for Video Technology*, Sept. 1992. pp 285-296.

APPENDICES

VLD Layout

