ALGORITHMS AND PROCESSOR STRUCTURES FOR MOTION PICTURE COMPRESSION

by

HUI Wai-Lam, BEng (Hons)

A thesis submitted for the Degree of Master of Philosophy
in the Department of Electronic and Information Engineering
of the Hong Kong Polytechnic University

Department of Electronic and Information Engineering
The Hong Kong Polytechnic University

September 1998

Abstract

of the thesis entitled

"Algorithms and Processor Structures for Motion Picture Compression"

submitted by HUI Wai-Lam

for the degree of Master of Philosophy

at The Hong Kong Polytechnic University in September, 1998.

Recent progress in video compression algorithms and VLSI technology has made it possible to store and transmit digital video in many applications. Related standard activities in video compression are also moving rapidly. Two central components of these emerging standards are the motion estimation and the Discrete Cosine Transform (DCT), which intend to remove the spatial and temporal redundancies in video sources. And it is known that the computational requirements are the most critical for real-time applications. It is necessary to develop custom hardware and efficient algorithms to enable real-time applications while to reduce the manufacturing cost of the Video Codec.

It is well known that the DCT approaches the statistically optimal KLT for highly correlated signals. Fast algorithms for its implementation are developed to reduce the computational complexity. With the growing applications of the DCT, it is desirable to develop a modular and reusable DCT processor to reduce system cost. For this purpose, a low hardware complexity Discrete Cosine Transform Processor, with high speed operation and regular structure, is designed and implemented. The DCT processor features bit-serial approach and parallel operation with multiple operation elements. With the use of signed digit representation of the DCT coefficients technique, more than 75 percent reduction on the hardware for the DCT kernel matrix multiplication can be saved. The optimized kernel matrix multiplier exhibits highly pipeline operations and the maximum achievable operation frequency of the processor is 45MHz. The implementation of the DCT processor including external interface circuits uses about 4500 gates.

The widely used block motion estimation and compensation algorithms are considered to be the most efficient and yet a simple technique for the reduction of temporal redundancies. In this thesis, a new pixel decimation technique based on a set of the pixel patterns for block motion vector estimation is proposed to compensate the drawback in the approach using pixel decimation. For regular pixel decimation, regular patterns are used for computing the matching criterion to estimate the motion vector. The results can easily be misled by some image textures. Thus, we define some "*most representative pixel patterns*" and make the selection according to the content in each image block for the matching criterion. Our approach can efficiently compensate the drawback in uniform pixel decimation. Computer simulations show that this technique is close to the performance of the full search, and has a significant reduction on computational complexity as compared with other pixel decimation algorithms in the literatures. Also, it is more convenient for hardware realization as compared with the fully adaptive pixel decimation.

Adaptive algorithm has many advantages over fixed strategy algorithms. However, the variation on the execution time for different video scene is difficult for its realization. A multiprocessor system is designed for the realization of the newly proposed Edge Oriented Adaptive Motion Algorithm. With the technique of task decomposition and a predetermined execution profile, a simple scheduling strategy is derived and the idling time of each processor is reduced dramatically. For the realization using ASIC design, a speedup of 3.5 is achievable by making use of four Processing Elements with an overhead of less than 15% hardware complexity.

One of the problems of the block motion estimation is that blocks located on boundaries of moving objects are not estimated accurately. It is considered to be the most objectionable distortion by human observer. In this thesis, a new edge-masked matching criterion for block motion estimation and its hardware architecture are also presented. The proposed matching criterion makes use of edge features to modify the conventional matching criterion for computing the motion information. Experimental results and a custom hardware design show that the new matching criterion has removed the most visually disturbing artifacts with a slight increase in hardware complexity, and the motion-compensated prediction frames are virtually error-free.

Acknowledgements

I would like to express my greatest thank to my supervisor Professor Wan-Chi Siu, who has been very supportive and encouraging, giving me valuable advice from the outset of my studies.

During my study, my life has been enriched by interaction with my colleagues, especially Mr. Stephen Kwok, Mr. Jamin Tse, Mr. Steven Choy, Mr. T. W. shen, Mr. Joseph So, Dr. Y. L. Chan, Dr. Clifford Choy, Dr. Bity Chau and Dr. James Kwok and those who have worked with me throughout my study. The countless discussions with them have proved to be fruitful and inspiring.

I have also thank all members of staff of the department, especially, Mr. W. H. Wong, Mr. S. M. Tse, Mr. K. C. Choi, Dr. Christie Chan, Dr. Daniel Lun and the clerical staffs in the General Office. They have created a supportive environment for me to work in.

It is my pleasure to acknowledge the Research Degrees Committee of the Hong Kong Polytechnic University for its generous support over the years.

Above all, I thank my parents, my brother and my sister for their constant love and support. Without their understanding and patience, it is impossible for me to complete this research study.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

1.1 Video Compression Development

Recent progress in digital technology has made the widespread use of compressed digital video signals practical. Standardization has been very important in the development of common compression methods to be used in the new services and products that are now possible. This allows the new services to interoperate with each other and encourages the investment needed in integrated circuits to make the technology inexpensive. These developments have been made possible by the convergence of VLSI technology with the image and video compression theory developed in the past, with the support of international standardization activities.

With the integration of video in the computer environment, it has been made possible, setting the starting point for multimedia and virtual reality technologies. Application of data compression is also possible in the development of fast algorithms where the number of operations required to implement an algorithm is reduced by working with the compressed data. There are two primary applications of data compression: transmission and storage of information. For transmission, compression techniques are greatly constrained by real time and on-line considerations which tend to limit severely the computational complexity of the compression scheme. On the other hand, for storage applications, the requirements of the encoder are less stringent because much of the processing can be done off-line. However, the processing time for the decoding or decompression process should be minimized to reduce the retrieval time. This asymmetrical compression scheme [17, 18] has benefited the video broadcast services since once the video is encoded, the encoded video bit-stream is then broadcast to all subscribers and the decoding process is the same for each subscriber. The reduced complexity in the decoder has leads to lower the manufacturing cost of the decoder which is an important factor that affects the acceptance of digital video broadcasting services.

In recent years, video conferencing application is another evolved application which requires real-time capability and symmetrical compression configuration. For practical realization of video conferencing system, development of effective video compression algorithms for real-time processing are required. The efficiency of a compression algorithm is measured by its data compressing ability, the resulting distortion and as well by its implementation complexity. The complexity of the compression algorithms is a particularly important consideration in their hardware implementation. There are a number of proposed compression schemes in the literature, namely, Discrete Cosine Transform (DCT) based coding [1], Vector Quantization (VQ) [12] based coding, sub-band and wavelet coders [13].

Among these available video coding schemes, the most successful video coding technique is the hybrid scheme formed by combining motion compensated prediction in the temporal domain and a decorrelation technique in the spatial domain. The hybrid compression scheme is also widely accepted as international coding standards such as H.263 [16] targets for low bit rate video conference applications, MPEG-1 [17] and MPEG-2 [18] target for broadcast quality video compression.

The envisaged mass application of the discussed video communication services call for coding equipment of low manufacturing cost and small size. The source coding schemes recommended by international standardization groups are very sophisticated in order to achieve high compression rate under the constraint of the highest possible picture quality. These coding schemes will result in high computational complexity for real-time implementation. However, general purpose processors are often not capable to handle such high computation rates. Thus, cost effective implementation of these high complexity systems rely on custom designed VLSI.

The continuing trend of further increase in the level of integration (i.e. the number of transistors) has lead to a huge 'management of complexity' problem in chip design. Nowadays, VLSI design methodology has been moving into the direction of modularity and reusability. With the aim of modular design, the system is decomposed into various small functional blocks and each functional block is designed independently with predefined specification. The well-designed and detailed documentation of each functional block has also made it possible for future uses. Furthermore, with the decomposition of the compression system into small functional blocks, programmability to adapt to different algorithms is possible by changing the interconnection between different functional blocks.

1.2 Motivation and Research Objectives

The continuous development of sophisticated video coding techniques has contributes to improved coding performance and better video quality. However, the required computation rates are many orders of magnitude greater than those obtainable from a general purpose processor. As a result, there is often demands for efficient implementation for real-time applications.

The advances and decreased costs in Very Large Scale Integration (VLSI) technology now allow many sophisticated algorithms to be implemented in a cost-effective manner in hardware. There is also an increasing interest in the design of dedicated or application-specific chips and chip sets. It is very clear that the VLSI technology will be available more cheaply in the future, it is also true that the increase in complexity of the coding algorithm will still make it necessary to develop architectures that map video coding algorithms with the lowest complexity.

Among other motion estimation techniques, block matching techniques have been developed for video coding schemes based on the DCT such as those adopted by the recent international video coding standards [14-19]. Hence, coding applications often rely on block matching motion estimation techniques. Despite their widespread use, block matching techniques share several common drawbacks: unreliable motion fields in the sense of the true motion in the scene, block artifacts and poor motion compensated prediction along moving edges. In addition, it is also difficult to incorporate the most important part of a video coding system, the human visual system, into the coding model. The widely used mean square distortion measure keeps little relation with the human visual system. It is desirable to develop efficient algorithms that produce improved visual quality which is sensitive to human observer while maintain the compatibility to current video coding standards [14-19].

Thus, in this thesis, we intend to develop efficient algorithms that made improvement on the visual quality of the motion estimation predicted frame. Apart from this, architectures that efficiently realize those algorithms are also our interest.

1.3 Organization Of The Thesis

It is important to review the basics in which the current video coding techniques are established. Thus, a review on the video coding techniques that are commonly adopted in various international video coding standards will be given in Chapter 2. Of course, there are so many issues related to video coding, and we have confined the review that is related to our works. This includes the background of the Discrete Cosine Transform (DCT) and the motion estimation/compensation (ME/MC). Besides the discussion on some fast algorithms that can reduce the computational complexity for both the DCT and ME/MC, some hardware structures for their realization will also be introduced.

In Chapter 3 we will show the design and the implementation of a Discrete Cosine Transform processor. With the low gate count and high operating frequency features, the architecture proposed is very suitable for low cost applications.

The ME/MC is the most important component in various video coding schemes, it determines the coding efficiency and the performance of the video coder. Unfortunately, most fast algorithms for the ME/MC have the common problem that the estimation process is easily trapped by local minimum. Hence, in Chapter 4, we tried to tackle the problem by reducing the number of pixels used in the matching criterion and by using a novel pixel pattern selection scheme.

Adaptive algorithms for the ME/MC have the ability to reduce the computational complexity while the prediction quality is improved as compared to other fast search algorithms. However, the realization would be complicated. In Chapter 5, a multiprocessor system with multiple-bus architecture for the realization of the Edge Oriented Adaptive Motion Estimation algorithm is proposed.

The reason behind the wide adoption of the block based motion estimation technique is its simplicity. However, the loose relationship to Human Visual System produces annoying artifacts, such as the discontinuities of the moving object's edges caused by the segmentation on the objects. In Chapter 6, we propose a scheme to produce an accurate prediction on the motion vector along the moving edges.

Finally, the thesis is concluded in Chapter 7 with a summary of the work done. Suggestions on the future directions of the present work are also discussed.

Chapter 2

Video Coding Techniques

2.1 Introduction

The rapid development of VLSI technologies, computer architectures, communications, advance techniques for video coding in the past decade emerged many applications has made the transmission, storage and manipulation of digital video data in a more efficient way. These emerged applications including digital TV, medical imaging, virtual reality, multimedia applications with video, audio and hypertext, but not to mention video-conferencing, which challenge many researchers to develop algorithms that can achieve the highest compression ratio while produce the best video quality.

Transmission of digital video data in its original form requires huge amount of bandwidth. For example, to transmit a CCIR-601 format video signal without any compression, the required bandwidth would be:

$$(720\times 480+2\times 360\times 480)\frac{bytes}{frames}\times 30\frac{frames}{\sec ond}=20.7 Mbytes/\sec$$

Although the advancement of telecommunication technologies increases the transmission bandwidth dramatically, it is still not economical to transmit the video data with such high data rate. The huge data rate of the digital video data could be reduced if suitable video coding algorithms are applied since there is a large amount of redundancies [1] resided in the video data.

Redundancies in the video data are regarded as those signals which can be discarded or can be transformed into another domain such that the number of bits required to represent those signals is the lowest and the system still maintains good video quality. In the framework of video coding, categorize the statistical redundancies residing in the digital video data into two classes, namely the *spatial and the temporal redundancies*. The former, also called *intraframe* redundancy, refers to the redundancy that exists within a single frame of video, while the latter, also called *interframe* redundancy, refers to the redundancy that exists between consecutive frames within a video sequence.

Removing of spatial redundancy within a video frame is based on the fact that every natural image exhibits high spatial correlation among neighbouring pixels and contains a significant amount of perceptually irrelevant information. Statistical coding [3, 4] such as Huffman coding, which assigns shorter code words to pixel values that occur more frequently and longer code words to values that occur rarely. This kind of coding technique is known as lossless image coding. Lossless image coding has the advantage that no distortion is imposed into the original image. But the coding efficiency of lossless coding does not sufficient for low bit rate video application. On the contrary, lossy image coding applies the rate distortion theory [2] on the encoding process. The quality of the decoded image is proportional to the output bit rate. In fact, lossy image coding is commonly used for the encoding of video sequences since it can achieve a much higher compression ratio than lossless coding schemes. Moreover, the information contained in the sequence, that is not perceived by a human observer, can be eliminated.

In this chapter, a review on various techniques for video coding and their hardware implementations will be given. First, an overview of the Discrete Cosine Transform (DCT), its implementations and application to image coding are given. Second, the predictive coding using block motion estimation will be discussed, including its implementation and several techniques to reduce the computational complexity. Finally, a hybrid technique combined with the transform and predictive coding will be discussed.

2.2 Spatial Image Coding

Spatial image coding is the process to eliminate the statistical redundancy in an image and there exist two main approaches: predictive approach and transform approach. In predictive approach, the current image pixel is computed on the basis of previously processed pixel samples, and the prediction error is encoded and transmitted. The performance of predictive coding is highly related to the statistics of the image. An alternative for the elimination of redundancy in the image is the transform approach where the image is transformed into another domain and the redundancy can be easily identified and eliminated.

Transform coding has been found to have relatively good capability for bit-rate reduction, which comes about mainly from two mechanisms. First, not all of the transform domain coefficients need to be transmitted in order to maintain good image quality, and second, the coefficients that are coded need not to be represented with full accuracy.

In a typical transform coding system, the image is divided into non-overlapped rectangular blocks with block size of $N \times N$ pixels, where size of 8×8 and 16×16 are most commonly used. A larger block size offers a higher compression efficiency with the disadvantage of an increased computational complexity. The purpose of using transformation is to decompose the correlated image signal into a set of uncorrelated spectral coefficients with energy concentrated in only a few of the coefficients. The performance of the transform coding system depends on the ability of the transform used to decorrelate and compact the image data.

The Karhunen-Loeve Transform (KLT) [7, 8], is an orthonormal transform whose base vectors are the eigenvectors of the autocorrelation matrix of the image signal. The KLT is optimal for a compression scheme based on discarding the coefficients of less energy. But the direct use of KLT for decorrelation is difficult because of its dependency on the statistics of the image signal and the lack of efficient algorithms for its computation. Both problems are partially solved by the Discrete Cosine Transform (DCT) [9]. In fact, the DCT has been proven to be a very efficient transformation and has adopted as a standard component in various international standards for image and video coding such as JPEG [14], H.261 [15], H.263 [16], MPEG 1 [17] and MPEG 2 [18].

The popularity of the DCT is based on the properties that the basis functions of the DCT are image independent, which is a property different from the KLT. It is shown in [9] that the DCT approximates the KLT for a first order Gauss-Markov process when the correlation coefficient approaches one. In practices, for natural images, the DCT approximates the de-correlation and energy compaction property of the KLT very well. Another advantage of the DCT is that there exist fast algorithms that reduce the computational complexity for its realization.

2.3 Discrete Cosine Transform

There are several different forms of the DCT. The one used in the standards is the DCT-II which is of the following form,

$$X(u,v)=\frac{2}{N}C(u)C(v)\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}x(m,n)\cos\left(\frac{\pi(2m+1)u}{2N}\right)\cos\left(\frac{\pi(2n+1)v}{2N}\right) \qquad (2\text{-}1)$$

where $X(u,v)$ is the spectral coefficient in the transform domain at location (u,v), $0 \le u,v \le N-1$ and $x(m,n)$ is the image signal in spatial domain at location (m,n), $0 \le m,n \le N-1$. The function $C(i)$ is defined to be one except when i is equal to zero, in which case $C(0)=1/\sqrt{2}$. This definition of the scaling function $C(i)$ results in a unitary transform.

Direct realization of the 2-D DCT requires huge amount of computing power, hence, fast algorithms [22-24] were developed to provide efficient realization of the 2-D DCT algorithm. Note that the DCT is separable, which means that higher dimension DCT can be implemented by a series of 1-D DCT computations. Emerge of these fast algorithms and the inherent properties of the DCT further increase the popularity the DCT on image coding applications.

2.3.1 DCT Based Image Coder

Figure 2-1 and Figure 2-2 shows the simplified operations of the DCT based image encoder and decoder respectively.

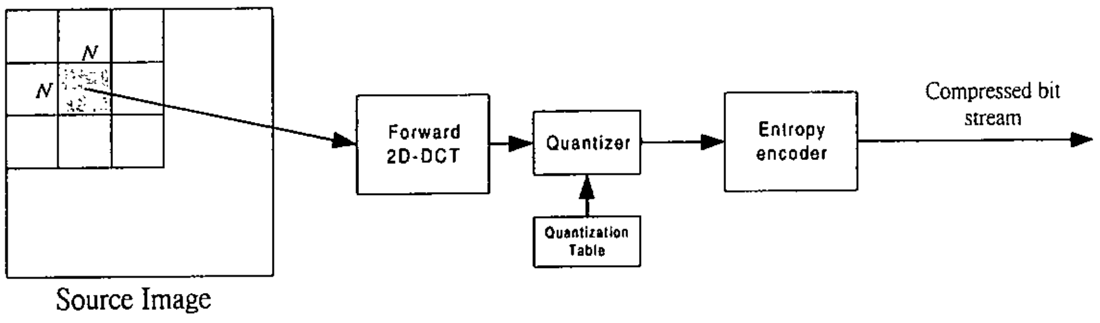


Figure 2-1 : Block diagram of the DCT based image encoder

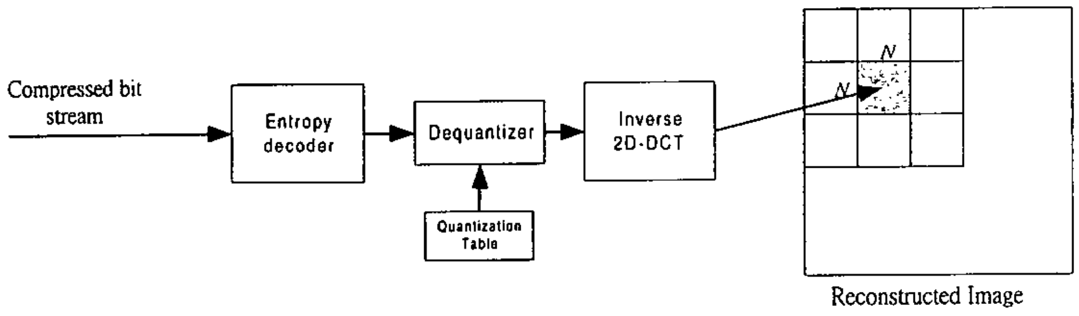


Figure 2-2 : Block diagram of the DCT based image decoder

At the encoder side, the source image is divided into a number of $N \times N$ image blocks before to perform the forward 2D-DCT. After performing forward 2D-DCT on the image block, most of the energy is represented by the low frequency coefficients. For lossy coding, these coefficients can be scalar quantized independently using a uniform quantizer which is specified by a quantization table $Q_{i,j}$, for $i, j = 0,..,N-1$ with a different step size for each coefficient. The lower frequencies are quantized more finely whereas a coarser quantization is used for the higher frequencies. This weighting of the lower frequencies is an attempt to incorporate some properties of the human visual system (HVS) into the coding scheme.

After quantization, most of the transform coefficients are zero and the coefficients are scanned in a zig-zag order as depicted in Figure 2-3. The resulting string is then run-length and entropy encoded which results in a very compact representation of the quantized DCT coefficients.

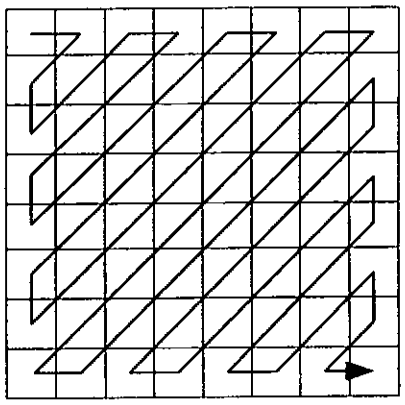


Figure 2-3 : Zig-Zag scanning order for the DCT coefficients

As shown in Figure 2-2, the operations of the decoder perform the inverse processes of the encoder. The received bit stream is feed into the entropy decoder and the decoded quantized DCT coefficients are then dequantized according to the quantization table, and finally, the inverse 2D-DCT is used to reconstruct the image. The inverse 2-D DCT is defined as,

$$x(u,v) = \frac{2}{N} C(u)C(v) \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} X(m,n) \cos\left(\frac{\pi(2m+1)u}{2N}\right) \cos\left(\frac{\pi(2n+1)v}{2N}\right) \qquad (2\text{-}2)$$

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

The relationship of the DCT to the KLT, its energy compaction ability, and its fast algorithms contribute to the popularity of the DCT used in various image coding standards. However, the advances of VLSI technologies resulting in lower manufacturing cost and faster development time also urge the applications of DCT to a wider area.

2.4 DCT Implementation

The 2D-DCT algorithm has become the standard component in various image compression standards, efficient realization which can perform in real time, high speed computing architecture is often preferred. Table 2-1 shows the processing time requirement of the 2D-DCT of various international video compression standards. The hardware implementation of an algorithm requires careful mapping of the algorithm into an architecture that can achieve small chip size, low power consumption and high processing throughput. These criteria set an important guideline at the design stage of the implementation.

Standard	**Format**	**Luminance (Y)**	**Chrominance (U/V)**	**Frame rate (fps)**	**pixel rate (pps)**	**8x8 pixels (μS)**	**8 pixels (μS)**
H.263	CIF	352x288	176x144	30	4.56M	14.03	1.754
	QCIF	176x144	88x72	30	1.14M	56.117	7.015
MPEG-1	CIF	352x288	176x144	30	4.56M	14.03	1.754
	SIF	352x240	176x120	30	3.8M	16.835	2.1044
MPEG-2	MP@ML (DTV)	720x576	360x288	30	18.66M	3.4294	0.4287
	MP@HL (HDTV)	1920x1152	960x576	60	200M	0.3215	0.04019

Table 2-1 : Processing time requirement of various international video compression standards

Most of the implementations of the 2D-DCT algorithm using either matrix decompositions or based on the properties and fast algorithms of the 2D-DCT, can be classified as row-column approach (RCA) and non-row-column approach (NRCA). RCA utilizes the separability of the 2D-DCT property to separate the computation into 1D-DCT over the rows, a matrix transposition and 1D-DCT over the columns. On the contrary, the NRCA avoids the matrix transposition either using array of processors or fast algorithms to implement the 2D-DCT algorithm directly.

2.4.1 Row-Column Approach

It is well known that the 2D-DCT is separable and the 1D-DCT is computed as,

$$z_u = \frac{C(u)}{2}\sum_{m=0}^{N-1} x(m)\cos\left(\frac{\pi(2m+1)u}{2N}\right) \qquad \text{for } u = 0,1,\ldots\ldots,N-1. \qquad (2\text{-}3)$$

It can also be expressed in vector-matrix form as $z = Tx'$, where T is an $N \times N$ matrix whose elements are the cosine function values defined in equation (2-3), $x = [x_0, x_1, \ldots, x_{N-1}]$ is a row vector, and z is a column vector. From equation (2-1), the 2D-DCT can be expressed as,

$$X(u,v) = \frac{C(u)}{2}\sum_{m=0}^{N-1}\left[\frac{C(v)}{2}\sum_{n=0}^{N-1} x(m,n)\cos\left(\frac{\pi(2m+1)u}{2N}\right)\right]\cos\left(\frac{\pi(2n+1)v}{2N}\right) \qquad (2\text{-}4)$$

The above equations imply that the 2D-DCT can be obtained by first performing 1D-DCT of the rows of $x(m,n)$ followed by 1D-DCT of the columns of the intermediate result. Hence, the 2D-DCT of an image $x(m,n)$ can be expressed in matrix notation,

$$X = TxT' \qquad (2\text{-}5)$$

The separability of the 2D-DCT reduces the implementation complexity dramatically and most of DCT implementations [25-27,29] reported were based on the RCA. As shown in

Figure 2-4 the 2D-DCT algorithm is separated as two independent 1D-DCT operations. For the first stage, a 1D-DCT operation is done on every row of the image block and the intermediate result is transposed through the transposition memory. For the second stage, 1D-DCT operations is done again for each column of the transposed intermediate result, to obtain the transformed image.

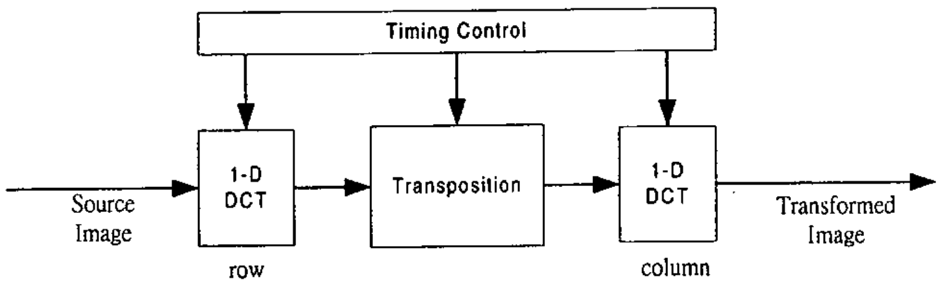


Figure 2-4 : Block diagram of the row-column 2D-DCT processor

Row-Column approach enables an easy implementation of the 2D-DCT algorithm. Any fast algorithm developed for the 1D-DCT can be easily converted into two-dimensional architecture with additional control circuits and transpose memory. Since the 1D-DCT algorithm involves many multiplications, butterfly structures with fewer multiplications can be found in [20, 21] and many other proposed fast algorithms for reducing the number of multiplications required for the computation of 1D-DCT can also be found in the literatures. Sophisticated fast algorithms for the 1D-DCT reduce the computational complexity, but have a drawback of complicated data-flow and irregular structure that may not be suitable for hardware implementation.

2.4.1.1 Distributed Arithmetic

Implementation of DCT algorithm using the distributed arithmetic approach has been reported in [27, 28]. Since distributed arithmetic is used for the computation of the matrix-vector product, and thus different fast algorithms are actually mapped into the hardware structure. This type architecture exhibits multiple processing elements (PE) architecture, bit-serial and bit-parallel data structures to implement the vector inner products concurrently.

The input data sequence $x(i)$ in equation (2-3) can be expressed in 2's complement form

$$x_m = -x_m^{(n-1)} 2^{n-1} + \sum_{j=0}^{n-2} x_m^{(j)} 2^j \tag{2-6}$$

where $x_m^{(j)}$ is the j^{th} bit of x_m which has a value of either 0 or 1, n is the number of bits to represent x_m and $x_m^{(n-1)}$ is the sign bit. By substituting equation (2-6) into (2-3),

$$z_u = -\sum_{m=0}^{N-1} x_m^{(n-1)} 2^{n-1} \cos\frac{(2m+1)u\pi}{2N} + \sum_{j=0}^{n-2} \sum_{m=0}^{N-1} x_m^{(j)} 2^j \cos\frac{(2m+1)u\pi}{2N} \tag{2-7}$$

$$= -F(x_m^{(n-1)}, u) 2^{n-1} + \sum_{j=0}^{n-2} F(x_m^{(j)}, u) 2^j \tag{2-8}$$

where

$$F(x_m^{(j)}, u) = \sum_{m=0}^{N-1} x_m^{(j)} \cos\frac{(2m+1)u\pi}{2N} \qquad \text{for } j = 0,1,\ldots\ldots,N-1 \tag{2-9}$$

Note that F is a function of bit patterns of x_m and the coefficients $\cos\frac{(2m+1)u\pi}{2N}$. Since the coefficients are fixed, thus F can be pre-computed and stored in ROM for all possible bit patterns of x_m. Consequently, z_u can be computed concurrently for $u = 0,1,\ldots\ldots,N-1$.

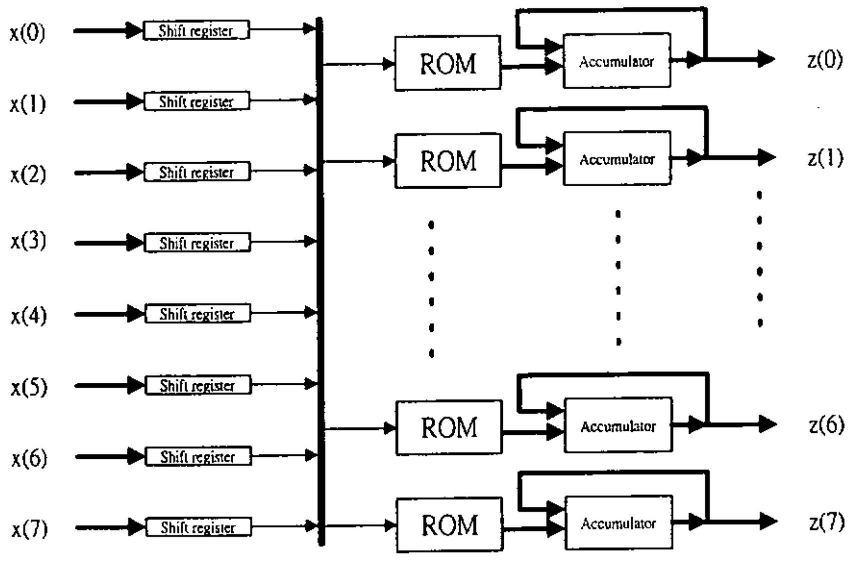


Figure 2-5 : Architecture of distributed arithmetic DCT aritechmetic DCT

As shown in Figure 2-5, the regularity of this approach results in an implementation that is a direct matrix-vector multiplication architecture. The pre-computed look-up tables for specific sizes of transforms are stored in ROMs. The shift register at the input stage shifts the image data serially to form the look-up table address. The ROM and the accumulator perform the matrix multiplications of the 1D-DCT concurrently.

2.4.1.2 Systolic Array

The key to VLSI parallel architecture design is to reduce the interconnection complexity and to keep the overall architecture highly regular, parallel and pipelined operation. Systolic array [10] architecture is very suitable for VLSI implementation because it features regularity, modularity and local communications and is capable of achieve high degree of pipelining. A systolic array is a network of processors which concurrently compute and pass data through the system. For DCT algorithms which feature with regular and recursive properties and with the help of the derivation of the Dependence Graph (DG) [10] to give a regular structure can be represented by a grid model for systolic realization.

Figure 2-6 shows a systolic array architecture for the 1D-DCT realization. This array composes of adder/subtractor cells for the pre-processing of the input data. The multiplier/adder cell stores the intermediate results and evaluates the data passed from the neighbouring cells. Extension of this architecture to 2D-DCT has been proposed in [30]. This new systolic array proposed for the 2D-DCT based on the row-column decomposition which does not require matrix transposition. Instead, internal registers and different architectures for evaluating the row and column transforms are used. Avoiding the matrix transposition will lead to obvious reduction in chip area and latency between the input data and the output result can also be reduced.

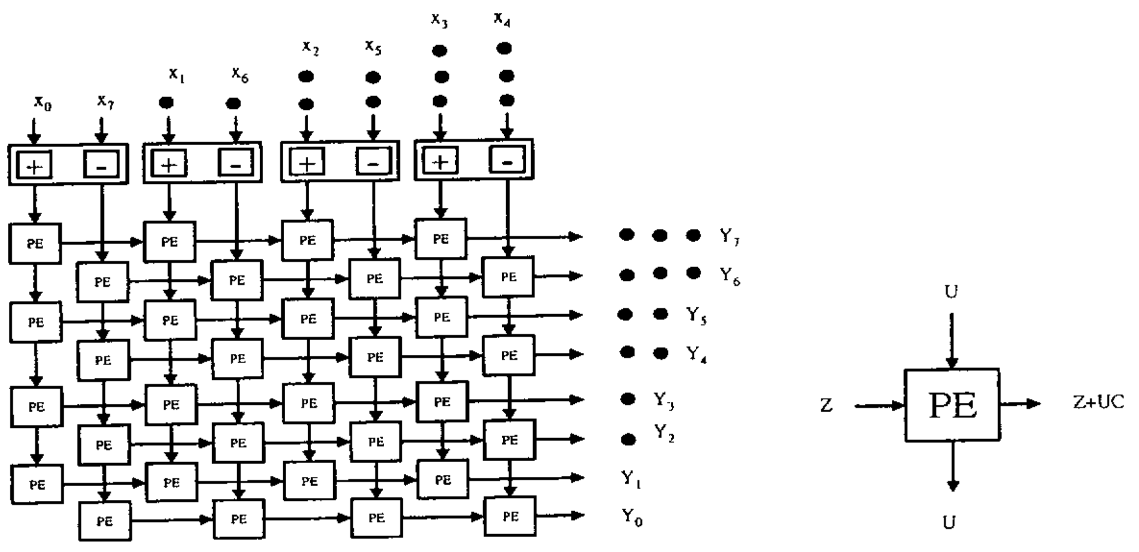


Figure 2-6 : Systolic Array architecture of DCT processor

2.4.2 Non-Row-Column Approach

In contrast to the RCA, the NRCA manipulates the 2D data set directly and requires a fewer number of multiplications. Many fast algorithms [22-24] have been proposed for the direct implementation of the 2D-DCT. In [22], Vetterli proposed to use an indirect polynomial transform approach which saves the number of multiplications by 50 to 75 % as compared to conventional RCA. Cho and Lee [23] proposed to use trigonometric decomposition of the 2D-DCT which can reduce the number of multiplications to 50% of the conventional RCA. In general, the NRCA architectures do not consist of matrix transposition. However, they usually involve high communication complexity and irregular data flow.

2.5 Motion Compensated Video Coding

The main idea behind the motion estimation/compensated [5] video coding is to use the temporal and spatial correlation such that the current frame can be predicted as accurately as possible from the previously decoded frames. The more accurate the prediction is, the smaller is the entropy of the prediction error, which is usually transmitted using a lossy compression scheme to remove the perceptual irrelevant information. If the entropy of the prediction error is small, only a small amount of information needs to be transmitted and hence the compression ratio can be very high.

In general, motion compensation is defined as the process of compensating the displacement of moving objects from one frame to another. In practice, motion compensation is preceded by motion estimation, the process of finding the motion information of the object among image frames. Among other approaches, *temporal* redundancy can be efficiently reduced by combining the motion estimation and compensation process.

2.5.1 Motion Estimation

Given a reference frame and the current video frame, the objective of the motion estimation is to determine the image object in the reference frame that better matches the content of the image object in the current frame. The current frame is defined as an image at time t, and the reference frame is either defined as an image at past time $t-n$, for forward motion estimation or at future time $t+k$ for backward motion estimation.

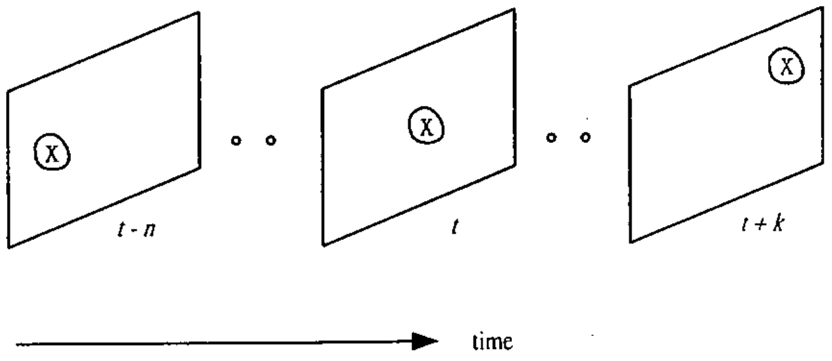


Figure 2-7 : Temporal correlation in a video sequence

Figure 2-7 shows that object X moves from left to the center of the image from time $t-n$ to t, and it moves to the upper right corner after time $t+k$. The motion of the object is usually represented by a motion vector. Block based translation motion model [37] is commonly used and this model does not consider any rotation and scaling of the objects, which simplifies the motion estimation process at the expense of decreased accuracy. A motion vector can be specified in integer or fractional pixel increments. Fractional pixel motion compensation involves interpolation of the pixel values in the reference frame.

For block based motion estimation algorithm (BMA), each image frame is divided into regular non-overlapping small blocks of $N \times N$ pixels as shown in Figure 2-8. For each block in the current frame, its motion vector is estimated from the reference frame over a predefined searching area by evaluating a matching criterion that yields the best matching. The relative positions of the two blocks define a motion vector associated with the current image block.

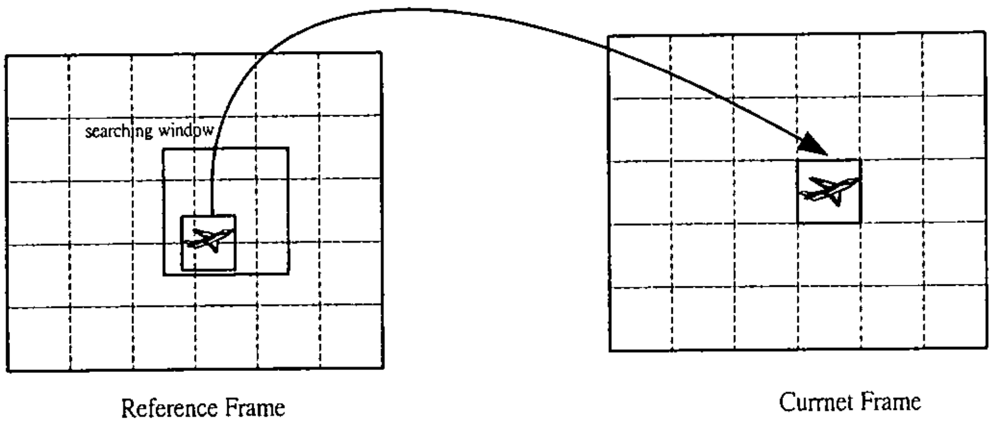


Figure 2-8 : Block Based Translation Motion Estimation

2.5.2 Matching Criteria

A matching criterion is used to measure the similarity between the current block and the reference block to determine the accuracy of the motion estimation process. Block based similarity measure is commonly used in the matching criterion since the block matching estimation algorithm assumes that all pixels within an image block have the same amount of movement.

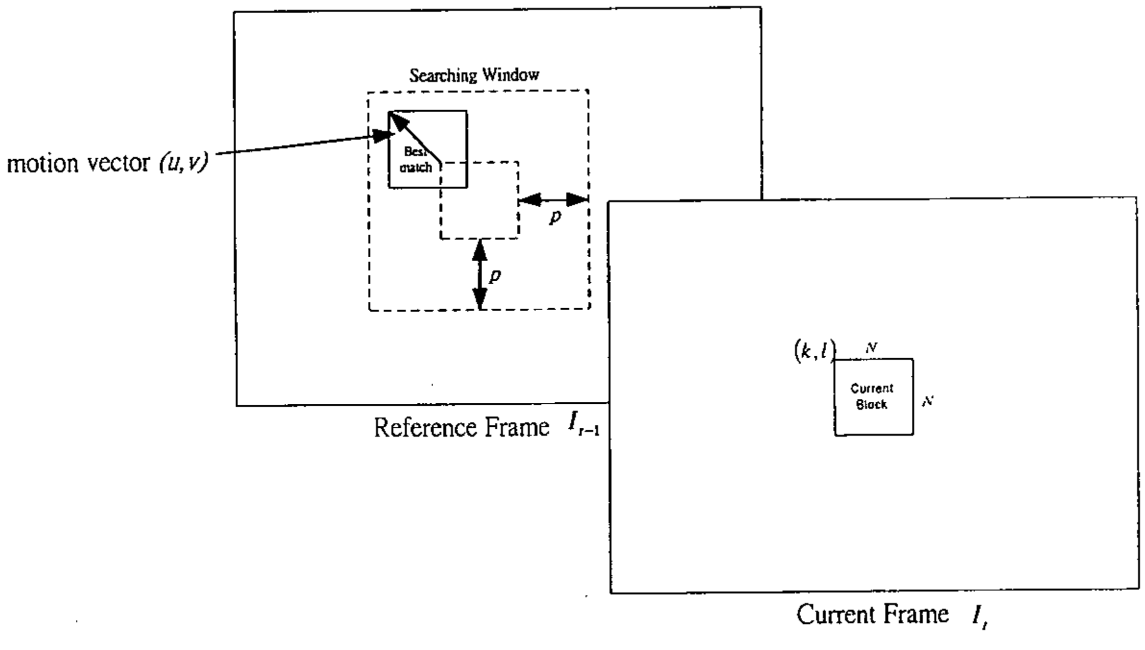


Figure 2-9 : Motion vector estimation process

Let $I_t(k,l)$ be the pixels of the image block with size $N \times N$ in the current frame at position (k,l) and $I_{t-1}(k-u,l-v)$ be the pixels in the reference frame where u,v are defined in $-p \le u,v \le p-1$ and p is the search range of the searching window as depicted Figure 2-9. The motion vector, $\vec{d}$, is chosen to minimize the matching criterion,

$$\vec{d} = \arg \min_{(k,l)} \sum_{i,j} \left\| I_t(i,j) - I_{t-1}(i-u, j-v) \right\| \qquad \text{for } i, j = 0,1,\ldots\ldots, N-1 \quad (2\text{-}10)$$

The widely used distance measures are the quadratic norm $\|x\| = x^2$ and the absolute value $\|x\| = |x|$. Hence, the Mean Square Error (MSE) is defined as,

$$MSE(u,v) = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left[I_t(i,j) - I_{t-1}(i-u, j-v) \right]^2 \qquad (2\text{-}11)$$

and the Mean Absolute Difference (MAD) is defined as,

$$MAD(u,v) = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left| I_t(i,j) - I_{t-1}(i-u, j-v) \right| \qquad (2\text{-}12)$$

The MSE measure gives the most robust result on the error measure between the current block and the reference block. However, the MSE involves multiplication, which introduces computational burden to the motion estimation process. Instead, MAD is usually employed due to the fact that simple arithmetic is needed. The simplicity of the MAD enables the error measure to be easily mapped into VLSI which further speedups the motion estimation process. Other matching criteria such as the Pel Difference Classification (PDC) [38] have been proposed. For the PDC, each pixel in the image block is classified as the matching pixel and mismatched pixel, and a threshold is used to perform the classification on the pixels. The displaced position from the current block with the maximum number of matched pixels in the reference block is then chosen as the motion vector. This approach provides a low complexity alternative to the traditionally used matching criterion since it involves counting and comparison, and no addition and multiplication is needed, which is greatly suitable for VLSI implementation.

2.6 Fast Algorithms For Motion Estimation

Apart from the reduction on the computational complexity of the matching criterion, the searching process still requires to find the best-matched block within the search window. However, the optimal motion vector can only be guaranteed by performing an exhaustive search which evaluates the matching criterion over the search window for every possible candidate displacement vectors. This is called the Full Search Algorithm (FSA). The FSA appears to be able to find the global minimum of the MAD value but is highly computationally expensive. To perform motion estimation in real time, the number of operations required by the FSA is often too high. To reduce the computational complexity, a number of fast search algorithms have been proposed [33, 39-43, 47-50]. These algorithms reduce the number of operations by either limiting the number of locations searched or by computing fewer pixel differences per search location to find the motion vector but may not be able to guarantee securely the minimum MAD value can be found. Moreover, these techniques rely on an assumption that the MAD monotonically increases as the search location moves away from the direction of global optimal distortion.

The computational requirement of the FSA can be estimated as follows. For each image block with $N \times N$ pixels and has a search range of $\pm p$ for both horizontal and vertical directions. Then for each motion vector there are $(2p+1)^2$ search locations. At each search location (u,v), $N \times N$ pixel errors have to be computed. Each pixel error computation requires three operations, namely, a subtraction, an absolute value evaluation and one addition. If we assume these three operations require the same computational resources, then the total number of operations per image block is $(2p+1)^2 \times N \times N \times 3$ operations. For an image with 720×480 at 30 frames per second and the image block with 16×16 pixels, the number of operations required would be 29.89 GOPS and 6.99 GOPS for $p = 15$ and $p = 7$ respectively. With this extremely high computational complexity requirement of the FSA, it is infeasible to implement the motion estimation algorithm in real-time application. Hence, several fast Block Matching Algorithm (BMA) have been developed to reduce the heavy computations of FSA, such as three-step hierarchical search algorithm [43], parallel hierarchical one dimensional search [44], pixel decimation [33], the adaptive approach [47-50], etc. In the following of this section, we will discuss some of the fast algorithms that are widely used in video coding applications.

2.6.1 Three-Step Hierarchical Search

Many fast block matching algorithms have been developed successfully to reduce the computational complexity of the motion estimation process. Among all of these BMAs, the three-step hierarchical search (TSHS) [43] is considered as one of the best algorithms in terms of its significant reduction on the computational complexity and its block matching performance. The TSHS algorithm reduces the number of computations by limiting the number of locations searched to find the motion vector in a coarse-to-fine manner. Figure 2-10 illustrates the procedures of the TSHS algorithm, with an example of motion vector $(-7,5)$. In the first step, nine sparsely located candidates are picked out and their MAD is computed. In the second step, the search is focused on the area centered at the winner of the first step (L2) which has the lowest MAD among the nine candidates, but the distance between candidate locations are shortened by half. In the same manner, step three compares MAD's of nine locations around the winner of the second step and then gives the final motion vector. For a larger search window, additional search steps may be incorporated. The maximum number of search location is reduced to $8k+1$ where k is the number of steps involved in the searching process. Hence, the computational complexity can be dramatically reduced.

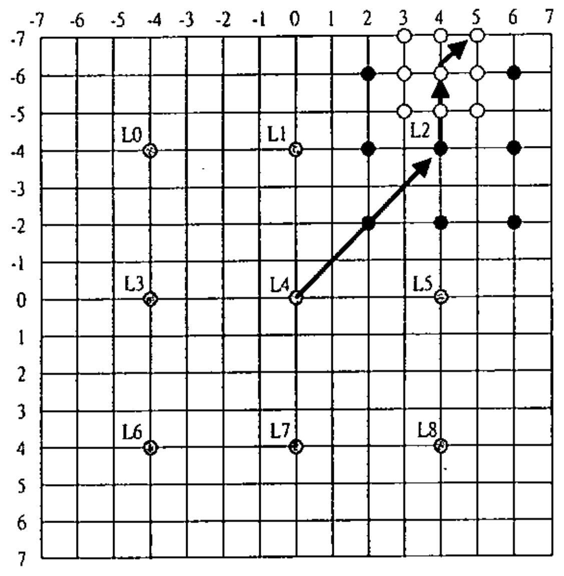


Figure 2-10 : The Three-Step Hierarchical Search (TSHS) block matching algorithm

It is no doubt that the TSHS greatly reduces the motion estimation computational effort, however, the checking points in the first step of TSHS are allocated uniformly in the search window which may not be very appropriate for some blocks in which small motions are present. Therefore a new three-step search algorithm (NTSS) [41] was proposed to accommodate this problem. In this algorithm, eight extra checking points are added, which are the eight neighbours of the search window center in the initial step. Due to this center-biased checking point pattern in the first step, the NTSS is more robust, and produces smaller motion compensation errors as compared to the TSHS.

2.6.2 Pixel Decimation

Hierarchical search algorithms such as TSHS and NTSS rely on a monotonically increasing MAD around the location of the optimal vector to iteratively determine the current block displacement location. However, the searching process may be trapped by local minimum since not all the search locations are visited. The final motion vector estimated might have a higher MAD than that of the optimal motion vector. Different from limiting the search locations, we may reduce the number of pixels that are involved in the computation of the matching criterion. This is another approach that can effectively reduce the computational complexity of the motion estimation process. Since block matching is based on the assumption that all pixels within a block have the same amount of motion, the motion of the block can be estimated by using only a fraction of pixels in the block in principle. However, if too few pixels are used, this will eventually reduces the accuracy of the motion vector estimated. Hence, pixel decimation must be done in a careful manner so as not to reduce the motion estimation accuracy. In [43], Koga et al. proposed to subsample an unfiltered image by a factor of 4 as shown in Figure 2-11. Only the shaded pixels are used in the computation of the matching criterion. As mentioned before, since only a uniform fraction of pixels enters into the matching criterion, the error of the motion vector estimated could be increased.

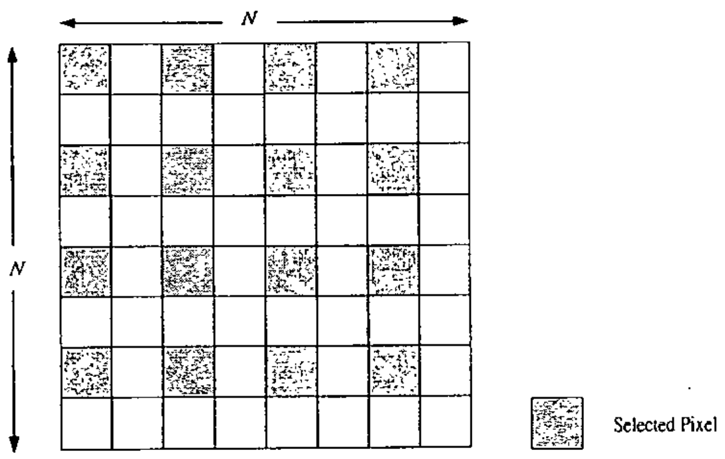


Figure 2-11 : 4-to-1 pixel decimation

In order to reduce the drawback introduced by the pixel decimation technique, Liu and Zaccarin [33] proposed to use an alternating pixel decimation pattern scheme such that all pixels in the image block will be used in the computation of the matching criterion. Figure 2-12 shows a block of 8×8 pixels with each pixel labelled as **a**, **b**, **c**, or **d** in a regular pattern. Then **A**, **B**, **C** and **D** are the subsampled pattern that consist of all the **a**, **b**, **c**, and **d** pixels respectively. If only one pattern of pixels, e.g. **A** is used in the block matching, the computation complexity is reduced by a factor of 4 as compare to the full search. However, if only one fixed pattern is used in the block matching, the accuracy of the motion vector estimated will be degraded. Thus, by alternating the pixel patterns and associating a different pattern for each neighbouring search location, all the pixels within the image block are covered within four search locations.

a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d
a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d
a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d
a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d

Figure 2-12 : Pattern of pixels used for computing the matching criterion with a 4-to-1 pixel decimation

The alternating pattern pixel decimation process is explained as follows. If pattern **A** is used in the search location (x, y), then it is also used at $(x+2i, y+2j)$ for i, j integers within the search area. Similarly, pattern B, C and D are used for locations $(x+2i, y+1+2j)$, $(x+1+2i, y+1+2j)$ and $(x+1+2i, y+2j)$ respectively. For each of the subsampling patterns, we choose the displacement that minimizes the MAD over the search locations where the pattern is used as the motion vector for that pattern. Then for each of the four motion vectors obtained, we compute the matching criterion again but using all the pixels. The one that has the minimum MAD among the four is selected as the final motion vector.

Besides to reduce the search locations and pixel subsampling techniques, hierarchical motion estimation techniques [34-36] are also considered to be efficient schemes to improve the performance of the motion estimation process. By using multi-resolution approach, a typical Gaussian pyramid is formed and the motion search ranges are allocated among different pyramid levels. The block matching is started at the lowest resolution pyramid level to obtain an initial estimation of the motion vectors. The computed motion vectors are then propagated to the next higher resolution level, where the motion vector is refined and again propagated to the next level until the highest resolution level is reached and the final motion vector is obtained.

Although many fast algorithms were proposed to reduce the computational complexity of the block matching algorithm, the number of operations required is still very high that is difficult to implement in software for real time applications. Table 2-2 shows the number of operations per second required to perform the motion estimation using various block matching algorithms. Recent advances of VLSI technology enable many applications that require high computational complexity can be efficiently implemented using hardware. The BMA which require huge computational is desirable to implement in VLSI to reduce the computational burden of the processor.

Search Algorithm	Operations per block	Operations per second For 720×480 at 30 frames/s	
		P=7	P=15
Full Search	$(2p+1)^2\times N\times N\times3$	6.998 GOPS	29.891 GOPS
3-Step Search [43]	$(8k+1)\times N\times N\times3$	0.7776 GOPS	1.026 GOPS
Pixel Decimation [33]	$\frac{(2p+1)^2\times N\times N\times3}{4}$	1.7495 GOPS	7.4727 GOPS
Adaptive pixel decimation based on neighbour pixels [48]	$(2p+1)^2\times44\times3$	1.203 GOPS	5.1375 GOPS

Table 2-2 : Operations requirement of various block matching algorithms using MAD error criterion

2.7 Motion Estimation Processor Implementation

The BMA is computationally very demanding and it is always desirable to develop special VLSI architectures for the BMA. A large number of architectures have been proposed for various BMAs [53-61]. Most of these architectures use array of Processor Elements (PE) to achieve high computation rates. FSA is the algorithm that is commonly implemented in VLSI architectures [53-58] due to the fact that its data access pattern is highly regular and lends itself naturally to parallel processing. Among other fast algorithms, three-step search algorithm also has many proposed implementations [60, 61] since it is computationally less intensive. In this section, we will discuss two typical VLSI architectures for block matching algorithm used in motion estimation. The first architecture maps the full search algorithm into VLSI architecture that is based on data-flow designs and allows sequential inputs but performs parallel processing with 100 percent efficiency of resources utilization. For the second architecture, three-step search algorithm is mapped into a fully pipelined parallel architecture with sophisticated data arrangement and memory configuration. Furthermore, the memory interleaving technique of the approach reduces the memory access bandwidth.

2.7.1 Full Search Motion Estimation Processor

Several dedicated hardware implementations of the BMA have been reported and most of the realizations are concern full search because of its regular data flow and low control overhead. Principally, these realizations are systolic array with high local communication and exhibit regular structure. In [54], a VLSI architecture of FSBMA utilizing multiple processor elements (PE) which perform subtraction, absolute value evaluation and accumulation was proposed. Special data flow has been derived to keep the PE's as busy as possible. As shown in Figure 2-13, $a(k,l)$ and $b(u,v)$ are the pixels from the current block and the search area respectively. p and p' denote sequences of previous frame data from different portions of the searching area.

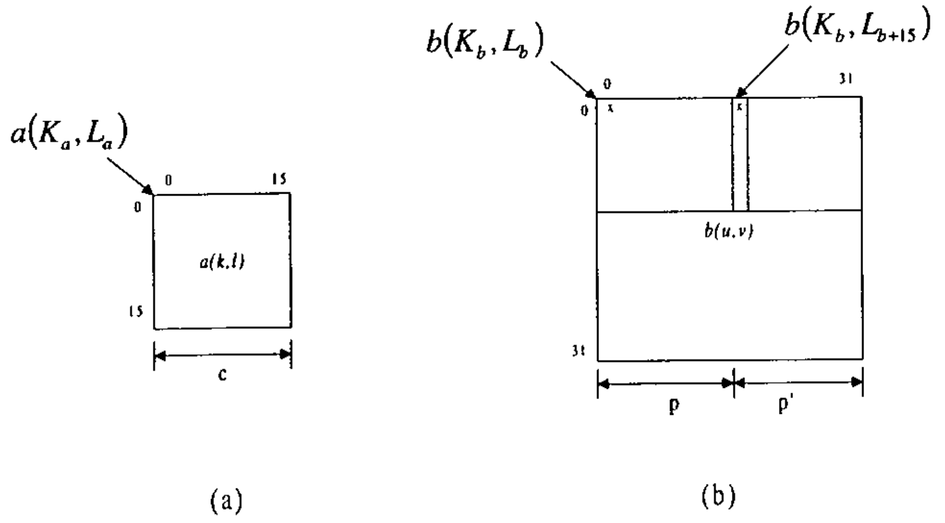


Figure 2-13 : Data access (a) From current block, (b) From previous frame

The pixel, $b(K_b, L_b + 15)$, is used at 16 different searching positions in the computation of the matching criterion $MAD(u, v)$.

$$MAD(u,v) = \sum_{i=0}^{15} \sum_{j=0}^{15} \left| a(K_a + u, L_a + v) - b(K_b + u + i, L_b + v + j) \right| \qquad \text{for } v = 0,1,\cdots,15$$

The computation of $MAD(u, v)$ for all values of v can be computed in parallel with multiple PEs while the pixel $b(K_b, L_b + 15)$ is broadcasted to all processors that need it. Since the reference frame pixel is being broadcast to all processors, the memory access to the reference block is greatly reduced. Moreover, with careful arrangement on data flow of the pixel data, both of the pixel data from current block and reference block can be input sequentially to the motion estimation processor. The novel data flow arrangement and broadcast of the reference frame pixel data to all PEs contribute to a regular architecture and achieve 100 percent efficiency.

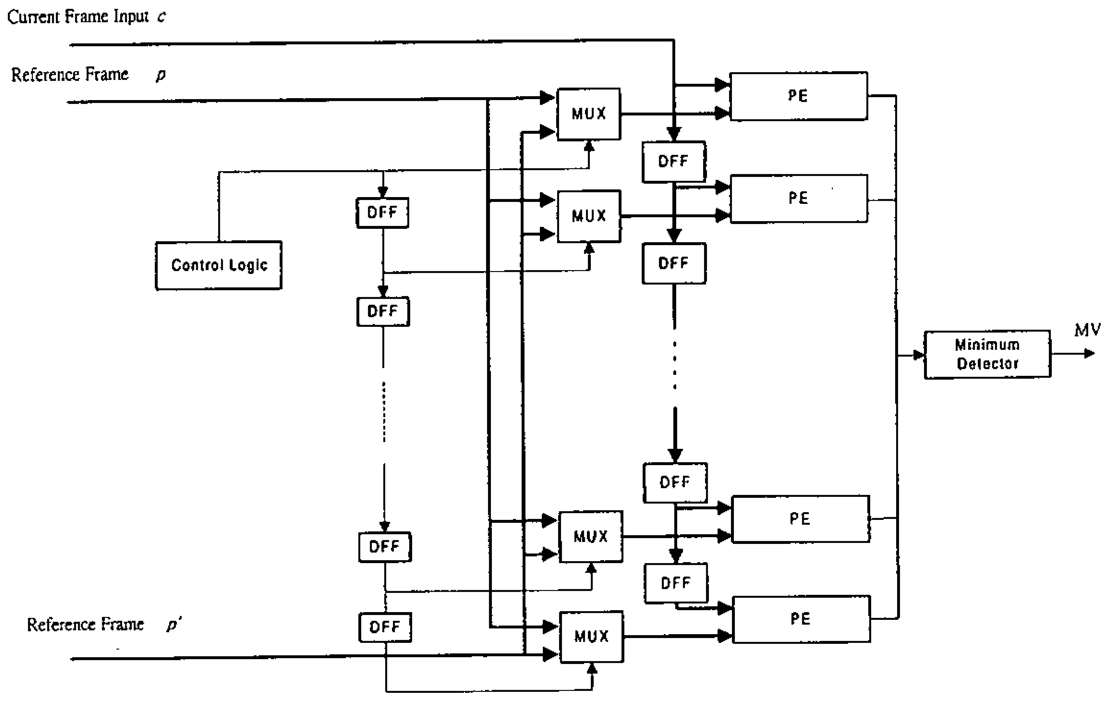


Figure 2-14 : Implementation of the FSBMA (broadcast reference frame data)

Figure 2-14 shows the implementation of the FSBMA with reference frame data broadcasted to all PE's. As shown in the figure, the pixel data of the current block and reference block are inputted sequentially. The current block pixel c is fed into a common bus structure with delay elements where the pixel data are available to one PE during one cycle and available to the next PE in the next clock cycle. The reference pixel data p and p' are broadcasted through two common buses and one of them will be input to all PE through a 2-to-1 multiplexer. For each PE, the absolute difference between the current pixel and reference pixel is computed and accumulated. The block diagram of a PE is shown in Figure 2-15.

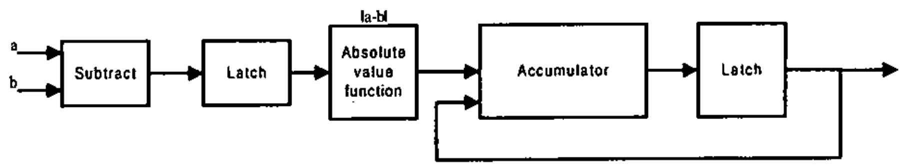


Figure 2-15 : Architecture of the PE

The broadcast reference frame motion estimation processor exhibits regular data flow and low control overhead that is suitable to implement in VLSI. However, the number of clock cycle required to evaluate the motion vector for one image block would be 256 for $p = 7$ and even longer with a wider search range.

2.7.2 Three-Step Search Motion Estimation Processor

The low complexity and the possibility for high accuracy of the TSHS is a potential solution for high-speed video applications. However, some architectural considerations prevent this algorithm from being widely used in real-time systems. The variable distances between candidate locations and the unpredictable data access pattern complicate the control scheme which will lower the efficiency of computation kernel, and make it difficult to reuse data for reducing the number of external memory accesses. Moreover, for the dependence between successive steps, the execution must be in sequential, so the latency of each step needs to be as short as possible to achieve high-speed operation.

The regular and continuous data flow of the FSBMA allows the data buffer to be distributed within or nearby the PE's. However, for TSHS, the variable distances between candidate locations break the continuity of the data flow and hence a memory buffer with multi-port access is more suitable. John et al. [60] proposed a hardware architecture that maps the TSHS using pipeline with 9 PE's. The architecture addressed the problem of complex data addressing by (1) using on chip buffer to reduce external memory access, (2) the residual memory interleaving for parallel data accesses, and (3) the PE function redistribution for eliminating the interconnection overheads between PE's and on-chip buffers.

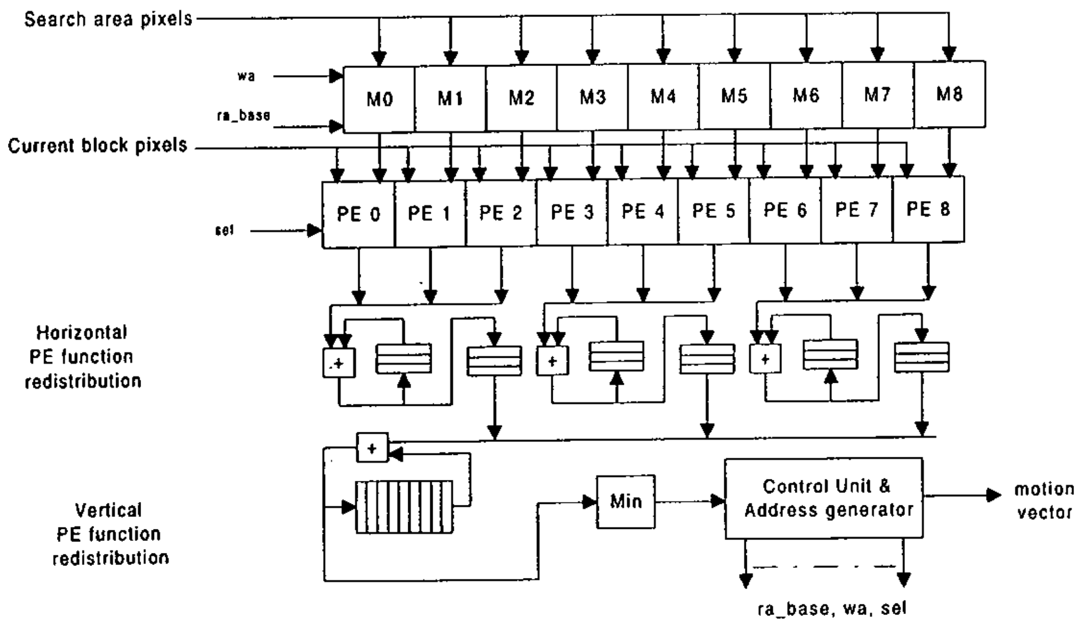


Figure 2-16 : 9-PE TSHS architecture with PE function redistribution

The architecture shows in Figure 2-16 sequentially inputs current block pixels and broadcasts them to all PE's. M0 ~ M9 are on chip buffers to store the search area pixels and they are internally available to the corresponding PE. The on chip memory buffer reduces the external access to the reference pixel, while a residual memory interleaving scheme is used to allow 9 PE's to access the search area pixels from the memory buffer simultaneously. Instead of evaluating a certain candidate location using a fixed PE, each PE evaluate partial results of different candidate locations at different clock cycles. This function redistribution technique eliminates the need of switching network between the memory modules and PE's.

Hardware implementation of the DCT algorithms and block matching algorithms bridge the gap between the advanced video coding techniques and real-time application requirement. A generic hybrid video coding scheme [15-18] was developed which removes the spatial and temporal redundancies by combining the use of the DCT and ME/MC algorithm respectively and this scheme is widely used in various video compression standard.

2.8 Hybrid Video Coding

The Discrete Cosine Transform (DCT) and Motion Estimation/Compensation techniques have been successfully applied into various international video compression standards [15-19]. A mixed scheme using these techniques as a generic hybrid video coding scheme is found to be effective and efficient in terms of its compression capability and decoded video quality. A block diagram of the generic hybrid video coder is shown in Figure 2-17. At a higher level, the encoder first decides whether it is going to code the input frame using intraframe or interframe coding [45, 46]. For intraframe coding, 2D-DCT will be applied to the image blocks of the video frame where the transform coefficients are quantized, zig-zag scanned and run-length / Huffman coded. If interframe coding is selected, the encoder performs motion estimation to find the motion vector for each macro block within the video frame.

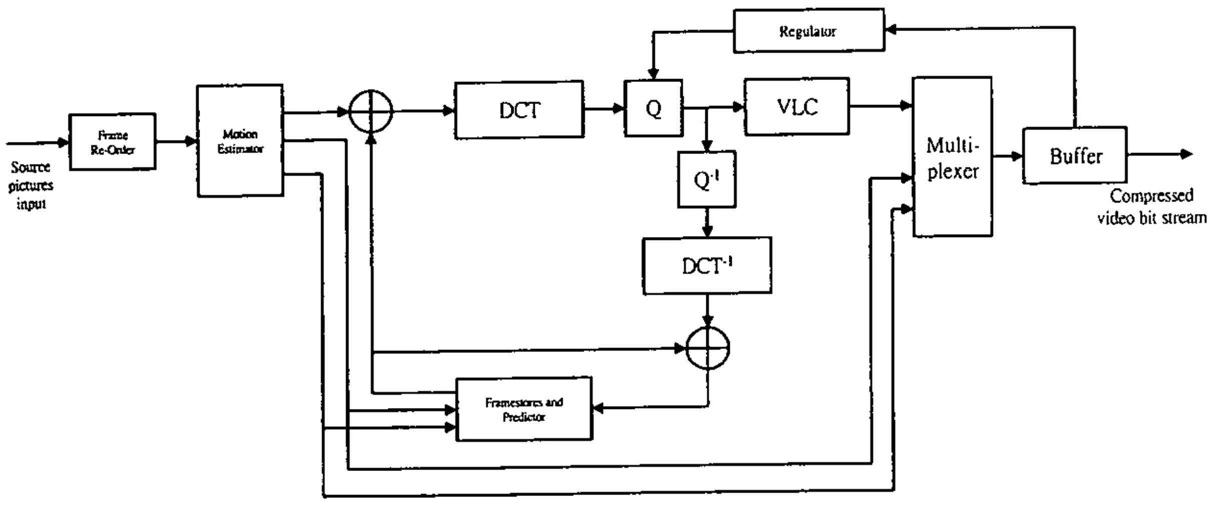


Figure 2-17 : Generic Hybrid Video Coder

An advanced technique called bi-directional prediction [15-19] which make uses of two reference frames, one for forward and one for backward motion predictions, and interpolates the results to form the predicted frame. In practices, multiple frame buffers are needed to store the reference frames for backward prediction and a certain delay will be imposed to the encoding and decoding process. However, for real-time applications, such as video-conferencing, the distance between successive reference frames is kept small to reduce the delay. Figure 2-18 illustrates various frame types and the frame dependencies in the motion prediction process.

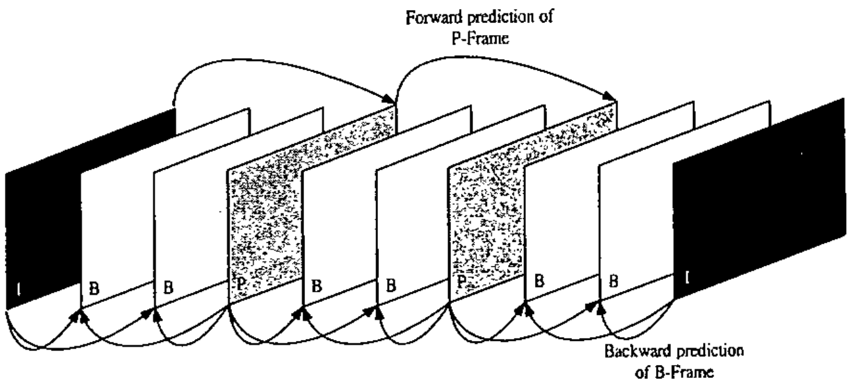


Figure 2-18 : Frame types and dependencies for forward/backward prediction

A digital signal processor (DSP) is expected to give a good solution for a flexible hardware implementation of the generic hybrid video coding scheme. However, due to the high computational complexity of the compression scheme, it is not possible to achieve real-time operation using conventional DSP's. Owing to the rapid advances in Very Large Scale Integrated Circuits (VLSI), dedicated VLSI chips [65] have been developed for various parts of the video coding applications. Special purpose video coding DSP's [66, 67] have also been developed which are capable to encode the real-time video into compressed bit streams.

Chapter 3

Discrete Cosine Transform Processor Realization

3.1 Introduction

The application of the Discrete Cosine Transform [9] has been spread over various fields of signal processing, especially for speech, image and video processing. This is mainly due to the fact that the DCT approaches the statistically optimal Karhunen-Loeve Transform (KLT) [7, 8] for highly correlated signals. With the recent advances of VLSI technology, the real-time video processing for the multimedia applications appears possible. Several international standards for image and video coding algorithms, such as CCITT H.263 [15, 16], MPEG [17-19] have been recommended. These standards are based on the hybrid video coding scheme that combines the transform coding and predictive coding techniques.

For high transform rates, highly parallel data-flow architectures are used, and for low transform rates, the main focus has been on designing processor based architectures for the DCT. In the later case, typically one multiplier-accumulator block is re-used for computing all DCT operations, which leads to the problem of efficient pipelining, register design, I/O design and scheduling policies. Considering the VLSI implementation for the DCT in the literature, the distributed arithmetic [26-29] is the most widely used technique since it provides an efficient way to realize the multiplications involved in the DCT computation by read only memory (ROM).

Bit-parallel data-flow architecture is the key to achieve high transform speed. However, for applications that the transform speed is not high enough to justify a bit-parallel data-flow architecture, the alternative is to map the algorithm execution such that operational elements are re-used for different operations. Two methods exist for carrying this out. The operations of the algorithm can be implemented at the word-level, leading to processor architectures or the algorithms that can be implemented using the bit-level, leading to a bit-serial architecture. The advantages of a bit-serial architecture implementation are that the interconnection overhead becomes almost negligible since only one wire is needed for each interconnection. In the contrary, word-level design needs a large number of buses for interconnection where each bus contains multiple wires which consume a large amount of chip area. In addition to using serial approach, the DCT processor hardware complexity can be further reduced by using fixed coefficient multipliers.

In this chapter, a Discrete Cosine Transform Processor architecture which uses bit-serial approach and features low complexity, high speed operation and regular structure will be presented. Although data conversion buffers are required at the input and output stage, a large amount of hardware complication can be saved with the use of bit-serial approach. The use of signed digit representation of the DCT coefficient also leads to an efficient architecture while preserving the speed performance of the serial multiplier. An interface circuit is specially designed to provide convenient data input/output arrangement that is suitable to interface to the PCI bus controller. A prototype of the processor is implemented using a EPLD device to evaluate the real-time performance.

3.2 Serial Multiplier

In order to meet the demand for high speed operations, many DCT processor were implemented making use of parallel array multipliers [29]. However, parallel array multiplier requires a large amount of hardware and consequently consume a large amount of power [70]. With applications that the chip area is critical, serial multiplier is often the preferred alternative. The shift-add with the carry propagation technique performs the multiplication one row at a time as illustrated in Figure 3-1. The multiplicand b is to multiply a for each row with one digit shifted after each operation. By using this natural process, one can implement the multiplier using shift-add in parallel. However, the carry generated from each partial product must be propagated at full length for every row multiplication, which introduces excess delay and reduces the speed of the multiplier. Carry look-ahead adder can be used to improve the speed of the carry propagation adder with the expenses of additional hardware.

$$\begin{array}{cccccccc}
 & & & & a_3 & a_2 & a_1 & a_0 \\
 & & & & b_3 & b_2 & b_1 & b_0 \\
\cline{5-8}
 & & & & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 \\
 & & & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & \\
 & & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & \\
 & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & & \\
\hline
y_7 & y_6 & y_5 & y_4 & y_3 & y_2 & y_1 & y_0 \\
\hline
\end{array}$$

Figure 3-1 : Multiplication matrix

Serial multiplier using Carry Save Add Shift (CSAS) [68] technique directly maps the multiplication using simple hardware. Due to its inherent structure, high speed operation and low hardware complexity can possibly be achieved. With the pipeline register between consecutive adder, no carry propagation is needed for every row multiplication. The block diagram of a 4-bit by n-bit CSAS serial multiplier is shown in Figure 3-2 with a and b as the multiplicands and the result is y_i.

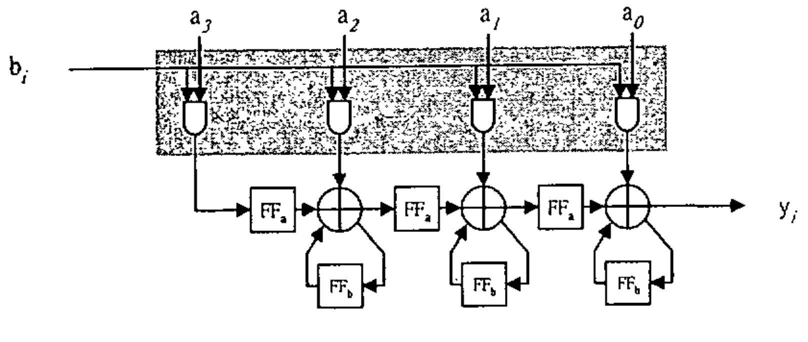


Figure 3-2 : 4 by n-bit serial multiplier

The multiplicand a is parallel input for every multiplication while b is serially shifted into the multiplier with LSB first and the product y_i is serially shifted out from the serial multiplier with LSB first. The shaded area performs the binary multiplication between a and b_i as shown in Figure 3-1. The flip-flop FF_a is the pipeline register and FF_b is used to store the carry of the previous partial product. The multiplier completes one multiplication using $L_a + L_b$ clock cycles, where L_a and L_b is the wordlength of multiplicands a and b respectively. The complexity of the serial multiplier with serial-in-serial-out configuration is $O(n)$ which is greatly reduced as compared with $O(n^2)$ of the parallel multiplier. Reduction on the hardware complexity also leads to lower power consumption but at the expense of longer computation time. The computation time can be reduced if L_b-bit carry look-ahead parallel adder is used after L_b clock cycles [68].

3.3 Signed Digit Representation

Serial multiplier offers reduction on the hardware complexity for general multiplication. Since the DCT kernel coefficients are fixed, further simplification on the multiplier is possible. When we consider to represent the DCT coefficients using the radix-2 signed digit (SD) which is represented in the general form as,

$$c = \sum_{j=0} s_j 2^{-j} \tag{3-1}$$

where $s_j \in \{-1,0,1\}$. The number of nonzero bits used to represent the fractional number determines the accuracy of the representation. Most numbers can be represented with fewer nonzero digits by using the SD representation. Since only a finite number of nonzero digits is used to represent the coefficients, certain error will be introduced to the multiplication result.

However, the multiplication result is also represented in a fixed-point format. With enough number of nonzero bits, the error magnitude can be reduced to an acceptable level.

From equation (2-3), the DCT kernel matrix can be written as,

$$T=\begin{bmatrix} d & d & d & d & d & d & d & d \\ a & c & e & g & -g & -e & -c & -a \\ b & f & -f & -b & -b & -f & f & b \\ c & -g & -a & -e & e & a & g & -c \\ d & -d & -d & d & d & -d & -d & d \\ e & -a & g & c & -c & -g & a & -e \\ f & -b & b & -f & -f & b & -b & f \\ g & -e & c & -a & a & -c & e & -g \end{bmatrix} \quad \text{where} \quad \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{bmatrix} = \sqrt{\frac{2}{8}} \begin{bmatrix} \cos\frac{\pi}{16} \\ \cos\frac{2\pi}{16} \\ \cos\frac{3\pi}{16} \\ \cos\frac{4\pi}{16} \\ \cos\frac{5\pi}{16} \\ \cos\frac{6\pi}{16} \\ \cos\frac{7\pi}{16} \end{bmatrix} \qquad (3\text{-}2)$$

The coefficients of the DCT matrix consist of seven distinct values: a, b, c, d, e, f, and g. These are coefficients existing in every column of the matrix T with different permutations. By using the SD representation, the number of addition or subtraction needed and the error for each DCT kernel coefficients are shown in Table 3-1.

Coefficient	**value**	**SD representation**	**No. of additions**	**Error Magnitude**
a	0.49039264	$2^{-1}-2^{-7}-2^{-9}+2^{-13}=0.490356$	4	3.619×10^{-5}
b	0.461939766	$2^{-1}-2^{-5}-2^{-7}+2^{-10}=0.461914$	4	2.5703×10^{-5}
c	0.415734806	$2^{-2}+2^{-3}+2^{-5}+2^{-7}+2^{-9}-2^{-12}=0.415771$	6	3.66782×10^{-5}
d	0.35355339	$2^{-2}+2^{-4}+2^{-5}+2^{-7}+2^{-9}=0.353516$	5	3.77656×10^{-5}
e	0.277785116	$2^{-2}+2^{-5}-2^{-8}+2^{-11}=0.277832$	4	4.69147×10^{-5}
f	0.191341716	$2^{-3}+2^{-4}+2^{-8}-2^{-14}=0.191345$	4	3.49861×10^{-6}
g	0.097545161	$2^{-4}+2^{-5}+2^{-8}-2^{-13}=0.097534$	4	1.09813×10^{-5}

Table 3-1 : Signed digit representation of the cosine coefficients

As shown in the above table, four to six nonzero bits are sufficient to represent the DCT kernel coefficients with the accuracy conformed to the specification that is derived in section 3.5.

With the bit-serial approach and SD representation of the DCT kernel coefficient technique, a bit-serial parallel DCT processor architecture was implemented and the architecture features regular structure and highly parallel operation.

3.4 Serial DCT Processor Architecture

The DCT algorithm (equation 2-3) requires 64 multiplications, for which the parallel implementation requires a large amount of multipliers. Several fast algorithms [20, 21] have been proposed to reduce the number of multiplications required. However, most of them require complex data flow and are not suitable for hardware implementation. Hence, in our implementation, only one stage of matrix decomposition based on the DCT kernel property is carried out to reduce the number of multiplications required and to maintain the regularity of the DCT processor.

By observing the coefficients of the DCT kernel matrix in equation (3-2), the even rows are even-symmetric and odd rows are odd-symmetric. Thus, by separating the even and odd rows of the kernel matrix T, we have

$$\begin{bmatrix} z(0) \\ z(2) \\ z(4) \\ z(6) \end{bmatrix} = \begin{bmatrix} d & d & d & d \\ b & f & -f & -b \\ d & -d & -d & d \\ f & -b & b & -f \end{bmatrix} \begin{bmatrix} x(0)+x(7) \\ x(1)+x(6) \\ x(2)+x(5) \\ x(3)+x(4) \end{bmatrix} \tag{3-3}$$

$$\begin{bmatrix} z(1) \\ z(3) \\ z(5) \\ z(7) \end{bmatrix} = \begin{bmatrix} a & c & e & g \\ c & -g & -a & -e \\ e & -a & g & c \\ g & -e & c & -a \end{bmatrix} \begin{bmatrix} x(0)-x(7) \\ x(1)-x(6) \\ x(2)-x(5) \\ x(3)-x(4) \end{bmatrix} \tag{3-4}$$

The number of multiplications required is reduced to half since the $N \times N$ matrix multiplication is replaced by two $\frac{N}{2}$ x $\frac{N}{2}$ matrix multiplications. Since the computation of the even and odd kernel is independent of each other, the even and odd kernel multiplications can be computed in parallel. By using the same technique, further decomposition on the even kernel matrix is possible but will lead to a more complicated data flow architecture. The overall architecture of the serial DCT processor is shown in Figure 3-3. The processor consists of four operation stages 1) parallel to serial conversion, 2) data pre-processing, 3) kernel matrices multiplication and 4) serial to parallel conversion.

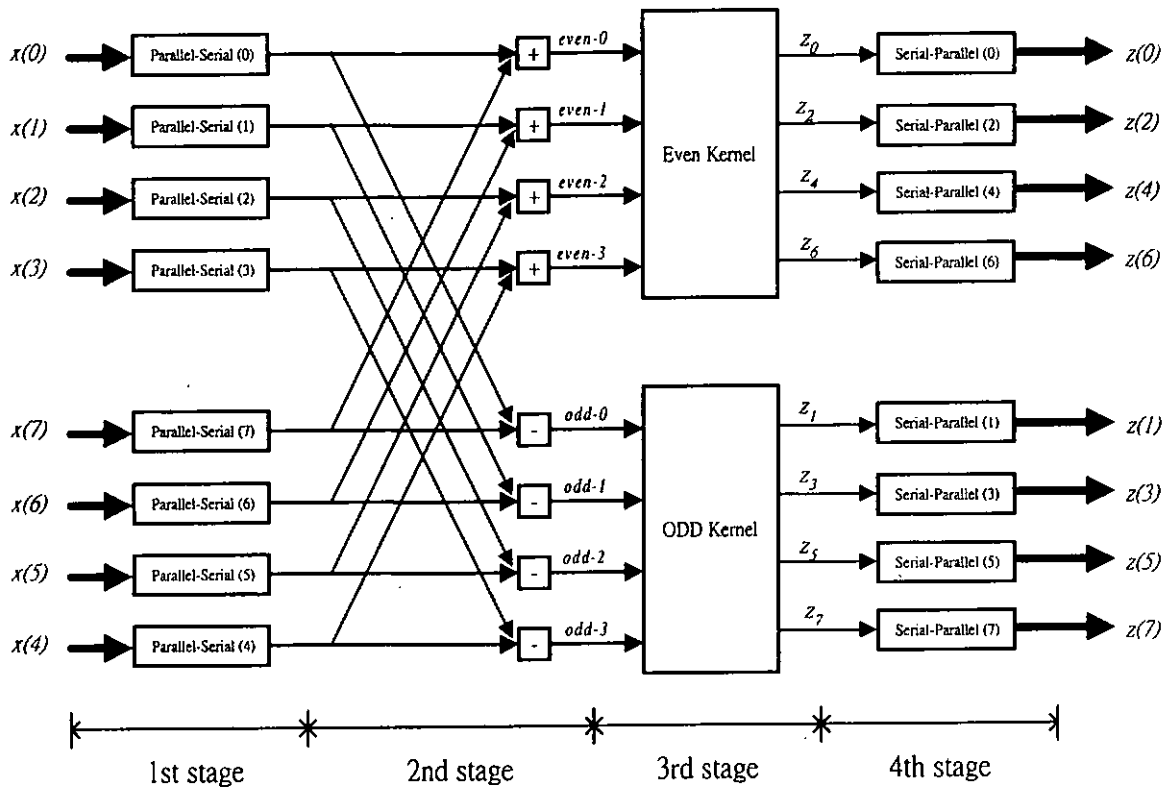


Figure 3-3 : Block diagram of the serial DCT processor

The first stage converts the parallel input data into serial format that will be used in the rest of the computation. At the second stage, the input data is pre-processed according to equations (3-3) and (3-4). Since the data is already in serial format, serial adders/subtractors are used with very low hardware complexity. After pre-processing the input data, the even and odd kernels have to perform the matrix multiplication that will be discussed in the following section. At the last stage, the transformed result is converted back into parallel form by the serial-to-parallel shift register.

3.4.1 Kernel Matrix Multiplication

The decomposition of the DCT algorithm results in two 4×4 matrices. The first matrix has three distinct values *b*, *d* and *f* and the second matrix has four distinct values *a*, *c*, *e* and *g*. The two matrix multiplications are independent of each other and hence can be computed in parallel. Figure 3-4 and Figure 3-5 show the odd and even kernel respectively. For the even kernel, it accepts the pre-processed data *even-0, even-1*, *even-2* and *even-3* and the odd kenrel accepts *odd-0*, *odd-1*, *odd-2* and *odd-3* from the second stage. The multiplication of the pre-processed data and the coefficients is performed by a bank of kernel multipliers which are optimized to achieve the optimal performance and chip area. The even and odd parts of the transformed results are outputted parallelly from the even and odd kernels respectively and will be converted back to parallel form at the last stage.

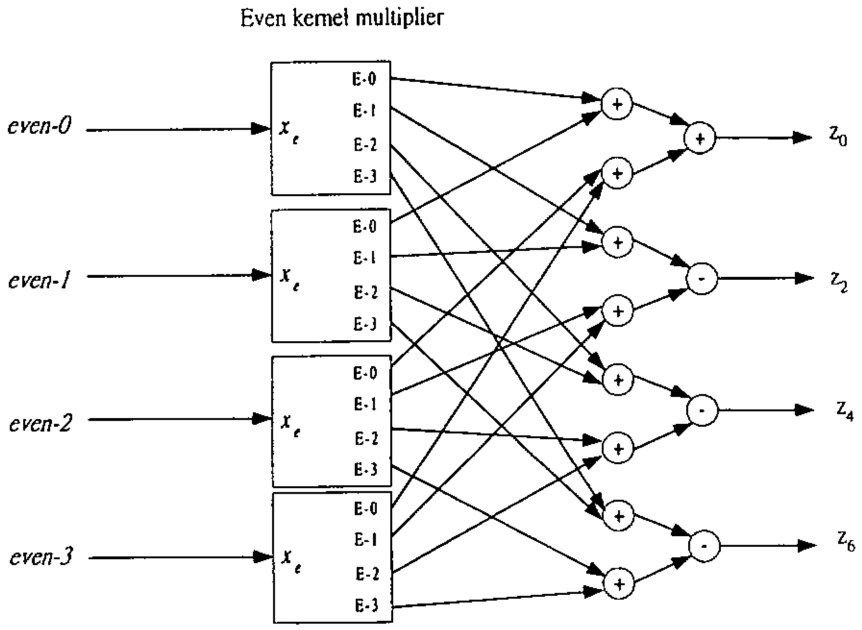


Figure 3-4 : Even Kernel

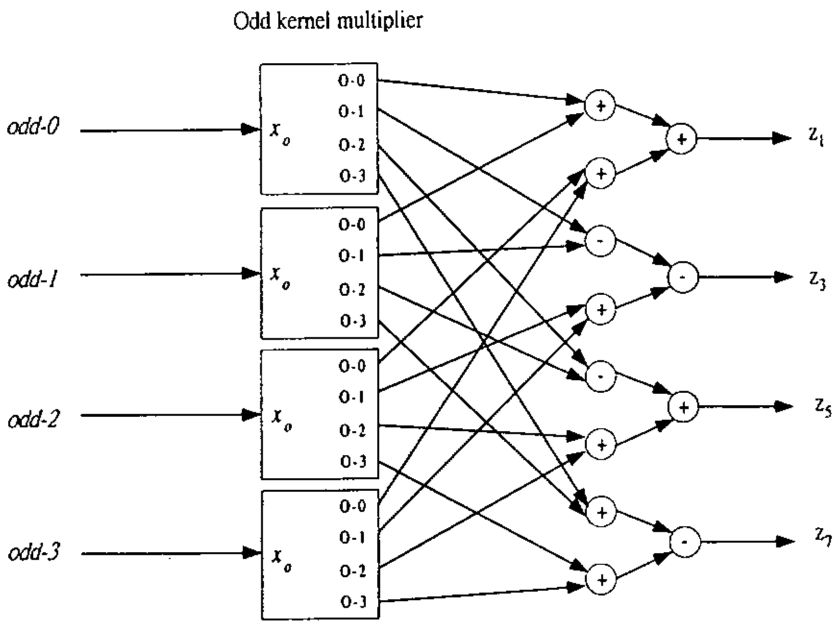


Figure 3-5 : Odd Kernel

In order to reduce the chip area and to achieve high performance, optimization on the kernel multipliers is carried out. With the SD representation of the DCT kernel coefficients as discussed in the previous section. A fewer number of nonzero digits is used to represent the coefficients as compared with the fixed point representation. When we rearrange the SD representation of the DCT kernel coefficients, the addition/subtraction can be implemented in a tree structure. The even and odd DCT coefficients can be written as,

Even coefficients:

$$\begin{aligned}
b &= \left(2^{-1}+2^{-10}\right)-\left(2^{-5}+2^{-7}\right)\\
d &= \left(2^{-2}+2^{-4}\right)+\left(2^{-5}+2^{-7}\right)+2^{-9}\\
f &= \left(2^{-3}+2^{-4}\right)-\left(2^{-8}-2^{-14}\right)
\end{aligned}$$

Odd coefficients:

$$\begin{aligned}
a &= \left(2^{-1}+2^{-13}\right)-\left(2^{-7}+2^{-9}\right)\\
c &= \left(2^{-2}+2^{-3}\right)+\left(2^{-5}+2^{-7}\right)+\left(2^{-9}-2^{-12}\right)\\
e &= \left(2^{-5}-2^{-8}\right)+\left(2^{-2}+2^{-11}\right)\\
g &= \left(2^{-4}-2^{-13}\right)+\left(2^{-5}+2^{-8}\right)
\end{aligned}$$

With the rearranged DCT coefficients with SD representation, we can map the coefficient multiplication into a structure that takes the advantage of the SD representation. Figure 3-6 and Figure 3-7 shows the even and the odd kernel multipliers respectively. As shown in the figures, each kernel multiplier performs four multiplications at a time and the hardware required is about 25 percent of the serial multiplier as described in section 3.2. The even kernel multiplier accepts the input x_e and is serially shifted into the flip-flop to generate the shifted version of the multiplicand. The shifted version of x_e inputted to a series of adder/subtractors corresponding to the SD representation as shown in Table 3-1. The products $b \cdot x$, $d \cdot x$ and $f \cdot x$ are available in parallel. The odd kernel multiplier performs exactly the same operation as the even kernel multiplier but it outputs the $a \cdot x$, $c \cdot x$, $e \cdot x$ and $g \cdot x$ in parallel.

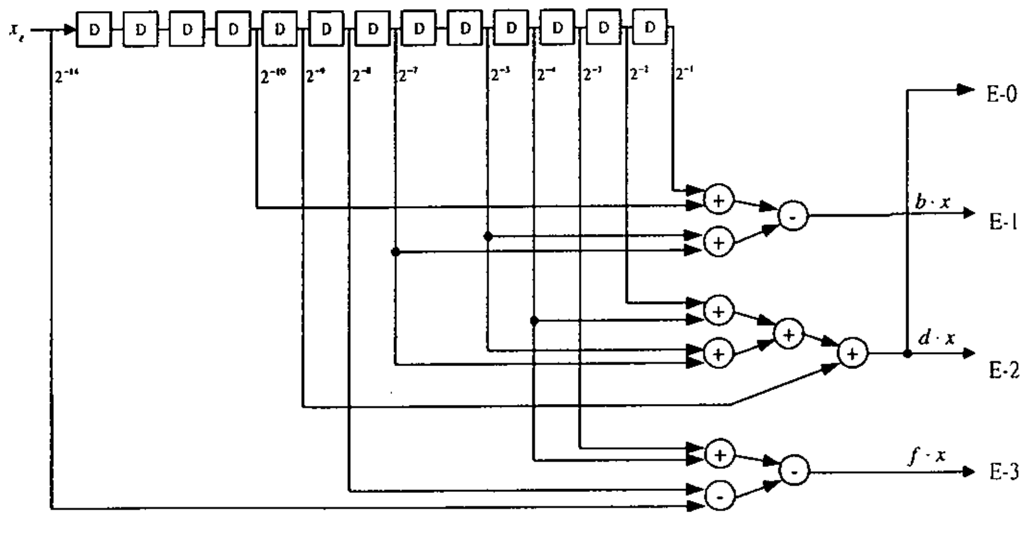


Figure 3-6 : Even Kernel Multiplier

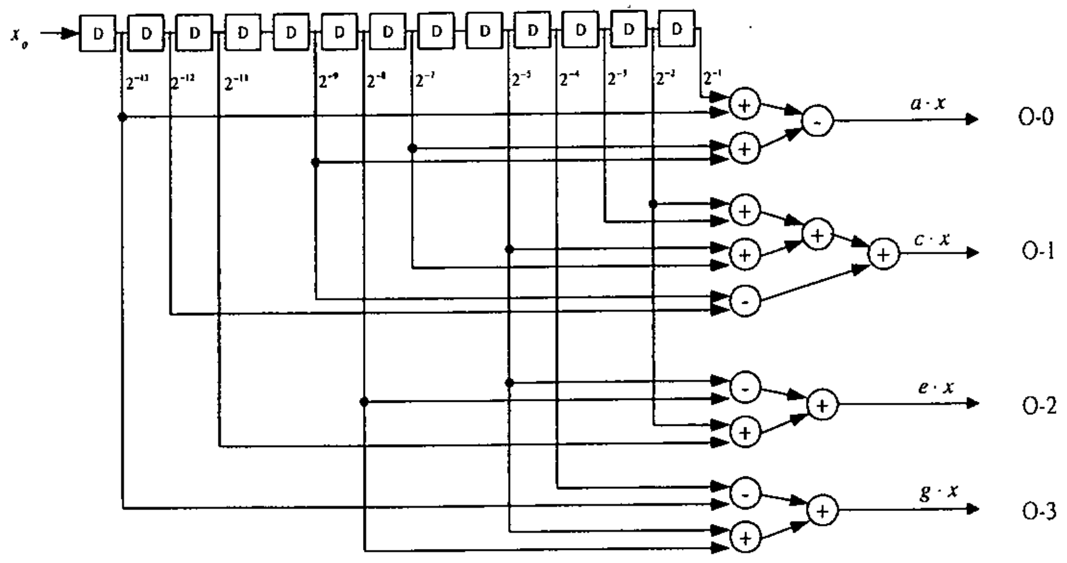


Figure 3-7 : Odd Kernel Multiplier

Since the multipliers consume most of the hardware resources in the whole architecture, a simplification of the multiplier structure should contribute greatly to an efficient implementation of the serial DCT processor. With a special arrangement, the input buffers and the parallel-to-serial shift register are merged together to reduce the hardware complexity. With the optimization on the kernel multiplier and the reduction of input buffers, a low complexity DCT processor architecture is achieved.

3.4.2 Processor Interface

The processor architecture described above accepts parallel data and outputs the transformed results in parallel. However, in general processor architecture, it is usually to have a common data path for the input and output data. In a common data bus structure, the data are input/output sequentially to and from the processor. Hence, a special circuitry should be designed to allow a convenient interface to the processor. Figure 3-8 shows the interface circuit for the processor. Input buffers are required to hold the input data since the DCT processor requires the data available in parallel. In fact, the DCT processor consists of a bank of parallel-to-serial registers and can be used as buffers to store the input data. The input data $x(0)$, $x(1)$, ... and $x(7)$ are inputted to the corresponding register which is selected by the logic controller. The transformed results $z(0)$, $z(1)$, ... and $z(7)$ are output sequentially through the multiplexer and is also controlled by the logic controller. The sequence of input and output data is shown in Figure 3-9.

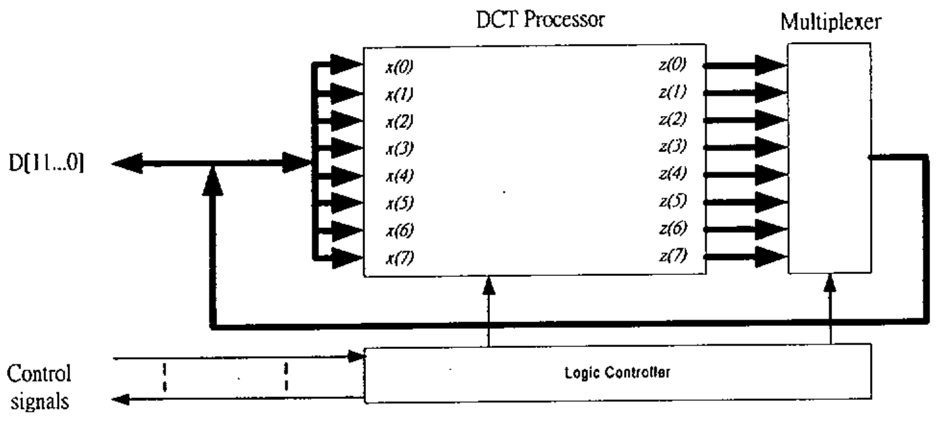


Figure 3-8 : Interface circuit for the DCT processor

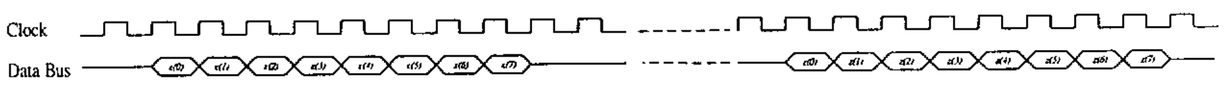


Figure 3-9 : Timing diagram of the DCT processor

3.4.3 DCT Processor Prototype

After a thorough simulation and verification on the techniques and architectures described above, a serial DCT processor has been designed and implemented using the ALTERA EPF10K50-3 EPLD device [72]. The EPLD device features RAM based look-up-table (LUT) for flexible interconnection configuration. The estimated gate count for the overall architecture including the serial DCT processor core and the control circuits for external interface is about 4500 gates. The critical path of the processor determines the maximum obtainable operating frequency. In most of the DCT processor implementations, the critical path is mainly located at the multiplier or the adder/accumulator [25-27, 29] since they use bit-parallel approach. However, due to bit-serial approach which is used in our implementation, the critical path is mainly located at the interface circuitry. The serial adder/subtractor, including implicit pipeline registers that the accumulate gate delay is equal to the total gate delay of the full-adder. The simulated maximum operating frequency of the overall architecture including the interface circuit is above 45MHz.

The PCI bus has become a standard bus in the PC architecture and breaks the PC data transfer bottleneck. With a PCI bus controller, our serial DCT processor is interfaced to the PC through the PCI bus and the data transfer rate is above 66 Mbytes/sec. Figure 3-10 shows the interconnections between the DCT processor and the PCI bus controller.

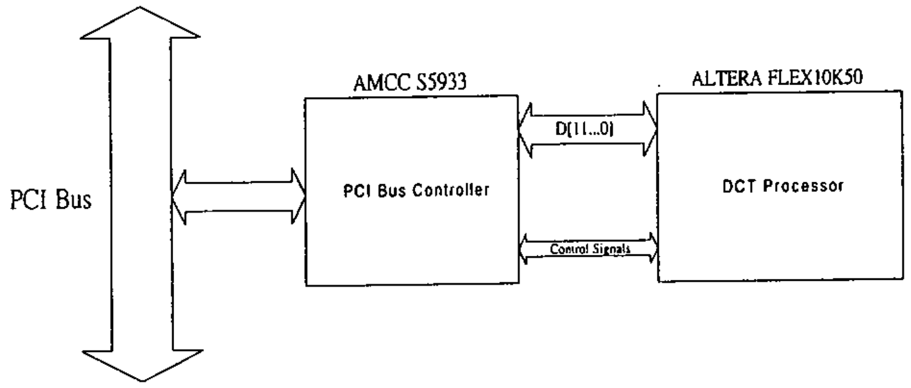


Figure 3-10 : PCI Bus Interface

The PCI bus controller is configured as bus master which means that the data transfer is originated and controlled by the PCI bus controller rather than the PC. This configuration relieves the burden on the PC CPU. Figure 3-11 and Figure 3-12 show the prototype board of the DCT Processor and the PCI bus controller board respectively.

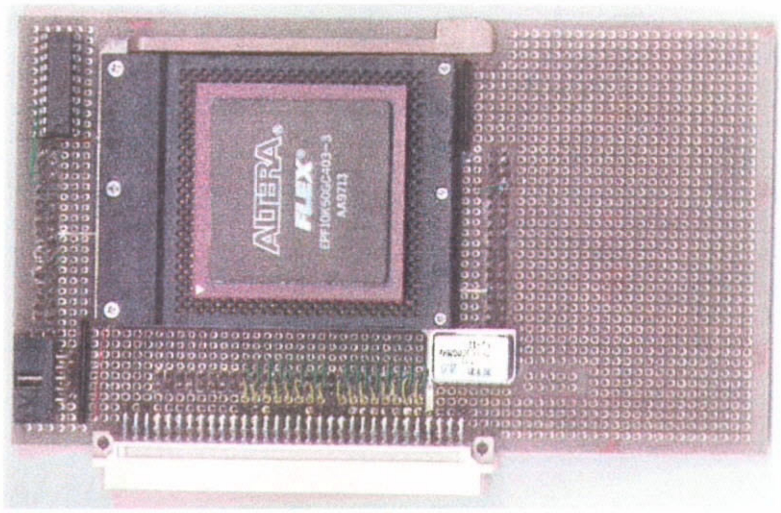


Figure 3-11 : Serial DCT Processor prototype board

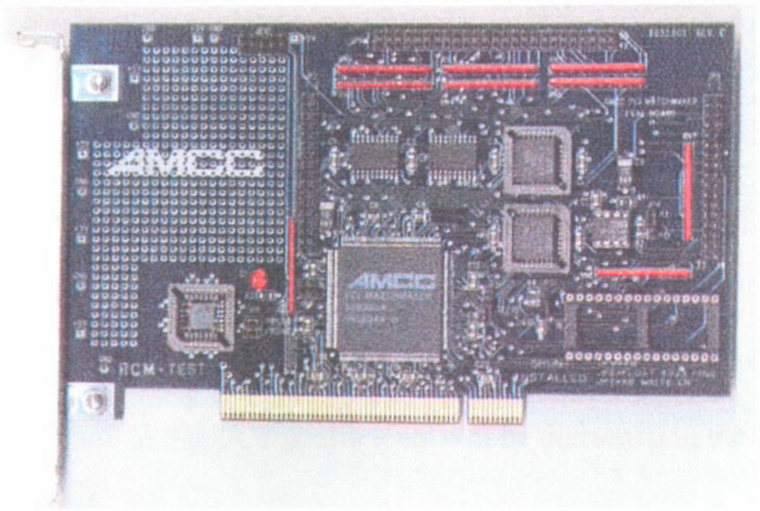


Figure 3-12 : PCI Interface board

3.4.4 DCT Processor Specification

According to the architecture that is described in the previous section, the DCT processor accepts 12-bits signed digit input and output 12-bits signed digit results. The total clock cycles to complete one 1D-DCT is 32 which include the input and output data conversion and internal delay. The summarized characteristics of the DCT processor are shown in Table 3-2.

Inputs	12 bits
Outputs	12 bits
Max. Clock Rate	45MHz
Clock cycles to complete one 1D-DCT	32
No. of gates	4500

Table 3-2 : Serial DCT processor characteristics

3.5 Error Analysis

Fixed-point arithmetic is usually employed in hardware implementation of digital signal processing algorithms because of high speed and small area requirement. In order to reduce the hardware complexity, short fixed-point wordlength is usually preferred. However, the balance between shorter wordlength and better accuracy must be carefully adjusted to achieve the optimal performance. Recently, there has been a number of fixed-point error analyses on several DCT algorithms [74, 75]. For the completeness of our DCT processor implementation, an analysis on the errors produced from the finite wordlength effect and the signed digit representation of the DCT coefficients will be discussed in the rest of this section. The error model used in our analysis is based on the assumptions that all error sources are uncorrelated with each other and all errors are uncorrelated with the input.

There are two sources of error in our implementation, namely the coefficient quantization and the finite wordlength effect. The coefficient quantization error is due to the signed digit representation of the coefficients. Since bit-serial approach is used in our implementation, internal wordlength during computation will not be truncated, however, the truncation of wordlength of the result generates the second error.

3.5.1 Signed Digit Representation Error

Since direct form of the DCT algorithm is used for our implementation, the fixed-point error analysis is rather straightforward. In our implementation, the signed digit representation is used to represent the DCT coefficients to reduce the hardware complexity of the kernel multiplier. However, due to a finite number of nonzero digits is used and the error imposed will affect the accuracy of the computation result and the computation error should be minimized to an acceptable level. Moreover, the computation results are also represented in fixed-point, and the maximum error should be less than 1 LSB. The upper bound of the maximum allowable error ensured that the error of the transform results should not exceed 1 LSB. The error produced due to the quantization on the coefficient can expressed as,

$$e_u = \left| \frac{C(u)}{2} \sum_{m=0}^{7} x(m) \cos\left(\frac{\pi(2m+1)u}{16}\right) - \frac{C(u)}{2} \sum_{m=0}^{7} x(m) \left[\cos\left(\frac{\pi(2m+1)u}{16}\right)\right]_q \right| \tag{3-5}$$

where $\left[\cos\left(\frac{\pi(2m+1)u}{16}\right)\right]_q$ is the quantized coefficient. Minimization of e_u required to minimze the error of the quantized coefficients. When we consider the maximum error that will be introduced from the quantization error, equation (3-5) can be modified to,

$$e_u = \sum_{m=0}^{7} x_{\max} \times \left| \cos\left(\frac{\pi(2m+1)u}{16}\right) - \left[\cos\left(\frac{\pi(2m+1)u}{16}\right)\right]_q \right| \tag{3-6}$$

The above error expression assumed that all coefficients have equal probability to contribute errors to the computation and e_u will become maximum when the input is maximum. From the above derivation, the maximum error of each quantized coefficient should be,

$$e_{\max} = \frac{e_u}{x_{\max} \times 8} \tag{3-7}$$

The upper bound of the quantization error for each coefficient guarantees that the maximum error of e_u will not exceed 1 LSB. The minimum number of nonzero bits to represent the DCT kernel coefficients is based on this criterion.

3.7 Experimental Results

In order to verify the correctness and to evaluate the performance of the DCT processor, a series of experiments were carried out to measure the speedup for using the DCT processor as compared to software implementation. Throughout these experiments, we have used a Pentium PC with 133 MHz operation frequency and PCI bus interface. With the use of a custom device driver for the PCI bus interface controller (AMCC S5933) and a prototype board for the DCT processor, the execution time of each step involved in the computation of the 2D-DCT is measured.

For software implementation of the 2D-DCT computation, row-column decomposition approach is used and the computation time is $t_{Software\_DCT} = 306\mu S$. There are three distinct steps necessary for the computation of the 2D-DCT using the DCT processor.

Step 1 : Setup and initiate the DMA transfer using the PCI bus controller (t_{setup_DMA})

Step 2 : Compute the 2D-DCT in the DCT processor ($t_{hardware\_DCT}$)

Step 3 : Check interrupt status from the PCI bus controller (t_{status})

The speedup for using the DCT processor can be computed as,

$$Speedup = \frac{t_{Software\_DCT} \times n}{t_{setup\_DMA} + (t_{hardware\_DCT}) \times n + t_{status}} \tag{3-8}$$

where n is the number of 8×8 blocks used in each computation. The more the blocks used, the higher the efficiency for computation since the time required to setup the DMA transfer and checking the status from the PCI interface controller is saved. The time used in each step and the speedup when using different numbers of 8×8 blocks in each computation is illustrated in Table 3-3.

$$t_{setup\_DMA} = 14\mu S$$

$$t_{hardware\_DCT} = 32.5\mu S$$

$$t_{status} = 9\mu S$$

n, **number of** 8×8 **blocks**	**Speedup**
1	5.514
5	8.248
20	9.0936

Table 3-3 : Speedup when using differene number of 8x8 blocks

Apart from measuring the speedup in a perfect situation, we also consider the speedup in a real environment where a MPEG-2 software encoder is used to measure the actual speedup to be achieved when the DCT processor is in use. The time required to encode a 352×240 video sequence with 150 frames, and using I-frame only, is 139 second when we use software DCT implementation. When we use the DCT processor, only 75 second is needed and the speedup is 1.853 when $n = 20$. The reason for achieving a lower speedup as compared to the ideal case for using the MPEG-2 encoder is that the computation time of the DCT involved in the encoding process is about 40%. This shows that a further development on efficient algorithms to reduce the computation time of the remaining encoding process, such as the motion estimation, coefficients quantization, variable length coding (VLC), etc. is necessary.

3.8 Summary

A low hardware complexity with high speed operation and regular structure Discrete Cosine Transform Processor architecture is presented in this chapter. The DCT takes the advantages of simple hardware and low interconnection complexity of the bit-serial approach and parallel operation with multiple operation elements. With the signed digit representation of the DCT coefficients technique, more than 75 percent reduction on the hardware for the DCT kernel matrix multiplication can be saved. The optimized kernel matrix multiplier exhibits highly pipeline operations and the maximum achievable operation frequency of the processor is 45MHz. The implementation of the DCT processor including external interface circuits uses about 4500 gates. Experimental results show that the maximum speedup is above 9 when using 20 8×8 blocks in each computation. The computation time of the DCT is about 40% of the entire encoding process, the achievable speedup for the whole system should be lower than this figure since most of the computation time, in this case would spend on the motion estimation process when full MPEG mode is employed (with P-frame and B-frame). In order to reduce the computation time of the overall encoding process, efficient algorithms and hardware structures for the motion estimation should be developed and we will show some of our work to relief the computation burden of the motion estimation in the following Chapters.

Chapter 4

Adaptive Motion Estimation Algorithm

4.1 Introduction

As mentioned in chapter 2, Discrete Cosine Transform (DCT) [9] and motion estimation/compensation (ME/MC) [5] are very efficient to remove the spatial and temporal redundancies respectively. Hybrid video coding that combines the aforementioned algorithms is widely adopted in international video coding standards [14- 19]. However, the computation complexity of the DCT and ME/MC is very high that real-time implementation using software algorithm is almost infeasible. Hence, many fast algorithms have been proposed to reduce the computational complexity of the DCT [20- 24] and ME/MC [33- 42]. Also, architectures for the DCT [25- 31] and ME/MC [53-61] are proposed. Most of these architectures use bit-parallel approach and require a large amount of hardware resources. However, a low complexity implementation of the DCT algorithm is often desired. For this purpose, a DCT processor features low complexity, high performance and regular structure has been discussed in Chapter 3.

ME/MC video coding technique achieves high compression ratio by applying the rate distortion theory [2]. The quality of the decoded image is directly proportional to the output bit rate. Thus in the motion compensated coding scheme, the coding performance depends heavily on the accuracy of the estimated motion vector. Block based motion estimation algorithm has been widely used in video coding because of its simplicity and coding efficiency of motion vectors. For the block matching algorithm, the present frame is divided into two-dimensional small blocks of $N \times N$ pixels. Each block in the current frame estimates its motion vector by evaluating some matching criteria over the blocks in the reference frame and selecting the block which yields the closest matching. Currently, the mean absolute difference (MAD) criterion is considered as a good candidate for low bit-rate video applications. This is mainly due to its relative ease of hardware implementation, although it may suffer from degraded performance. More robust criteria such as the mean square error (MSE) require multiplications and are not suitable for hardware realization.

As mentioned before, the accuracy of the motion estimation process affects the coding performance heavily. Hence, the trade-off between quality and speed of the motion estimation algorithms must be carefully justified. Among all search algorithms, the Full Search Algorithm (FSA), for which all possible displaced candidates within the search area in the reference frame are searched, gives the best solution in the viewpoint of prediction error. However, it is well known that the FSA requires extensive computations. Thus, many fast and efficient algorithms [33- 40] have been developed to reduce the computational complexity by limiting the number of search locations. Nearly all algorithms rely on the assumption that the distortion function increases monotonically as the search location moves away from the global optimum location. However, this assumption does not always hold exactly true for some image sequences that have multiple local minima. As a consequence, the fast search algorithm would easily be trapped at a local minimum and this results inaccurate estimate on the motion vector.

The drawback of limiting the number of search locations has been partly resolved by using hierarchical search algorithms [35, 36]. An alternative approach that reduces the number of pixels contributing to the computation of the matching criterion was proposed by Koga et al. [43] that uses a pixel decimation technique to reduce the computational complexity. However, since only a regular fraction of the pixels enters into the matching computation, the use of these regular subsampling techniques can seriously affect the accuracy of motion vector detection. Liu and Zaccarin [33] have successfully improved the simple pixel decimation technique by using a scheme with alternating pixel decimation patterns. The subsampling patterns are alternated over the locations searched so that all pixels of a block contribute to the computation of the motion vector. This approach could gives a good estimation of the motion vector when the image block has a nearly uniform intensity. However for some image blocks that have high activities, some details may be neglected and it probably would introduce excess prediction errors.

In [48], an adaptive pixel decimation for block motion vector estimation is firstly introduced. This approach adaptively selects a set of representative pixels in each block for matching. Only one pixel among its neighbours is selected and sufficiently enough when the neighbour pixels have similar intensity. But if the pixels have significantly distinct value, all pixels are used in order to prevent the image edges to give misleading results for the matching criterion. However, the drawback is that the number of selected pixels has large variation in each block and it is difficult to implement in practice.

Thus, in this chapter, a number of pixel patterns are predefined which are based on the edge content of possible blocks. Since the selected pixel patterns used here have to be predefined before the actual implementation, it is easy for hardware realization. Furthermore, the prediction error as compared with the uniform pixel decimation is significantly reduced by devising the predefined pattern set properly. The result is very close to the exhaustive search without pixel decimation. Furthermore, the performance of our algorithm is better than that of Liu and Zaccarin's [33] algorithm.

Details of the algorithm are shown in the following sections while part of the results of this chapter have been published in [49].

4.2 Pattern Based Pixel Decimation

In regular pixel decimation, a regular pixel pattern is usually used for the computation of the matching criterion. However, it could easily be misled by some image textures that would probably introduce excessive prediction errors. Thus, an adaptive pixel decimation [48] was proposed which is to vary the number of selected pixels to be used for the computation of the matching criterion. The algorithm uses less pixels when the block has a uniform intensity. But for high activity blocks, more pixels are employed for the MAD matching criterion. This adaptive approach can reduce the prediction error as compared with the regular pixel decimation. However, due to the adaptive nature of the algorithm, the variation of the number of pixels selected is very large and is not suitable for real-time implementation. In order to resolve the difficulty for a practical realization of this adaptive technique, a predefined set of pixel patterns is employed in our algorithm.

Each pixel pattern of an image block represents a certain local activity such as textures and edges. Hence the number of possible patterns increases exponentially with the size of the image block. Unfortunately, recent video coding standards [14-19] often use 8×8 or 16×16 block sizes. Our strategy to be adopted for devising the pixel patterns is that, the pattern size is chosen such that the corresponding image part of which could at most have one edge. It is reasonable to assume that a block of 4×4 pixels would only contain a single edge, and let us refer it to as a "sub-block". In our algorithm, we firstly divide the 8×8 or 16×16 image block into four or sixteen 4×4 sub-blocks respectively. Then, each 4×4 sub-block selects its corresponding predefined pixel pattern according to the information contained such as the intensity and edge direction. Finally, these four or sixteen selected pixel patterns will give the most representative pixels for the image block. And also, these representative pixels are to be involved in the MAD matching criterion to compute the motion vector. The selected representative pixels greatly affect the accuracy of the estimated motion vector and hence a pixel selection scheme should be devised.

4.2.1 Pixel Selection In Uniform Sub-Block And Edge Sub-Block

The predefined set of pixel patterns should be designed such that important information within the sub-block can be represented. Based on the nature of sub-blocks, they can be classified as *uniform* sub-block and *edge* sub-block. For *uniform* sub-block, only the simplest comparison is required. Since the sub-block has uniform intensity, only two pixels as shown in Figure 4-1 could contribute to the computation of the matching criterion and this could be sufficiently enough. For blocks that have sufficient intensity variation are classified as *edge* sub-block. For this type of sub-blocks, more pixels have to be involved in the matching criterion to reduce the prediction error.

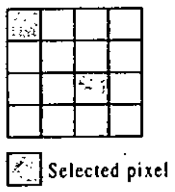


Figure 4-1 : Predefined pixel pattern for uniform sub-block

There are an excessively large number of possible physical situations for a sub-block that contains an edge. Since small blocks are used, the edge within the block is confined to a resolution of 45°. Eight basic edge patterns (c_i: where i is the edge orientation) with orientations 0°, 45°, 90°, 135°, 180°, 225°, 270° and 315° are defined. Figure 4-2 shows the 0°, 45°, 90° and 135° edge patterns. The remaining orientations are 180°, 225°, 270° and 315° which have contrast in the opposite direction. These are defined in order to devise a set of pixel patterns to be used as edge sub-blocks for the matching criterion.

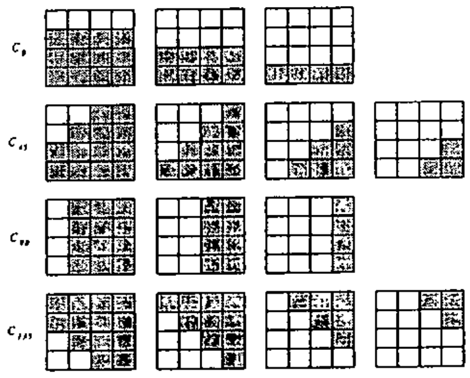


Figure 4-2 : Edge patterns with different orientations

A small set of predefined pixel patterns containing the important edge information is constructed and is according to the assumption that only a single edge is contained within each sub-block. The derivation of the predefined edge patterns is similar to those used in classified DCT or adaptive DCT image coding, however, the edge patterns defined here are used for the computation of the matching criterion, but not for the coding of the image. As depicted in Figure 4-3, we define the normal lines to different edge orientations. Since these lines contain significant intensity variations, a pair of pixels is selected along a normal line for the case that two adjacent pixels are in different intensity regions. As a result, different pixel patterns have been designed as shown in Figure 4-3.

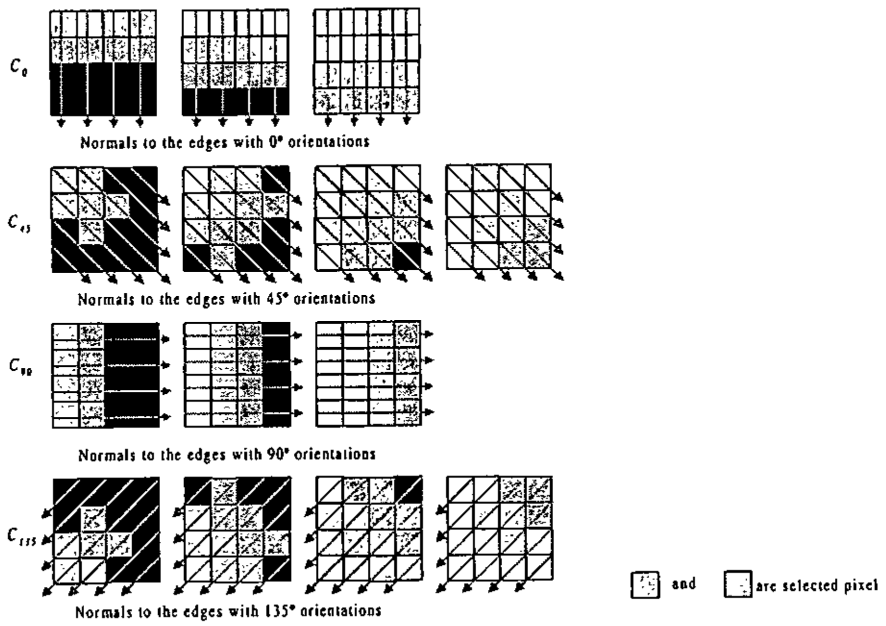

C_0
Normals to the edges with 0° orientations
C_{45}
Normals to the edges with 45° orientations
C_{90}
Normals to the edges with 90° orientations
C_{135}
Normals to the edges with 135° orientations
and
are selected pixel

Figure 4-3 : Possible edge patterns with its corresponding pixel pattern

4.2.2 Edge Pattern Matching

The matching scheme to maps each 4×4 sub-block into their corresponding pixel patterns must be of low computational complexity that does not introduce excessive burden to the motion estimation process. For each 4×4 sub-block, we denote the intensity of each pixel as $I(i,j)$ where $0 \le i,j \le 3$. A natural measure of the edge is the discrete gradient $\nabla I(i,j) = \{\Delta_x I(i,j), \Delta_y I(i,j)\}$ where the directional derivative can be approximated as [11],

$$\Delta_x I(i,j) = \sum_{i=2}^{3}\sum_{j=0}^{3} I(i,j) - \sum_{i=0}^{1}\sum_{j=0}^{3} I(i,j) \tag{4-1}$$

and

$$\Delta_y I(i,j) = \sum_{i=0}^{3}\sum_{j=2}^{3} I(i,j) - \sum_{i=0}^{3}\sum_{j=0}^{1} I(i,j) \tag{4-2}$$

The gradient magnitude and gradient orientation with each image sub-block are given respectively by,

$$|\nabla I(i,j)| = \sqrt{(\Delta_x I(i,j))^2 + (\Delta_y I(i,j))^2} \tag{4-3}$$

and

$$\angle \nabla I(i,j) = \tan^{-1}\frac{\Delta_y I(i,j)}{\Delta_x I(i,j)} \tag{4-4}$$

The gradient magnitude is used to determine the type of the sub-block. If $(|\nabla I(i,j)|)^2 \le (|\nabla I|_{\min})^2$, where $|\nabla I|_{\min}$ is the minimum value of the discrete gradients containing edge feature, the block is determined to be a uniform sub-block; otherwise it is an edge sub-block. The gradient orientation $\angle \nabla I(i,j)$ is quantized by an increment of 45° and it maps the sub-block into their corresponding edge pattern subspace, c_i. A unique mapping is achieved by selecting the distribution of shaded and unshaded pixels in the edge pattern, as depicted in Figure 4-2, to be the closest to that of the sub-block. This can easily be accomplished by simply counting the number of shaded and unshaded pixels in the sub-block, and choosing the edge patterns having the best match. And then, an appropriate pixel pattern according to Figure 4-3 is assigned for the sub-block.

After we can assign appropriate pixel patterns to a number of 4×4 sub-blocks within each 8×8 or 16×16 block, the most representative pixels are obtained. Only these pixels contribute to the computation of the matching criterion (MAD). We select k motion vectors which are the ones to give the smallest MAD for further comparison. Then, we compute the MAD matching criterion for each of the k motion vectors using all pixels. The one that has the minimum MAD among k motion vectors is selected as the final motion vector.

4.3 Computational Complexity

The computational complexity of an algorithm affects the feasibility of its implementation. The improvement on the performance should not introduce a large amount of extra computational effort. The computational complexity of a motion estimation algorithm can be described by a function in terms of the number of search locations and the number of selected pixels. To measure the computation complexity of our adaptive pixel decimation algorithm, we denote the number of pixels selected as S. If all candidate locations are searched, the total number of search locations required is $(2p+1)^2$ with $\pm p$ search range for both vertical and horizontal directions. Then the total number of pixels required for evaluation is $S\times(2p+1)^2$. To compute the MAD of each candidate location, each pixel involve three operations namely subtraction, absolute value and accumulation. If we assume the three operations require the same amount of computational complexity, the total number of operations to evaluate one motion vector is $3\times S\times(2p+1)^2$. In our algorithm, the edge pattern matching requires to compute the gradient magnitude and gradient orientation of each 4×4

sub-block as defined in equations (4-1) to (4-6). This extra computation complexity can be estimated as follows.

To compute the discrete gradient $\nabla I(i,j)$ for each 4×4 sub-block, we have to compute $\Delta_x I(i,j)$ and $\Delta_y I(i,j)$ first. If we consider to precompute part of the summations first, then only 18 additions are needed to compute both $\Delta_x I(i,j)$ and $\Delta_y I(i,j)$. Then equation (4-1) and (4-2) can be written as,

$$\Delta_x I(i,j) = (a+b)-(c+d)$$

$$\Delta_y I(i,j) = (b+d)-(a+c)$$

where $a=\sum_{i=2}^{3}\sum_{j=0}^{1} I(i,j)$, $b=\sum_{i=2}^{3}\sum_{j=2}^{3} I(i,j)$, $c=\sum_{i=0}^{1}\sum_{j=0}^{1} I(i,j)$ and $d=\sum_{i=0}^{1}\sum_{j=2}^{3} I(i,j)$

After we compute the discrete gradient, the computation of the gradient magnitude $(|\nabla I(i,j)|)^2$ for each 4×4 sub-block requires two multiplications and one addition. No further computation is required if the sub-block is classified as uniform sub-block; otherwise if the sub-block is classified as edge block, one addition multiplication is required to compute the gradient orientation $\angle\nabla I(i,j)$. Table 4-1 shows the number of operations required for pattern matching of each 4 x 4 sub-block.

	No. of additions	**No. of multiplications**
Uniform sub-block	19	2
Edge sub-block	19	3

Table 4-1: Number of operations required for pattern matching of each 4 x 4 sub-block

If the 16×16 block size is used, then there are totally 304 additions and 48 multiplications required for the worst case situation. However, the operations for the pixel pattern matching can be neglected since the computation of the matching criterion dominates the computation complexity. The number of operations required for the FSA and 4-to-1 regular pixel decimation are shown in Table 4-2.

Algorithm	No. of operations (p=7)	No. of operations (p=15)
Full Search (FSA)	57600	246016
Regular 4-to-1 pixel decimation	14400	61504

Table 4-2 : Number of operations required for different motion estimation algorithms

4.4 Experimental Results

In order to evaluate the performance of the pattern based pixel decimation algorithm, many sequences of motion pictures have been tested; these include *80* frames of the "football", "tennis" and "salesman" sequences with 352×240 pixels. The maximum allowed displacement *(p)* is set to *8* with a block size of 16×16. We compare different algorithms using the prediction errors (MSE) of the motion-compensated frames. Although the simulations were run using many sequences, we only present the results on "football" sequence since they are typical ones among all results from the sequences.

The prediction errors (MSE) of a regular 4-to-1 pixel [43] and that of the pattern based pixel decimation algorithm with $|\nabla I|_{\min} = 20$ and $k=4$ are shown in Figure 4-4. It is seen that our pattern based algorithm is significantly better than that of the regular 4-to-1 pixel decimation [43]. Note that, the performance of the FSA without pixel decimation, the Liu and Zaccarian's [33] pixel decimation and adaptive pixel [48] are also shown in Figure 4-4. It is seen that the pattern based pixel decimation algorithm has an MSE very close to other pixel decimation approaches. In Table 1, the average numbers of selected pixels for different pixel decimation algorithms are shown. The average numbers of pixels used for our pattern based pixel decimation are 1.6 less than that of the regular 4-to-1 pixel [43] and the Liu and Zaccarin's [33] approach. The results also show that the pattern based pixel decimation algorithms are very effective.

Algorithm	**Average pixel selected**
Without pixel decimation	256
Regular 4-to-1 pixel decimation [43]	64
Liu and Zaccarian's pixel decimation [33]	64
Adaptive pixel decimation based on neighbour pixels [48]	44
Pattern based pixel decimation	40

Table 4-3 : Comparison of average selected pixels for different algorithms

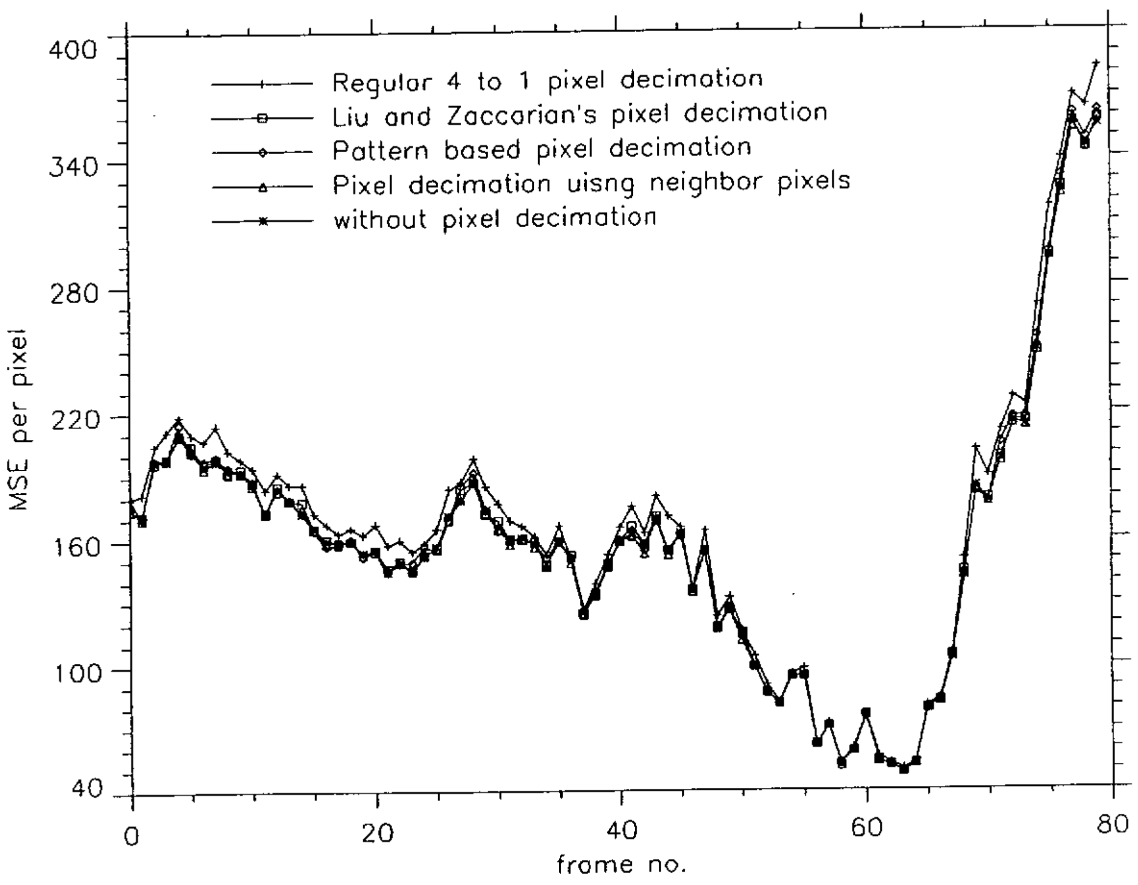


Figure 4-4 : MSE produced by different pixel decimation algorithms for the "Football" sequence

4.5 Summary

In this chapter, a new pixel decimation block matching algorithm is proposed to compensate the drawback in regular pixel decimation techniques. We define some "most representative pixel patterns" and make the selection according to the image content in each block for the matching criterion. The pattern matching requires very low computational complexity by the use of discrete gradient measure. Our approach can efficiently compensate the drawback in uniform pixel decimation. Computer simulations show that this technique is close to the performance of the full search, and is about 1.6 times faster than that of the famous approach given by Liu and Zaccarin [33]. Also, it is more convenient for hardware realization as compared with the fully adaptive pixel decimation.

Chapter 5

Multiprocessor Realization of Adaptive Motion Estimation Algorithm

5.1 Introduction

It is no doubt that many fast algorithms [33-40] for the Block Motion Vector Estimation have successfully reduced the computational complexity to an acceptable level as compared to the Full Search Algorithm. Some VLSI implementations of these fast algorithms have also been reported in [53- 61]. However, the limited searching location or the reduction on the number of pixels involved in the matching criterion has posed the common problem that the searching process would be trapped into a local minimum easily. Adaptive algorithms [47-49] have been proposed to solve the aforementioned problem by adjusting the position and the number of selected pixels to be used in the matching criterion according to the features of the image block. Since block based motion model is assumed and the matching criterion is computed using the pixel intensity difference between two image blocks. Objects are segmented and resided in different blocks, the inconsistent prediction of these blocks will create serious problems.

Overlapped window approach [52] could be used to overcome the problem of blocking artifacts in the motion compensated frame. However, it is not compatible with the current video coding standards [14-19] which employ non-overlapped blocks with a fixed size. The edge-oriented motion estimation algorithm [50] using a fixed block size can obtain more accurate motion prediction along moving edges and produce better visual quality as compared with the FSA.

In this chapter, a multiprocessor structure is proposed for the realization of a new adaptive algorithm [50] for Block Motion Vector Estimation in motion picture compression. It is found that edge information is extremely important for enhancing the picture quality, which could even outperform the Full Search Algorithm. Since the algorithm is adaptive, it appears that the realization of the approach could be complicated. However, this problem can be resolved by making use of a multiprocessor system with multiple-bus structure. With multiple buses, the bottleneck of data transfer between the Processing Elements and memory modules has been reduced. With a predetermined execution profile and the decomposition on the algorithm into several distinct tasks, the idling time of each processor is reduced. For the realization using ASIC design, a speedup of 3.5 is achievable by making use of four Processing Elements with overheads of less than 15% hardware complexity for the switching network.

Some preliminary results in this chapter have been reported in [63].

5.2 Edge Oriented Adaptive Motion Estimation Algorithm

Let us begin to review a basic problem of the motion vector estimation. It is commonly known that the block size of a block matching algorithm gives a significant effect to its realization. If the block size is too small, there could be too many blocks that match to the block under question while if the block size is too large, it is more likely to have blocks with a mixture of stationary background and moving objects. The latter problem is significant in many situations, including cases using commonly used block sizes, such as 8×8 and 16×16. This becomes worse if the stationary background occupies a larger share of the block while the moving object only shares a smaller part of the block. It is often that the moving object is the major target of the scene to be observed by the observer. In this case, the motion vector estimated will be the motion vector of the background instead of the object, which gives rise to annoying artifacts of having some disconnected objects. We resolve this problem by extracting the edge patterns of the current and the previous frames. The major idea is that the edge can help us to track the motion object such that we can use the information to form the motion vector which could reflect the motion more accurately. Four types of blocks are classified which are based on the characteristics of the blocks. The procedures to identify the type of each block of the algorithm are summarized as shown below.

1. To form edge frames

Since the detection of boundary edges of the image sequence is an important factor, a simple gradient mask as shown in equation (5-1), the Sobel Operator is used to obtain the edge frame, with the gradients equal to $S_n(i,j)$. Before finding the approximated directional derivatives [11], the image must be smoothed so that the ripples, spikes and high frequency noises from the image may be removed.

$$g_x = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \qquad g_y = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{5-1}$$

$$S_n(i,j) = \left| I'_n(i,j) * g_x \right| + \left| I'_n(i,j) * g_y \right| \tag{5-2}$$

2. To detect the level of motion inside blocks

This step is to classify blocks of the current frame into three types according to the contents of the block using the following Frame Difference equation,

$$FD_n = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left| I_n(i,j) - I_{n-1}(i,j) \right| \tag{5-3}$$

where n is the frame number, and $I_n(i,j)$ is the intensity at location (i,j) of the n^{th} frame. This is an initial check to see possible motion in the frames. Should the value of FD_n is lower than a predefined threshold value, we assume it as a block without much motion and the normal MAD matching criterion will be used for locating the motion vector of the block. Should FD, is larger than a predefined value, it is a block containing some level of activities and the following edge matching criterion will be used.

3. To identify the block containing moving edges

Since the FD_n is larger than the predefined value, we have to check if it contains some major moving edges. This is achieved by converting the corresponding block of the edge frame, $S_n(i,j)$, to become a binary edge block (with 0 or 1 value only, and with an appropriate thresholding value).

$$B(i,j)=\begin{cases}1 & \text{if } S(i,j)>T, \\ 0 & \text{othwise}\end{cases} \tag{5-4}$$

$$B_{count}=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}B(i,j) \tag{5-5}$$

If the sum of this binary edge block B_{count} is larger than another predefined value, it is considered as a block with active moving edges. The Binary Edge Mean Absolute Difference, BEMAD, as shown in equation (5-6) is used as the matching criterion to locate the motion vector (instead of the normal MAD).

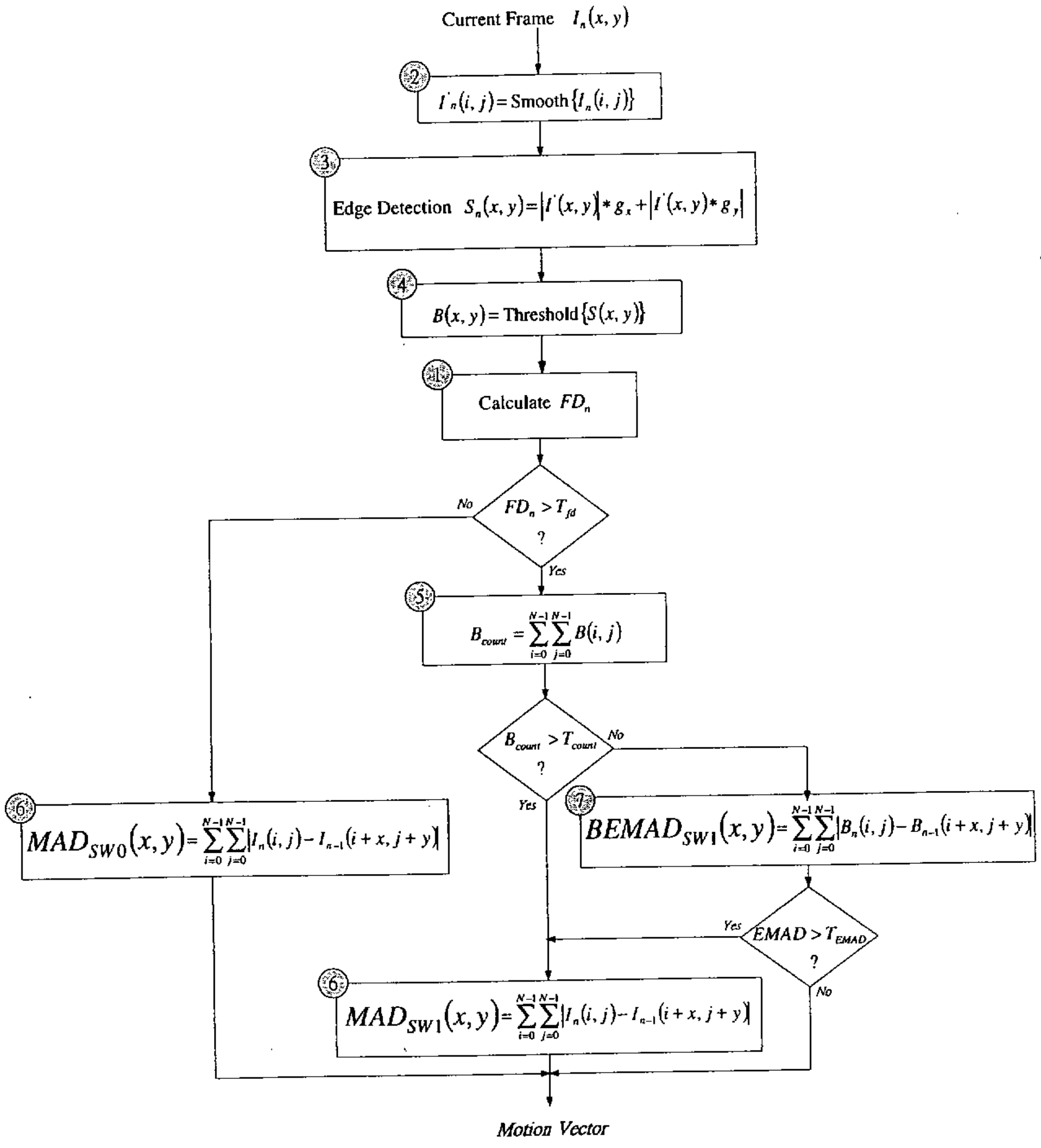
$$BEMAD(x,y)=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left|B_n(i,j)-B_{n-1}(i+x,j+y)\right| \tag{5-6}$$

4. Blocks on moving objects

If the sum of the elements in the binary edge block is smaller than the predefined value, it is considered as a block inside a moving object. In this case some neighboring motion vectors could be used to shorten the searching range for using the MAD for motion vector estimation.

5.2.1 Task Decomposition

With the procedures described above, the complete operation of the Edge-Oriented Adaptive Motion Estimation Algorithm can be illustrated by a flowchart as shown in Figure 5-1.



Current Frame $I_n(x,y)$
2
$I'_n(i,j)=\text{Smooth}\{I_n(i,j)\}$
3
Edge Detection $S_n(x,y)=|I'(x,y)*g_x|+|I'(x,y)*g_y|$
4
$B(x,y)=\text{Threshold}\{S(x,y)\}$
1
Calculate FD_n
$FD_n > T_{fd}$?
No
Yes
5
$B_{count}=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}B(i,j)$
$B_{count} > T_{count}$?
No
Yes
6
$MAD_{SW0}(x,y)=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|I_n(i,j)-I_{n-1}(i+x,j+y)|$
7
$BEMAD_{SW1}(x,y)=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|B_n(i,j)-B_{n-1}(i+x,j+y)|$
$EMAD > T_{EMAD}$?
Yes
No
6
$MAD_{SW1}(x,y)=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|I_n(i,j)-I_{n-1}(i+x,j+y)|$
Motion Vector

Figure 5-1 : Flowchart of the Edge-Oriented Adaptive Motion Estimation Algorithm

As indicated in the above flowchart, the algorithm consists of several distinct tasks and is decomposed as follows.

Task 1 : Evaluate the Frame Difference, $FD_n = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|I_n(i,j) - I_{n-1}(i,j)|$

Task 2 : Smooth the Current Frame , $I'_n(i,j) = \text{smooth}\{I_n(i,j)\}$

Task 3 : Perform the edge detection, $S_n(x,y) = |I'(x,y)| * g_x + |I'(x,y) * g_y|$

Task 4 : Generate the binary edge, $B(x,y) = \text{Threshold}\{S(x,y)\}$

Task 5 : Count the binary edges within the block, $B_{count} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} B(i,j)$

Task 6 : Compute the MAD, $MAD(x,y) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|I_n(i,j) - I_{n-1}(i+x, j+y)|$

Task 7 : Compute the BEMAD, $BEMAD(x,y) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|B_n(i,j) - B_{n-1}(i+x, j+y)|$

With these seven distinct tasks, we can implement the algorithm using multiple processors with a proper scheduling policy to perform parallel processing to speedup the motion estimation process. However, the data dependence between tasks should be carefully identified. The data dependence graph of the algorithm can be obtain from its operation flow. The flowchart illustrated in Figure 5-1 shows the sequence of different tasks of the algorithm.

5.3 ASIC Realization

Since the algorithm is adaptive, the realization of the approach using hardware could be complicated. We resolve this problem by making use of a multiprocessor system with multiple-bus structure. The major idea is that since the algorithm is adaptive, the realization times for different blocks would be different. A system with multiple buses connected to different memory modules could resolve the problem of idling time of processors. By decomposing the algorithm into several distinct tasks and with proper task scheduling, the idling time of each processor can be used to perform other outstanding task. This is because the processing time of each distinct task is fixed but with different combinations for each block. Thus, the processing time for each task can be pre-determined and the scheduling can be easily adopted. Furthermore, the present of our design can allow the searching of motion vectors to be done very quickly since multiple processors are used. The data dependence graph of the algorithm derived from the flowchart as shown in Figure 5-1 is illustrated in Figure 5-2 which forms the basis of the task scheduling policy.

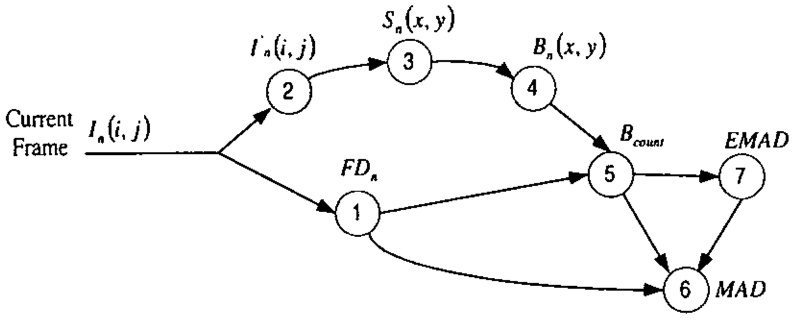


Figure 5-2 : Task dependence graph of the Edge Oriented Adaptive Motion Estimation Algorithm

There are two possible approaches for its parallelism that can be implemented using the multiprocessor structure: 1) frame level parallelism and 2) block level parallelism. In the former approach, coarse-level parallelism is explicitly achieved by using each processor to perform the motion estimation of one entire frame. The implementation of this approach is straightforward. Multiple and identical copies of modules consisting of processing elements and memory modules with simple connections are sufficient for the simple dataflow of the system. However, the processing time for each frame varies significantly and the synchronization between the input frame sequence and output sequence will introduce a large amount of idling time. Furthermore, the excess delay between the input frame and the output motion vector is not acceptable in many applications such as video conferencing. On the contrary, the second approach, fine-grain parallelism is obtained by distributing each block into an idling processor. This fine-grain approach is very attractive for the implementation since the idling time of processors can be minimized and the delay between the input frame and the output motion vector is reduced dramatically. However, it would be very complicated for the implementation since the data access pattern is very irregular. The balance point of the parallelism should be carefully selected such that the resulting architecture can be efficiently implemented with a reasonable amount of extra hardware.

5.3.1 The Multiprocessors System

The general architecture of the whole system is illustrated in Figure 5-3. As shown in the figure, the memory is divided into equal size memory blocks. Connections between the processors and the memory modules are achieved via a switching network. The switching network should ensure that each processor receives their required data in each clock cycle and the time delay between the input and output should be minimized. Two separate switching networks are needed in the system: one is used to connect up the data bus from the memory modules to the processor, while another is used to connect up the appropriate address bus of the processor to the memory module.

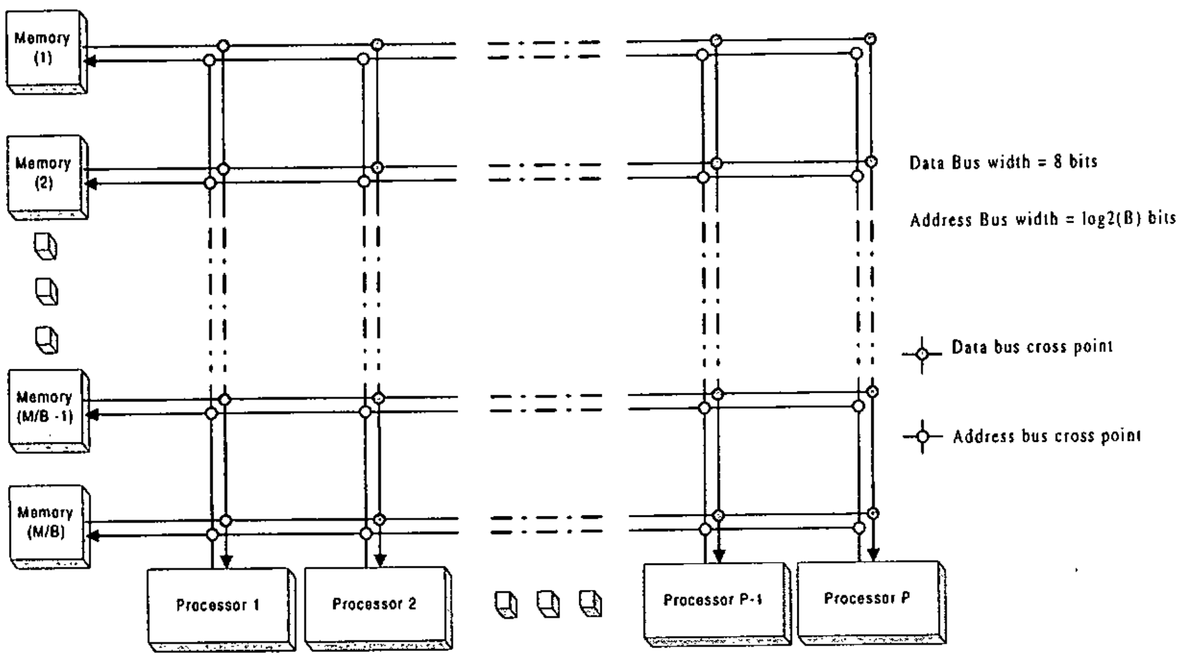


Figure 5-3 : Architecture of the multiprocessors system

5.3.2 Design And Implementation Of The Switching Network

The Performance of multiprocessor system always bounded by the switching network. Since we intent to implement block level fine-grain parallelism, low latency through the network is particularly important. The switching network is the most important part of the architecture, which affects the performance of the whole system. As stated in the early part, the switching network which is used to connect up the data from each memory block to the appropriate processor and to guarantee that each processor can receive correct data through the data bus in each clock cycle. Note that the subject network topology is a well-understood problem among researchers of high performance parallel computing using multiprocessors.

There are many techniques for the implementation of the switching network, including multi-stage networks and crossbar network. At the same level of complexity, multi-stage networks have longer latency than single-stage (crossbar) network. Moreover, in order to avoid the entire system to be stalled due to network conflicts, non-block multi-stage network is a commonly used technique to guarantee there is always connection between the memory and the processor. On the other hand, irregular structure of the network makes it difficult for its implementation.

From a system-level perspective, single-stage (crossbar) network is more preferable than the switching network implementation. However, the delay and hardware complexity of the crossbar network scale up exponentially with the number of ports. We, however, believed that the crossbar network can be implemented efficiently using customized ASIC design.

5.3.2.1 Multiplexer Based Crossbar Network

There are many implementation approaches [76, 77] for the crossbar network. Among the four possible implementations of the multiplexer cell as described in [77], a tree of OR gates has the optimal speed and area performance. The resulting structure of the crossbar network also exhibits regular structure which is very suitable for VLSI implementation. An example of a 4×4 crossbar network is illustrated in Figure 5-4.

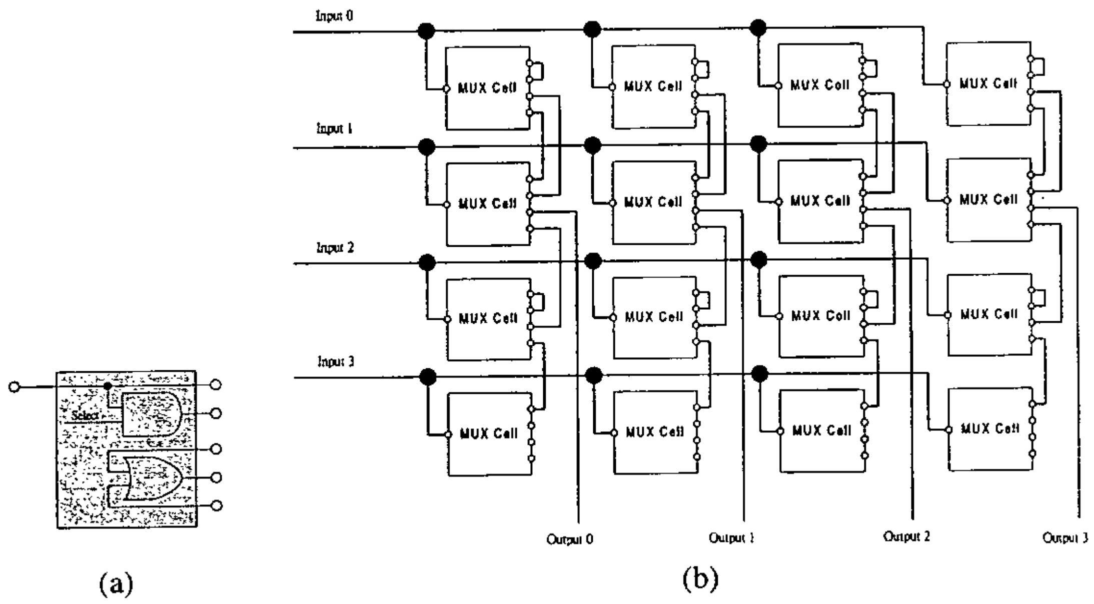


Figure 5-4 : The implementation of the crossbar network using multiplexer , (a) mutiplexer cell, (b) cell matrix of 4x4 crossbar

When we consider the overall performance, implementation cost and available technologies, the multiplexer is the most reasonable choice if the architecture is implemented using either Erasable Programmable Logic Device (EPLD) or Field Programmable Gate Array (FPGA).

The 2-to-1 multiplexers based design of the crossbar network features a $[\log_2(k)-1]$-stage multiplexer at each output to provide switching. In order to realize a n-bit network, the single-bit design as shown in Figure 5-4 is replicated $n-1$ times. The implementation of the crossbar network with P processors requires P-column of multiplexers and the implementation cost of each multiplexer should be estimated.

For the purpose of illustration, the implementation cost of a 4-to-1 multiplexer is used as an example and it is shown in Figure 5-5.

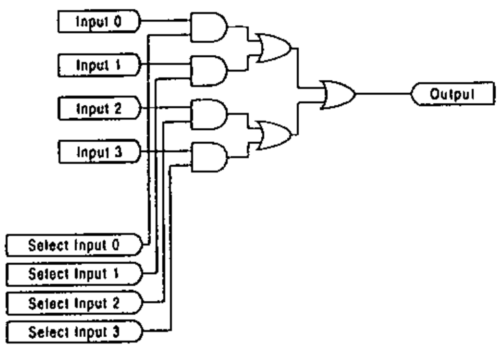


Figure 5-5 : 4-to-1 multiplexer

The 4-to-1 multiplexer here uses 7 logic gates and in general, the multiplexer is implemented as a tree structure, the cost of the multiplexer can be formulated as,

$$\text{Cost}_{\text{multiplexer}} = \sum_{i=0}^{\log_2(k)} \frac{k}{2^i} \qquad (5\text{-}7)$$

where k is the number of inputs to the multiplexer.

5.3.3 Implementation Cost Of The Switching Network

Since there are two separate switching networks which are required for the architecture, two types of multiplexers should be used. The cost functions of these two types of multiplexer are as shown in the following equations.

$$\text{Cost}_{\text{Data Bus multiplexer}} = \left(\sum_{i=0}^{\log_2(k)} \frac{k}{2^i} \right) \times P \times \text{Bits per word} \tag{5-8}$$

$$\text{Cost}_{\text{Address Bus multiplexer}} = \left(\sum_{i=0}^{\log_2(P)} \frac{P}{2^i} \right) \times k \times \log_2(B) \tag{5-9}$$

where ***P*** is the number of processors, ***B*** is the memory block size and ***k*** is the number of memory modules used. Hence the total cost of the switching network is the sum of the two costs.

$$\text{Cost}_{\text{Switching Network}} = \left(\sum_{i=0}^{\log_2(k)} \frac{k}{2^i} \right) \times P \times \text{Bits per word} + \left(\sum_{i=0}^{\log_2(P)} \frac{P}{2^i} \right) \times k \times \log_2(B) \tag{5-10}$$

Usually the absolute cost of the switching network does not reflect much idea on the amount of resources used for the whole system. Hence the ratio between the cost of the switching network and the cost of the whole system is used as a relative measure.

$$\text{Cost Ratio} = \frac{\text{Cost of Processors} + \text{Cost of Memory} + \text{Cost of Switching Network}}{\text{Cost of Processors} + \text{Cost of Memory}}$$

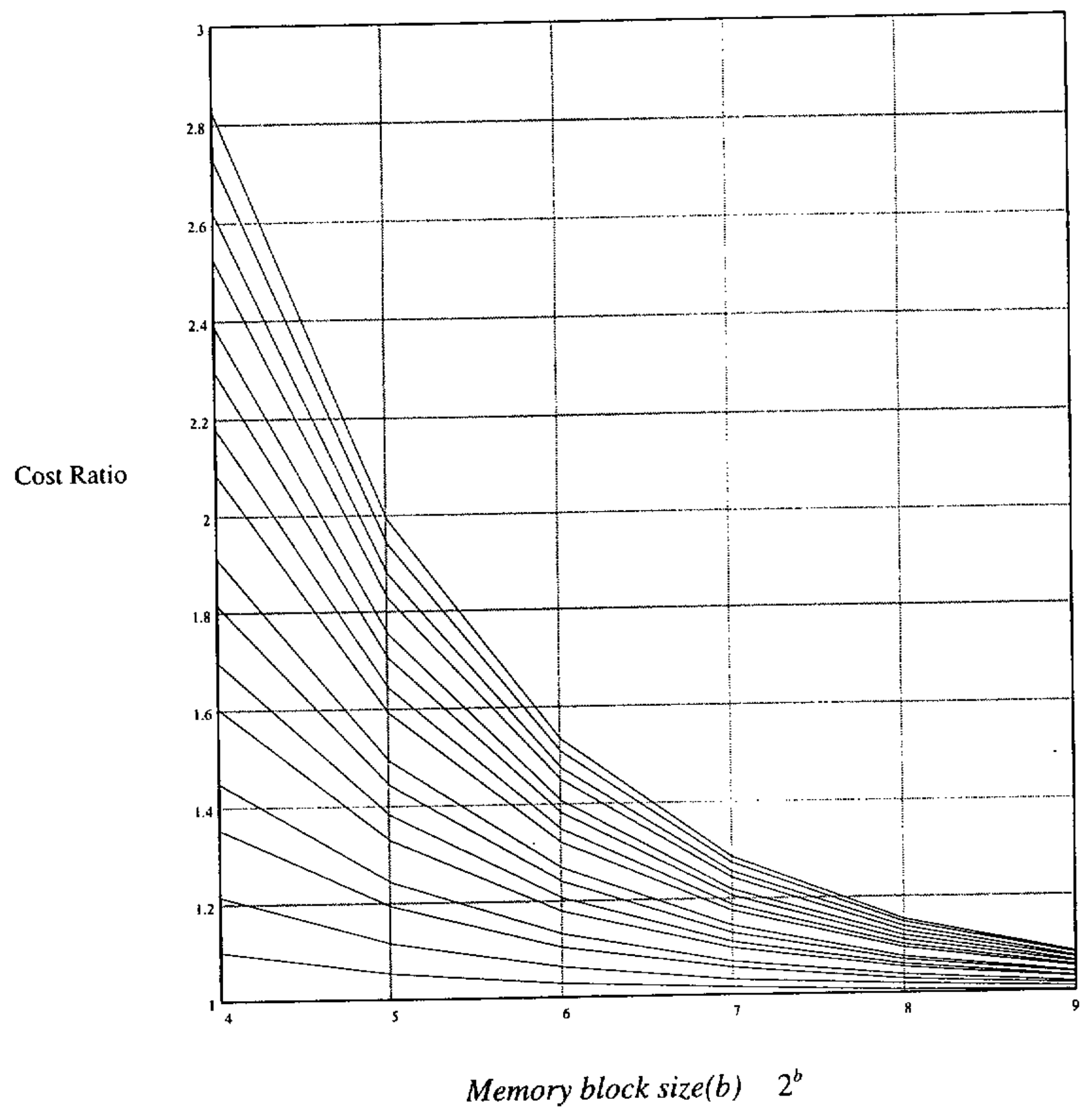


Figure 5-6 : Cost ratio of the switching network when using multiplexer

The cost ratio was computed by assuming that the memory size is 2^{16} or 65536 words with 8 bits per word. Cost ratio using 1 to 16 processors are shown using different curves with $P = 1$ for the lowest curve, $P = 16$ for the highest curve. The graph in Figure 5-6 shows that the cost ratio is exponentially decreasing as the number of memory block is reduced. For example, if the number of processors to be used is 4 and the size of each memory block is 8 x 8 words, then the cost ratio is equal to 1.12.

5.4 Results

With the algorithm and the architecture described above, the algorithm is realized via block-level parallelism using the multiprocessor architecture. The efficiency of the architecture depends on the reduction of the idling time of each processor. As illustrated in Figure 5-2, there are four possible execution paths in the algorithm. The execution time of each execution path is derived from the sequential execution of the algorithm. Since the execution time of each execution path is deterministic, straightforward scheduling strategy based on the derived execution time of each task can be used. The four execution paths are identified as,

Path a : *Task 1, Task 6*

Path b : *Task 1, Task 5, Task 6*

Path c : *Task 1, Task 5, Task 7*

Path d : *Task 1, Task 5, Task 7, Task 6*

The data dependence graph shown in Figure 5-2 indicated that the edge detection process is preceded the motion estimation processor. This dependence requirement introduces some idling time on the processor which is needed to wait for the completion of the edge detection process. The relative execution time of the four execution paths a, b, c and d is measured as shown in the follow table, and the data of which are used for the task scheduling decision.

Path	Relative execution time
t_a	0.032595
t_b	0.30335
t_c	0.7191
t_d	1

Table 5-1 : Relative execution time for different paths

The performance of the multiprocessor system is simulated using different number of processors and the speed-up for the realization of the edge oriented adaptive motion estimation algorithm for the sequence "table tennis" is shown in Figure 5-7.

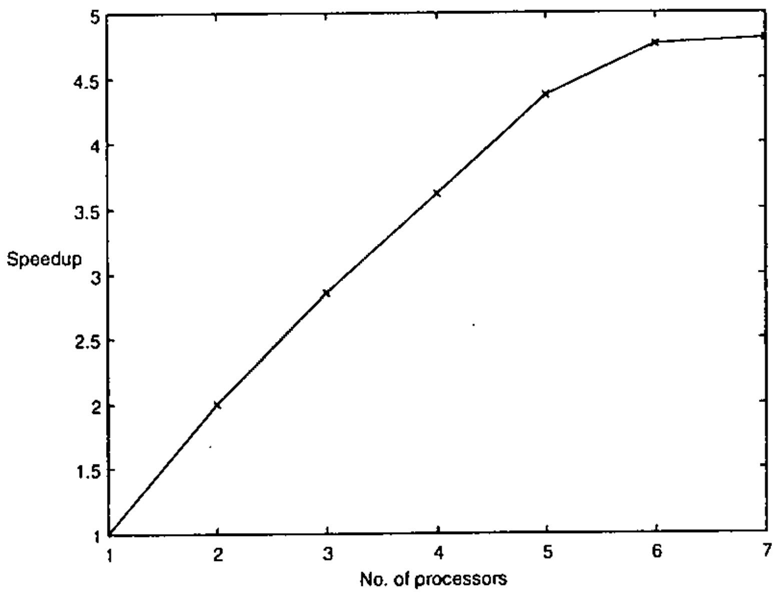


Figure 5-7 : Speed-up for the "table tennis" sequence

5.5 Summary

It is seen that we have presented an algorithm using edge mean absolute difference for motion estimation and its realization using an efficient multiprocessor structure using multiple-bus. The algorithm is more efficient as compared to other algorithms, including the Full Search Algorithm, in the literature, while the realization using the multiple-bus system requires little overheads. With task decomposition and block-level parallelism, the idling time of each processor is reduced and the speedup of the realization approaches 3.5 when four processors are used, with less than 15% of hardware overhead for the switching network.

Chapter 6

Edge Masked Motion Estimation

6.1 Introduction

As we have discussed in the previous chapters, Block Matching Algorithms (BMA) [37] divide the image frame into regular blocks and do not consider any rotation and scaling of the objects to simplify the motion estimation process. Although this approach may produce poor results for some scenes that contain a large amount of zoom, object translation and camera panning, the simplicity and coding efficiency for motion information are the primary consideration and adopted by a number of video coding standards [15-19]. Many fast search algorithms [33, 35-42] have been developed to reduce the computational complexity by either limiting the search location or reducing the pixel involved in the matching criterion. The advances in VLSI technology also facilitate a cost effective hardware realization of the BMA even to implement the full search which requires the highest hardware complexity.

The objective measure of the matching determines the accuracy of the estimated motion vector. The Mean Absolute Difference (MAD) [43] criterion has been considered as a good candidate for low bit rate video applications for it requires simple operations. More accurate criteria such as the cross-correlation and the Mean Squared Error (MSE) [51] are too complex for hardware realization. By the use of the above mentioned objective measures, the error between the current block and the reference block is evaluated in a point by point manner. However, the human visual system (HVS) perceives an image as a whole rather than individual pixel. The most annoying problem is that blocks located on boundaries of moving objects are not estimated accurately. The discontinuity of the edges of the object may be considered as the most objectionable distortion by human observers. Thus the conventional matching criteria cannot produce the best subjective result that is closely related to the HVS even using the full search algorithm. Some adaptive algorithms [47, 49] had tried to reduce the drawback of reducing the number of pixels in the matching criterion by considering the block activities to select the "most representative pixels". The disadvantage of using intensity matching criterion which was pointed out in [50] is that even for the global optimum full search approach, it may lead to the "missing edge effect", and which making use of edge features in consecutive frames could result in better motion-compensated prediction frames.

In this chapter, an edge-masked matching criterion has been found to be most suitable for low bit-rate video applications. The proposed matching criterion makes use of edge features to modify the conventional matching criterion for computing the motion information. Experimental results show that the proposed matching criterion performs as well as the more complex criteria and is able to remove the most visually disturbing artifacts with a slight increase in hardware complexity. In the following sections of this chapter, we will establish some preliminaries related to the study and define the problem. Details of the new matching criterion will be discussed and a custom architecture will be given.

6.2 Conventional Matching Criterion for Block Motion Estimation

Let us start to recall that the motion model for block matching algorithms (BMA) usually assumes that an image is composed of rigid objects in translational motion, while the motion field over the blocks of pixels moves as a group. The present frame is divided into two-dimensional small blocks of $N \times N$ pixels. For each block in the current frame, we evaluate a certain matching criterion over nearby blocks in the reference frame and select the block which yields the closest matching. This closest matching block is then used as a predictor for the present block. The relative position of these two blocks defines a motion vector associated with the present block.

Let us assume $I_t(m,n)$ to be the intensity of the pixel at location, (m,n) (upper most left in the block) of the current frame, t, and $I_{t-1}(m+k,n+l)$ to be the intensity of the pixel on the reference frame, $t-1$, displaced by the k pixels and l lines within the search window. The displacement (k,l) which has the closest matching with the current block is selected as the motion vector. The accuracy of the estimate depends on the matching criterion applied in the search. Let us recall some of these matching criteria.

Mean Squared Error (MSE) [51]

$$MSE(k,l)=\frac{1}{N^2}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}[I_t(m,n)-I_{t-1}(m+k,n+l)]^2 \qquad (6\text{-}1)$$

In this measure, the smallest $MSE(k,l)$ within the search window, $p \le k,l \le -p$, represents the best match. That is, the estimation of the motion vector (u,v) is taken to be the value of (k,l) which minimizes the MSE.

$$(u,v)=\arg\min_{(k,l)} MSE(k,l) \qquad (6\text{-}2)$$

Mean Absolute Difference (MAD) [43]

However, the MSE criterion is not commonly used in VLSI implementations because it is difficult to realize the square operation in hardware. Instead, the MAD criterion, defined as

$$MAD(k,l)=\frac{1}{N^2}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\left|I_t(m,n)-I_{t-1}(m+k,n+l)\right| \qquad (6\text{-}3)$$

is the most popular choice for VLSI implementations because it does not require any multiplication and produces similar performance as the MSE criterion. Then the motion vector (u,v) is given by

$$(u,v) = \arg \min_{(k,l)} MAD(k,l) \tag{6-4}$$

<u>*Matching Pixel Count (MPC)*</u> [38]

Another alternative is the Matching Pixel Count (MPC) criterion. In this approach, each pixel within the block is classified as either a matching pixel or a mismatching pixel according to

$$T(m,n;k,l) = \begin{cases} 1 & if \mid I_t(m,n,t) - I_{t-1}(m+k,n+l) \mid \leq a \\ 0 & otherwise \end{cases} \tag{6-5}$$

where a is a predetermined threshold. Then the number of matching pixels within the block is given by

$$MPC(k,l) = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} T(m,n;k,l) \tag{6-6}$$

and

$$(u,v) = \arg \max_{(k,l)} MPC(k,l) \tag{6-7}$$

That is, the motion vector (u,v) is the value of (k,l) which gives the highest number of matching pixels. The MPC criterion requires a threshold comparator, and a $\log_2(N^2)$ counter.

Among the above matching criteria, the MSE requires multiplications, thus it is costly to implement in real time processing. The MAD is the most widely used in matching criterion for the BMA due to its lower complexity. Although the MPC requires less hardware than MAD does, it is less appropriate to use in existing coding due to the degradation of the performance.

6.3 Edge-Masked Matching Criterion

Although the above criteria have a good physical and theoretical basis, they correlate poorly with the subjective measure of two successive blocks. Much of the reason for this is due to the fact that the human visual system does not process the image in a point-by-point fashion, but extracts certain spatial features. Thus, the conventional criteria cannot reflect the image characteristics such as edge fidelity, image contrast, etc. Consequently, certain spatial features are also very important to predict the compensated frame. Our proposed edge-masked matching criterion tries to obtain more accurate motion prediction along moving edges to which human visual system is very sensitive.

In our proposed scheme, an edge mask is used to enhance the conventional MAD matching criterion such that the motion-compensated prediction frames are tied more closely to the physical features. An edge detector is used to generate the edge mask. For choosing an edge detection algorithm, we consider its speed and precision. Before finding the directional derivatives, we have to smooth the image. For the smoothing, ripples, spikes or high frequency noises from the image could possibly be removed. A simple mean smoothing filter that performs equally weighted smoothing using a square window with the size of 5 has been employed in our work. Then the edge detection algorithm is applied to the smoothed current frame. For simplicity, the 3×3 Sobel gradient convolution masks, as shown in equation (6-8), have been used.

$$g_x = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad g_y = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{6-8}$$

Then edge frame S_t is obtained from equation (6-9).

$$S_t(m,n) = | I'_t(m,n) * g_x | + | I'_t(m,n) * g_y | \tag{6-9}$$

where I'_t is the smoothed image with the dimension same as the original image. Let us define the edge mask, B_t, as shown in equation (6-10)

$$B_t(m,n) = \begin{cases} \beta & if\ S_t(m,n) > T \\ 1 & otherwise \end{cases} \tag{6-10}$$

where T is a predefined threshold and β is the edge enhancement factor.

Then the Edge-Masked Mean Absolute Difference, $EMMAD(k,l)$, is defined as follows,

$$EMMAD(k,l) = \frac{1}{N^2}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\left|I_t(m,n) - I_{t-1}(m+k,n+l)\right| \times B_t(m,n) \tag{6-11}$$

From equation (6-11), we should observed that an increase in β will enhance the influence of edge features in the matching criterion. The selection criterion of β also depends on the architecture for its realization as described below.

6.4 The Architecture

The Edge-Masked Motion Estimator consists of two stages in which the function of the mask generating stage is to form the edge mask, B_t. It includes a 5×5 smooth filter, a 3x3 Sobel Operator and a comparator. The second stage estimates the motion vector of the current block from the reference searching window with the matching criterion enhanced by the edge mask. The proposed general architecture of the edge-masked motion estimator is shown in Figure 6-1, which obviously consists of the first and the second stages.

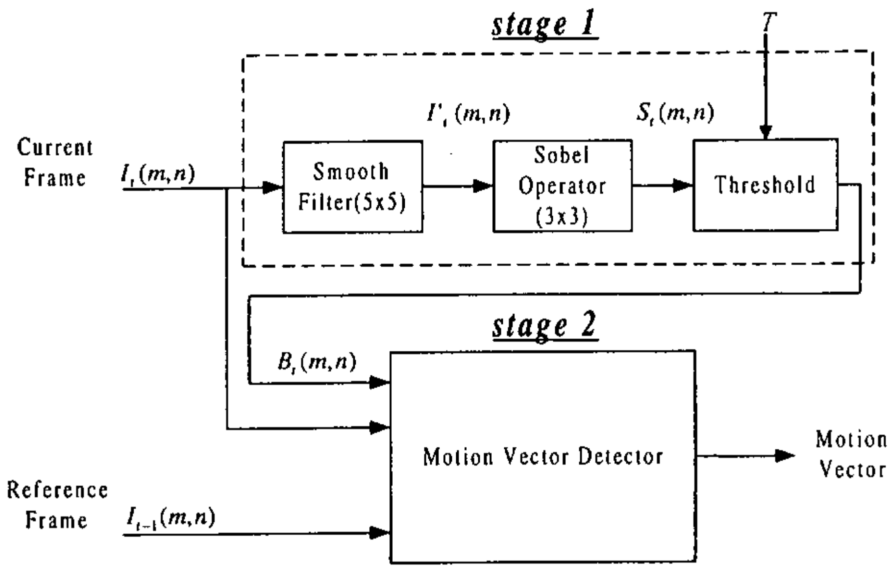


Figure 6-1 : Architecture of the Edge-Masked Motion Estimator

6.4.1 Smooth Filter

The proposed structures of the mask generating stage includes the smooth filter and the Sobel edge detector. The 5×5 smooth filter performs the smoothing on the input image block through a convolution operation and the smoothed image can be expressed as,

$$I_t'(m,n)=I_t(m,n)*s \tag{6-12}$$

where $$s=\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

From equation (6-12), the convolution can be implemented by a series of addition, and the partial result of each row can be reused. Figure 6-2 shows the architecture of the 5×5 smooth filter.

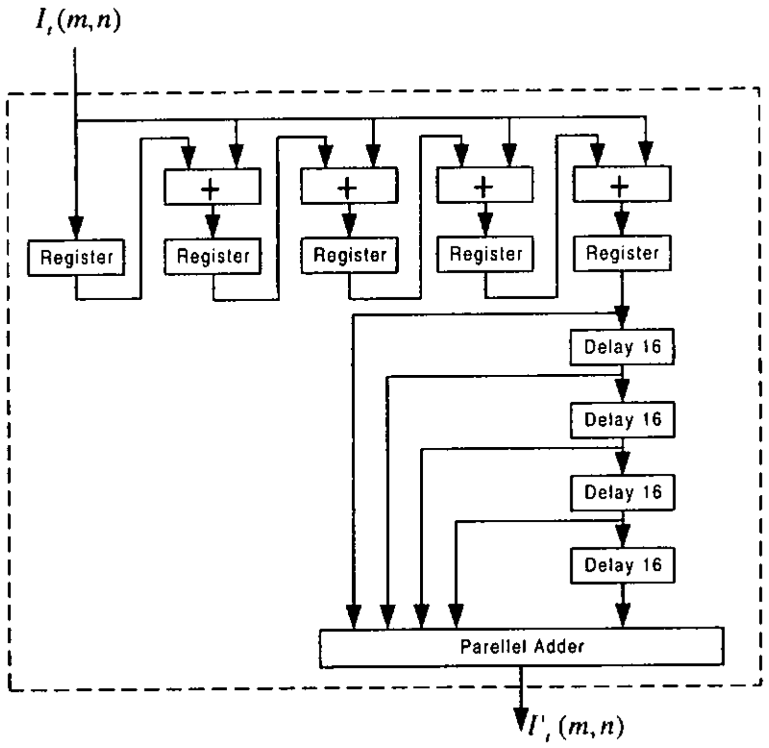


Figure 6-2 : 5x5 Smooth Filter

6.4.2 Edge Detector

The Sobel Operator performs similar operations as the smooth filter. With the derivation as shown below, the architecture for the Sobel Operator is obtained and is greatly simplified.

Equation (6-9) can be rewritten as,

$$S_t(m,n) = |h_x(m,n)| + |h_y(m,n)| \tag{6-13}$$

where $h_x(m,n) = I_t^{'}(m,n) * g_x$ and,

$$h_y(m,n) = I_t^{'}(m,n) * g_y$$

Let $I_t^{'}(m,n)$ be a 16×16 image block. The image pixel enters into the system sequentially from left to right and top to bottom, the 2-D index $I_t^{'}(m,n)$ can be mapped onto a

1-D time index, i.e. $I_t^{'}(k)$. Then the computation of h_x and h_y can be expressed as shown in the following equations.

$$h_x(k) = I'(k-17) + 2I'(k-16) + I'(k-15) - [I'(k+15) + 2I'(k+16) + I'(k+17)] \tag{6-14}$$

and,

$$h_y(k) = I'(k-17) - I'(k-15) + 2I'(k-1) - 2I'(k+1) + I'(k+15) - I'(k+17) \tag{6-15}$$

Apply the Z-transform to equations (6-14) and (6-15), we have

$$H_x(z) = z^{-17}I'(z) + 2z^{-16}I'(z) + z^{-15}I'(z) - \left[z^{15}I'(z) + 2z^{16}I'(z) + z^{17}I'(z)\right]$$

$$z^{-17}H_x(z) = z^{-32}\left[z^{-2}I'(z) + 2z^{-1}I'(z) + I'(z)\right] - \left[z^{-2}I'(z) + 2z^{-1}I'(z) + I'(z)\right] \tag{6-16}$$

and,

$$H_y(z) = z^{-17}I'(z) - z^{-15}I'(z) + 2z^{-11}I'(z) - 2zI'(z) + z^{15}I'(z) - z^{17}I'(z)$$

$$z^{-17}H_y(z) = z^{-32}\left[z^{-2}I'(z) - I'(z)\right] + 2z^{-16}\left[z^{-2}I'(z) - I'(z)\right] + \left[z^{-2}I'(z) - I'(z)\right] \tag{6-17}$$

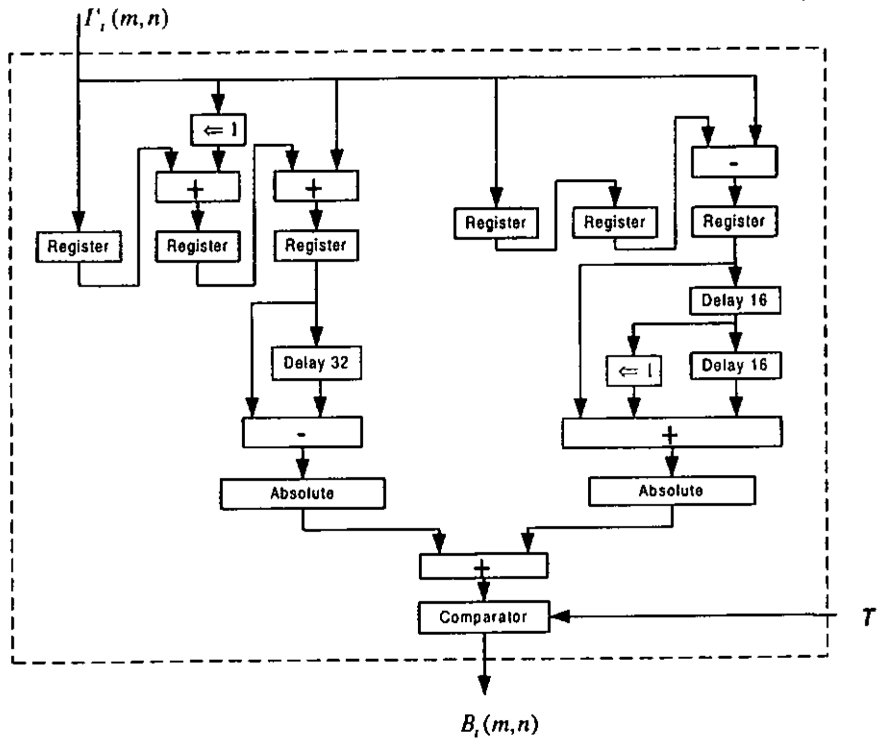


Figure 6-3 : 3x3 Sobel Operator

Equations (6-16) and (6-17) can be directly mapped onto hardware realization where z^{-1}, of course, stands for a delay of 1 clock cycle. The implementation of the 3×3 Sobel Operator is shown Figure 6-3.

6.4.3 Motion Vector Detector

In this proposed algorithm, the introduction of the edge enhancement factor improves the visual quality significantly at the expenses of just a minor modification of the MAD computation. Thus, the implementation of the motion estimator can be achieved from the conventional architecture with a slight modification. In our work, we modify the broadcasting architecture proposed in [54], and only the modification is shown here. The complete data flow and control timing of the architecture are the same as [54].

As shown in Figure 6-4, the edge mask B_t generated from the Sobel edge detector enters into the system in synchronization with the bit-stream of the current frame to each PE. In each PE, as shown in Figure 6-5, the absolute difference is multiplied by the edge enhancement factor β according to the edge mask and is accumulated to form the EMMAD. In order to reduce the complexity to implement the multiplication of the absolute difference and β, we choose the values of β to be multiples of 2, as depicted in the following equation,

$$\beta = 2^n \qquad (6\text{-}18)$$

where n is any positive integer. A simple Barrel Shifter is used in our design where the value of the absolute difference has either to shift left by n bits or just pass through to the accumulator. The shaded areas in Figure 6-4 and Figure 6-5 indicate the modified portion of the original architecture.

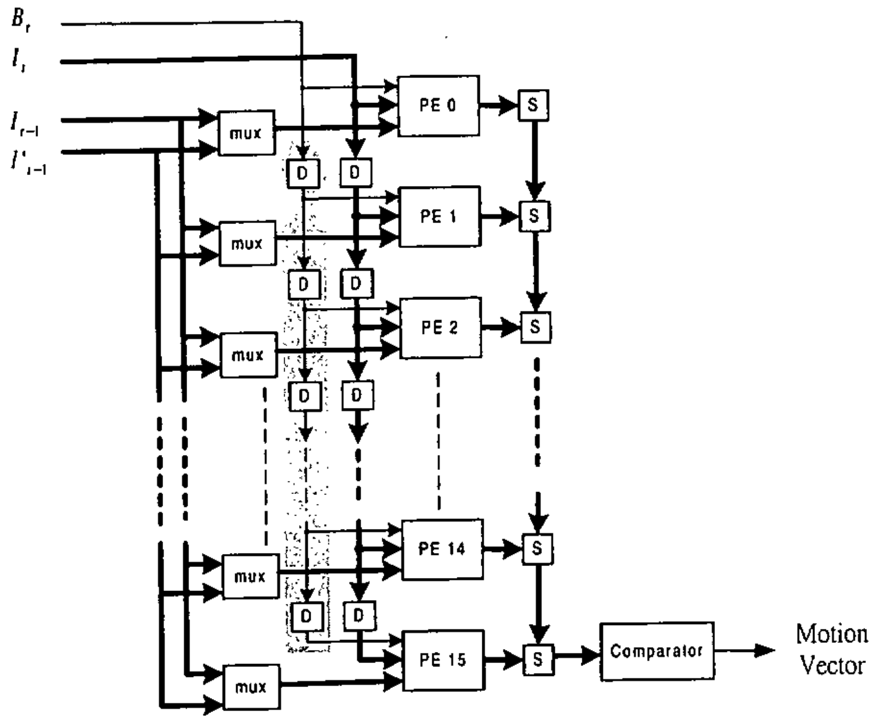


Figure 6-4 : Block diagram of the modified Motion Estimator

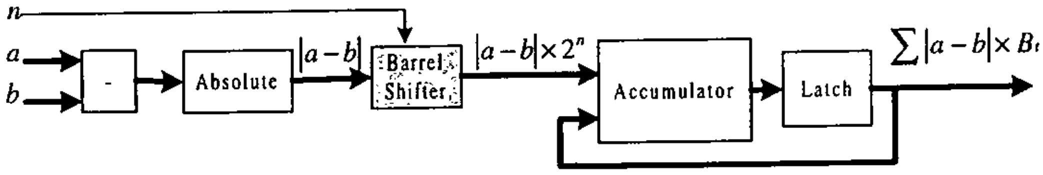


Figure 6-5 : PE with edge enhancement factor

6.5 Simulation Results

In our simulations, motion vectors using full-search were estimated according to the MSE, the MAD, the MPC and the proposed EMMAD. We have tested the performance of the proposed algorithms for a large variety of real image sequences, including the "Table Tennis", the "Football" and the "Mobile & Calendar". We have examined the performances of these algorithms in terms of their Mean Square Errors as well as the visual quality of the motion-compensated prediction frames. The Mean Square Error at frame t (MSE_t), defined in terms of the motion-compensated prediction frame $\hat{I}_t(x,y)$ and the original frame $I_t(x,y)$, is given as shown below:

$$MSE_t = \frac{\sum_{x=0}^{P-1}\sum_{y=0}^{L-1}\left[\hat{I}_t(x,y) - I_t(x,y)\right]}{P \times L} \tag{6-19}$$

where P is the number of pixels per line and L is the number of lines in each frame.

The maximum displacement p was set to *15* and a block size of 16×16 was considered. Parameter T in equations (6-10) for the mask generation of the proposed EMMAD was set to *40*, and the enhancement factor β was set to 4 (i.e. $n = 2$ in equation (6-18)).

The MSE results are shown in Figure 6-6, Figure 6-7 and Figure 6-8 for the "Table Tennis", the "Football" and the "Mobile & Calendar" respectively. The MSE performance for the MSE, the MAD and the EMMAD appear to be very close. These results also show that the MPC is inferior to other matching criteria though it requires less hardware. Typically, the MSE is not a good metric for subjective measurement since the MSE relates little to the response of the human visual system. In fact, due to the sophisticated psychovisual behaviour of human beings, there is a lack of benchmark to objectively measure the subjective image quality of any image. In comparing the visual behaviour of different matching criteria, human evaluation could be the best judgement. In this chapter, the proposed EMMAD tries to obtain more accurate motion prediction along moving edges to which human visual system is very sensitive. Figure 6-9 shows the formation of the motion-compensated prediction frames of the "Table Tennis" sequence using the MSE, the MAD, the MPC and the EMMAD. Figure 6-9(b), Figure 6-9(c) and Figure 6-9(d) show the incorrect prediction of the ball and the bat along the edge using the traditional MSE, MAD and MPC, while the edge of the ball and the bat can be preserved by using the proposed EMMAD as shown in Figure 6-9(e). This visual inspection indicates that the EMMAD is even better than the MSE in terms of subjective view, though the MSE obtains the most optimal result in terms of objective measurement. The major reason is that the EMMAD can remove most of the visually disturbing artifacts of motion estimation and compensation, hence frames produced by this approach are virtually error free. In low bit rate applications, for which there is insufficient bandwidth to reconstruct the prediction error adequately, the artifact produced by the motion estimation can remain in the final decoded frame of the traditional approach. However, our proposed EMMAD can definitely achieve a good subjective quality.

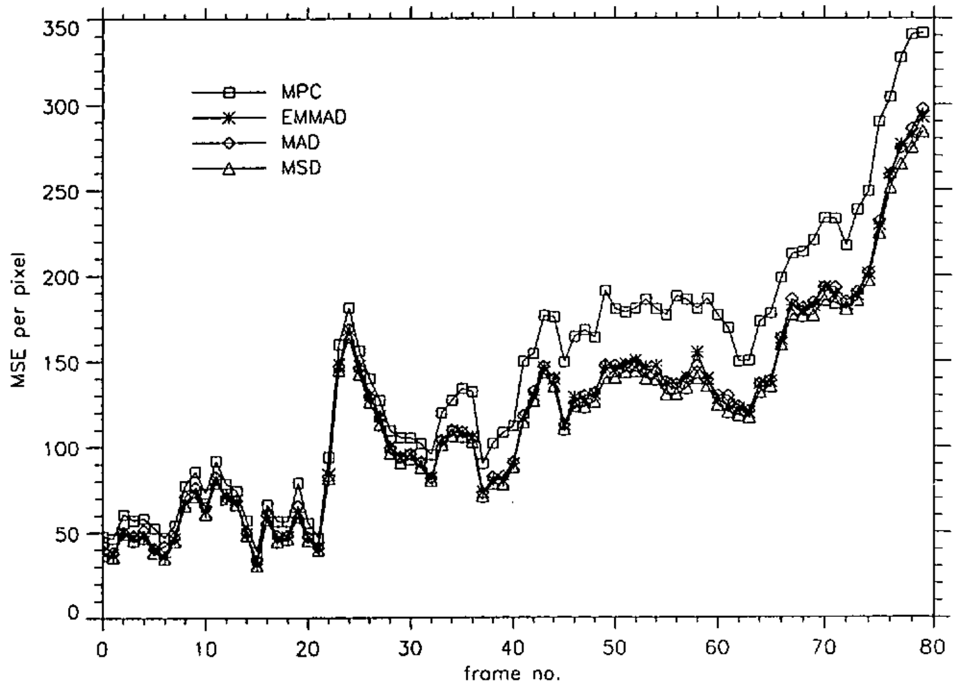


Figure 6-6 : MSE produced by motion estimation and compensation using different matching criteria for the "Table Tennis" sequence

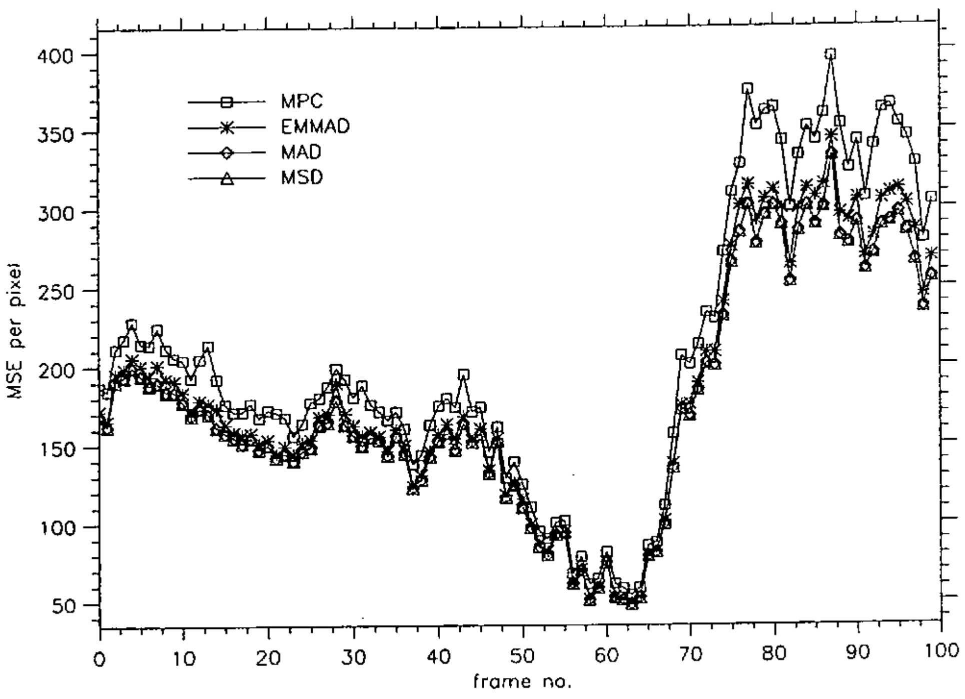


Figure 6-7 : MSE produced by motion estimation and compensation using different matching criteria for the "Football" sequence

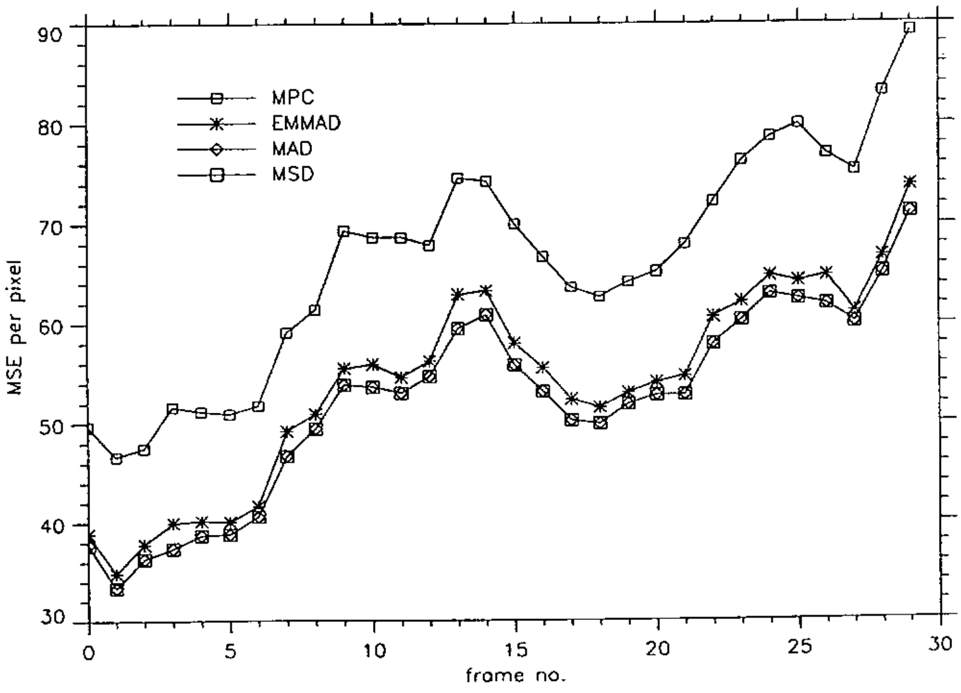
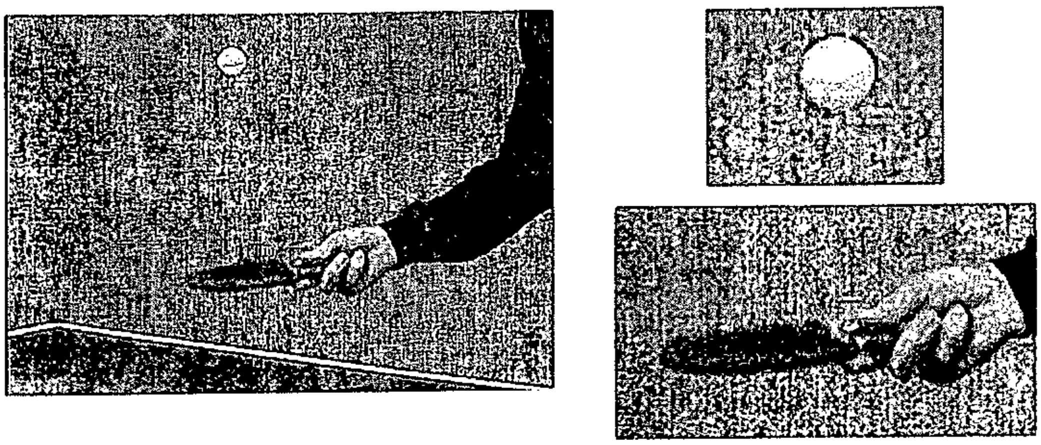
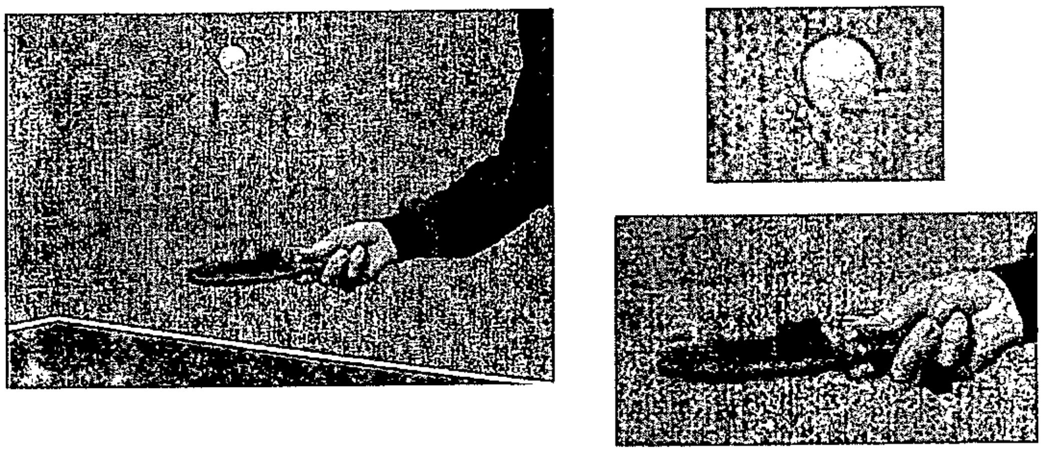
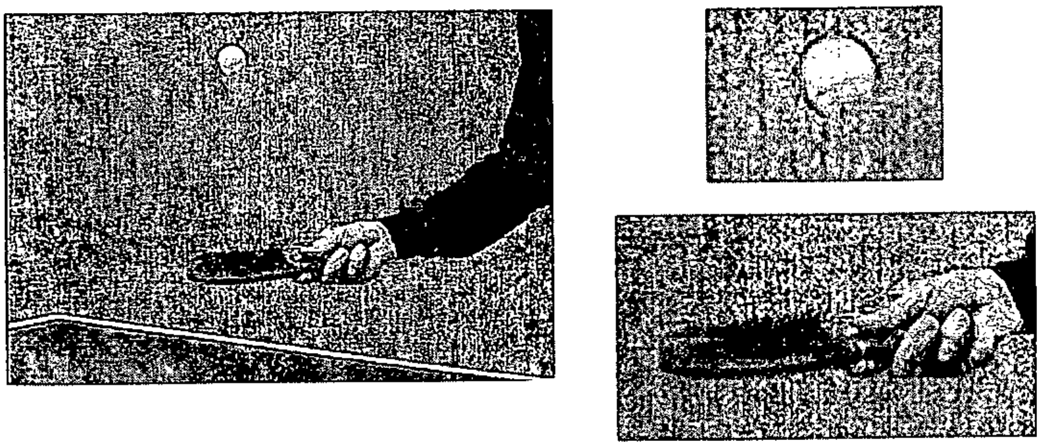
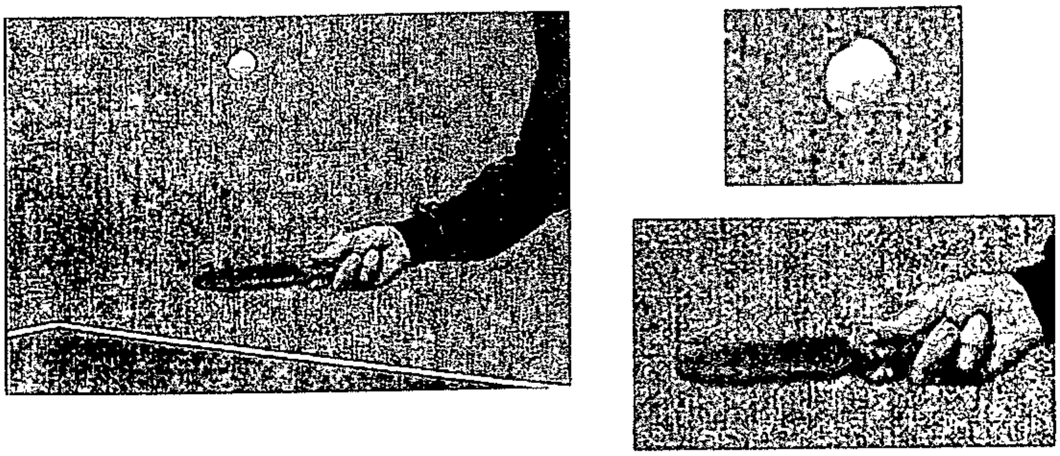


Figure 6-8 : MSE produced by motion estimation and compensation using different matching criteria for the "Mobile & Calendar" sequence

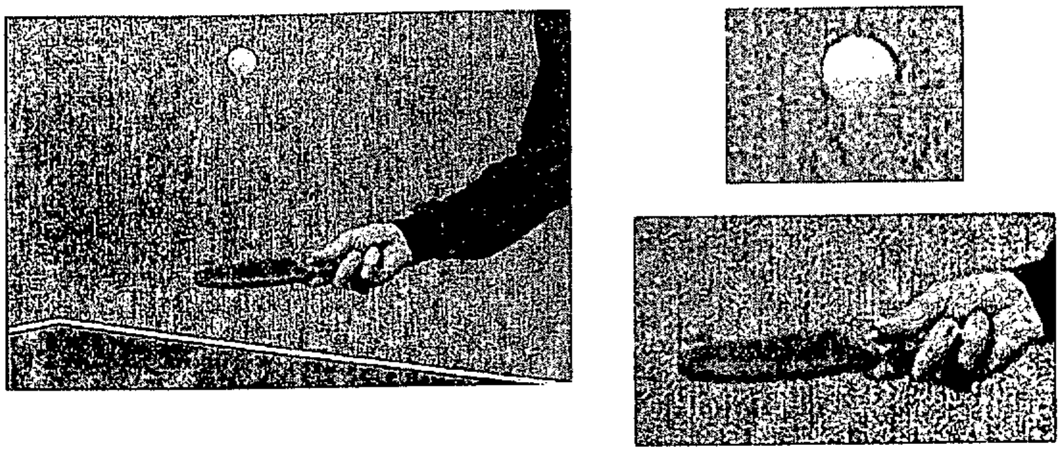

(a) Original


(b) Motion-compensated prediction frame using MSE


(c) Motion-compensated prediction frame using MAD



(d) Motion-compensated prediction frame using MPC



(e) Motion-compensated prediction frame using EMMAD

Figure 6-9 : Motion-compensated prediction frame in "Table Tennis"

6.6 Summary

We have presented a new matching criterion for motion estimation based on the edge features and given the architecture for its hardware implementation. The edge-masked matching criterion (EMMAD) is proposed to compute the motion vectors such that better motion-compensated prediction along moving edges is obtained. The computed motion vectors are more reliable, especially for edge portions of the moving objects. Experimental results show that our new EMMAD has comparable mean square error performance as compared to the traditional matching criterion with only a slight increase in hardware complexity, while the poor prediction along the moving edges, which is very annoying around moving objects, is substantially reduced. In addition, since edges are more closely tied to physical features in a scene as compared to individual pixel intensities, accurate moving edges are likely to be useful in other processing parts of a video compression system.

Chapter 7

Conclusions and Future Directions

7.1 Conclusions

The success of video coding relies heavily on an efficient implementation of the video coding scheme. In the domain of video coding, the spatial and temporal redundancies can be efficiently reduced by using transform coding and block based motion estimation techniques. The aim of this thesis is to develop efficient hardware architectures and algorithms which can provide efficient implementation of various video coding schemes.

In Chapter 3, a low complexity and high-speed processor structure for the Discrete Cosine Transform is described. The key to reduce the hardware complexity of the structure is by the use of bit-serial approach. With careful optimization on the DCT kernel multiplications, more than 75 percent of hardware resources has been saved. The realization of the DCT processor using EPLD achieves a high operating frequency of 45MHz and used only 4500 gates.

An adaptive pattern based pixel decimation algorithm as described in Chapter 4 which makes use of the information contained in the image block to select a predefined set of pixel pattern for the block matching has been introduced. In conventional pixel decimation technique, a fixed set of pixel is usually used in the matching criterion computation, some details of the image block will be neglected and hence the accuracy of the predicted motion vector is degraded. In our approach, a predefined set of pixel patterns is designed to represent most of the properties of the sub-block. The "most representative pixels" are selected based on the classification of each sub-block and the results show that the performance of the algorithm is comparable to the full search and is 1.6 times faster than the regular pixel decimation algorithm.

Adaptive algorithm has many advantages over fixed strategy algorithms. However, the variation on the execution time is difficult for its realization. In Chapter 5, we have mapped the Edge Oriented Adaptive Motion Algorithm into a multiprocessor system. With the technique of task decomposition, the execution profile of each possible execution pattern can be determined. With a simple scheduling strategy using the predetermined execution profile, the idling time of each processor is reduced. The multiprocessor system when it is using four processors, up to 3.5 times of speedup is achieved and only 15% of extra hardware complexity is needed for the switching network.

Many fast algorithms for the block matching have done a very good job to reduce the computational complexity of the motion estimation process. Nearly all of the fast algorithms, even the full search algorithm, use the objective measurement to determine the accuracy of the motion vector. It is well known that the Human Visual System (HVS) is very sensitive to object continuities rather than individual pixel intensity. The reliability of the predicted motion vector is doubtful for some images that contain large amount of moving objects. Thus, in Chapter 6, we proposed a scheme which modifies on the conventional matching criterion to improve the visual quality of the predicted frame. With the introduction of the enhancement factor, an emphasis on the object edges is achieved in the computation of matching criterion. We have emphasized to use edges of objects for the computation of the matching criterion and, as a result, the motion vectors can be accurately predicted. Although the simulation results show that the objective quality has a little degradation as compare to the full search, the subjective quality of the predicted frame using our new matching criterion has been improved.

7.2 Future Directions

As video coding algorithms become more and more sophisticated. The computational complexity of the algorithm has grown into a level that the general purpose processor and even specially designed video processor cannot handle this huge computational requirement. Special hardware with programmability should be developed as a co-processor to general purpose processor, which provides flexible configuration to adapt to different video coding algorithms and high speed processing capability to achieve real-time operations.

For many years, the block-based hybrid video coding approaches as discussed in Chapter 2 have been the most successful video coding techniques that exhibits simple and efficient coding performance. However, the absence of consideration for the Human Visual System (HVS) in the design of the coder which leaves many problems, such as the blocking effect, which are annoying to viewers. Furthermore, this block-based coding scheme has many limitations to low bit-rate application. Although we have shown that the modification on the conventional matching criterion improves the visual quality of the estimated frame, it is still desirable to develop a more generic video coding techniques to further improve the visual quality and is suitable for low bit-rate applications.

Model based video coding which has been considered as one of the candidate that provides both coding efficiency and visual quality improvement. However, the very high computational complexity has limited the wide spread application of the model based coding algorithm and hence efficient algorithms to reduce the computational complexity should be developed.

References

[1] A. N. Netravali and B. G. Haskell, Digital Pictures Representation and Compression. New York: Plenum, 1994.

[2] L. D. Davisson, "Rate-Distortion theory and application", Proceedings of the IEEE, Vol. 60, No. 7, pp. 800-808, July 1972.

[3] A. N. Netravali and J. O. Limb, "Picture coding: a review", Proceedings of IEEE, Vol. 68, No. 3, pp. 366-406, March 1980.

[4] A. K. Jain, "Image data compression: a review", Proceedings of IEEE, Vol. 69, pp. 349-389, March 1981.

[5] H. G. Musmann, P. Pirsch and H. J. Grallert, "Advances in picture coding", Proceedings of IEEE, Vol. 73, No. 4, pp. 523-548, April 1985.

[6] M. Kunt, A. Ikonomopoulos and M. Kocher, "Second generation image coding techniques", Proceedings of IEEE, Vol. 73, No. 6, pp. 549-575, April 1985.

[7] H. Karhunen, "Uber lineare methoden in der Wahrsheinlich-Keitsrechnug", Ann. Acad. Science Fenn, Ser. A.I. 37, Helsmki, 1947.

[8] M. Loeve, "Fonctions aleatoires de seconde ordre", in P. Levy, Processus Stochastiques et Mouvement Brownien. Paries, France: Hermann, 1948.

[9] K.R. Rao and R. Yip, Discrete Cosine Transform – Algorithms, Advantages and Applications. New York: Academic Press, 1990.

[10] S. Y. Kung, VLSI Array Processors, Englewood Cliffs, NF: Prentice Hall, 1988.

[11] B. K. P. Horn, Robot Vision, Cambridge, MA: M.I.T. Press 1986.

[12] A. Gersho and R. M. Gray, "Vector Quantization and Signal Compression", Kluwer Academic Publishers, 1992.

[13] C. I. Podilchuk, N. S. Jayant, N. Farvardin, "Three-dimensional subband coding of video", IEEE Transactions on Image Processing, Vol. 4, No. 2, pp. 125-139, February 1995.

[14] ISO/IEC JTC1 / SC2 / WG8 N933, "JPEG Technical Specification, Revision 5", Jan. 16 1990.

[15] ITU-T Recommendations H.261, "Video codec for audivisual services at p x 64 kbit/s"

[16] Draft ITU-T Recommendations H.263, "Video coding for narrow telecommunication channels at < 64 kbit/s", Technical report, ITU, July 1995.

[17] ISO/IEC IS 11172(MPEG-1), "Coding of moving pictures and associated audio for digital image storage media at up to 1.5Mbits/sec", Technical report, Motion Picture Experts Group, 1993.

[18] ISO/IEC JTC1 / SC29 / WG11 Moving Picture Experts Group, "MPEG2 Test model 4", 1993

[19] ISO/IEC JTC1 / SC29 / WG11 Moving Picture Experts Group, "MPEG-4 Proposal Package Description", July 1995.

[20] W.H. Chen, C. H. Smith, and S. C. Fralick, "A fast computational algorithms for the Discrete Cosine Transform", IEEE Trans. on Communication, Vol. COM-25, pp. 1004-1009, Sept. 1997.

[21] M. Vetterli and H. J. Nussbaumer, "Simple FFT and DCT algorithms with reduced number of operations", Signal Processing, Vol. 6, No. 4, pp. 267-278, Aug. 1984.

[22] Martin Vetterli, "Fast 2-D Discrete Consine Trasnform", Intl. Conf. On Acoust., Speech, and Signal Processing, pp. 1538-1541, Tampa, FL, March 26-29 1985.

[23] N. I. Cho, S. U. Lee, "Fast Algorithm and Implementation of 2-D Discrete Cosine Transform", IEEE Trans. on Circuits and Systems, Vol 38, No. 3, March 1991.

[24] M. A. Haque, "A Two-Dimensional Fast Cosine Trasnform", IEEE Trans. on acoustics, speech, and signal processing, Vol. ASSP-33, No. 6, December 1985.

[25] T. Masaki, Y. Morimoto, T. Onoye, I. Shirakawa, "VLIS Implementation of Inverse Discrete Cosine Transformer and Motion Compensator for MPEG2 HDTV Video Decoding", IEEE Trans. on Circuits and Systems for Video Technology, Vol. 5, No. 5, October 1995.

[26] S. Uramoto, Y. Inoue, A. Takabatake, et. al, "A 100-MHz 2-D Discrete Cosine Transform Core Processor", IEEE Journal of Solid-State Circuits, Vol. 27, No. 4, April 1992.

[27] M. T. Sun, T. C. Chen, A. M. Gottlieb, "VLSI Implementation of a 16 x 16 Discrete Cosine Transform", IEEE Trans. on Circuits and Systems, Vol.36, No. 4, April 1989.

[28] Y. H. Chan and W. C. Siu, "On the realization of discrete cosine transform using the distributed arithmetic", IEEE Transactions on Circuits and Systems, Vol. 39, No. 9, pp. 705-712, September 1992

[29] A. Madisetti and A. N. Willson, "A 100 MHz 2-D 8x8 DCT/IDCT Processor for HDTV Applications", IEEE Trans. on Circuits and Systems for Video Technology, Vol. 5, No. 2, April 1995.

[30] Y. T. Chang, C. L. Wang, "New Systolic Array Implementation of the 2-D Discrete Cosine Transform and Its Inverse", IEEE Trans. on Circuits and Systems for Video Technology, Vol. 5, No. 2, pp. 150-157, April 1995.]

[31] L. W. Chang, M. C. Wu, "A Unified Systolic Array for Discrete Cosine and Sine Transforms", IEEE Trans. on Signal Processing, Vol. 39, No. 1, pp. 192-194, Jan 1991.

[32] F. Dufaux and F. Moscheni, "Motion Estimation techniques for Digital TV: A Review and New Contribution", Proceedings of IEEE, Vol. 83, No. 6, June 1995.

[33] B. Liu, A. Zaccarin, "New fast algorithms for the estimation of block motion vectors", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 3, No. 3, pp 148-157, Oct. 1993.

[34] M. Bierling, "Displacement estimation by hierarchical block matching", SPIE VCIP'88, Vol. 1001, pp. 942-951, 1988.

[35] X. Lee, Y. Q. Zhang, "A Fast Hierarchical Motion-Compensation Scheme for Video Coding Using Block Feature Matching", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 6, No. 6, pp. 627-635, December 1996.

[36] Y. Q. Shi, X. Xia, "A Thresholding Multiresolution Block Matching Algorithm", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 7, No. 2, pp. 437-440, April 1997.

[37] J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding", IEEE Transactions on Communications, Vol. COM-29, pp. 1799-1808, Dec. 1981.

[38] H. Gharavi and M. Mills, "Blockmatching motion estimation algorithms – new results", IEEE Transactions on Circuits and Systems, Vol. 37, No. 5, pp. 649-651, May 1990.

[39] M. Ghanbari, "The Cross-Search Algorithm for Motion Estimation", IEEE Transactions on Communications, Vol. 38, No. 7, pp. 950-953, July 1990.

[40] M. J. Chen, L. G. Chen, T. D. Chiueh, "One Dimensional Full Search Motion Estimation Algorithm For Video Coding", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 4, No. 5, pp. 504-509, October 1994.

[41] R. Li, B. Zeng, M. L. Liou, "A New Three Step Search Algorithm for Block Motion Estimation", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 4, No. 4, pp. 438-442, August 1994.

[42] L. M. Po, W. C. Ma, "A Novel Four-Step Search algorithm for Fast-Block Motion Estimation", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 6, No. 3, pp. 313-317, June 1996.

[43] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, T. Ishiguro, "Motion compensated interframe coding for video conferencing", Proc. Nat. Telecommun. Conf., pp. G.5.3.1-5.3.5, Nov. 29-Dec. 3, 1981.

[44] L. G. Chen, W. T. Chen, Y. S. Jehng, and T. D. Chiueh, "An Efficient Parallel Motion Estimation Algorithm for Digital Image Processing", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 1, No. 4, 378-385, Dec. 1991.

[45] S. Zafar and Y. Q. Zhang, "Predictive Block-Matching Motion Estimation for TV Coding-Part I: Inter-Block Prediction", IEEE Transactions on Broadcasting, Vol. 37, No. 3, pp. 97-101, September 1991.

[46] Y. Q. Zhang and S. Zafar, "Predictive Block-Matching Motion Estimation for TV Coding-Part II: Inter-Frame Prediction", IEEE Transactions on Broadcasting, Vol. 37, No. 3, pp. 102-105, September 1991.

[47] Y. L. Chan and W. C. Siu, "Adaptive multiple-candidate hierarchical search for block matching Algorithm", Electronics Letters, Vol. 31, No. 19, pp.1637-39, Sept. 1995.

[48] Y. L. Chan and W. C. Siu, "New adaptive pixel decimation for block motion vector estimation", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 6, No. 1, pp.113-118, February. 1996.

[49] Y. L. Chan, W. L. Hui and W. C. Siu, "A Block Motion Vector Estimation Using a Pattern Based Pixel Decimation", Proceedings of the IEEE International Symposium on Circuits and System (ISCAS'97), pp.1153-1156, June 1997.

[50] Y. L. Chan and W. C. Siu, "Block motion vector estimation using edge matching: an approach with better frame quality as compared to full search algorithm", Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'97), pp. 1145-1148, June 1997.

[51] S. Kappagantula and K.R. Rao, "Motion compensated predictive coding", SPIE 27th Proc. 432, pp. G4-70, 1983.

[52] C. Auyeung, J. Kosmach, M. Orchard, and T. Kalafatis, "Overlapped block motion compensation", Proceedings of the SPIE: Visual Communications and Image Processing, Boston, MA, Vol. 1818, PP. 561-571, 1992.

[53] Seung-Hyun Nam, Jong-Seob Baek and Moon-Key Lee, "Flexible VLSI Architecture of Full Search Motion Estimation for Video Applications", IEEE Transactions on Consumer Electronics, Vol. 40, No. 3, pp. 176-183, May 1994.

[54] Kun-Min Yang, Ming-Ting Sun and Lancelot Wu, "A Family of VLSI Design for the Motion Compensation Block-Matching Algorithm", IEEE Transactions on Circuits and Systems, Vol. 36, No. 10, pp. 1317-1325, Oct. 1989.

[55] L. De Vos and M. Stegherr, "Parameterizable VLSI architectures for the full-search block-matching algorithm", IEEE Transactions on Circuits and Systems Vol. 36, pp. 1309-1316, Oct. 1989.

[56] C. H. Hsieh and T. P. Lin, "VLSI architecture for block-matching motion estimation algorithm", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 2, pp. 169-175, June 1992.

[57] T. Komarek and P. Pirsh, "Array architectures for block matching algorithms", IEEE Transactions on Circuits and Systems, Vol. 36, pp 1301-1308, Oct. 1989.

[58] H. Yeo and Y.H. Hu, "A novel modular systolic array architecture for full-search block matching motion estimation", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 5, pp 407-416, Oct. 1995.

[59] S. B. Pan, S. S. Chae, R. H. Park, "VLSI Architecture for Block Matching Algorithms Using Systolic Arrays", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 6, No. 1, pp 67-73, February 1996.

[60] H. M. Jong, L. G. Chen, T. D. Chiueh, "Parallel architectures for 3-Step Hierarchical Search Block Matching Algorithm", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 4, No. 4, pp. 407-415, August 1994

[61] B. M. Wang, J. C. Yen and S. Chang, "Zero Waiting-Cycle Hierarchical Block Matching Algorithm and its Array Architecture", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 4, No. 1, February 1994.

[62] S. C. Cheng, H. M. Hang, "A Comparison of Block-Matching Algorithms Mapped to Systolic-Array Implementation", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 7, No. 5, October 1997.

[63] W. C. Siu, W. L. Hui and Y. L. Chan, "Realization of Enhanced Adaptive Algorithm for Block Motion Vector Estimation Using Multiple bus Structure", Proceedings of the IEEE TENCON'97, pp.765-768, December 2-4 1997.

[64] P. Pirsch, N. Demassieux and W. Gehrke, "VLSI Architectures for Video Compression – A Survey", Proceedings of IEEE, Vol. 83, No. 2, pp. 220-246, Feb 1995.

[65] H. Fujiwara, M. L. Liou, M. T. Sun, et al., "An All-ASIC Implementation of a Low Bit-Rate Video Codec", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 2, No. 2, PP. 123-134, June 1992.

[66] T. Murakami, K. Kamizawa, M. Kameyama, S. I. Nakagawa, "A DSP Architectural Design for Low Bit-Rate Motion Video Codec", IEEE Transactions on Circuits and Systems, Vol. 36, No. 10, pp. 1267-1274, October 1989.

[67] M. Toyokura, H. Kodama, E. Miyagoshi, et al, "A Video DSP with Macroblock-Level-Pipeline and a SIMD Type Vector-Pipeline Architecture for MPEG2 CODEC", IEEE Journal of Solid-State Circuits, Vol. 29, No. 12, PP. 1474-1481, December 1994.

[68] R. Gnanasekaran, "A Fast Serial Parallel Binary multiplier", IEEE Transactions on Computers, Vol. 34, No. 8, August 1985.

[69] S. Sunder, F. E. Guibaly, A. Antoniou, "Two's Complement Fast Serial Parallel Multiplier", IEEE Proceedings on Circuits Devices System, Vol. 142, No. 1, February, 1995.

[70] C. R. Baugh, B. A. Wooley, "Digital Multiplier Power Estimates", IEEE Transactions on Communications, COM-23, pp. 1287-1288, November 1975.

[71] I. C. Wu, "A Fast 1-D Serial-Parallel Systolic Multiplier", IEEE Transactions on Computers, Vol. C-26, No. 10, pp. 1243-1247, October 1987.

[72] "Altera Data Book 1997", Altera, 1997

[73] "AMCC S5933 PCI Controller Data Book", AMCC, 1996.

[74] I. D. Yun and S. U. Lee, "On the fixed-point error analysis of several Fast DCT algorithms", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 3, pp. 27-41, February 1993.

[75] A. J. Jalali and K. R. Rao, "Limited wordlength and FDCT processing accuracy", Proceeding of ICASSP'81, pp. 1180-1183, March 1981.

[76] F. E. Barber, W. E. Werner, P. A. Wilford, et al, "A 64x17 Non-Blocking Crosspoint Switch", Proceedings of IEEE International Solid-State Circuits Conference, pp. 116-117, 322, February 1988.

[77] K. Choi and W. S. Adams, "VLSI Implementation of a 256x256 Crossbar Interconnection Network", Proceedings of IEEE International Parallel Processing Symposium, pp. 289-293, March 1992.