National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

*Your file Votre référence*

*Our file Notre référence*

## NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.



## AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.



Canada

# Novel Motion Estimators for Video Compression

**Eric Wai Chi CHAN**

A THESIS

submitted to the School of Graduate Studies and Research

in Partial Fulfillment of the Requirements

for the Degree of

MASTER OF APPLIED SCIENCE

in

Electrical Engineering

Ottawa-Carleton Institute of Electrical Engineering

Department of Electrical Engineering

Faculty of Engineering

University of Ottawa

OTTAWA, ONTARIO, K1N 6N5

ISBN 0-315-95895-2

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

*To my family,*

# Acknowledgments

First, I would like to express my gratitude and appreciation to my supervisor Dr. S. Panchanathan. His support, guidance and encouragement were the driving force for the successful completion of this thesis. Special thanks are due to Dr. Xiping Wang and Mr. Rakesh Gandhi for their valuable comments and suggestions.

I would also like to thank all the past and current members of the Multimedia Communications Research Laboratory, especially Grant Henderson for his technical computing consultations.

The help and co-operation of all the supporting staffs of the Department of Electrical Engineering, especially Michèle Roy and Amanda Lauzon are appreciated.

Finally, I am grateful to my parents, my sisters and my family for their consistent support, patient and understanding, without which this thesis would not have been possible.

# Abstract

Video compression is becoming increasingly important with the advent of the compression standards and broadband networks. Typical applications include high-definition television, multimedia communications, etc. There are two kinds of redundancies that exist in a video sequence, namely, temporal and spatial which are exploited to achieve compression. The spatial redundancies are removed by using techniques such as discrete cosine transform, vector quantization, etc. while the temporal redundancies are removed by the motion estimation/compensation techniques. Recently, several block based motion estimation algorithms have been reported in the literature. However, these algorithms are either computationally expensive or converge to a local optimum.

In this thesis, the problem of motion estimation is addressed from two perspectives, namely, hardware architecture and reduced complexity algorithms in the spatial and transform domains. First, a VLSI architecture which implements the full search block matching algorithm in real time is presented. The interblock dependency is exploited and hence the architecture can meet the real time requirement in various applications. Most importantly, the architecture is simple, modular and cascadable. Hence the proposed architecture is easily implementable in VLSI as a codec.

The spatial domain algorithm consists of a layered structure and alleviates the local optimum problem. Most importantly, it employs a simple matching criterion, namely, a modified pixel difference classification (MPDC) and hence results in a reduced computational complexity. In addition, the algorithm is compatible with the recently proposed MPEG-1 video compression standard. Simulation results indicate that the proposed algorithm provides a comparable performance (compared to the algorithms reported in the literature) at a significantly reduced computational complexity. In addition, the hardware implementation of the proposed algorithm is very simple because of the binary operations used in the matching criteria.

Finally, we present a wavelet transform based fast multiresolution motion estimation (FMRME) scheme. Here, the wavelet transform is used to exploit both the spatial and temporal redundancies resulting in an efficient coder. In FMRME, the correlations among the orientation subimages of the wavelet pyramid structure are exploited resulting in an efficient motion estimation process. In addition, this significantly reduces side information for motion vectors which corresponds to significant improvements in coding performance of the FMRME based wavelet coder for video compression. Simulation results demonstrate the superior coding performance of the FMRME based wavelet transform coder.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Video compression is becoming increasingly important with the advent of the compression standards (JPEG, MPEG, H.261, etc.) and broadband networks (ISDN, ATM, etc.) [1], [2]. The higher channel bandwidth and storage capacity required for a video signal necessitate the use of compression techniques in applications such as high-definition television, video conferencing, multimedia communications, etc. [3], [4]. There are two kinds of redundancies in a video sequence, namely, temporal and spatial. Compression is essentially achieved by exploiting these redundancies. Typically, the temporal redundancies are removed by using motion estimation/ compensation techniques [5], [6] which is followed by spatial redundancy removal techniques such as discrete cosine transform [7] , vector quantization [8], etc.

Two most commonly used approaches for motion estimation in the literature are pel-recursive and block matching techniques. In the pel-recursive algorithms, the motion information is estimated for individual pixels [17], [18], [19]. This class of algorithms do not require the transmission of motion information since they recursively use the relative luminance change to determine the motion in a frame. However, these algorithms involve extensive computations. In addition, failure of correct estimation on the edges is a major drawback of pel-recursive algorithms [20].

Block matching algorithms [5] have been proposed to reduce the computational burden in pel-recursive algorithms and to improve the prediction performance. In the block matching process, the current frame ($t$) of a video sequence is divided into non-overlapping reference blocks. Each reference block in the current frame is compared with candidate blocks from a search area in the previous frame ($t-1$) in order to obtain the closest match block with respect to a prespecified error criterion. The offset between the reference block and the best match candidate block specifies the motion vector. The most intuitive approach for block matching is to use the full search algorithm (FSA) where all possible candidate blocks are searched to obtain the best match block for each reference block. However, this is a computationally intensive procedure and hence requires parallel architecture for real time execution.

Recently, hardware architectures which implement FSA have been reported in the literature [37-48]. However, these architectures are either complex and/or non-cascadable. This results in an increased hardware requirement and reduced flexibility in VLSI implementation.

The computational burden of FSA can be decreased by reducing the number of candidate blocks required to be searched. Recently, several fast algorithms [23-28] have been reported in the literature. These fast algorithms are based on the assumption that the error (e.g. mean absolute error (MAE)) increases monotonically as the search moves away from the direction of minimum distortion (the global optimum position). This results in a reduced number of candidate blocks to be searched. However, this may lead to a motion vector which converges to a local optimum rather than a global optimum. This reduces the accuracy of the estimated motion vectors resulting a poorer performance compared to the FSA. In addition, the fast algorithms requires complicated control structure which corresponds to expensive hardware realizations. Hence it is desirable to have reduced complexity motion estimation algorithms which have a comparable performance to FSA and in addition be easily implementable in hardware.

Since block matching algorithms individually estimate a motion vector for each nonoverlapping reference block, this may result into block edge discontinuities causing disturbing artifacts in the reconstructed frame. An alternative approach for motion estimation is the transform domain technique which execute the cross-correlation function via the frequency domain [16], [66]. Motion vectors are calculated using overlapped, windowed regions of the transform coefficients of the image data. This reduces the block edge discontinuities resulting in a smoother motion field. However, we note that the execution of the cross-correlation function is computationally expensive resulting in a complex codec design.

We recall that the temporal redundancies are typically removed by using motion estimation/compensation techniques while the spatial redundancies are removed by techniques such as discrete cosine transform, vector quantization, etc. We note that these redundancies are exploited independently. In addition, the residual video signal (i. e. the difference between the motion compensated previous frame $(t-1)$ and the current frame $(t)$) tends to be highly

nonstationary [11]. Hence it is necessary to use a smaller block size to reduce the nonstationarity of each block. However, this increases the number of reference blocks in a frame resulting in a computationally expensive motion estimation algorithm. Recently, a wavelet transform based motion estimation technique, namely multiresolution motion estimation (MRME) [11], [15] has been proposed in the literature to reduce the computational complexity in motion estimation. Here, wavelet transform is used to exploit both the spatial and temporal redundancies. In addition, we note that wavelet transform is flexible in representing nonstationary video signals [11] resulting in a simple and efficient video coder. In MRME, the cross-correlations among each level of the wavelet pyramid structure is exploited. However, the motion vectors are estimated separately for each wavelet subimage at each pyramid level. Hence the MRME scheme is still a computationally expensive procedure.

In this thesis, the problem of motion estimation is addressed from two perspectives, namely, hardware architecture and reduced complexity algorithms. First, a VLSI architecture which implements the full search block matching algorithm (FSA) in real time is presented. The architecture consists of a two-dimensional structure of basic cells (BC's) where each BC is capable of computing the mean absolute error (MAE). The interblock dependency is exploited and hence the architecture can meet the real time requirement in various applications. Most importantly, the architecture is simple, modular and cascadable. Hence the proposed architecture is easily implementable in VLSI as a codec.

We recall that since the FSA requires all possible candidate blocks to be searched, it is a computationally expensive procedure. In this thesis, we present two novel motion estimation algorithms in the spatial and transform domains for video compression. The spatial domain algorithm consists of a layered structure and alleviates the local optimum problem of fast search algorithms. Most importantly, it employs a simple matching criterion, namely, a modified pixel difference classification (MPDC) and hence results in a reduced computational complexity compared to FSA. In addition, the algorithm is compatible with the recently proposed MPEG-1 video compression standard. Simulation results indicate that the proposed algorithm provides a

comparable performance to the FSA using MAE at a significantly reduced computational complexity. In addition, the hardware implementation of the proposed algorithm is very simple because of the binary operations used in the matching criteria.

Finally, we present a fast multiresolution motion estimation (FMRME) scheme based on wavelet transform coding. In FMRME, the set of orientation subimages at each level of the wavelet pyramid structure are first combined together into a single (all-orientation) subimage, and then the motion estimation is applied on the newly formed subimage. The motion vectors of an all-orientation subimage at a lower level are predicted from the motion vectors at the preceding higher level and are refined at each step. This reduces the search time for the motion vectors by 66% of that MRME scheme. In other words, the FMRME scheme has the advantages of significantly reduced side information for motion vectors and search time. This corresponds to significant improvements in coding performance of the FMRME based wavelet coding compared to the MRME based wavelet coding for video compression. Simulation results show that the performance of FMRME is comparable to the MRME approach and confirm the reductions in execution time.

This thesis is organized as follows: In chapter 2, we present a review of motion estimation techniques in both spatial and transform domains. The new VLSI architecture for the FSA is presented in chapter 3. In chapter 4, we present the reduced complexity block matching based motion estimation algorithm. The FMRME scheme for wavelet transform coding is detailed in chapter 5. Finally, the conclusions and suggestions for future research work are given in chapter 6 followed by the bibliography.

# Chapter 2

# Review of Motion Estimation

In this chapter, we present a review of motion estimation. First, spatial domain motion estimation techniques such as pel-recursive, block matching, etc. are reviewed. Since block matching is widely used for motion estimation, the various block matching (BM) based motion estimation algorithms are detailed. These algorithms are classified into three categories, namely, fast algorithms, layered structure algorithms and inter-block motion field prediction algorithms. This is followed by a review of transform domain motion estimation techniques such as the multiresolution motion estimation (MRME) scheme in the wavelet transform domain. Finally, various video compression schemes which use combinations of different inter/intra frame coding techniques are detailed.

## 2.1 Motion Estimation

We recall from chapter 1 that temporal redundancies correspond to the correlations between successive frames in a video sequence. These redundancies can be greatly reduced by using motion estimation/compensation techniques [5], [6]. This results in a considerable improvement in compression performance compared to simple interframe replenishment coding, i.e. coding only the differences between two successive frames. The two mainstream approaches for motion estimation are pel-recursive and block matching techniques. These techniques are used to obtain (estimate) the motion information for the pixels/group of pixels in a frame. This motion information is then used to predict the current frame $(t)$ from the previous frame $(t-1)$ resulting in a motion compensated previous frame $(t-1)$. The difference between the motion compensated pervious frame $(t-1)$ and the current frame $(t)$ is then coded and transmitted to the receiver.

### 2.1.1 Pel-Recursive Techniques

In the pel-recursive techniques, the motion information is estimated for individual pixels [17], [18], [19]. For each pixel in the current frame $(t)$ with luminance intensity $I_t(x,y)$ (x and y are the coordinates of the pixel), a pixel with the corresponding intensity level $I_{t-1}(x+dx, y+dy)$ in the previous frame $(t-1)$ is found where $D=(dx,dy)$ is the displacement vector, i.e.

$I_t(x,y) = I_{t-1}(x+dx, y+dy)$. In pel-recursive algorithms, an initial estimate of the displacement vector $D_i$ is used to obtain an improved estimate $D_{i+1}$ defined as follows:

$$D_{i+1} = D_i + U_i \qquad \text{--- (2.1)}$$

where $U_i$ is the *update term of iteration i*. The accuracy of the estimate $D_i$ is measured by a function called displaced frame difference (DFD) which is defined as follows:

$$DFD(x,y,D_i) = I_t(x,y) - I_{t-1}(x+dx_i, y+dy_i) \qquad \text{--- (2.2).}$$

A good estimate of $D_i$ corresponds to a small value of DFD resulting in a fast convergence of equation (2.1). Hence the DFD function (equation (2.2)) can be used as a criterion for calculating the estimate $D_{i+1}$.

Recently, Netravali *et al.* [17] have proposed a pel-recursive algorithm which minimizes the square value of the DFD criterion using the gradient method:

$$\begin{aligned} D_{i+1} &= D_i - \frac{1}{2}\varepsilon \nabla_{D_i}[DFD(x,y,D_i)]^2 \\ &= D_i - \varepsilon DFD(x,y,D_i)\cdot \nabla_{D_i}[DFD(x,y,D_i)] \end{aligned} \qquad \text{--- (2.3)}$$

where $\nabla_{D_i}$ is the gradient operator with respect to $D_i$ and ε is a positive constant. The procedure for estimating the displacement vector $D$ is illustrated in Figure 2.1. We note that the update term $U_i$ at each iteration is proportional to the gradient $\nabla_{D_i}[DFD(x,y,D_i]^2$ and constant ε (equation (2.3)). Substituting equation (2.2) into equation (2.3) yields:

$$\begin{aligned} D_{i+1} &= D_i - \varepsilon DFD(x,y,D_i)\cdot \nabla I_{t-1}(x-dx_i, y-dy_i) \\ &= D_i - \varepsilon DFD(x,y,D_i)\cdot \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \cdot I_{t-1}(x-dx_i, y-dy_i) \end{aligned} \qquad \text{--- (2.4)}$$

where $\nabla$ is a gradient operator with respect to the horizontal and vertical coordinates $(x,y)$. A larger value of ε corresponds to a faster convergence but results in a less accurate estimate of the displacement vector.

We note that the pel-recursive algorithms do not require the transmission of motion information to the receiver since they recursively use the relative luminance change to determine the motion in a frame. In other words, the receiver is also required to execute the motion estimation

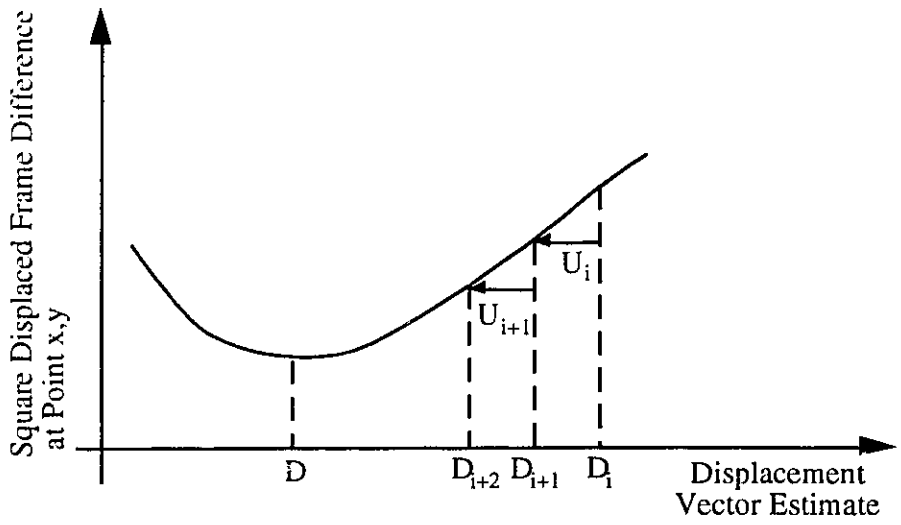



**Figure 2.1: Illustration of Pel-Recursive Algorithm**

algorithm [73]. We note that the pel-recursive algorithms are executed on each pixel of a frame independently and require a few iterations of execution. This results in a computationally expensive motion estimation process.

Wang *et al.* [20] have shown that pel-recursive algorithms are susceptible to noise. In other words, the recursive operation (equation (2.1)) is invalidated if any one of the values used in the calculation of the gradient (equation (2.4)) is corrupted by noise. In addition, failure of correct estimation on the edges is a major drawback of pel-recursive algorithms [20].

### 2.1.2 Block Matching Techniques

Block matching techniques reduce the computational burden in pel-recursive algorithms and improve the prediction performance. Here, all pixels within a block are assumed to have the same motion activity [5]. In the block matching process, the current frame ($t$) of a video sequence is divided into blocks of size $n \times n$. For each block (reference block) in the current frame ($t$), the previous frame ($t-1$) is searched within a neighborhood (search area) in order to obtain the best match block with respect to a prespecified error criterion. This process is illustrated in Figure 2.2. The search area consists of $(n+2p)\times(n+2p)$ pixels, where $p$ is the maximum allowed

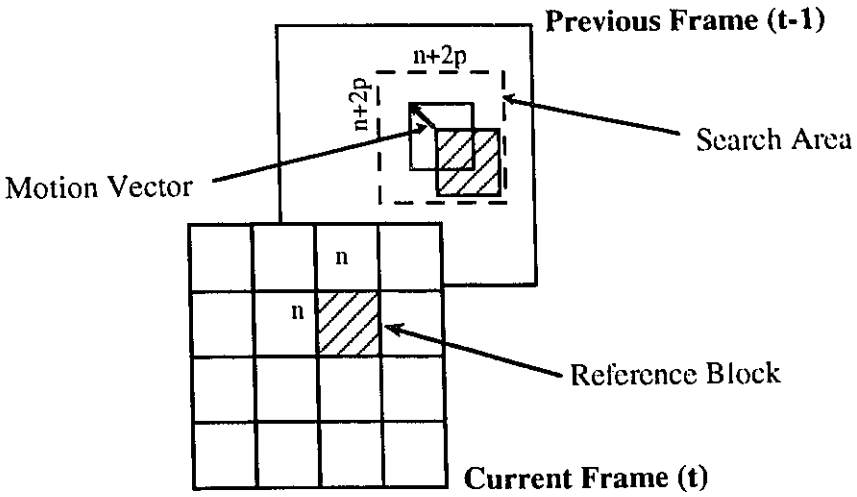



**Figure 2.2: Block Matching Motion Estimation Process**

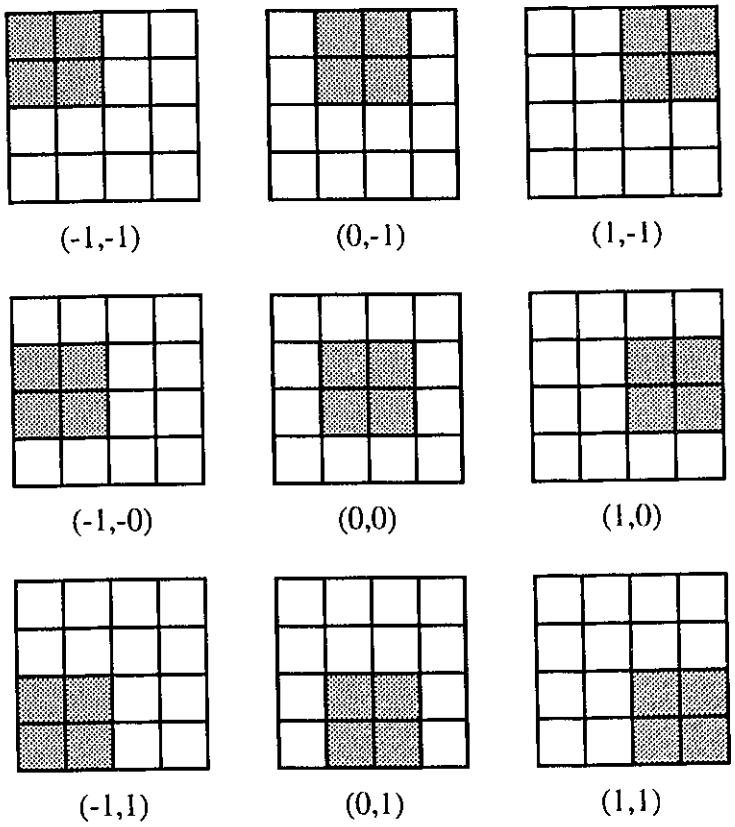



**Figure 2.3: Possible Candidate Blocks for n=2 and p=1**

displacement. We note that the total number of possible candidate blocks is $(2p+1)^2$ and the corresponding motion vectors range from $(-p,-p)$ to $(p,p)$ as illustrated in Figure 2.3 for $n=2$

and $p = 1$. The best match block is used as a prediction estimate for the reference block. The relative displacement between the reference block and the best match block constitutes the motion vector. The motion vectors together with the prediction error image (i.e. the difference between the motion compensated previous frame $(t-1)$ and the current frame $(t)$) are then coded and transmitted to the receiver. This contrasts with the pel-recursive techniques where motion information is not required to be transmitted.

## 2.2 Block Matching Algorithms

The most intuitive approach for block matching is to use the full search algorithm (FSA). For each reference block, all possible $(2p+1)^2$ candidate blocks are searched to obtain the best match block using a criterion such as mean square error (FSA-MSE), mean absolute error (FSA-MAE) or pixel difference classification (FSA-PDC).

The mean square error (MSE) is defined as follows:

$$MSE(\Delta x, \Delta y) = \sum_{i=1}^{n}\sum_{j=1}^{n}(S(i+\Delta x, j+\Delta y) - R(i,j))^2 \qquad -p \le \Delta x, \Delta y \le p \qquad \text{--- (2.5)}$$

where $R(i,j)$ is a reference block's pixel in the current frame $(t)$ and $S(i+\Delta x, j+\Delta y)$ is a candidate block's pixel within a search area in the previous frame $(t-1)$. The candidate block which minimizes the MSE function is selected as the best match block and the corresponding displacement $(\Delta x, \Delta y)$ is the motion vector $(V)$. In algorithmic form, the FSA-MSE can be written as follows:

**Initialization:** • Δx = -p;

• Δy = -p; /* starting from the upper leftmost candidate block (Figure 2.3) */

• Champion = ∞;

**Step:** While (Δy ≤ p)

While (Δx ≤ p)

• Distortion = MSE(Δx, Δy);

- if (Distortion < Champion) then
    - * Champion = Distortion;
    - * Motion Vector = (Δx,Δy);
  - end if
  - Δx = Δx + 1;
- end of while loop
- Δx = -p;
- Δy = Δy + 1;

end of while loop

- Motion Vector (V) = (Δx,Δy).

We note that execution of equations (2.5) involves $n^2$ accumulation, $n^2$ subtraction and $n^2$ squaring operations for each candidate block. For example, in the MPEG standard [9] with $n = 16$ and $p = 7$, there are 255 (i.e. $(2p+1)^2$) candidate blocks required to be searched for each reference block. This corresponds to 65280 (i.e. $255 \times 16^2$) accumulations, subtractions, and squaring (multiplication) operations to be executed. We note that the multiplication operations contribute significantly to the high complexity of FSA-MSE.

The computational complexity of FSA-MSE can be reduced by using the mean absolute error (MAE) as the matching criterion which is defined as follows:

$$MAE(\Delta x, \Delta y) = \sum_{i=1}^{n}\sum_{j=1}^{n}|S(i+\Delta x, j+\Delta y) - R(i,j)| \qquad \text{--- (2.6)}$$

We note that the multiplication operation in the MSE is replaced by the magnitude operation in the MAE. The superior performance of FSA-MAE has been reported by Orchard [21]. However, execution of equation (2.6) still involves $n^2$ accumulation, $n^2$ subtraction and $n^2$ magnitude operations for each candidate block. For the same example ($n = 16$ and $p = 7$), this corresponds to 65280 accumulations, subtractions and magnitude operations to be executed. Although FSA-MAE is less complex compared to FSA-MSE, it is still a computationally expensive algorithm.

In order to reduce the computational burden in the MSE and MAE, Gharavi *et al.* [22] have proposed a simple matching criterion, namely, pixel difference classification (PDC). In PDC, each pixel in a block is classified as either a *matching pixel* or *mismatching pixel* with respect to a threshold $t$, i.e.

$$T(i,j,\Delta x,\Delta y) = 1 \quad if\ |S(i+\Delta x, j+\Delta y) - R(i,j)| \le t$$
$$= 0 \quad otherwise \qquad \text{--- (2.7)}$$

where $T(i,j,\Delta x,\Delta y)$ is the binary representation of the pixel difference with a value of one or zero corresponding to a *matching* or *mismatching pixel.* The PDC criterion is defined as follows:

$$PDC(\Delta x,\Delta y) = \sum_{i=1}^{n}\sum_{j=1}^{n} T(i,j,\Delta x,\Delta y) \qquad -p \le \Delta x,\Delta y \le p \qquad \text{--- (2.8)}$$

The candidate block which maximizes the PDC function is selected as the best match block and the corresponding displacement $(\Delta x,\Delta y)$ is the motion vector $(V)$. The search procedure of FSA-PDC can be written in algorithmic as follows:

**Initialization:**
- Δx = -p;
- Δy = -p; /* starting from the upper leftmost candidate block (Figure 2.3) */
- Champion = 0;

**Step:** While (Δy ≤ p)

  While (Δx ≤ p)

    - Matching = PDC(Δx, Δy);
    - if (Matching > Champion) then
      * Champion = Matching;
      * Motion Vector = (Δx,Δy);

    end if

    - Δx = Δx + 1;

  end of while loop

  - Δx = -p;
  - Δy = Δy + 1;

end of while loop

• Motion Vector (V) = ($\Delta x$,$\Delta y$).

We note that PDC represents the matching algorithm as a binary process which consequently reduces the computational complexity. For example, execution of equation (2.8) involves only $n^2$ comparison and $n^2$ increment operations which corresponds to 65280 comparison and increment operations to be executed for the example of $n = 16$ and $p = 7$. The performance of the PDC criterion, however, depends on the choice of the threshold $t$. For example, a smaller threshold value performs better than a larger threshold for a slow motion sequence. This is a result of the large number of candidate blocks with the same PDC value for a large threshold. However, the situation is reversed for a fast motion sequence. Hence the performance of FSA-PDC can vary significantly based on the motion in the video sequence.

## 2.3 Fast Algorithms

The computational burden of FSA can be decreased by reducing the number of candidate blocks required to be searched. Recently, several fast algorithms [23-28] have been reported in the literature. These fast algorithms are based on the assumption that the error (MSE or MAE) increases monotonically as the search moves away from the direction of minimum distortion (the global optimum position). This results in a reduced number of candidate blocks to be searched.

For example, the orthogonal search (OS) algorithm [26] tracks the direction of minimum distortion by examining three candidate blocks in each step of the search procedure. The search procedure for $p = 5$ is illustrated in Figure 2.4. We note that each intersection (of two lines) represents one possible candidate block and the distance between two adjacent intersections (candidate blocks) is one pixel (Figure 2.3). In the first search step, the candidate block corresponding to the motion vector (0,0) is called the search center. The errors (MSE or MAE) of three candidate blocks in the horizontal direction ((-S,0), (0,0), (S,0) where S is the distance between the search center and the adjacent candidate block) are calculated. The candidate block

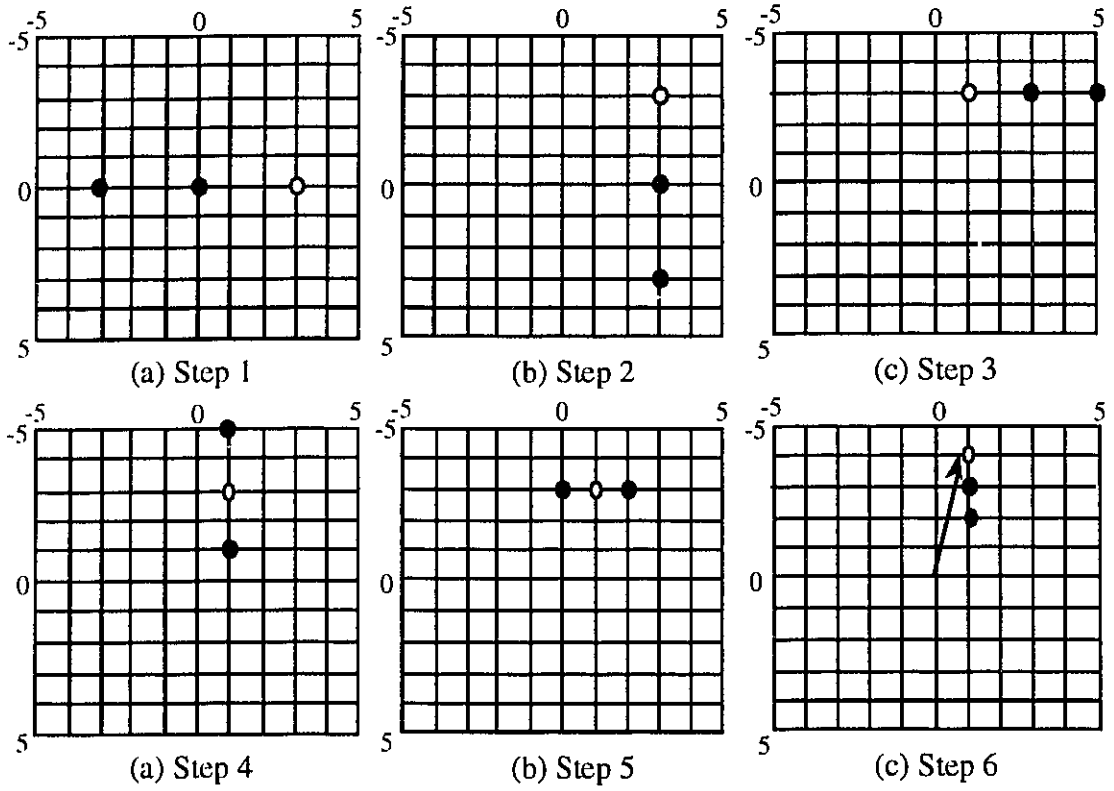



**Figure 2.4: Illustration of Fast Algorithm - OS**

with the smallest error becomes the search center in the next step ((S,0) in our example). This is followed by the calculations of the errors of three candidate blocks in the vertical direction ((S,-S), (S,0), (S,S)). Once again, the candidate block with the smallest error becomes the search center in the next step ((S,-S) in our example). Then the distance between the candidate blocks is reduced. This procedure is repeated until the distance between the candidate blocks is one pixel. In algorithmic form, it can be written as follows:

**Initialization:**
- $S = [p/2]$; ([x] denotes the smallest integer that is greater than x)
- $\Delta x = 0, \Delta y = 0$;
- Stop = False;

**Step 1:** While (S > 0) and (Stop = False)

• (Δx, Δy) = the location of min (MAE(Δx-S, Δy), MAE(Δx, Δy), MAE(Δx+S, Δy));

• (Δx, Δy) = the location of min (MAE(Δx, Δy-S), MAE(Δx, Δy), MAE(Δx, Δy+S));

• if (S = 1) then Stop = True;

else S = [S/2];

end of while loop

**Step 2:** Motion Vector (V) = (Δx, Δy).

We note that the OS algorithm involves sequential search steps, i.e. the set of candidate blocks from the previous step is used as an initial set for the current step. For example, it requires 6 search steps and 13 search points (candidate blocks) for a search area size of $p = 7$ and $n = 16$. This corresponds to 3328 (i.e. $13 \times 16^2$) accumulations, subtractions and magnitude operations to be executed.

Another example of a fast algorithm is the parallel hierarchical one-dimensional search (PHODS) algorithm [28]. In PHODS, the two-dimensional motion vector for each reference block is represented as two one-dimensional motion vectors (horizontal and vertical directions) which is located independently (in parallel) along the two directions within the search area. The search procedure of this algorithm for $p = 5$ is illustrated in Figure 2.5. In each search step, the errors of

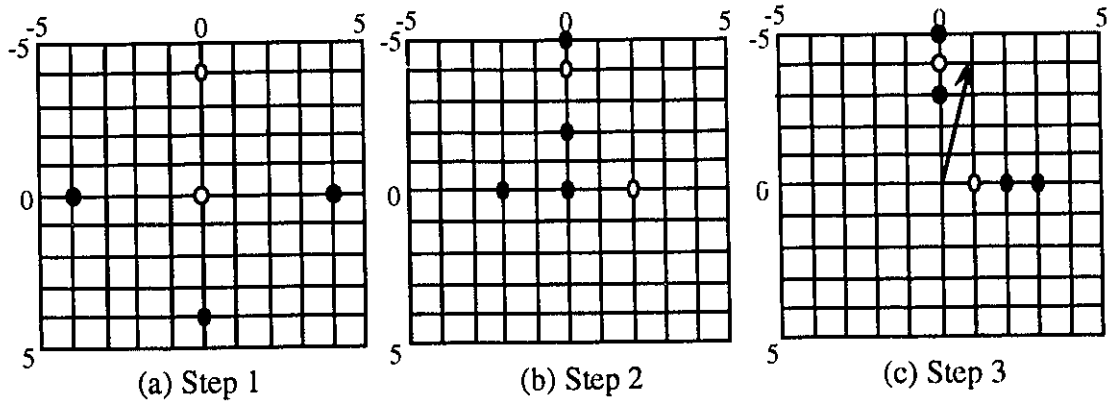



**Figure 2.5: Illustration of Fast Algorithm - PHODS**

three candidate blocks are computed (the candidate block in the middle position is called the search center). The candidate block with the smallest error becomes the search center in the next search step and the distance between the candidate blocks is also reduced. Proceeding in this manner, the optimum position in each direction is found when the distance between two candidate blocks is one pixel. In algorithmic form, it can be written as follows:

**Initialization:** • $S = 2^{\log_2 p}$

• $\Delta x = 0, \Delta y = 0;$

**Step 1:** While ($S > 0$)

PAR DO

• *x axis*: ($\Delta x$, 0) = the location of min (MAE($\Delta x$-S, 0), MAE($\Delta x$, 0), MAE($\Delta x$+S, 0));

• *y axis*: (0, $\Delta y$) = the location of min (MAE(0, $\Delta y$-S), MAE(0, $\Delta y$), MAE(0, $\Delta y$+S));

END PAR DO

• $S = S/2;$

end of while loop

**Step 2:** Motion Vector (V) = ($\Delta x$, $\Delta y$).

**Note**: PAR DO refers to the parallel execution of the statements

We note that the PHODS algorithm requires 3 sequential search steps and 13 search points (candidate blocks) for the same example with $p = 7$ and $n = 16$. This also corresponds to 3328 accumulations, subtractions and magnitude operations to be executed. However, we note that the PHODS algorithm determines the motion vectors in the horizontal and vertical directions in parallel. In other words, PHODS reduces the search time by 50% compared to OS.

Although the fast algorithms may result in a reduced number of candidate blocks to be searched, it may lead to a motion vector which converges to a local optimum rather than a global optimum. This reduces the accuracy of the estimated motion vectors. Hence the performance of the fast algorithms are generally poorer compared to the FSA.

## 2.4 Layered Structure Algorithms

In order to alleviate the local optimum problem, Chun *et al.* [29] have proposed a layered structure algorithm. In this algorithm, each candidate block is organized into a mean pyramid structure [57], [58]. For example, a candidate block A (size $n \times n$ with $n = 2^p$, $p \geq 0$) is written as,

$$A = \{A(i,j);\ i = 0, 1, ....., n-1;\ j = 0, 1, ....., n-1\} \qquad \text{--- (2.9).}$$

The corresponding p-level mean pyramid $\{A_k\}$ is formed by successively averaging $2 \times 2$ neighboring pixels as shown in Figure 2.6. For example, a $4 \times 4$ candidate block can be organized as a 3 level pyramid. The block sizes of levels 2, 1 and 0 are $4 \times 4$, $2 \times 2$ and 1 pixel, respectively. In algorithmic form, the mean pyramid decomposition can be written as:

**Step 1. Bottom Level:** • let k = p and level k be the original data, i.e. $A_k = A$;

**Step 2. Level k-1:** • level k-1 is obtained by calculating the mean for each spatially contiguous, non-overlapping block of 2 x 2 at level k,

$$A_{k-1}(i,j) = \frac{A_k(2i,2j) + A_k(2i,2j+1) + A_k(2i+1,2j) + A_k(2i+1,2j+1)}{4}$$

$$where\ i = 0,1,.....,n/2^{p-k+1} - 1$$

$$and\ j = 0,1,.....,n/2^{p-k+1} - 1$$

**Step 3. Termination:** • k = k - 1;

• if $k \neq 0$ then goto Step 2

else Stop

In the first step of the algorithm, each candidate block is organized into a mean pyramid structure. The FSA-MAE algorithm is executed at the top level (level 0) of the pyramids. A candidate block is assigned to a candidate set of motion vectors if its error (MAE) is smaller than a prespecified threshold. In the next step, the search is performed on level 1 of the pyramid and the corresponding errors of the blocks in the candidate set obtained from the first step are calculated. We note that the number of blocks in the candidate set varies according to the search step. This

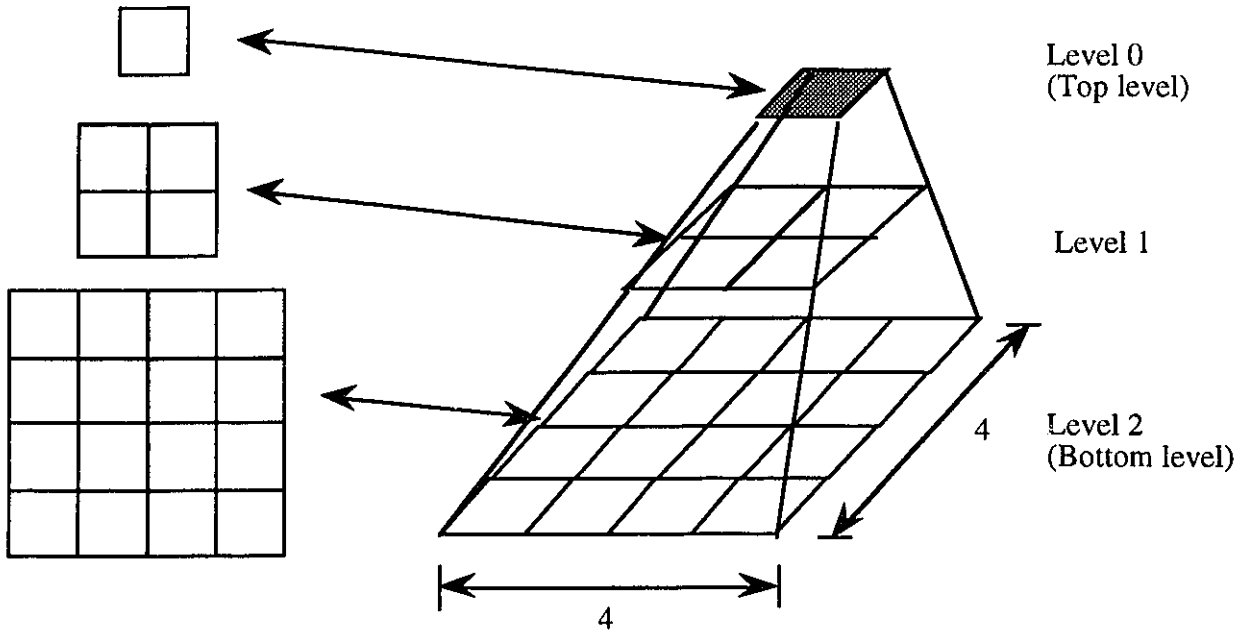



**Figure 2.6: Mean Pyramid Decomposition of a 4 x 4 Block**

process is repeated until the bottom level (level 2 in Figure 2.6) of the pyramids are searched, i.e. the motion vector is determined. An example to illustrate the search procedure for $p = 5$ is given in Figure 2.7. In algorithmic form, it can be written as follows:

**Initialization:**
- Candidate Set C1 = all possible candidate blocks within the search area
- Arrange the reference block and C1 into a set of p-level pyramid structures;
- S = 1;

**Step 1:** While (S ≤ p)
- Apply FSA-MAE on the pyramid level #S of C1;
- Select Candidate Set C1 = all candidate blocks within a prespecified threshold;
- S = S + 1;

end of while loop

**Step 2:** Motion Vector (V) is determined.

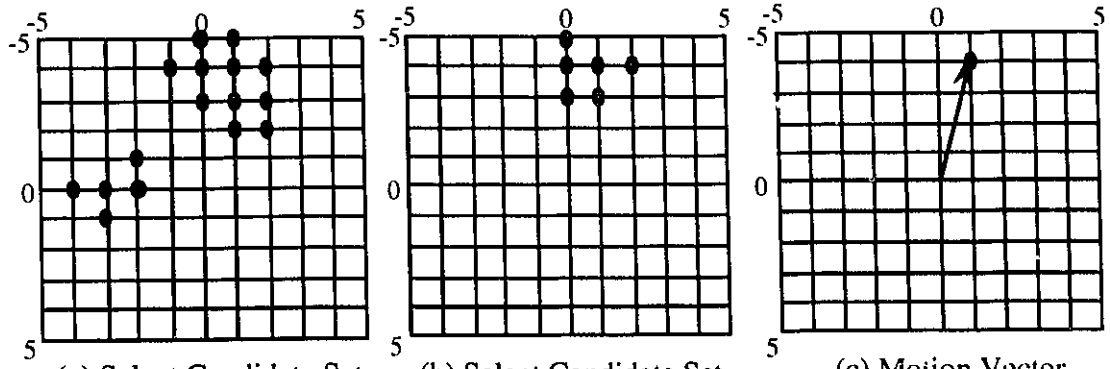



**Figure 2.7: Illustration of Layered Structure Algorithm**

Simulations reported by Chun *et al.* [29] show that the performance of this algorithm is comparable to the FSA-MAE. However, this algorithm requires extensive amount of data preparation since each candidate block has to be organized as a pyramid structure. Hence it is computationally expensive.

## 2.5 Inter-block Motion Field Prediction Algorithms

An alternative approach to improve the search efficiency is to incorporate the prediction of the inter-block motion information [30], [31], [32] in the search procedure. Since many blocks in a large homogeneous area of a frame are likely to move in the same direction with similar velocities, the motion vector information between two adjacent reference blocks are highly correlated. Hence it is possible to predict the motion vector of a reference block from the motion vectors of the adjacent blocks. This approach is illustrated in Figure 2.8 where a $2n \times 2n$ block is divided into four $n \times n$ blocks (A, a1, a2 and a3). The FSA algorithm is first applied on blocks A, B, C and D. Block a1 is assigned the motion vector of either block A or B depending on whether the motion vector of A or B results in a better match. Likewise, block a2 is assigned the motion vector of either block A or C; and block a3 is assigned the motion vector of either A, B, C or D. Hence the total number of operations is approximately 25% of the conventional FSA algorithm.

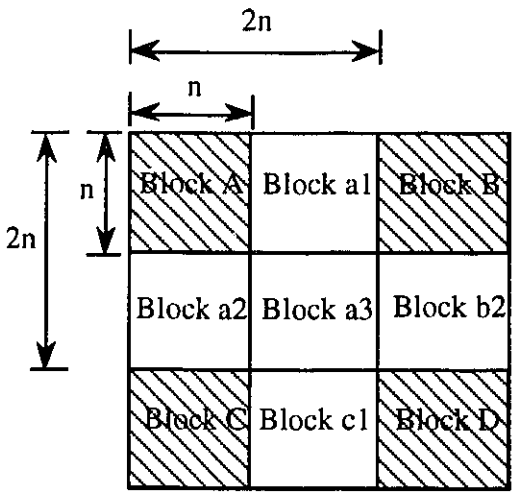



**Figure 2.8: Illustration of Inter-Block Motion Prediction Algorithms**

## 2.6 Transform Domain Motion Estimation

### 2.6.1 Frequency Domain Cross Correlation Based Motion Estimation

We recall that block matching algorithms divide a video frame into a set of nonoverlapping reference blocks of size $n \times n$ pixels and estimate a motion vector for each block. In other words, each motion vector is calculated without reference to any image information outside the nonoverlapping reference block. This may result into block edge discontinuities causing disturbing artifacts in the reconstructed frame. An alternative approach for motion estimation is the transform domain approach where cross correlation functions are executed via the frequency domain [16], [66]. In frequency domain algorithms (FDA), the motion vectors are calculated using overlapped, windowed regions of the transform coefficients of the image data. This reduces the block edge discontinuities resulting in a smoother motion field. We note that this requires the use of overlapped transforms such as lapped orthogonal transform (LOT), complex lapped transform (CLT), etc. where $2n$ data samples are mapped into $n$ transform coefficients [74]. Recently, Young *et al.* [16] have proposed a frequency domain algorithm based on complex lapped transform (CLT). Here, a motion vector for each $n \times n$ block is calculated using information from a larger block of size $2n \times 2n$ pixels, centered on the $n \times n$ block. The CLT is executed on the $2n \times 2n$ block resulting in $n \times n$ transform coefficients. The motion vector is then estimated using the cross

correlation function via these $n \times n$ transform coefficients. However, we note that CLT involves complex exponential functions which is computationally expensive resulting in a complex codec design.

### 2.6.2 Wavelet Transform Based Motion Estimation

We recall that motion compensation is used as an efficient temporal prediction in many video coding schemes. However, the residual video signal (i. e. the difference between the motion compensated previous frame $(t-1)$ and the current frame $(t)$) tends to be highly nonstationary [11] after motion compensation. Hence it is necessary to use a small block size to reduce the nonstationarity of each block. However, this increases the number of reference blocks in a frame resulting in a computationally expensive motion estimation algorithm. Recently, wavelet transform has emerged as a promising technique for image processing applications due to its flexibility in representing nonstationary video signals. The superior performance of wavelet transform compared to the DCT for video compression has been reported in [12], [13]. This is a result of the absence of blocking effects and mosquito noise in wavelet transform coding [14]. We note that the improved quality of reconstructed images is desirable for very low bit rate applications.

#### Wavelet Transform

The basic concept of one-dimensional wavelet transform is to represent any arbitrary function as a superposition of basic functions $P$'s (wavelets). Here, a function $P$ and its associated scaling function $Q$ are defined such that the family of functions $\{P^J(x)\}$, $J \in Z$ where $P^J(x) = (2^J)^{1/2} P(2^J x)$ is orthonormal [13]. The wavelet transform can be realized by using quadrature mirror filters [69], [75] as shown in Figure 2.9 where $h(n) = 1/2\langle Q(x/2), Q(x-n)\rangle$ and $g(n) = (-1)^n h(1-n)$. We note that $h(n)$ and $g(n)$ correspond to the impulse response of a low pass filter and a high pass filter, respectively. The reconstruction filters have impulse response $h^*(n) = h(1-n)$ and $g^*(n) = g(1-n)$.

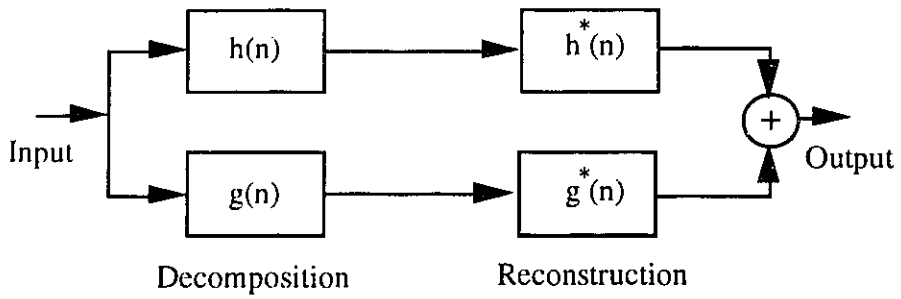



**Figure 2.9: Wavelet Decomposition and Reconstruction**

In image processing applications, a two-dimensional separable wavelet decomposition is implemented independently, first in the horizontal direction and then in the vertical direction. The filter outputs are subsampled by a factor of 2. This results in orientation selective high pass subimages and a low pass subimage. This process is then repeated on the low pass subimage to form the next level of the wavelet (pyramid) decomposition. In other words, wavelet transform decomposes an image into a pyramid structure of subimages with various resolutions corresponding to the different frequency bands [34], [35], [36]. Each level of the wavelet pyramid consists of three wavelet components, namely, the horizontal component $W^H$, the vertical component $W^V$, and the diagonal component $W^D$. As an example, a 3-level wavelet representation of an image $S_1$ of size $a \times b$ pixels and the corresponding pyramid structure are shown in Figures 2.10 and 2.11, respectively. A total of 9 orientation subimages and 1 low pass

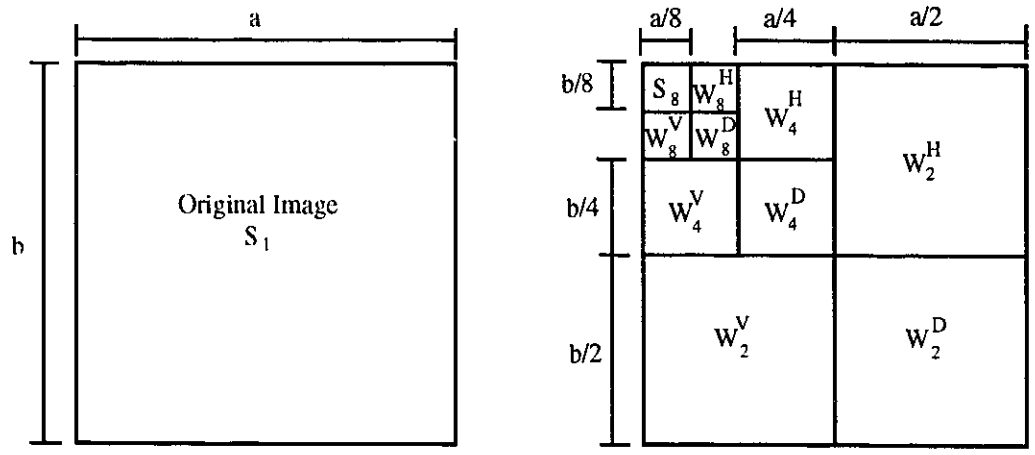



**Figure 2.10: Wavelet Transformed Image**

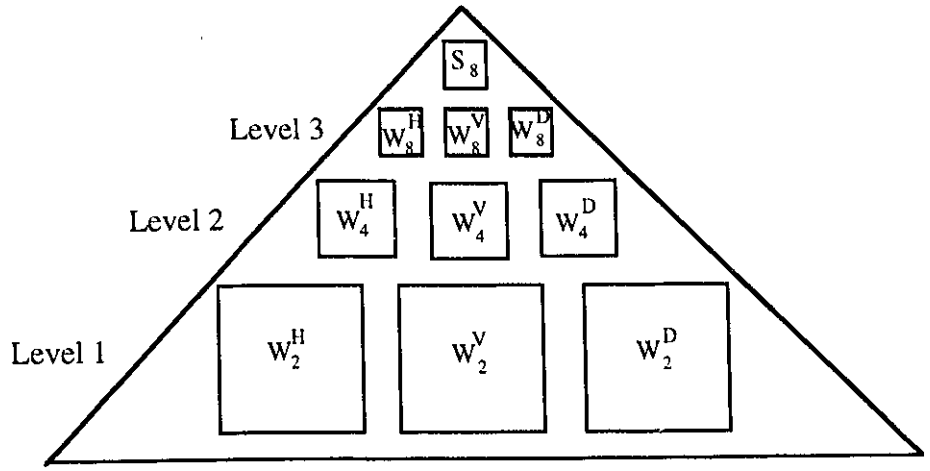



**Figure 2.11: Wavelet Pyramid**

subimage ( $S_8$ of size $(a/8)\times(b/8)$ pixels) are obtained. The inverse wavelet transform of the low pass subimage $S_8$ and wavelets $W_8^H$, $W_8^V$, $W_8^D$ results in a reconstructed subimage $S_4$ of size $(a/4)\times(b/4)$ pixels. Similarly, the inverse transform of $S_4$ and $W_4^H$, $W_4^V$, $W_4^D$ results in a reconstructed subimage $S_2$ of size $(a/2)\times(b/2)$ pixels while the original image $S_1$ is reconstructed by inverse transforming $S_2$ and $W_2^H$, $W_2^V$, $W_2^D$. Hence, the wavelet representation provides a multiresolution/ multifrequency representation of a signal with localization in both time and frequency domains [34]. This property of the wavelet transform makes a nonstationary image signal easier to code since the signal is decomposed into a set of multiscaled wavelets where each component becomes relatively more stationary.

### Multiresolution Motion Estimation for Wavelet Transform Based Coding

Recently, a multiresolution motion estimation scheme (MRME) [11], [15] has been reported for wavelet transform based video compression. This approach exploits the cross-correlations among each level of the wavelet pyramid structure in order to reduce the computational complexity of the motion estimation. In the MRME scheme, the motion vectors at the highest level of the wavelet pyramid are first estimated using the conventional block matching based motion

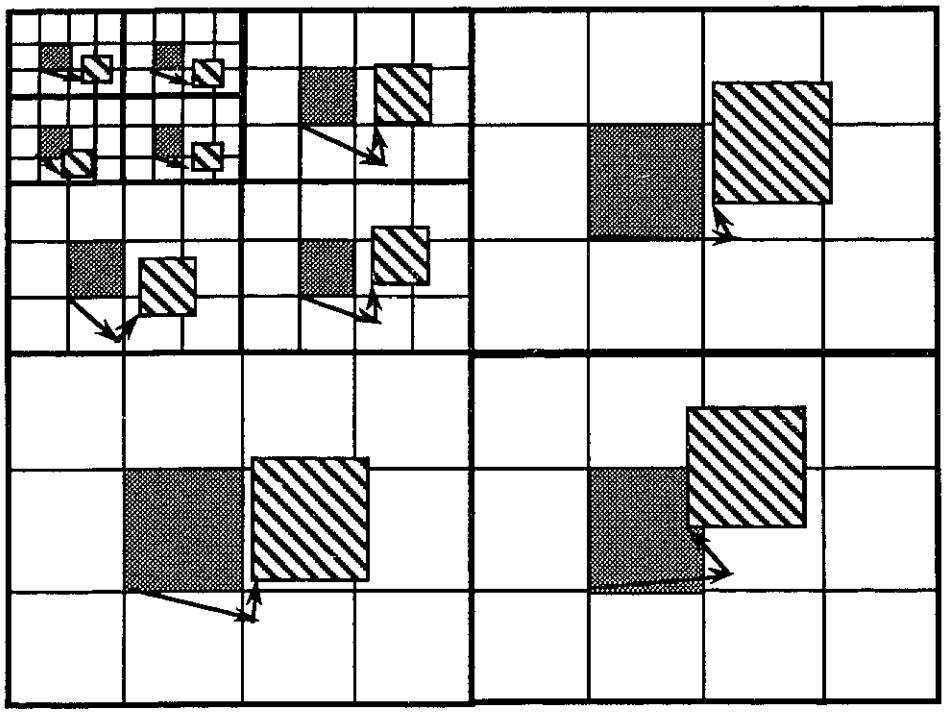


**Figure 2.12: Multiresolution Motion Estimation (MRME)**

estimation algorithm. The motion vectors at the next level of the wavelet pyramid are predicted from the motion vectors of the preceding level and are refined at each step. For example, the motion vectors in $W_4^H$, $W_4^V$, $W_4^D$ are predicted from the motion vectors in $W_8^H$, $W_8^V$, $W_8^D$, respectively. Likewise, motion vectors in $W_2^H$, $W_2^V$, $W_2^D$ are predicted from the motion vectors in $W_4^H$, $W_4^V$, $W_4^D$. This process is illustrated in Figure 2.12. We note that in [11], [15], motion vectors are estimated separately for each wavelet subimage at each level of the wavelet pyramid. In other words, the motion vectors in each subimage at each pyramid level (e.g. $W_4^H$, $W_4^V$, $W_4^D$ in level 2) are estimated independently.

## 2.7 Video Compression Schemes

We recall that there exists two kinds of redundancies in a video signal, namely, temporal and spatial. In this section, we review two video compression techniques (which exploit the

spatial and temporal redundancies), namely, interframe hybrid coding [5] and motion compensated wavelet transform coding [15].

### 2.7.1 Interframe Hybrid Coding

Interframe hybrid coding is a combination of transform and interframe prediction coding [6]. Typically, each frame is partitioned into two-dimensional non-overlapping blocks. Block matching algorithm is then executed on these blocks to obtain a set of motion vectors as discussed in section 2.2. The prediction error (i.e. the difference between the current frame and the motion compensated previous frame) is then transform coded and transmitted to the receiver along with the set of motion vectors. The block diagram of the coder is shown in Figure 2.13. The motion estimation module in the coder is discussed in section 2.1. The other three major modules are the discrete cosine transform, quantization and variable length coding modules. We now present the details of these modules.

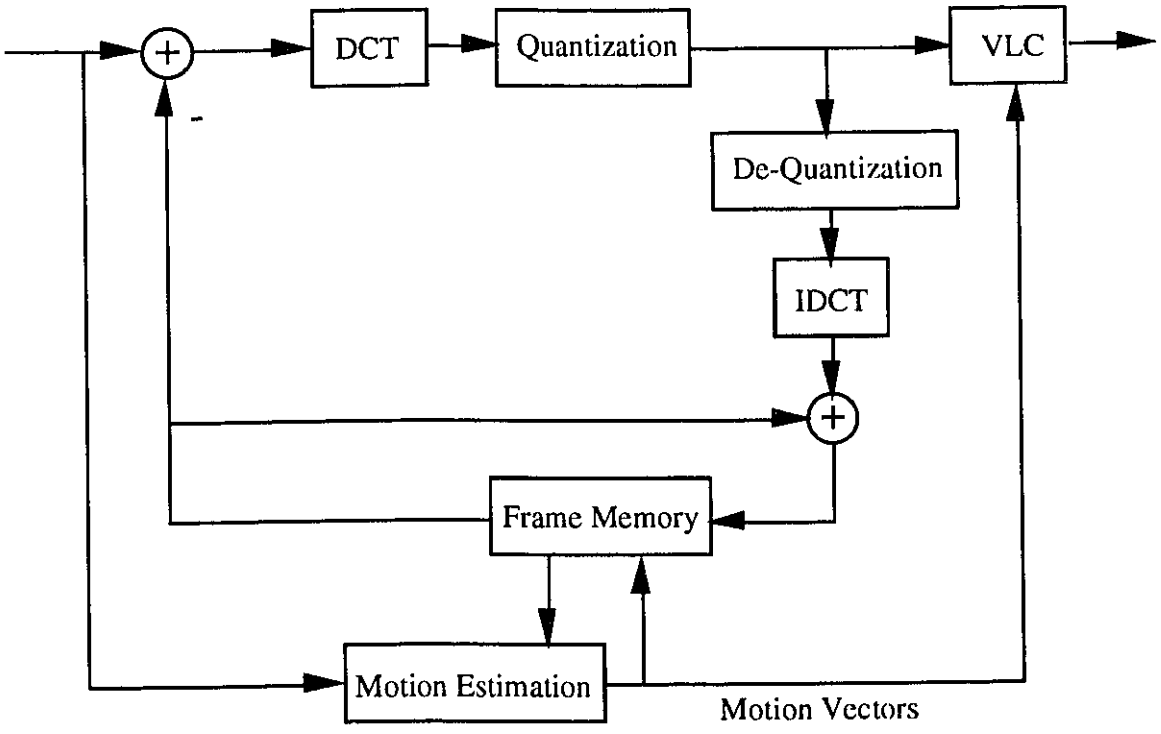



**Figure 2.13: Interframe Hybrid Coder**

**Transform Coding**

The fundamental concept of transform coding is to decorrelate the image data via an orthogonal transform. An optimum transform should result in statistically independent coefficients [60]. Clark [61] has shown that the Karhunen Loève transform (KLT) [71] is an optimum transform in terms of mean square error. However, it requires a large number of operations to be computed. Hence it is usually replaced by suboptimal transforms [62] such as discrete cosine transform (DCT). Rao *et al.* [7] have shown that the performance of DCT is close to KLT.

Given a $n \times n$ block (i.e. $f(x,y)$, $0 \le x,y \le n-1$), its DCT is given as follows [7]:

$$F(n_1,n_2) = c(n_1)c(n_2)\sum_{x=0}^{n-1}\sum_{y=0}^{n-1}\cos\frac{\pi n_1(2x+1)}{2n}\cos\frac{\pi n_2(2y+1)}{2n}f(x,y) \qquad \text{--- (2.10)}$$

where $c(0) = \frac{1}{\sqrt{n}}$ and $c(n_1) = c(n_2) = \sqrt{\frac{2}{n}}$ for $1 \le n_1,n_2 \le n-1$.

For a block of size $n \times n$, DCT transforms the block into $n^2$ unique two-dimensional spatial frequencies. The transform coefficient with zero frequency is called DC coefficient and remaining $n^2 - 1$ coefficients are called the AC coefficients. We note that the energy of the image is concentrated in the low frequency coefficients. The choice of block size is a trade-off between compression efficiency and image quality. A larger block size provides a better compression since more pixels are used to exploit the spatial redundancies [10]. However, this results in disturbing artifacts such as blocking effects. Experiment results shown that a block size of $8 \times 8$ or $16 \times 16$ is a good compromise [10]. The block can be reconstructed by executing the inverse DCT (IDCT) operation which is defined as follows:

$$f(x,y) = \sum_{n_1=0}^{n-1}\sum_{n_2=0}^{n-1}c(n_1)c(n_2)\cos\frac{\pi n_1(2x+1)}{2n}\cos\frac{\pi n_2(2y+1)}{2n}F(n_1,n_2) \qquad \text{--- (2.11).}$$

### Quantization

Compression of an image is achieved by quantizing the DCT coefficients using a predefined quantization table. Quantized coefficients are obtained by dividing each DCT coefficient by the quantizer step size. We note that the quantizer step size may be different for each DCT coefficient. Since the human visual system is not sensitive to degradations at high frequencies [10], a large quantizer step size can be used for high frequencies. This achieves further compression without degrading the subjective visual quality. The quantized coefficients are then zig-zag scanned as shown in Figure 2.14 into a one-dimensional symbol sequence. Finally, the symbol sequence is entropy coded and transmitted to the receiver.

### Variable Length Coding

In entropy coding (lossless coding), the redundancies that exist in the symbol sequence are exploited in such a way that the process is reversible, i.e. the symbol sequence can be recovered perfectly. A typical entropy coding technique is Huffman coding which assigns longer codewords to symbols with lower probabilities of occurrence [63], [64]. We illustrate the Huffman coding procedure in Figure 2.15. Here, we assume that a symbol sequence (source) to be coded consists

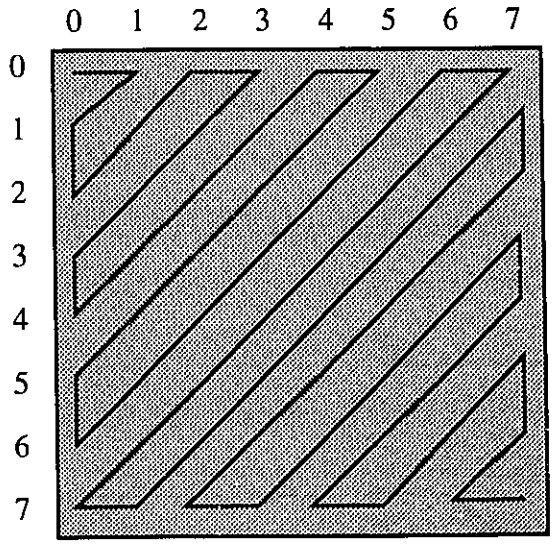



**Figure 2.14: The Zig-Zag Scan Path of the DCT Coefficients**

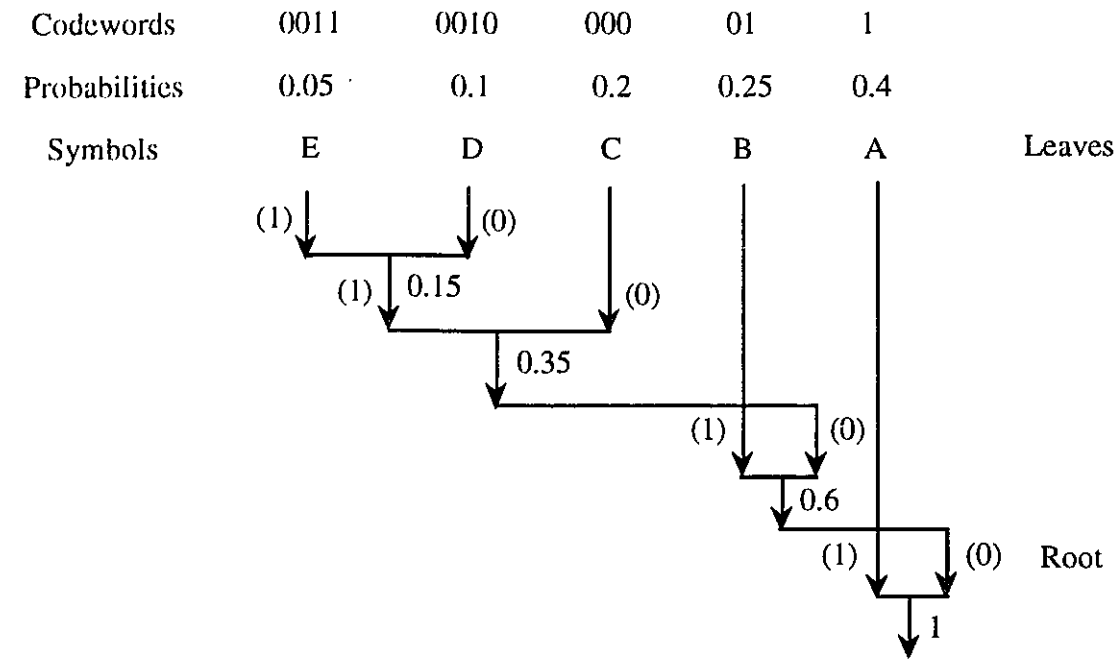



**Figure 2.15:Huffman Coding - An Example**

of five letters, namely, A, B, C, D and E with probabilities 0.4, 0.25, 0.2, 0.1 and 0.05 respectively. First, the probabilities of the source are arranged in decreasing order (0.4, 0.25, 0.2, 0.1, 0.05). Next, the two smallest probabilities (0.1, 0.05) are combined to obtain a new single value (0.15), resulting in a reduced set of probabilities. Once again, this set of probabilities is re-organized in the decreasing order (0.4, 0.25, 0.2, 0.15) and the two smallest probabilities (0.2, 0.15) are combined to yield a single value (0.35). This process is repeated until a single value (which is unity) is reached resulting in a binary tree (Huffman tree). Each node of the binary tree is then assigned a binary number "1" and "0" to the left and right member, respectively. The codeword for each letter is the sequence of 0's and 1's obtained by tracing the path from the root to the leaf of the tree. For example, the codewords for letter A and E are 1 and 0011, respectively. We note that the Huffman coding procedure results in a variable length code (VLC).

### 2.7.2 Motion Compensated Wavelet Transform Coding

Motion compensated wavelet transform coding is a combination of wavelet transform and interframe prediction coding. The block diagram of the coder is shown in Figure 2.16. This coder consists of four major modules, namely, wavelet transform, multiresolution motion estimation, quantization and variable length coding. The wavelet transform, multiresolution motion estimation and variable length coding modules are discussed in section 2.6.1, 2.6.2 and 2.7.1, respectively. We note that the wavelet decomposition is executed before the motion estimation/ compensation process. Hence the wavelet transform exploits both the spatial and temporal redundancies in a video sequence. We now present the details of the quantization module.

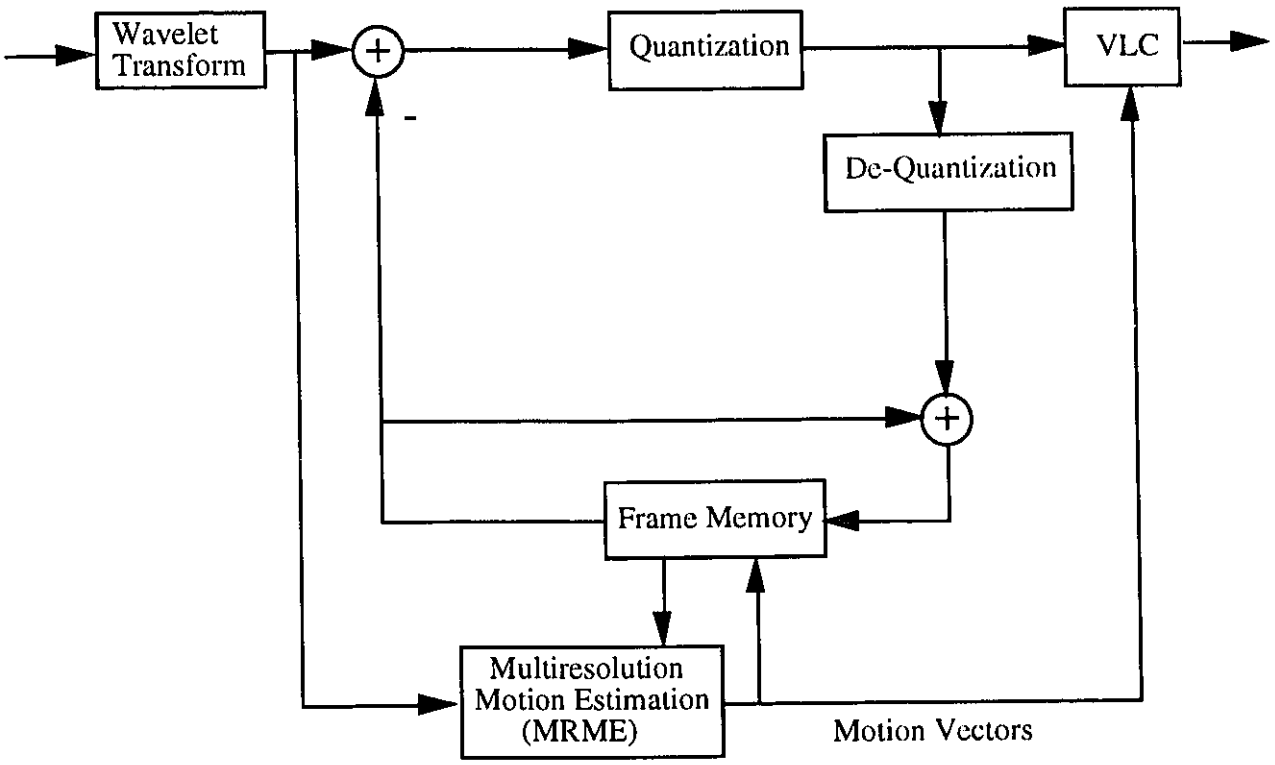



**Figure 2.16: Motion Compensated Wavelet Transform Coder**

### Quantization

We recall that compression of an image is achieved by the quantization of the transform coefficients. In wavelet transform coding, the adaptive truncation scheme [65] is typically used. This scheme involves a simple floating-to-integer conversion and consists of three steps. In the first step, the transform coefficients ($W$) of all subimages are subtracted by a threshold $T$, resulting in a smaller number of coefficients ($TW$) to be quantized. This process is defined as follows:

$$\begin{aligned} TW_m^k(i,j) &= W_m^k(i,j) - T \qquad if\ W_m^k(i,j) > T \\ &= 0 \qquad\qquad\qquad if\ W_m^k(i,j) \le T \\ for\ 0 \le i \le \frac{N}{2^{m-1}}\ and\ 0 \le j \le \frac{M}{2^{m-1}} & \end{aligned} \qquad \text{--- (2.12)}$$

where $N$ and $M$ are the video frame size and m is the wavelet pyramid level. The next step is to scale the coefficients by a quantizer step size $D_m$, resulting in a set of normalized coefficients ($NTW$) , i.e.

$$NTW_m^k(i,j) = \frac{TW_m^k(i,j)}{D_m} \qquad \text{--- (2.13).}$$

We note that the quantization threshold and quantizer step size may be different for each level of the wavelet pyramid. For example, the low pass subimage ($S_8$ in Figure 2.10) may have a smaller quantization threshold and quantizer step size compared to the high pass subimages ($W_2^H$, $W_2^V$, $W_2^D$ in Figure 2.10). This is because the low pass subimage contains higher portion of energy of an image. The normalized coefficients are then rounded to the next higher integer values to obtain the quantized coefficients. Finally, the quantized coefficients are converted into a one-dimensional symbol sequence which is entropy coded and transmitted to the receiver as discussed in section 2.7.1.

## 2.8 Video Compression Standards

Recently, the Consultative Committee on International Telephony and Telegraphy (CCITT) and the Motion Pictures Experts Group (MPEG) of the International Standards Organization (ISO) have proposed standards for video compression known as the H.261 and MPEG-1 standards,

respectively [9], [10], [67]. The H.261 standard is aimed at videophone and videoconferencing applications, where real-time encoding and decoding are essential. It is often called the p*64 standard since the targeting bit rate is $p \times 64$ kbit/sec where $p$ is in the range of 1 to 30 [67]. The MPEG-1 standard was originally developed for the storage applications (CD-ROM, etc.) at a bit rate below 1.5 Mbit/sec [9]. The MPEG committee of the ISO is currently drafting the MPEG-2 standard which is targeted for broader applications such as multimedia communications with bit rate above 3 Mbit/sec [76]. We review the structure of the MPEG-1 coder as it is used as a baseline for comparison with our simulation results in this thesis.

The block diagram of an MPEG-1 coder is shown in Figure 2.17. We note that this is essentially an interframe hybrid coder (discussed in section 2.7.1) which employs motion compensated prediction and interpolation followed by discrete cosine transform (DCT) coding. Here, the video sequence is organized into a set of group of pictures (GOP). Each GOP approach is a combination of one (typically) I (intra), one or two P (predicted) and the rest of B (bi-directional interpolated) frames as shown in Figure 2.18. This contrasts with the simple interframe hybrid coder where only the first frame of a video sequence is intra-frame coded and the rest are inter-framed coded (predicted). We note that in each GOP, the I and P frames are divided into blocks of 8 x 8 pixels where the DCT is performed. In other words, only the I and P frames are

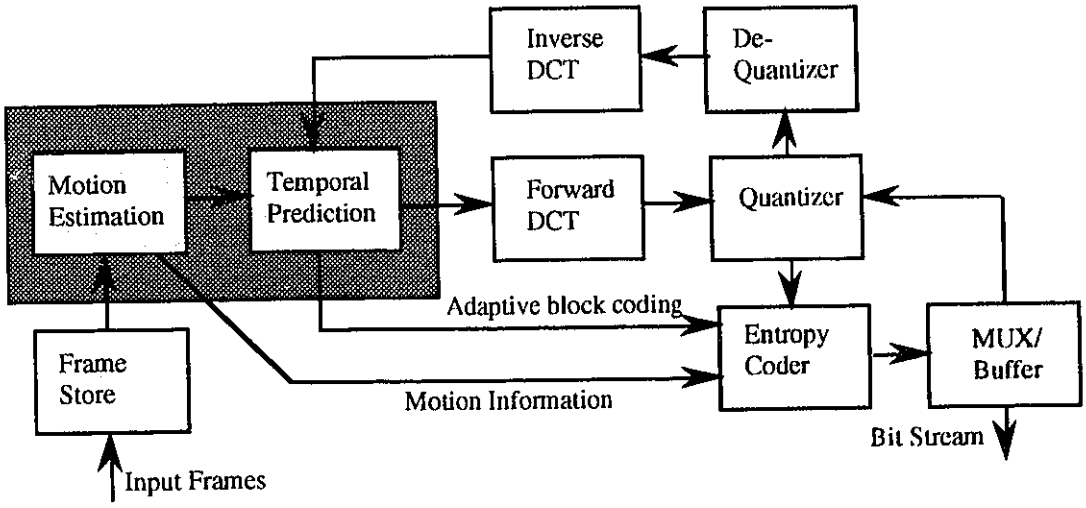



**Figure 2.17: MPEG-1 Video Coder**

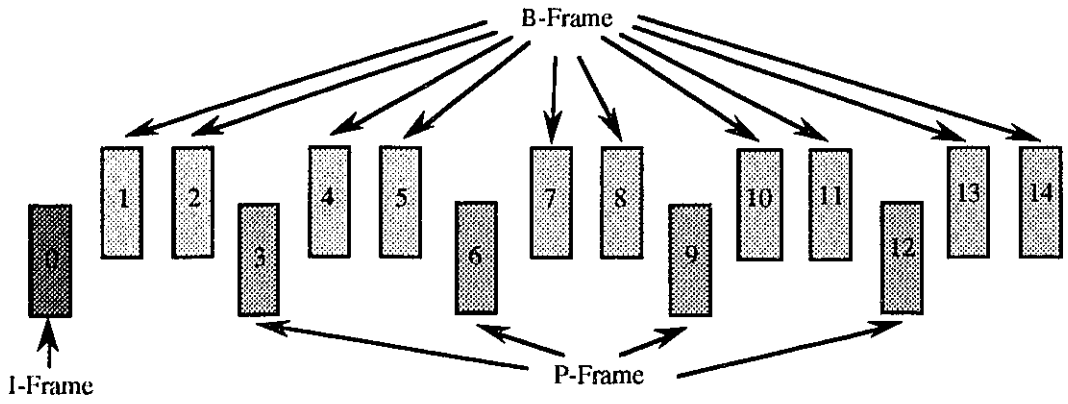



**Figure 2.18: MPEG-1 Group of Pictures (GOP)**

DCT coded. The resulting coefficients are zig-zag scanned and run-length/huffman coded to generate a bit-stream according to the syntax provided in the MPEG-1 draft. The B frames are used to achieve higher data compression and are motion compensation interpolated at the receiver from the already transmitted I and P frames. We note that the MPEG-1 standard specifies that the representation of the motion information is based on square subblocks (16 x 16 pixels) of a frame. Due to the block-based motion representation, block matching techniques are typically used for motion estimation in MPEG-1.

We note that MPEG-1 standard employs bidirectional motion estimation. Here, the previous frame $(t-1)$ and the next frame $(t+1)$ are used in determining the motion vector for each reference block. The motion vector corresponding to the frame ($(t-1)$ or $(t+1)$) which results in the smallest matching error is transmitted to the receiver. A bit plane is used to indicate to the receiver which frame ($(t-1)$ or $(t+1)$) results in the smallest matching error for all the reference blocks in the current frame. We note that bidirectional motion estimation deals with the problems of covered and uncovered areas resulting in an improved performance compared to unidirectional motion estimation [9]. For example, an uncovered area (e.g. scene change) may not be predictable from the previous frame $(t-1)$, but can however be predicted from the next fame $(t+1)$.

## 2.9 Summary

In this chapter, we have presented an overview of the basic techniques for motion estimation. We first discussed the concepts of block based motion estimation algorithms. The algorithms have been classified into two categories, namely, spatial and transform domain algorithms. Spatial domain algorithms such as full search algorithm, fast algorithms, layered structure algorithms and inter-block motion field prediction algorithms were reviewed. This followed with a review of transform domain algorithms. The motivation for the transform domain algorithms was identified. We presented the concepts of the complex lapped transform and wavelet transform based motion estimation algorithms. This followed with a review of video compression techniques such as interframe hybrid coding and motion compensated wavelet transform coding. Finally, we presented the details the MPEG-1 video compression standard.

# Chapter 3

# VLSI Architecture for Full Search Algorithm

We recall from chapter 2 that block matching techniques are widely used for motion estimation because of their simplicity. Recently, various block matching algorithms (e.g. full search algorithm (FSA) [6] and fast algorithms [23-28]) have been reported in the literature. We recall that FSA has a high computational complexity since all possible candidate blocks are searched for a match. On the other hand, fast algorithms reduce the computational burden by facilitating searching on a smaller set of candidate blocks. This however reduces the accuracy of the estimated motion vectors and also requires a complicated control structure. Hence, for hardware realization proposes, FSA is often preferred because of its regularity and lower control overhead requirements.

Recently, hardware architectures which implement FSA have been reported in the literature [37-48]. However, these architectures are either complex and/or non-cascadable. This results in an increased hardware requirement and reduced flexibility in VLSI implementation.

In this chapter, we propose a VLSI architecture which implements the FSA-MAE algorithm in real time. The proposed architecture consists of a two-dimensional structure of basic cells (BC's) where each BC is capable of computing the partial MAE. We note that the architecture is capable of executing the FSA-MSE algorithm by modifying the basic function executed by the BC's. The architecture exploits the interblock dependency and can hence meet the real time requirement in various applications. Most importantly, the architecture is simple, modular and cascadable. This makes possible VLSI implementation as a codec.

## 3.1 Review of Architectures for the Full Search Algorithm

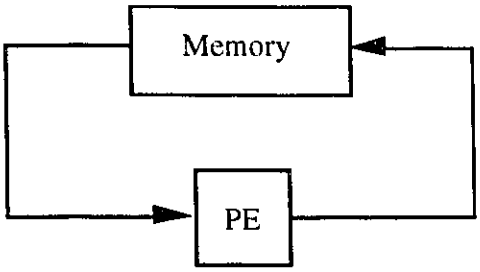
In this section, we present a review of hardware architectures recently reported in the literature for executing FSA. These architectures are classified into four categories, namely, general purpose multiprocessor systems, systolic arrays, tree architectures and content-addressable memory (CAM) architectures.

### 3.1.1 General Purpose Multiprocessor Systems

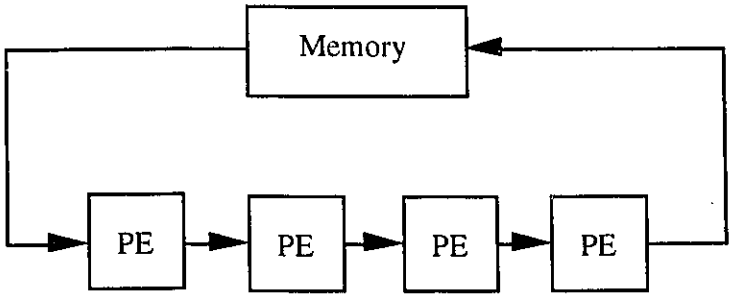
Several general purpose multiprocessor systems [37-39] have been proposed to execute FSA for low bit rate video coding applications. These systems consist of programmable processors. Data (reference block and candidate blocks) is preloaded into the local memories via the system bus and the matching task is distributed among several processors. For example, May [37] has proposed a system consisting of 8 digital signal processors (DSP) and a control processor for executing general tasks including FSA for the videophone application. Hoek [38] has proposed a chip which consists of 16 processors in a single instruction multiple data stream (SIMD) architecture where instructions are broadcast from a central control unit [49]. Tamitani *et al.* [39] have designed a real time video signal processor system which can execute the FSA. This system employs 3 multiprocessor clusters which are composed of a set of processing modules, two bus switch units and a variable delay unit. We note that these implementations have the disadvantages of high cost and sub-optimal mapping of FSA onto the set of processors [45]. In addition, communication bottlenecks as a result of multiple processors accessing the shared data resources [41] limits the overall computational rate. Hence the use of these systems is limited to a small class of applications.

### 3.1.2 Systolic Arrays

Recently, systolic arrays [50], [51] have been proposed in the literature to overcome the communication bottleneck in general purpose multiprocessor systems. The concept of systolic arrays is a general methodology for mapping high level computations onto hardware structures. Systolic arrays consist of a set of locally interconnected, pipelined processing elements (PE's) where each PE is capable of performing some simple operations. In addition, data flows from the system memory in a rhythmic fashion, passing through many PE's before returning to the memory. The basic concept of systolic arrays is illustrated in Figure 3.1. The main advantage of systolic arrays is that they provide a balance between the computation rate and the input/output




(a) Conventional Von Neumann Architecture




(b) Systolic Array

**Figure 3.1: Basic Principle of a Systolic Array**

(I/O) bandwidth. Moreover, the simplicity, regularity and modularity make systolic arrays easily implementable in VLSI.

Recently, systolic arrays implementations for executing FSA [40-45] have been proposed. These architectures differ in terms of the mapping strategy (i.e. mapping an algorithm onto systolic arrays). For example, Yang *et al.* [40] have proposed a one-dimensional semi-systolic array [50] which executes FSA on $16 \times 16$ blocks. However, the large number of off-chip memory accesses and the complexity of the address generation circuitry increases the input bandwidth and control requirements. Komarek *et al.* [41] have proposed a two-dimensional systolic architecture with local memories and shift registers in order to reduce the off-chip memory accesses. We note that this architecture requires a total number of $(4n+4)$ input/output lines (each line is either 8 or 16 bits wide) to compute a $2n \times 2n$ reference block using a cascade of four $n \times n$ chips. This results in a large pin count and is hence not an elegant solution for hardware implementation. Hsieh *et al.*

[45] have proposed a systolic architecture with serial data input to reduce the pin count. However, this architecture employs a parallel adder which causes the bottleneck.

### 3.1.3 Tree Architectures

The third class of architectures for real time implementation of FSA is based on tree structure. Recently, Jehng *et al.* [47] have proposed a tree architecture which maps FSA onto a binary tree topology [49]. This architecture uses a memory interleaving technique [52] to enhance the memory bandwidth (i.e. the number of words accessed per second). In addition, it employs a flexible technique called *tree-cut technique* which allows the execution of FSA on a $2n \times 2n$ reference block using a $n \times n$ block architecture without any additional complex control structure. However, the *tree-cut technique* requires four times more clock cycles to execute the FSA.

### 3.1.4 Content Addressable Memory (CAM) Architectures

The next class of architectures for executing FSA is based on content addressable memories (CAM's). In a CAM, the data is stored, retrieved or modified according to its content rather than its address [53], [54]. It allows parallel searching of a particular word pattern (word template) in the memory. Recently, Idris *et al.* [48], [78] have proposed a CAM architecture for executing FSA. This architecture stores all of the search area's pixels in the CAM. Each reference block's pixel is compared to all of the search area's pixels in parallel as shown in Figure 3.2. This results in a set of closest match pixels. The best match block is a candidate block where all of its pixels are closest match pixels. We note that this architecture requires all of the reference block's pixels to be compared with the search area in parallel in order to meet the real time requirement. In other words, operations A, B, C and D (shown in Figure 3.2) are required to be executed in parallel. This corresponds to an increase in the storage capacity of the CAM by a factor of $n^2$ (i.e. for a reference block of size $n \times n$). Hence this architecture is hardware intensive.

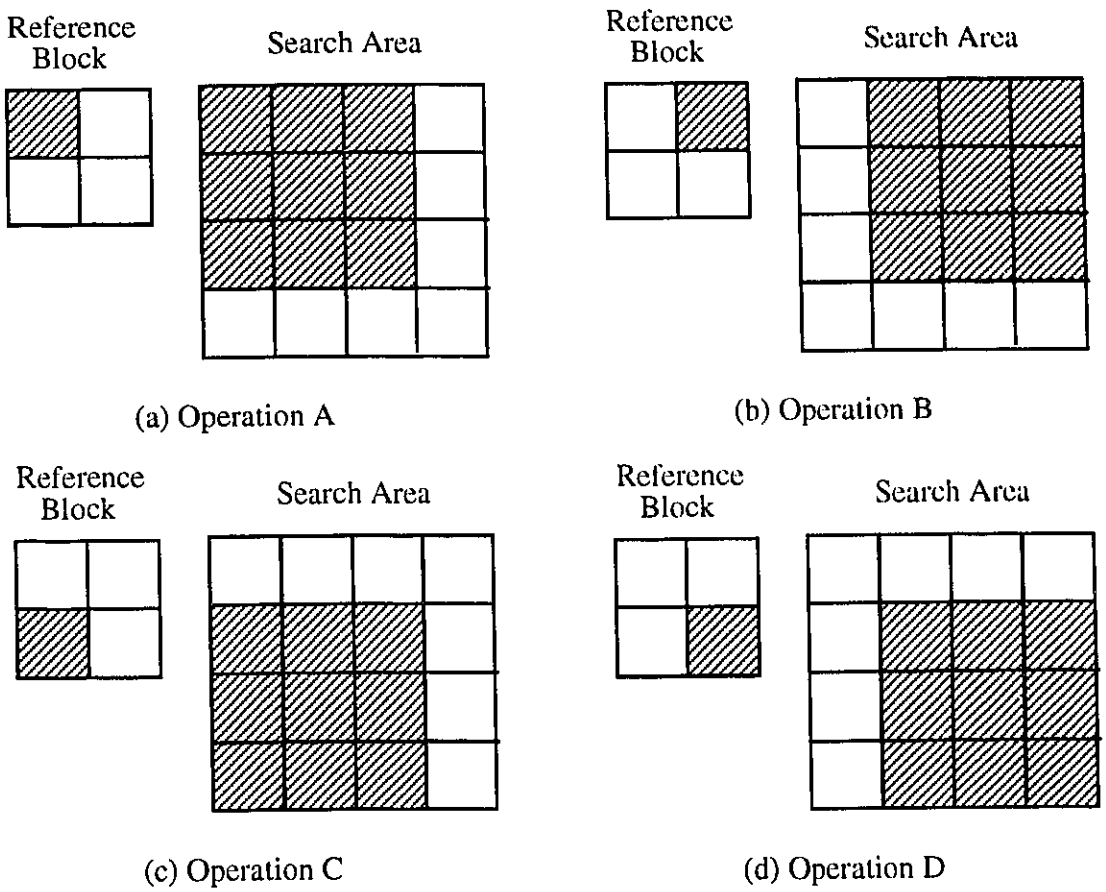



**Figure 3.2: Required Operations for Executing FSA using CAM's for Block Size n=2 and Maximum Allowed Displacement p=1**

## 3.2 VLSI Architecture for the Full Search Algorithm

In this section, we propose a VLSI architecture which implements the FSA-MAE algorithm in real-time. The architecture consists of a two-dimensional structure of basic cells (BC's) where each BC is capable of computing the MAE. We note that the architecture is capable of executing the FSA-MSE algorithm by modifying the basic function executed by the BC's. The architecture can meet the real time requirement in various applications by exploiting the interblock dependency. Most importantly, the architecture is modular and cascadable and hence makes possible computation of a $2n \times 2n$ reference block by a simple cascade of four $n \times n$ chips. The architecture can be easily implemented in VLSI as a codec.

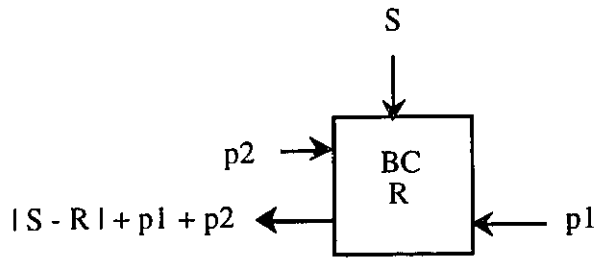



**Figure 3.3: The Basic Cell (BC)**

### 3.2.1 System Architecture

We recall from chapter 2 that the mean absolute error (MAE) is widely used for FSA because of its superior performance. The MAE equation is given as follows:

$$MAE(\Delta x,\Delta y)=\sum_{i=1}^{n}\sum_{j=1}^{n}|S(i+\Delta x,j+\Delta y)-R(i,j)| \qquad -p\le\Delta x,\Delta y\le p \qquad \text{--- (3.1)}$$

where $R(i,j)$ is a reference block's pixel in the current frame $(t)$ and $S(i+\Delta x,j+\Delta y)$ is a candidate block's pixel within a search area in the previous frame $(t-1)$.

The proposed architecture consists of a two-dimensional structure of $n\times n$ basic cells (BC's) where each BC is capable of computing the partial MAE between a reference block and a candidate block in the search area (Figure 3.3). The simplified architecture for $n=3$ and $p=2$ is shown in Figure 3.4. To start with, the reference block's pixels are pre-loaded and remain fixed in the BC's registers. A pixel of the search area is input at each clock cycle into the rightmost BC of each row (BC1, BC4 and BC7) and the corresponding partial MAE's are computed. The partial MAE outputs are shifted to the adjacent (left) BC as shown in Figure 3.4. We note that |S(1,1)-R(1,1)| + |S(1,2)-R(1,2)| + |S(1,3)-R(1,3)| is available at the output of BC3 at $t=3$ while |S(1,1)-R(1,1)| + |S(1,2)-R(1,2)| + |S(1,3)-R(1,3)| + |S(2,1)-R(2,1)| + |S(2,2)-R(2,2)| + |S(2,3)-R(2,3)| is available at the output of BC6 at $t=4$. Likewise, the MAE of the first candidate block is available at the output of BC9 at $t=5$. The sequence of operations is illustrated using the Occupancy vs Time chart [52] shown in Figures 3.5 and 3.6. In general, after an initial latency of $2n-1$ clock cycles, the MAE of the first candidate block is computed.

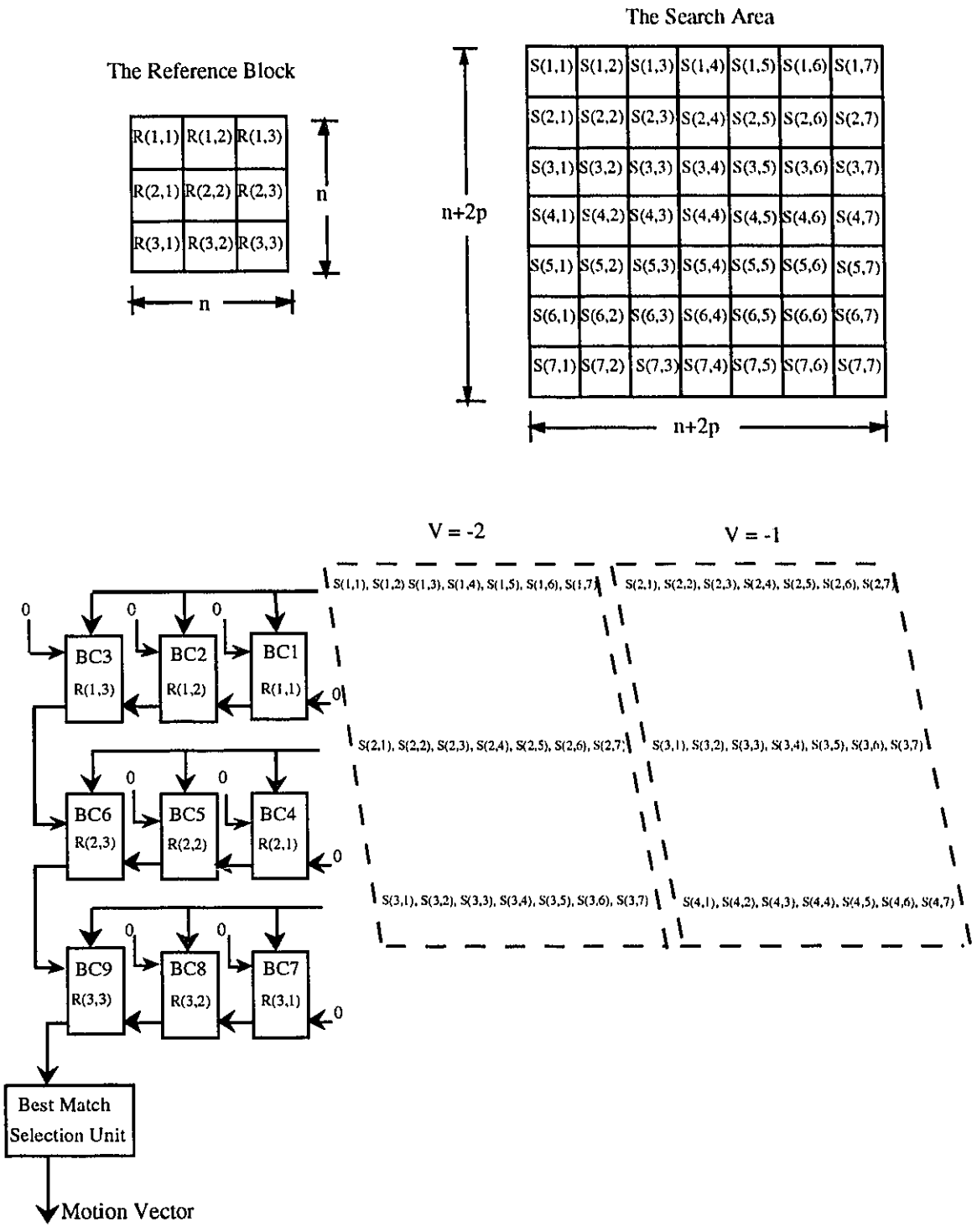



**Figure 3.4: The Simplified Architecture for n=3 and p=2**

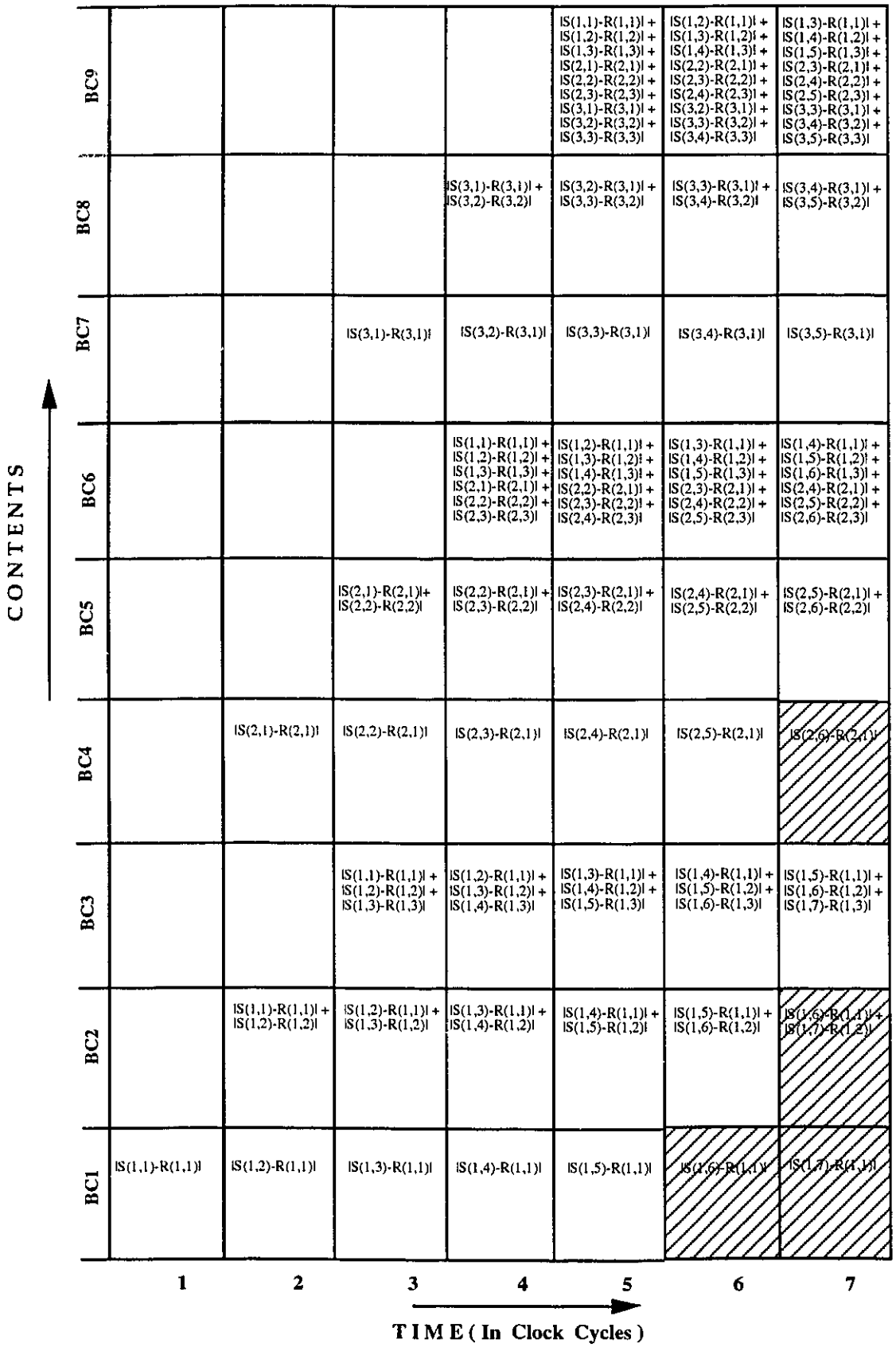



**Figure 3.5: Cell Occupancy Vs Time Chart (1)**

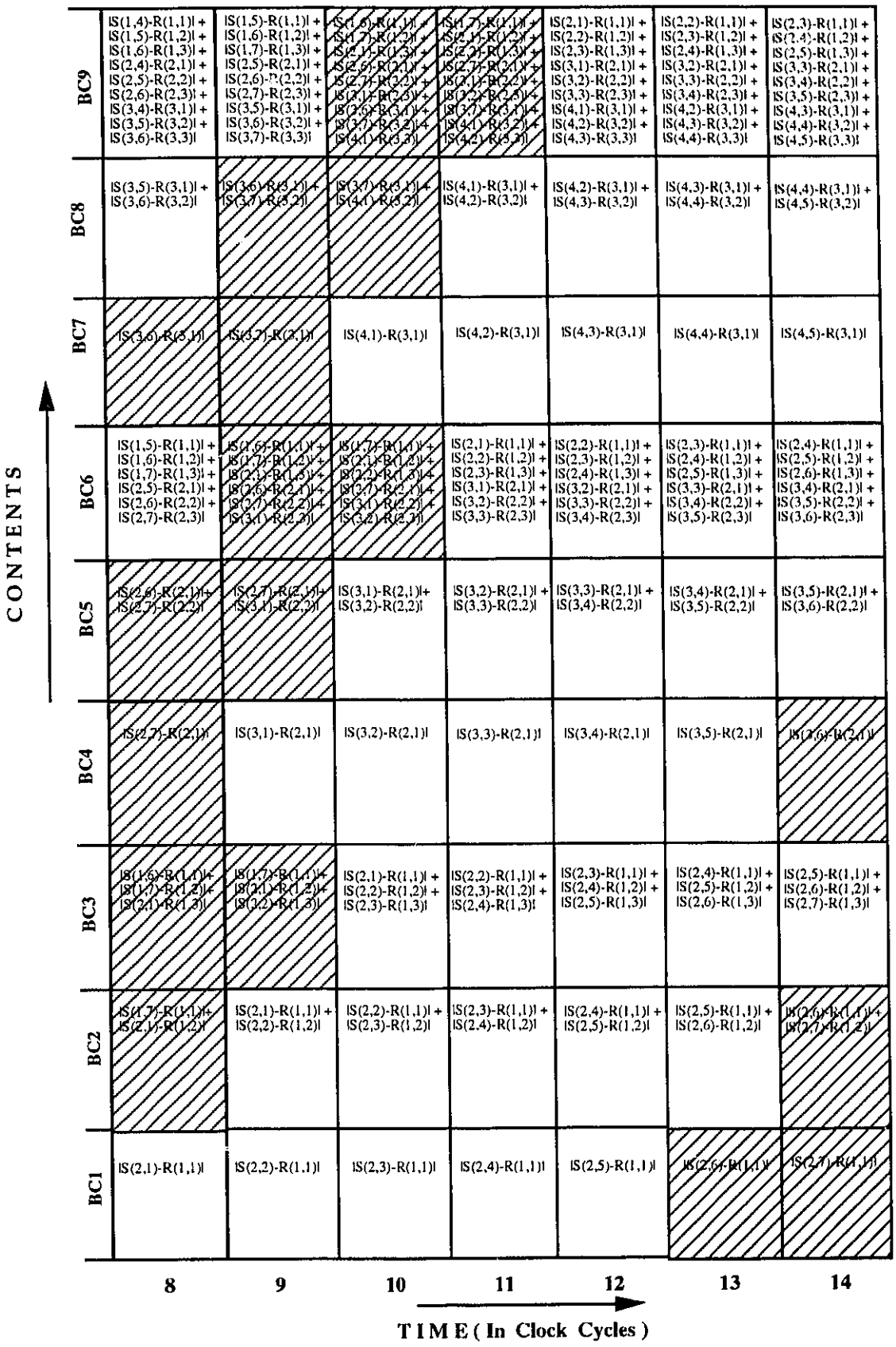



**Figure 3.6: Cell Occupancy Vs Time Chart (2)**

We note that the output of the leftmost BC in each row is the input to the leftmost BC in the next row. For example, the outputs of BC3 and BC6 are connected to the input of BC6 and BC9, respectively. This necessitates a delay of 1 clock cycle between any two successive rows of BC's. Hence there is a total delay of $n-1$ clock cycles in the last row of BC's with respect to the first row. In order to calculate all MAE's (as in equation (3.1)) for a fixed value of $\Delta y$ while $\Delta x$ is varied from $-p$ to $p$, $n+2p$ pixels of the search area are input to each row of BC's. Since $\Delta x$ is varied from $-p$ to $p$, there are a total of $2p+1$ possible values of $\Delta x$. Hence a total number of $(n+2p)(2p+1)$ pixels of the search area are required to be input to each row of BC's. The total processing time for computing all MAE's for a reference block is given as given as follows:

$$T_{parallel} = (n^2 + (n+2p)(2p+1) + n - 1) \text{ clock cycles} \qquad \text{--- (3.2)}$$

where $n^2$ clock cycles are required to load the reference block's pixels serially.

The MAE values are then entered into the best match selection unit (BMSU) which essentially selects the candidate block with the minimum value of MAE (the best match block) and the corresponding motion vector $(V)$ is determined.

If we assume that the BMSU requires only one clock cycle, the processing time for computing the motion vector $(V)$ for a reference block is given as follows:

$$\begin{aligned} T_{parallel} &= (n^2 + (n+2p)(2p+1) + n - 1 + 1) \\ &= (n^2 + (n+2p)(2p+1) + n) \text{ clock cycles} \end{aligned} \qquad \text{--- (3.3)}$$

### 3.2.2 BC and BMSU Design

The detailed design of the BC is shown in Figure 3.7. The reference block's pixel is stored in the register A while $p1$ and $p2$ correspond to the partial MAE's from adjacent BC's (refer to Figure 3.4). We note that the operations $p1 + p2$ and $|S - R|$ are performed in parallel. Hence the cycle time of the BC is calculated as follows:

$$\begin{aligned} t_{BC} &= t_{ABS} + t_{LATCH} + t_{ADDER} + t_{LATCH} \\ &= t_{ABS} + t_{ADDER} + 2t_{LATCH} \end{aligned} \qquad \text{--- (3.4)}$$

We assume that the absolute subtractor requires at least the same processing time as the adder.

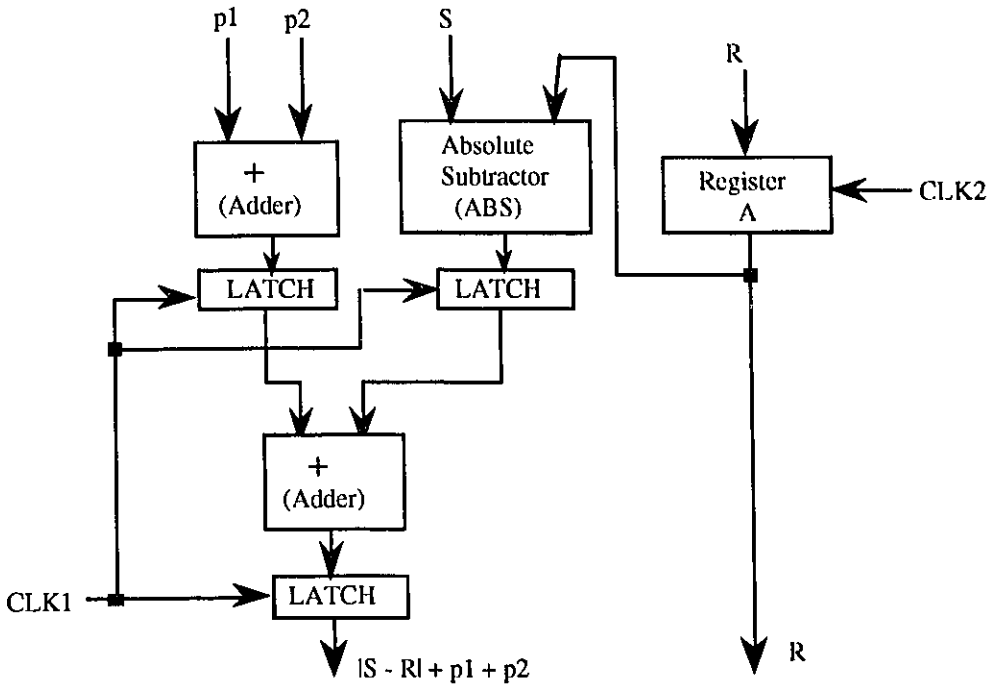



**Figure 3.7: The Detailed Design of BC**

The block diagram of the BMSU is illustrated in Figure 3.8. It consists of two units, namely, a Best Match Selector and a Valid Candidate Block Monitor. The Best Match Selector selects the smallest MAE value among the MAE values generated from the two-dimensional BC's structure. The champion register stores temporarily the smallest MAE value (current champion) and is initially set to 1's in all its bit positions (i.e. the maximum value). The incoming MAE values are compared with the current champion. If the MAE value is smaller, then this value becomes the current champion. The valid candidate block monitor keeps track and examines the indices $(\Delta x, \Delta y)$ of the incoming MAE values. If the incoming MAE belongs to an invalid region

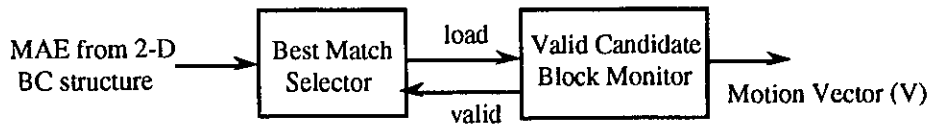
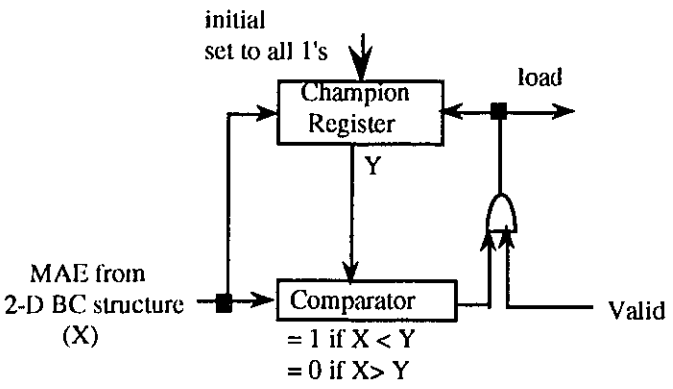
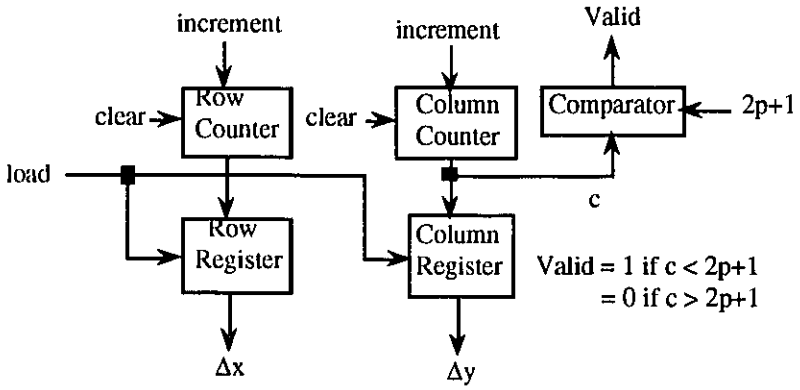



**Figure 3.8 (a): Block Diagram of Best Match Selection Unit (BMSU) Design**




(i) Best Match Selector




(ii) Valid Candidate Block Monitor

**Figure 3.8 (b): The Detailed Design of Best Match Selection Unit (BMSU)**

of the search area, i.e. the column counter has a value greater than or equal $2p+1$ (corresponding to the shaded regions in Figures 3.5, 3.6 and 3.9), an invalid signal (i.e. *valid* $= 0$) is generated and this signal will disable the output of the comparator in the Best Match Selector. This avoids the invalid MAE values from generating the wrong motion vector. We note that the row and column counters have the indices of the current MAE value, while the row and column registers contain the indices of the smallest MAE value. The column counter should be cleared every $(n+2p)$ clock cycles and the row counter should be incremented every $(n+2p)$ clock cycles since there are $(n+2p)$ pixels in each row of the search area.

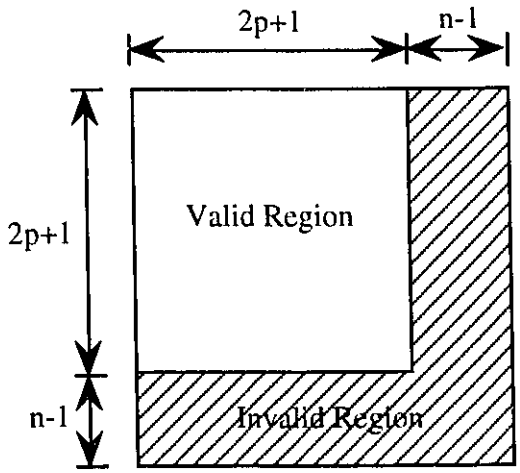



**Figure 3.9: The Regions of Valid and Invalid Motion Vectors**

### 3.2.3 Performance Analysis

We recall from chapter 2 that the execution of the MAE criterion involves $n^2$ operations. Since FSA requires that all $(2p+1)^2$ possible candidate blocks are searched for a match, the total number of operations for computing all MAE's for a reference block is given as follows:

$$\text{Number of Operations} = n^2(2p+1)^2 \qquad \text{--- (3.5)}$$

If FSA is executed using a general purpose processor and assuming that only one clock cycle is required to select the candidate block with the minimum value of MAE, the total processing time is calculated as follows:

$$T_{general} = n^2(2p+1)^2 + 1 \ clock\ cycles \qquad \text{--- (3.6).}$$

Hence the speed up factor of the proposed architecture is given by:

$$S_{parallel} = \frac{T_{general}}{T_{parallel}} = \frac{n^2(2p+1)^2+1}{n^2+(n+2p)(2p+1)+n} \qquad \text{--- (3.7).}$$

For example, if $n = p = 8$, this corresponds to a speed up, $S_{Parallel}$ of 39.

We now illustrate the capability of the proposed architecture for executing the motion estimation algorithm in real-time. For an image of size $512 \times 512$ pixels, with $n = 8$, $p = 8$, the

number of clock cycles required for computing the MAE's for each block (equation (3.3)) is 480. The total number of blocks in a frame is 4096 (i.e. (512/8) * (512/8)). Hence the execution time for a frame is 19.7ms (i.e. 480 * 4096 * 10ns) assuming that the clock cycle is 10ns [55]. This satisfies the real-time requirement (33ms).

In contrast to the architectures reported in the literature, the proposed architecture does not employ any parallel adders and hence removes the computation bottleneck. Most importantly, the architecture is modular and hence makes possible computation of a $2n \times 2n$ reference block by a simple cascade of four identical $n \times n$ chips. For example, Figure 3.10 illustrates the computation of a $4 \times 4$ reference block using a cascade of four $2 \times 2$ chips.

## 3.3 Summary

A VLSI architecture which implements the FSA-MAE algorithm has been presented. The architecture computes the MAE of different (search area) candidate blocks in parallel. This results

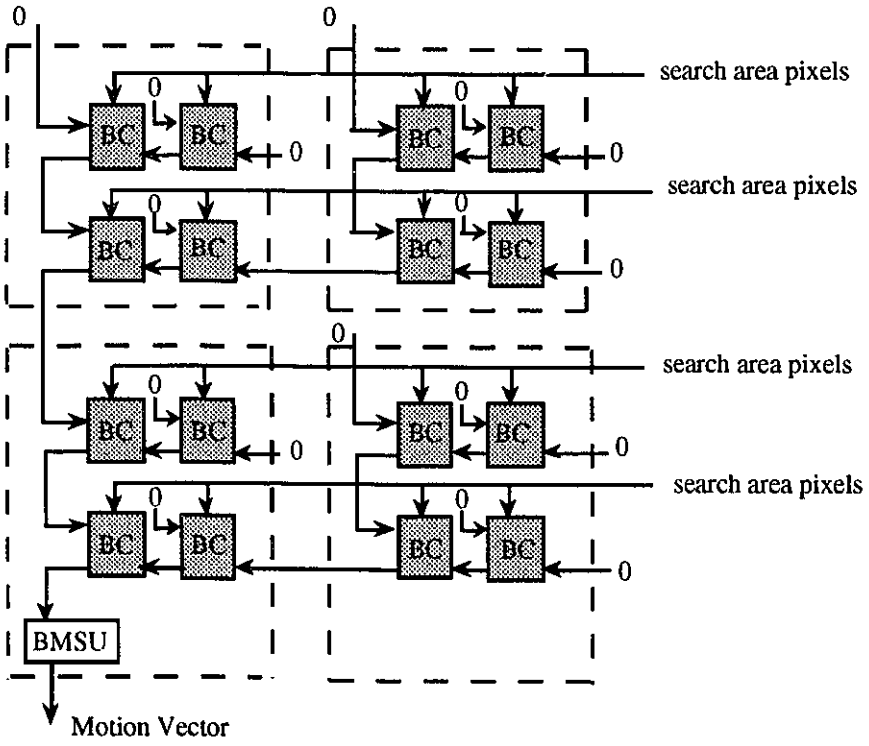



**Figure 3.10: The Cascaded System for n=4**

in real-time computation of the motion vector. Most importantly, the architecture is modular and cascadable and hence makes possible computation of a $2n \times 2n$ reference block by a simple cascade of four $n \times n$ chips. The proposed architecture is easily implementable in VLSI as a codec.

We recall from chapter 2 that the full search algorithm yields the optimum motion vectors. However, it is computationally expensive. Hence it is desirable to have reduced complexity motion estimation algorithms which have a comparable performance to FSA and in addition be easily implementable in hardware. In next chapter, we present a new reduced complexity block matching based motion estimation algorithm. This algorithm has a comparable performance to the FSA-MAE. In addition, the hardware implementation is simple because of the binary operations used in the matching criteria.

# Chapter 4

# Reduced Complexity Block Matching Algorithm for an MPEG Video Coder

We recall from chapter 2 that the most intuitive approach for block matching is to use the full search algorithm (FSA) [6]. For each reference block, all possible candidate blocks are searched to obtain the best match block. However, this is a computationally expensive procedure. Several fast algorithms [23-28] (e.g. parallel hierarchical one dimensional search (PHODS)) have been proposed to reduce the computational burden in FSA. These fast algorithms are based on the assumption that the error increases monotonically as the search moves away from the global optimum position. This results in a reduced number of candidate blocks to be searched. However, such an assumption may lead to a motion vector which converges to a local optimum rather than the global optimum. This reduces the accuracy of the estimated motion vectors. Hence the performance of the fast algorithms are generally poor compared to FSA.

In this chapter, we present a reduced complexity block matching based motion estimation algorithm which consists of a layered structure and hence alleviates the local optimum problem of the fast algorithms. Most importantly, it uses a simple matching criterion namely, the modified pixel difference classification (MPDC). The MPDC criterion represents the matching algorithm as a binary process which consequently reduces the computational complexity. In addition, the algorithm follows the guidelines provided by the MPEG-1 video compression standard (reviewed in section 2.8). The MPEG-1 video coder has been simulated using the proposed layered structure MPDC algorithm (LSA-MPDC) and other techniques such as FSA (using various matching criteria including mean square error (FSA-MSE), mean absolute error (FSA-MAE) and pixel difference classification (FSA-PDC)) and PHODS. Simulation results indicate that the LSA-MPDC algorithm performs comparable to the FSA-MSE and FSA-MAE algorithms at a significantly reduced computational complexity. In addition, it is better than the FSA-PDC and PHODS algorithms. We note that the hardware implementation of LSA-MPDC is very simple because of the binary operations used in the matching criteria.

## 4.1 Layered Structure MPDC Based Algorithm (LSA-MPDC)

We now present the layered structured and reduced complexity block matching algorithm for motion estimation. The algorithm is based on a modified pixel difference classification (MPDC) as the matching criterion. In MPDC, each pixel in a block is first classified as either a *matching pixel* or *mismatching pixel* with respect to a threshold t, i.e.

$$T(i,j,\Delta x,\Delta y) = 1 \quad if\ |S(i+\Delta x, j+\Delta y) - R(i,j)| \le t$$
$$= 0 \quad otherwise \qquad \text{--- (4.1)}$$

where $R(i,j)$ is a reference block's pixel in the current frame $(t)$, $S(i+\Delta x, j+\Delta y)$ is a candidate block's pixel within a search area in the previous frame $(t-1)$ and $T(i,j,\Delta x,\Delta y)$ is the binary representation of the pixel difference. We note that this is identical to the PDC criterion (equation (2.7)). A block is then classified as a *matching block* if all its pixels are *matching pixels*. The MPDC criterion is defined as follows:

$$MPDC(\Delta x,\Delta y) = 1 \quad if \sum_{i=1}^{n}\sum_{j=1}^{n} T(i,j,\Delta x,\Delta y) = n^2$$
$$= 0 \quad otherwise \qquad \text{--- (4.2)}$$

where $MPDC(\Delta x,\Delta y)$ with a value of one or zero corresponds to a *matching block* or a *mismatching block.*

In the search procedure, the number of candidate blocks varies according to the search level. At the first level, a full search is performed using the MPDC criterion with a fixed threshold (a power of 2) to obtain a smaller set of candidate blocks. Then, at each successive level, the threshold is either halved or doubled and a new set of candidate blocks is obtained from the preceding level. If the initial threshold is too low, it is increased (doubled) until the candidate set is not a null set. Similarly, if the initial threshold is too high, it is decreased (halved) until the candidate set is a null set or the threshold value becomes zero. We note that when the final set has more than one candidate block, the search is performed using the PDC criterion (equation (2.8)). The algorithm for $p = 5$ is illustrated in Figure 4.1. The search procedure can be written in algorithmic form as follows:

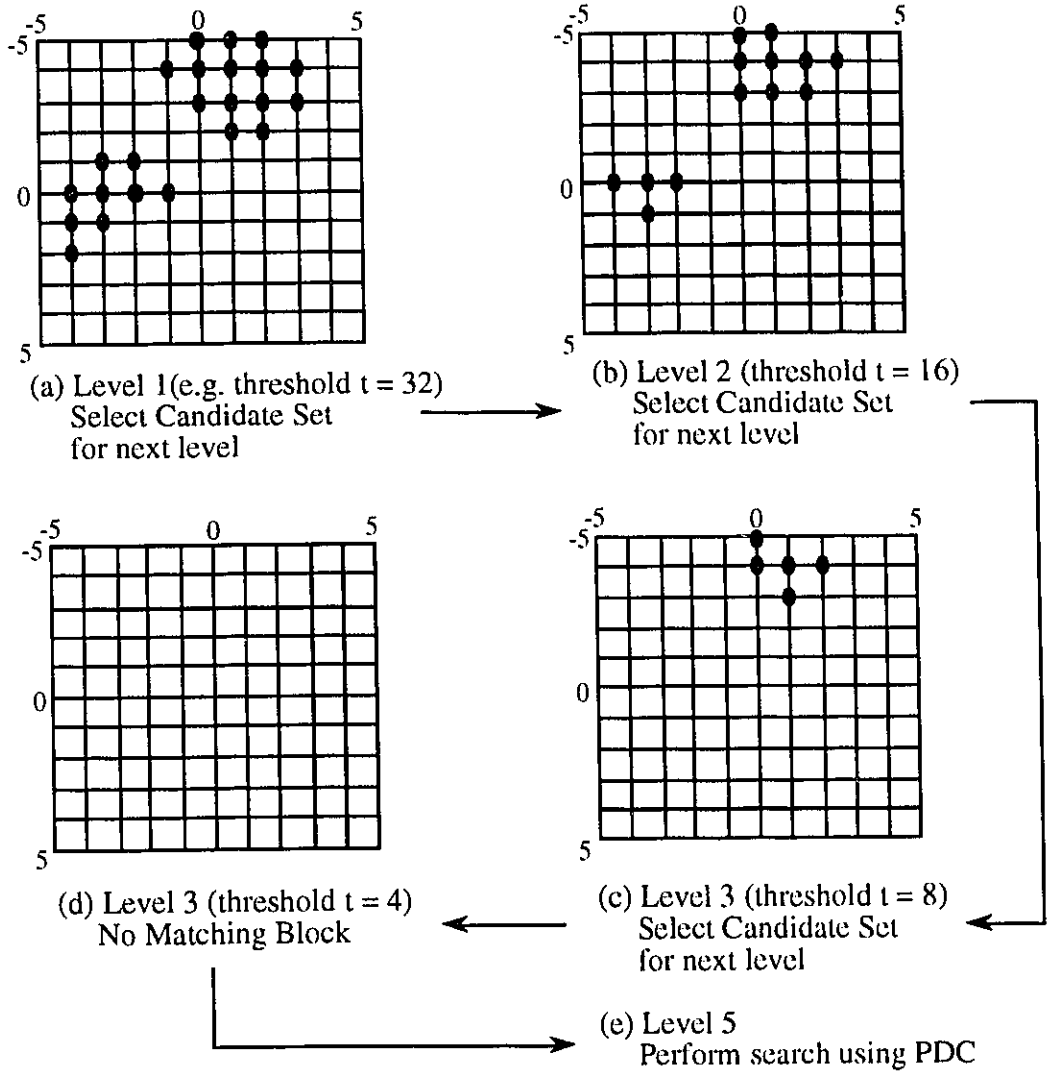



**Figure 4.1: Illustration of the Proposed LSA-MPDC Motion Estimation Algorithm**

**Initialization:** • Candidate set C1 = all possible candidate blocks;

• Set a variable UP = 2 and threshold $t = 2^m$, where $m > 0$;

**Step 1:** • Perform search on C1 using MPDC;

• Select candidate set C2 = all *matching blocks*;

• if C2 has 1 element then *Stop*;

else if (C2 = NULL and UP ≠ 0) m = m + 1, UP = 1;

else if (C2 ≠ NULL and UP ≠ 1) m = m - 1, UP = 0;

**Step 2:** • if ((m < 0) or (C2 ≠ NULL and UP = 1)) Perform search on C2 using PDC and *Stop*;

else if (C2 = NULL and UP = 0) Perform search on C1 using PDC and *Stop*;

else C1 = C2 and goto Step 1;

One of the drawbacks of the LSA-MPDC algorithm is the number of iterations required for convergence which depends on the motion in the sequence and the initial value of the threshold. However, this number can be reduced by selecting the previous block's final threshold as an initial threshold for the current block. In addition, the execution time for each iteration is smaller because of the use of a simple matching criterion (MPDC).

## 4.2 Simulation Results

Simulations have been performed using the standard test sequence, *"Miss America"* and *"PingPong"* of CIF format (Common Intermediate Format) quantized to 8 bits per pixel on a 486 DX2 66 MHz personal computer running *System V* operating system. The test sequence *"Miss America"* ($352 \times 288$ pixels) has slow to moderate motion with uniform background whereas *"PingPong"* ($352 \times 240$ pixels) has fast motion with high spatial details. The coding performance is measured using the peak signal to noise ratio (PSNR) [4] which is defined as follows:

$$PSNR = \frac{(Peak\ to\ Peak\ value\ of\ the\ original\ image\ data)^2}{average\ mean\ square\ \text{error}} \qquad \text{--- (4.3)}$$

and the average mean square error (AMSE) is defined as follows:

$$AMSE = \frac{1}{N \times M} \sum_{i=1}^{N} \sum_{j=1}^{M} (O(i,j) - R(i,j))^2 \qquad \text{--- (4.4)}$$

where $O(i,j)$ is the original frame and R(i,j) is the reconstructed frame of size $N \times M$ pixels. We note that the number of gray levels is 256 (i.e. 0 - 255) in a typical frame (quantized to 8 bits per pixel). Hence the peak to peak value of the original image data is 255 and equation (4.3) can be written as follows:

$$PSNR = \frac{255^2}{AMSE} \qquad \text{--- (4.5)}$$

Five sets of experiments have been performed which are detailed as follows:

In experiment #1, the FSA-PDC algorithm has been simulated using different threshold values ($t = 5$ and 20). A block size of $16 \times 16$ ($n = 16$) and a maximum displacement of 7 ($p = 7$) were used in this experiment. The PSNR of the motion compensated frame difference using FSA-PDC are tabulated in Table 4.1. The corresponding PSNR vs Frame number are plotted in Figures 4.3 and 4.4 for *"Miss America"* and *"PingPong"*, respectively. The simulation results confirm that the choice of the threshold ($t$) has a major impact on the performance of FSA-PDC, i.e. a smaller threshold value performs better than a larger threshold for a slow motion sequence while the situation is reversed for a fast motion sequence.

In experiments #2 - #5, the MPEG-1 video coder and decoder have been simulated using the various block matching algorithms, namely, FSA-MSE, FSA-MAE, FSA-PDC, PHODS and LSA-MPDC. We note that a threshold value of 10 was used in FSA-PDC in our experiments.

In experiment #2, the performance of the LSA-MPDC algorithm is compared with that of FSA-MSE, FSA-MAE, FSA-PDC and PHODS (reviewed in sections 2.2 and 2.3) for an MPEG-1 video coder for a block size of $16 \times 16$ ($n = 16$) and a maximum displacement of 8 ($p = 8$). The simulation results (PSNR) are tabulated in Tables 4.2 and 4.3 for the *"Miss America"* sequence at 1.18 Mbit/sec (compression factor 20:1) and *"PingPong"* sequence at 1.53 Mbit/sec (compression 13:1), respectively. The corresponding PSNR vs Frame number are plotted in Figure 4.5. The results show that the FSA-MSE algorithm achieves the best performance in terms of PSNR for both slow and fast moving sequences at an expense of a higher computational complexity. It can be seen that the LSA-MPDC algorithm has a marginally lower performance compared to FSA-MSE (and FSA-MAE) at a significantly reduced computational complexity. In addition, the performance of LSA-MPDC is better than FSA-PDC and PHODS. We note that the PHODS algorithm fails to capture the fast motion in the *"PingPong"* sequence and its performance is approximately 3 dB lower than LSA-MPDC. This is mainly because of the convergence to the local optimum of the PHODS algorithm in a fast motion sequence which in turns results in a poorer performance.

The subjective qualities of the reconstructed images (frame number 6 of *"Miss America"*) using various algorithms are shown in Figures 4.9-4.14. We note that although the FSA-MSE and FSA-MAE algorithms achieve the best performance in terms of PSNR, they do not out-perform the LSA-MPDC algorithm subjectively. This can be seen from the right eye of *"Miss America"* (Figures 4.9, 4.10, 4.11 and 4.14) where the reconstructed image using the LSA-MPDC algorithm is closer to the original image compared to the reconstructed images using FSA-MSE and FSA-MAE.

In experiment #3, simulations were performed for a smaller block size of $8 \times 8$ ($n = 8$) and a maximum displacement of 8 ( $p = 8$, same as the experiment #2). The simulation results (PSNR) are tabulated in Tables 4.4 and 4.5. The corresponding PSNR vs Frame number are plotted in Figure 4.6. We note that there is an overhead of 0.1 bits/pixel (i.e. 0.22 Mbit/sec) because of the increased number of motion vectors required to be transmitted. The bit rate including this overhead is 1.45 Mbit/sec for the *"Miss America"* sequence and 1.75 Mbit /sec for the *"PingPong"* sequence. Here, the behaviour of the LSA-MPDC algorithm is similar to the experiment #2. In general, a smaller block size should result in a better performance since the motion can be captured more easily. This is confirmed by the superior performance (in terms of PSNR) of FSA-MSE, FSA-MAE and LSA-MPDC. However, FSA-PDC does not perform better for smaller block sizes with the same threshold value. The reason for the inferior performance is that maximizing the number of matching pixels in the PDC criterion does not guarantee the minimum mean square error.

In experiment #4, a larger search area ( $p = 16$ ) with a block size of $16 \times 16$ ($n = 16$) was used to examine the effect of maximum allowed displacement on the coding performance. The simulation results are tabulated in Tables 4.6 and 4.7. The corresponding PSNR vs Frame number are plotted in Figure 4.7. We note that there is an overhead of 0.01 bits/pixel (i.e. 0.01 Mbit/sec) because of the larger dynamic range of the motion vectors. The bit rate including this overhead is 1.2 Mbit/sec for the *"Miss America"* sequence and 1.54 Mbit/sec for the *"PingPong"* sequence. The performance of the LSA-MPDC algorithm is similar to the previous sets of experiments. It

can be seen that the FSA-MSE, FSA-MAE and LSA-MPDC algorithms perform identical (in terms of PSNR) for $p = 8$ and $p = 16$ in the *"Miss America"* sequence. This is because slow motion with an uniform background can be captured adequately using a small search area. For the fast moving *"PingPong"* sequence, a larger search area helps to improve the accuracy of the estimated motion vectors and hence the coding performance.

In the last set of experiments, we have used a rectangular block of size $16 \times 8$ and a maximum displacement of 8 ($p = 8$). This choice of block size is based on the fact that the motion in a natural video sequence is typically in the horizontal direction. We note that there is an overhead of 0.04 bits/pixel (i.e. 0.07 Mbit/sec) because of the increased number of motion vectors required to be transmitted. The simulation results are tabulated in Tables 4.8 and 4.9. The corresponding PSNR vs Frame number are plotted in Figure 4.8 at a bit rate of 1.27 Mbit/sec for the *"Miss America"* sequence and at 1.6 Mbit/sec for the *"PingPong"* sequence. Once again, the behaviour of the LSA-MPDC algorithm is similar to the previous set of experiments. It can been that the performance gains using a block size of $16 \times 8$ compared to a block size of $16 \times 16$ are larger in the fast moving sequence than in the slow moving sequence. This is mainly because the *"PingPong"* sequence has more motion in the horizontal direction than the *"Miss America"* sequence.

In summary, the performance of the LSA-MPDC algorithm is comparable to the FSA-MSE and FSA-MAE techniques at a significantly reduced computational complexity. We note that the reduction in computational complexity is due to the binary operations in the matching criteria. The performance of LSA-MPDC does not depend on the initial threshold values as in the FSA-PDC algorithm. In addition, LSA-MPDC consists of a layered structure, and hence does not converge to a local optimum resulting in a superior performance compared to the PHODS algorithm.

## 4.3 Hardware Requirements

One of the major bottlenecks in executing the MPEG algorithm in real time is the computational complexity involved in the motion estimation process. The hardware components

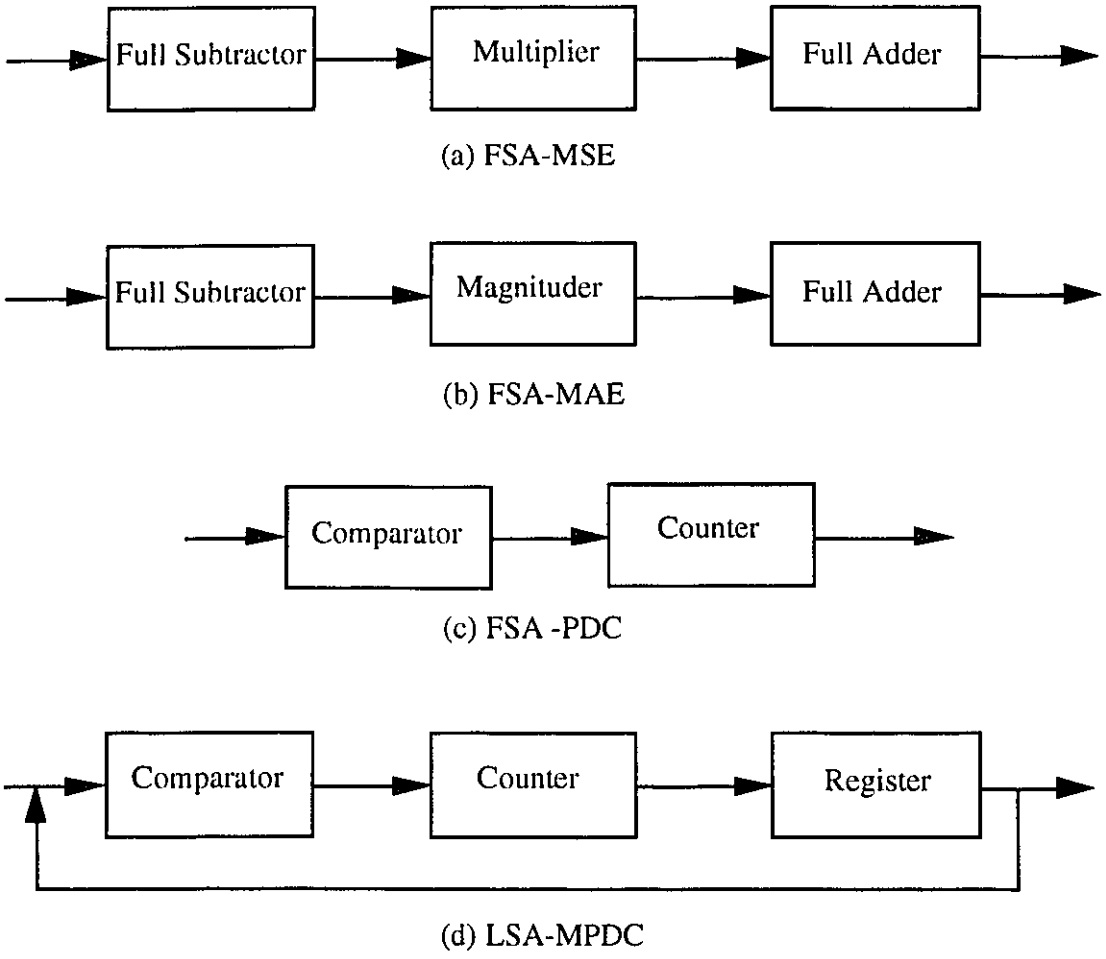



**Figure 4.2 Hardware Requirements for the Various Motion Estimation Algorithms**

required for the various block matching algorithms are shown in Figure 4.2. In the FSA-MSE and FSA-MAE algorithms, $(8+2\log_2 n)$ bit full adders and full subtractors (equations (2.5) and (2.6)) are required for a block size of $n \times n$ [22]. In addition, the FSA-MSE and FSA-MAE algorithms requires multipliers and magnituder, respectively. This makes the implementation of FSA-MSE/FSA-MAE expensive. We recall that execution of PDC criterion (equations (2.7) and (2.8)) involves only comparison and increment operations. Hence the FSA-PDC algorithm requires only threshold comparators and $2\log_2 n$ bit counters [22]. Since the MPDC criterion is an extension of the PDC criterion, the requirements for the LSA-MPDC algorithm is similar to FSA-PDC. We

note that the increment operations required in the PDC and MPDC criteria are binary operations. Hence the corresponding hardware implementations are simpler compared to FSA-MSE and FSA-MAE. We recall that the LSA-MPDC algorithm results in a set of intermediate *matching blocks* (candidate blocks) in each iteration and hence requires additional registers to store the intermediate *matching blocks*. However, we note that the each iteration of the algorithm results in a smaller subset of candidate blocks compared to the previous iteration and hence only fewer registers are required. In summary, the hardware requirements of the LSA-MPDC algorithm are significantly lower compared to FSA-MSE and FSA-MAE. This makes real-time MPEG encoding is possible with simple processing elements using the LSA-MPDC algorithm.

## 4.4 Summary

In this chapter, we have presented a reduced complexity block matching motion estimation algorithm for an MPEG video coder. The proposed algorithm consists of a layered structure and hence does not converge to the local optimum. Most importantly, it employs a simple matching criterion, namely, the modified pixel difference classification (MPDC). This overcomes the computational burden in the FSA using the MSE or MAE criterion. In addition, the performance of the MPDC criterion does not depend on the initial threshold value as in the PDC criterion. Hence it results in a superior performance for both slow and fast motion video sequences. The MPEG-1 video coder has been simulated using various block matching algorithms such as LSA-MPDC, FSA-MSE, FSA-MAE, FSA-PDC and PHODS. Simulation results indicate that the LSA-MPDC algorithm performs comparable to the FSA-MSE and the FSA-MAE algorithms at a significantly reduced computational complexity. In addition, the performance of LSA-PDC is better than the FSA-PDC and the PHODS algorithms. In addition, hardware implementation is very simple because of the binary operations used in the MPDC and PDC criteria.

We recall from chapter 2 that in the MPEG-1 standard, DCT and motion estimation/ compensation are used to exploit the spatial and temporal redundancies, respectively. However, these redundancies are exploited independently. In next chapter, we present a fast multiresolution

motion estimation algorithm based on wavelet transform coding. Here, wavelet transform is used to exploit both spatial and temporal redundancies resulting in an efficient video coding scheme. In addition, the cross-correlation among each subimage in the wavelet pyramid structure is exploited. This results in a reduced computational complexity in the motion estimation process.

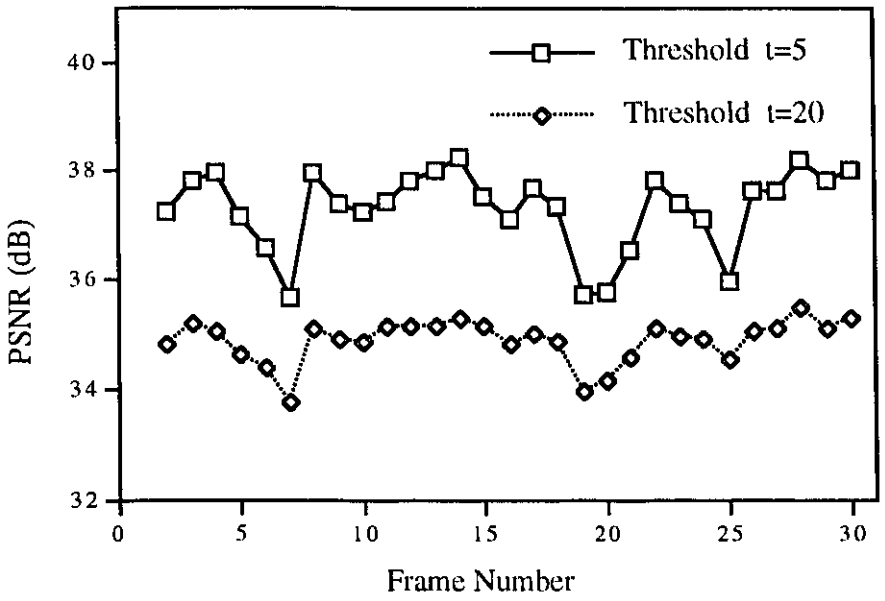



**Figure 4.3: PSNR Values of the Motion Compensated Frame Using FSA-PDC, *"Miss America"* (n=16 and p=7)**

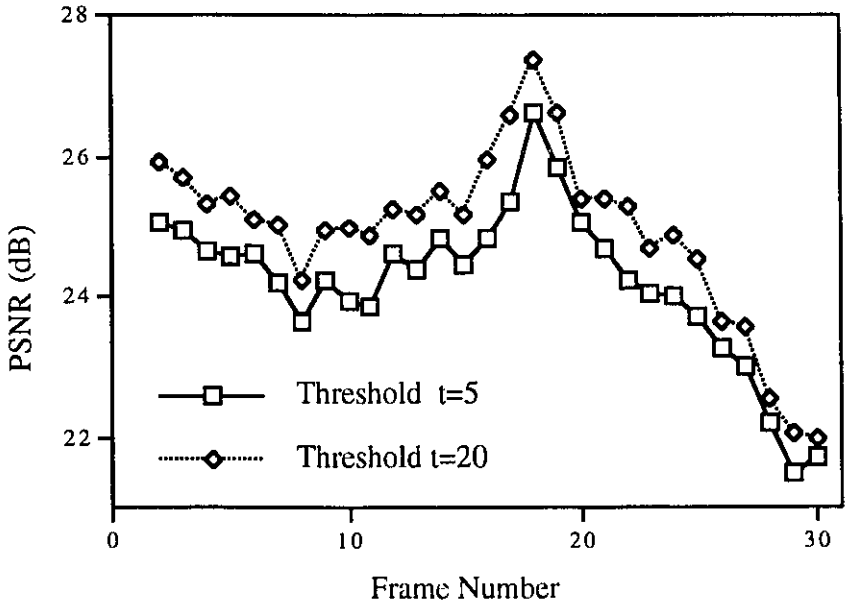



**Figure 4.4: PSNR Values of the Motion Compensated Frame Using FSA-PDC, *"PingPong"* (n=16 and p=7)**

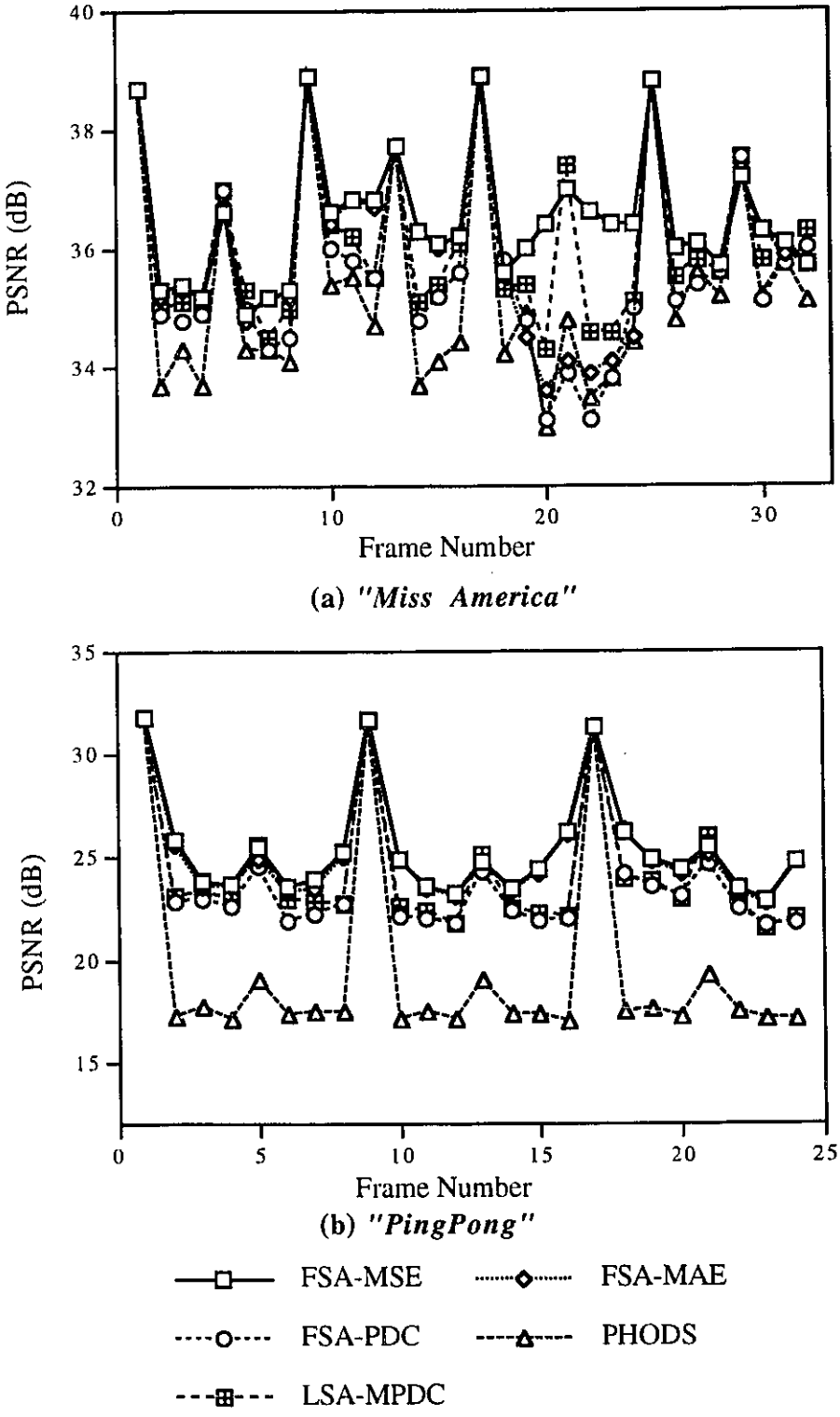



**Figure 4.5: PSNR Values of Coded Sequence Using the MPEG-1 Coder (n=16 and p=8)**

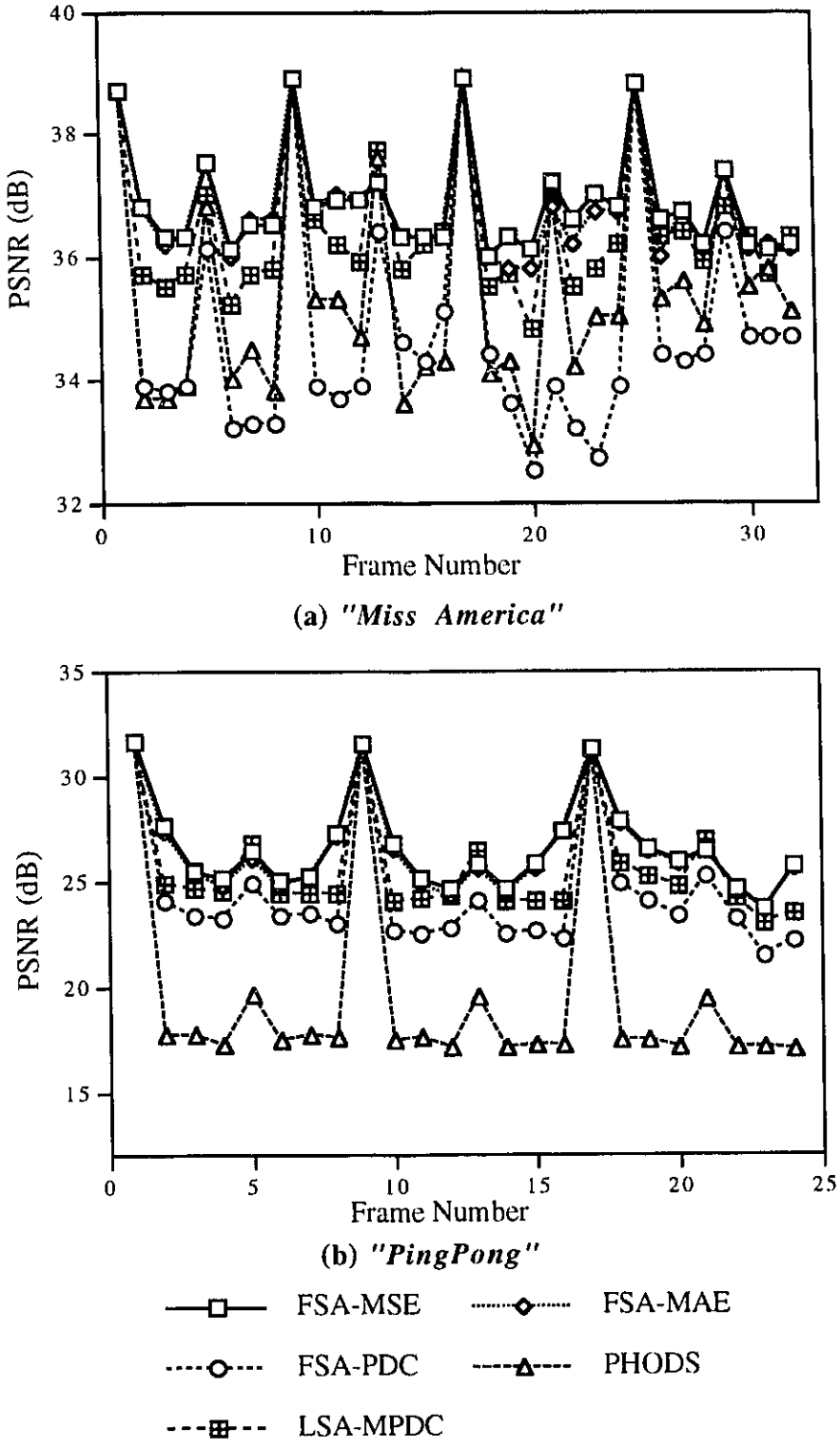



**Figure 4.6: PSNR Values of Coded Sequence Using the MPEG-1 Coder (n=8 and p=8)**

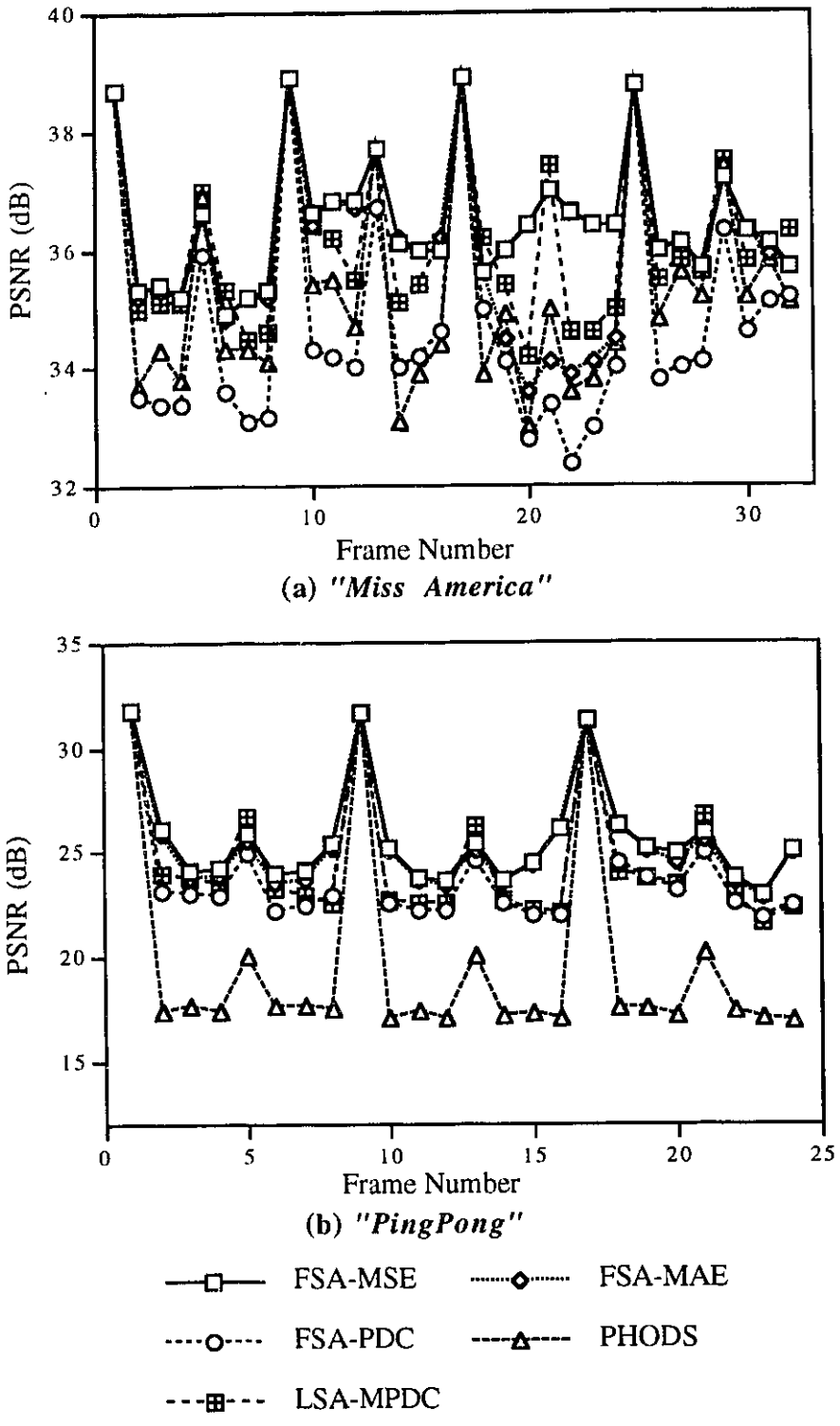



**Figure 4.7: PSNR Values of Coded Sequence Using the MPEG-1 Coder (n=16 and p=16)**

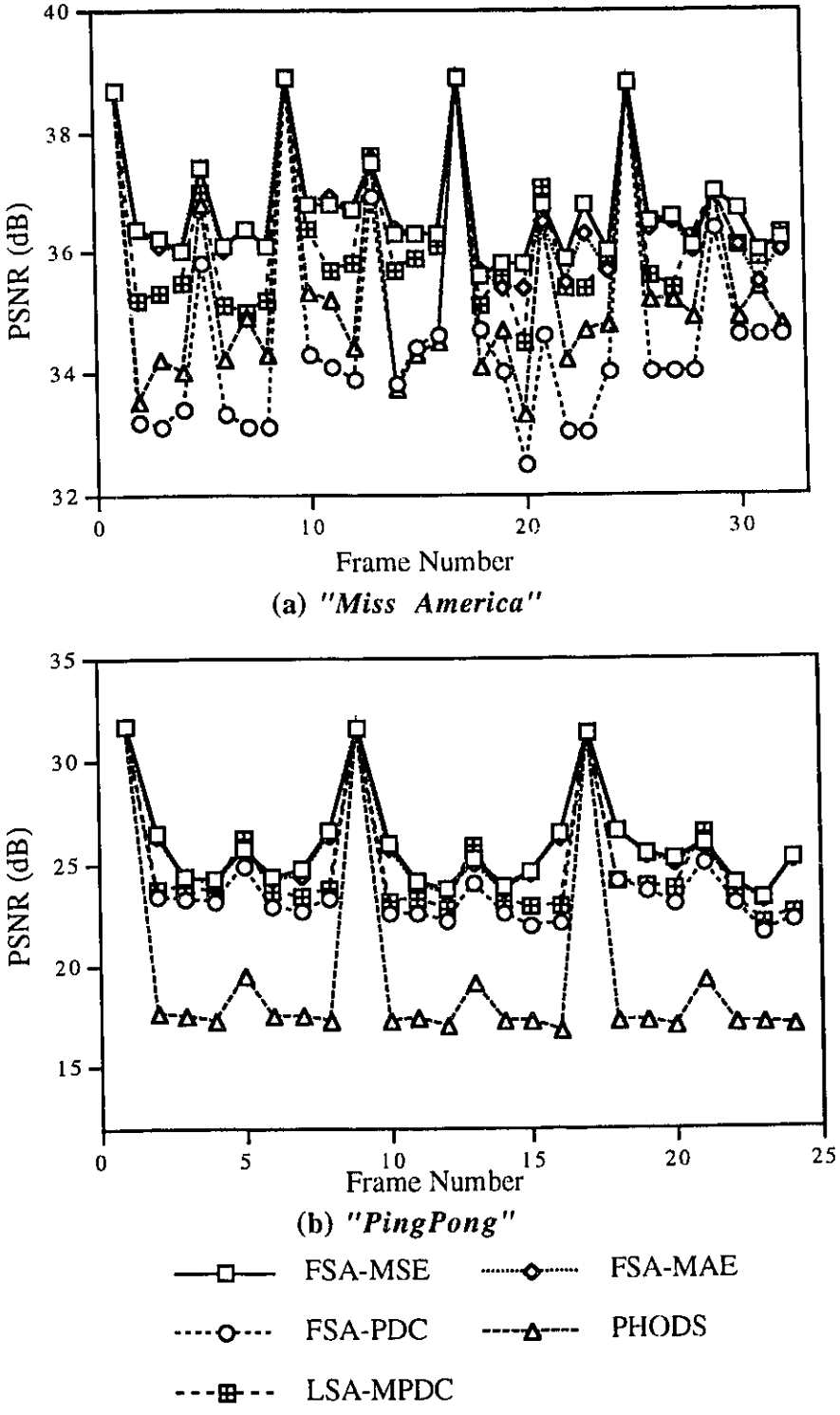



**Figure 4.8: PSNR Values of Coded Sequence Using the MPEG-1 Coder (Block Size 16 x 8 and p=8)**

| *Miss America Sequence* | | | *PingPong Sequence* | |
|---|---|---|---|---|
| Frame # | Threshold t = 5 | Threshold t = 20 | Threshold t = 5 | Threshold t = 20 |
| 2 | 37.26 | 34.79 | 25.06 | 25.93 |
| 3 | 37.83 | 35.17 | 24.96 | 25.71 |
| 4 | 37.99 | 35.04 | 24.63 | 25.32 |
| 5 | 37.18 | 34.62 | 24.59 | 25.42 |
| 6 | 36.58 | 34.38 | 24.62 | 25.09 |
| 7 | 35.66 | 33.76 | 24.21 | 25.02 |
| 8 | 37.96 | 35.09 | 23.63 | 24.25 |
| 9 | 37.39 | 34.92 | 24.22 | 24.94 |
| 10 | 37.27 | 34.87 | 23.93 | 24.98 |
| 11 | 37.46 | 35.14 | 23.88 | 24.86 |
| 12 | 37.82 | 35.15 | 24.61 | 25.26 |
| 13 | 38.00 | 35.13 | 24.38 | 25.19 |
| 14 | 38.24 | 35.28 | 24.82 | 25.52 |
| 15 | 37.54 | 35.16 | 24.47 | 25.19 |
| 16 | 37.10 | 34.79 | 24.83 | 25.95 |
| 17 | 37.68 | 35.00 | 25.36 | 26.61 |
| 18 | 37.35 | 34.84 | 26.62 | 27.37 |
| 19 | 35.74 | 33.96 | 25.83 | 26.65 |
| 20 | 35.79 | 34.12 | 25.06 | 25.38 |
| 21 | 36.55 | 34.57 | 24.69 | 25.39 |
| 22 | 37.81 | 35.09 | 24.25 | 25.29 |
| 23 | 37.40 | 34.94 | 24.05 | 24.70 |
| 24 | 37.11 | 34.92 | 24.01 | 24.88 |
| 25 | 35.97 | 34.52 | 23.72 | 24.54 |
| 26 | 37.63 | 35.03 | 23.25 | 23.65 |
| 27 | 37.63 | 35.10 | 23.01 | 23.56 |
| 28 | 38.23 | 35.48 | 22.23 | 22.55 |
| 29 | 37.83 | 35.12 | 21.52 | 22.06 |
| 30 | 38.00 | 35.30 | 21.72 | 22.00 |

**Table 4.1: PSNR Values of the Motion Compensated Frame Using FSA-PDC (n=16 and p=7)**

| Frame # | FSA-MSE | FSA-MAE | FSA-PDC | PHODS | LSA-MPDC |
|---|---|---|---|---|---|
| 1 | 38.7 | 38.7 | 38.7 | 38.7 | 38.7 |
| 2 | 35.3 | 35.2 | 34.9 | 33.7 | 35.0 |
| 3 | 35.4 | 35.3 | 34.8 | 34.3 | 35.1 |
| 4 | 35.2 | 35.2 | 34.9 | 33.7 | 35.1 |
| 5 | 36.6 | 36.7 | 37.0 | 36.8 | 37.0 |
| 6 | 34.9 | 34.8 | 35.0 | 34.3 | 35.3 |
| 7 | 35.2 | 35.2 | 34.3 | 34.3 | 34.5 |
| 8 | 35.3 | 35.2 | 34.5 | 34.1 | 35.0 |
| 9 | 38.9 | 38.9 | 38.9 | 38.9 | 38.9 |
| 10 | 36.6 | 36.4 | 36.0 | 35.4 | 36.4 |
| 11 | 36.8 | 36.8 | 35.8 | 35.5 | 36.2 |
| 12 | 36.8 | 36.7 | 35.5 | 34.7 | 35.5 |
| 13 | 37.7 | 37.7 | 37.7 | 37.7 | 37.7 |
| 14 | 36.3 | 36.3 | 34.8 | 33.7 | 35.1 |
| 15 | 36.1 | 36.0 | 35.2 | 34.1 | 35.4 |
| 16 | 36.2 | 36.2 | 35.6 | 34.4 | 36.1 |
| 17 | 38.9 | 38.9 | 38.9 | 38.9 | 38.9 |
| 18 | 35.6 | 35.6 | 35.8 | 34.2 | 35.3 |
| 19 | 36.0 | 34.5 | 34.8 | 34.9 | 35.4 |
| 20 | 36.4 | 33.6 | 33.1 | 33.0 | 34.3 |
| 21 | 37.0 | 34.1 | 33.9 | 34.8 | 37.4 |
| 22 | 36.6 | 33.9 | 33.1 | 33.5 | 34.6 |
| 23 | 36.4 | 34.1 | 33.8 | 33.8 | 34.6 |
| 24 | 36.4 | 34.5 | 35.0 | 34.4 | 35.1 |
| 25 | 38.8 | 38.8 | 38.8 | 38.8 | 38.8 |
| 26 | 36.0 | 36.0 | 35.1 | 34.8 | 35.5 |
| 27 | 36.1 | 36.1 | 35.4 | 35.6 | 35.8 |
| 28 | 35.7 | 35.7 | 35.6 | 35.2 | 35.6 |
| 29 | 37.2 | 37.2 | 37.5 | 37.4 | 37.5 |
| 30 | 36.3 | 36.3 | 35.1 | 35.2 | 35.8 |
| 31 | 36.1 | 35.9 | 35.7 | 35.8 | 36.1 |
| 32 | 35.7 | 35.7 | 36.0 | 35.1 | 36.3 |

**Table 4.2: PSNR Values of Coded *"Miss America"* Sequence Using the MPEG-1 Coder (n=16 and p=8)**

| Frame # | FSA-MSE | FSA-MAE | FSA-PDC | PHODS | LSA-MPDC |
|---|---|---|---|---|---|
| 1 | 31.7 | 31.7 | 31.7 | 31.7 | 31.7 |
| 2 | 25.8 | 25.6 | 22.9 | 17.3 | 23.1 |
| 3 | 23.8 | 23.6 | 23.0 | 17.7 | 23.3 |
| 4 | 23.7 | 23.5 | 22.6 | 17.2 | 23.0 |
| 5 | 25.4 | 24.9 | 24.5 | 19.1 | 25.6 |
| 6 | 23.6 | 23.3 | 21.9 | 17.4 | 23.0 |
| 7 | 23.9 | 23.4 | 22.3 | 17.5 | 22.8 |
| 8 | 25.2 | 25.0 | 22.7 | 17.5 | 22.7 |
| 9 | 31.6 | 31.6 | 31.6 | 31.6 | 31.6 |
| 10 | 24.9 | 24.9 | 22.1 | 17.1 | 22.6 |
| 11 | 23.5 | 23.4 | 22.0 | 17.5 | 22.4 |
| 12 | 23.2 | 23.1 | 21.8 | 17.1 | 21.8 |
| 13 | 24.8 | 24.8 | 24.3 | 19.0 | 25.1 |
| 14 | 23.4 | 23.3 | 22.4 | 17.4 | 22.5 |
| 15 | 24.4 | 24.2 | 21.9 | 17.4 | 22.3 |
| 16 | 26.2 | 26.1 | 22.0 | 17.0 | 22.1 |
| 17 | 31.3 | 31.3 | 31.3 | 31.3 | 31.3 |
| 18 | 26.2 | 26.3 | 24.1 | 17.5 | 23.9 |
| 19 | 24.9 | 24.7 | 23.6 | 17.6 | 23.8 |
| 20 | 24.4 | 24.2 | 23.1 | 17.3 | 23.0 |
| 21 | 25.5 | 25.1 | 24.6 | 19.3 | 25.9 |
| 22 | 23.4 | 23.2 | 22.5 | 17.5 | 22.8 |
| 23 | 22.9 | 22.7 | 21.7 | 17.1 | 21.6 |
| 24 | 24.8 | 24.8 | 21.8 | 17.1 | 22.0 |

**Table 4.3: PSNR Values of Coded *"PingPong"* Sequence Using the MPEG-1 Coder (n=16 and p=8)**

| Frame # | FSA-MSE | FSA-MAE | FSA-PDC | PHODS | LSA-MPDC |
|---|---|---|---|---|---|
| 1 | 38.7 | 38.7 | 38.7 | 38.7 | 38.7 |
| 2 | 36.8 | 36.8 | 33.9 | 33.7 | 35.7 |
| 3 | 36.3 | 36.2 | 33.8 | 33.7 | 35.5 |
| 4 | 36.3 | 36.3 | 33.9 | 33.9 | 35.7 |
| 5 | 37.5 | 37.5 | 36.1 | 36.8 | 37.0 |
| 6 | 36.1 | 36.0 | 33.2 | 34.0 | 35.2 |
| 7 | 36.5 | 36.6 | 33.3 | 34.5 | 35.7 |
| 8 | 36.5 | 36.6 | 33.3 | 33.8 | 35.8 |
| 9 | 38.9 | 38.9 | 38.9 | 38.9 | 38.9 |
| 10 | 36.8 | 36.8 | 33.9 | 35.3 | 36.6 |
| 11 | 36.9 | 37.0 | 33.7 | 35.3 | 36.2 |
| 12 | 36.9 | 36.9 | 33.9 | 34.7 | 35.9 |
| 13 | 37.2 | 37.2 | 36.4 | 37.6 | 37.7 |
| 14 | 36.3 | 36.3 | 34.6 | 33.6 | 35.8 |
| 15 | 36.3 | 36.3 | 34.3 | 34.2 | 36.2 |
| 16 | 36.3 | 36.3 | 35.1 | 34.3 | 36.4 |
| 17 | 38.9 | 38.9 | 38.9 | 38.9 | 38.9 |
| 18 | 36.0 | 36.0 | 34.4 | 34.1 | 35.5 |
| 19 | 36.3 | 35.8 | 33.6 | 34.3 | 35.7 |
| 20 | 36.1 | 35.8 | 32.5 | 32.9 | 34.8 |
| 21 | 37.2 | 36.8 | 33.9 | 36.9 | 36.9 |
| 22 | 36.6 | 36.2 | 33.2 | 34.2 | 35.5 |
| 23 | 37.0 | 36.7 | 32.7 | 35.0 | 35.8 |
| 24 | 36.8 | 36.7 | 33.9 | 35.0 | 36.2 |
| 25 | 38.8 | 38.8 | 38.8 | 38.8 | 38.8 |
| 26 | 36.6 | 36.0 | 34.4 | 35.3 | 36.3 |
| 27 | 36.7 | 36.7 | 34.3 | 35.6 | 36.4 |
| 28 | 36.2 | 36.1 | 34.4 | 34.9 | 35.9 |
| 29 | 37.4 | 37.4 | 36.4 | 37.4 | 36.8 |
| 30 | 36.2 | 36.1 | 34.7 | 35.5 | 36.3 |
| 31 | 36.1 | 36.2 | 34.7 | 35.8 | 35.7 |
| 32 | 36.2 | 36.1 | 34.7 | 35.1 | 36.3 |

**Table 4.4: PSNR Values of Coded *"Miss America"* Sequence Using the MPEG-1 Coder (n=8 and p=8)**

| Frame # | FSA-MSE | FSA-MAE | FSA-PDC | PHODS | LSA-MPDC |
|---|---|---|---|---|---|
| 1 | 31.7 | 31.7 | 31.7 | 31.7 | 31.7 |
| 2 | 27.6 | 27.4 | 24.0 | 17.8 | 24.9 |
| 3 | 25.5 | 25.4 | 23.4 | 17.7 | 24.7 |
| 4 | 25.1 | 24.8 | 23.2 | 17.3 | 24.5 |
| 5 | 26.4 | 26.1 | 24.9 | 19.7 | 26.8 |
| 6 | 25.0 | 24.8 | 23.4 | 17.5 | 24.4 |
| 7 | 25.3 | 25.2 | 23.5 | 17.7 | 24.4 |
| 8 | 27.3 | 27.2 | 23.0 | 17.6 | 24.4 |
| 9 | 31.6 | 31.6 | 31.6 | 31.6 | 31.6 |
| 10 | 26.8 | 26.6 | 22.6 | 17.5 | 24.0 |
| 11 | 25.1 | 24.9 | 22.5 | 17.6 | 24.2 |
| 12 | 24.6 | 24.3 | 22.7 | 17.2 | 24.3 |
| 13 | 25.9 | 25.6 | 24.0 | 19.5 | 26.4 |
| 14 | 24.6 | 24.4 | 22.5 | 17.2 | 24.1 |
| 15 | 25.8 | 25.6 | 22.6 | 17.3 | 24.1 |
| 16 | 27.4 | 27.4 | 22.3 | 17.3 | 24.1 |
| 17 | 31.3 | 31.3 | 31.3 | 31.3 | 31.3 |
| 18 | 27.9 | 27.7 | 24.9 | 17.5 | 25.8 |
| 19 | 26.6 | 26.4 | 24.1 | 17.5 | 25.2 |
| 20 | 26.0 | 25.8 | 23.4 | 17.1 | 24.8 |
| 21 | 26.4 | 26.4 | 25.2 | 19.4 | 26.9 |
| 22 | 24.7 | 24.7 | 23.2 | 17.1 | 24.2 |
| 23 | 23.7 | 23.6 | 21.4 | 17.1 | 23.0 |
| 24 | 25.7 | 25.6 | 22.2 | 17.0 | 23.5 |

**Table 4.5: PSNR Values of Coded *"PingPong"* Sequence Using the MPEG-1 Coder (n=8 and p=8)**

| Frame # | FSA-MSE | FSA-MAE | FSA-PDC | PHODS | LSA-MPDC |
|---|---|---|---|---|---|
| 1 | 38.7 | 38.7 | 38.7 | 38.7 | 38.7 |
| 2 | 35.3 | 35.2 | 33.5 | 33.7 | 35.0 |
| 3 | 35.4 | 35.3 | 33.4 | 34.3 | 35.1 |
| 4 | 35.2 | 35.2 | 33.4 | 33.8 | 35.1 |
| 5 | 36.6 | 36.6 | 35.9 | 36.9 | 37.0 |
| 6 | 34.9 | 34.8 | 33.6 | 34.3 | 35.3 |
| 7 | 35.2 | 35.2 | 33.1 | 34.3 | 34.5 |
| 8 | 35.3 | 35.2 | 33.2 | 34.1 | 34.6 |
| 9 | 38.9 | 38.9 | 38.9 | 38.9 | 38.9 |
| 10 | 36.6 | 36.4 | 34.3 | 35.4 | 36.4 |
| 11 | 36.8 | 36.8 | 34.2 | 35.5 | 36.2 |
| 12 | 36.8 | 36.7 | 34.0 | 34.7 | 35.5 |
| 13 | 37.7 | 37.7 | 36.7 | 37.7 | 37.7 |
| 14 | 36.1 | 36.2 | 34.0 | 33.1 | 35.1 |
| 15 | 36.0 | 36.0 | 34.2 | 33.9 | 35.4 |
| 16 | 36.0 | 36.2 | 34.6 | 34.4 | 36.1 |
| 17 | 38.9 | 38.9 | 38.9 | 38.9 | 38.9 |
| 18 | 35.6 | 35.6 | 35.0 | 33.9 | 36.2 |
| 19 | 36.0 | 34.5 | 34.1 | 34.9 | 35.4 |
| 20 | 36.4 | 33.6 | 32.8 | 33.0 | 34.2 |
| 21 | 37.0 | 34.1 | 33.4 | 35.0 | 37.4 |
| 22 | 36.6 | 33.9 | 32.4 | 33.6 | 34.6 |
| 23 | 36.4 | 34.1 | 33.0 | 33.8 | 34.6 |
| 24 | 36.4 | 34.5 | 34.0 | 34.4 | 35.0 |
| 25 | 38.8 | 38.8 | 38.8 | 38.8 | 38.8 |
| 26 | 36.0 | 36.0 | 33.8 | 34.8 | 35.5 |
| 27 | 36.1 | 36.1 | 34.0 | 35.6 | 35.8 |
| 28 | 35.7 | 35.7 | 34.1 | 35.2 | 35.6 |
| 29 | 37.2 | 37.2 | 36.3 | 37.4 | 37.5 |
| 30 | 36.3 | 36.3 | 34.6 | 35.2 | 35.8 |
| 31 | 36.1 | 35.9 | 35.1 | 35.8 | 36.1 |
| 32 | 35.7 | 35.7 | 35.2 | 35.1 | 36.3 |

**Table 4.6: PSNR Values of Coded *"Miss America"* Sequence Using the MPEG-1 Coder (n=16 and p=16)**

| Frame # | FSA-MSE | FSA-MAE | FSA-PDC | PHODS | LSA-MPDC |
|---|---|---|---|---|---|
| 1 | 31.7 | 31.7 | 31.7 | 31.7 | 31.7 |
| 2 | 26.1 | 25.9 | 23.2 | 17.4 | 24.0 |
| 3 | 24.1 | 23.8 | 23.0 | 17.7 | 23.7 |
| 4 | 24.2 | 23.9 | 22.9 | 17.4 | 23.5 |
| 5 | 25.9 | 25.5 | 24.9 | 20.0 | 26.7 |
| 6 | 24.0 | 23.5 | 22.2 | 17.6 | 23.3 |
| 7 | 24.1 | 23.6 | 22.4 | 17.6 | 22.9 |
| 8 | 25.4 | 25.2 | 22.9 | 17.5 | 22.5 |
| 9 | 31.6 | 31.6 | 31.6 | 31.6 | 31.6 |
| 10 | 25.2 | 25.1 | 22.6 | 17.1 | 22.7 |
| 11 | 23.7 | 23.6 | 22.2 | 17.4 | 22.5 |
| 12 | 23.6 | 23.4 | 22.2 | 17.1 | 22.5 |
| 13 | 25.4 | 25.1 | 24.6 | 20.0 | 26.2 |
| 14 | 23.6 | 23.5 | 22.6 | 17.2 | 22.7 |
| 15 | 24.5 | 24.3 | 21.9 | 17.3 | 22.2 |
| 16 | 26.1 | 26.1 | 22.0 | 17.0 | 22.1 |
| 17 | 31.3 | 31.3 | 31.3 | 31.3 | 31.3 |
| 18 | 26.3 | 26.2 | 24.5 | 17.5 | 24.0 |
| 19 | 25.2 | 25.0 | 23.8 | 17.5 | 23.7 |
| 20 | 24.9 | 24.4 | 23.2 | 17.2 | 23.4 |
| 21 | 25.9 | 25.5 | 24.9 | 20.2 | 26.7 |
| 22 | 23.7 | 23.5 | 22.5 | 17.4 | 22.9 |
| 23 | 22.9 | 22.8 | 21.8 | 17.1 | 21.6 |
| 24 | 25.0 | 24.9 | 22.4 | 16.9 | 22.3 |

**Table 4.7: PSNR Values of Coded *"PingPong"* Sequence Using the MPEG-1 Coder (n=16 and p=16)**

| Frame # | FSA-MSE | FSA-MAE | FSA-PDC | PHODS | LSA-MPDC |
|---|---|---|---|---|---|
| 1 | 38.7 | 38.7 | 38.7 | 38.7 | 38.7 |
| 2 | 36.4 | 36.4 | 33.2 | 33.5 | 35.2 |
| 3 | 36.2 | 36.1 | 33.1 | 34.2 | 35.3 |
| 4 | 36.0 | 36.0 | 33.4 | 34.0 | 35.5 |
| 5 | 37.4 | 37.4 | 35.8 | 36.8 | 37.0 |
| 6 | 36.1 | 36.0 | 33.3 | 34.2 | 35.1 |
| 7 | 36.4 | 36.4 | 33.1 | 34.9 | 35.0 |
| 8 | 36.1 | 36.1 | 33.1 | 34.3 | 35.2 |
| 9 | 38.9 | 38.9 | 38.9 | 38.9 | 38.9 |
| 10 | 36.8 | 36.8 | 34.3 | 35.3 | 36.4 |
| 11 | 36.8 | 36.9 | 34.1 | 35.2 | 35.7 |
| 12 | 36.7 | 36.7 | 33.9 | 34.4 | 35.8 |
| 13 | 37.5 | 37.5 | 36.9 | 37.6 | 37.6 |
| 14 | 36.3 | 36.4 | 33.8 | 33.7 | 35.7 |
| 15 | 36.3 | 36.3 | 34.4 | 34.3 | 35.9 |
| 16 | 36.3 | 36.3 | 34.6 | 34.5 | 36.1 |
| 17 | 38.9 | 38.9 | 38.9 | 38.9 | 38.9 |
| 18 | 35.6 | 35.7 | 34.7 | 34.1 | 35.1 |
| 19 | 35.8 | 35.4 | 34.0 | 34.7 | 35.6 |
| 20 | 35.8 | 35.4 | 32.5 | 33.3 | 34.5 |
| 21 | 36.8 | 36.5 | 34.6 | 36.5 | 37.1 |
| 22 | 35.9 | 35.5 | 33.0 | 34.2 | 35.4 |
| 23 | 36.8 | 36.3 | 33.0 | 34.7 | 35.4 |
| 24 | 36.0 | 35.7 | 34.0 | 34.8 | 35.8 |
| 25 | 38.8 | 38.8 | 38.8 | 38.8 | 38.8 |
| 26 | 36.5 | 36.4 | 34.0 | 35.2 | 35.6 |
| 27 | 36.6 | 36.5 | 34.0 | 35.2 | 35.4 |
| 28 | 36.1 | 36.0 | 34.0 | 34.9 | 36.2 |
| 29 | 37.0 | 37.0 | 36.4 | 37.0 | 37.0 |
| 30 | 36.7 | 36.1 | 34.6 | 34.9 | 36.1 |
| 31 | 36.0 | 35.5 | 34.6 | 35.4 | 35.9 |
| 32 | 36.2 | 36.0 | 34.6 | 34.8 | 36.3 |

**Table 4.8: PSNR Values of Coded *"Miss America"* Sequence Using the MPEG-1 Coder (Block Size 16 x 8 and p=8)**

| Frame # | FSA-MSE | FSA-MAE | FSA-PDC | PHODS | LSA-MPDC |
|---|---|---|---|---|---|
| 1 | 31.7 | 31.7 | 31.7 | 31.7 | 31.7 |
| 2 | 26.4 | 26.3 | 23.4 | 17.6 | 23.7 |
| 3 | 24.4 | 24.3 | 23.2 | 17.5 | 23.8 |
| 4 | 24.2 | 24.0 | 23.1 | 17.2 | 23.6 |
| 5 | 25.7 | 25.6 | 24.8 | 19.4 | 26.2 |
| 6 | 24.3 | 24.2 | 22.9 | 17.5 | 23.6 |
| 7 | 24.7 | 24.4 | 22.6 | 17.5 | 23.4 |
| 8 | 26.5 | 26.3 | 23.3 | 17.2 | 23.7 |
| 9 | 31.6 | 31.6 | 31.6 | 31.6 | 31.6 |
| 10 | 25.9 | 25.7 | 22.5 | 17.3 | 23.1 |
| 11 | 24.1 | 24.0 | 22.5 | 17.4 | 23.3 |
| 12 | 23.7 | 23.5 | 22.2 | 17.0 | 22.8 |
| 13 | 25.2 | 24.9 | 24.0 | 19.1 | 25.8 |
| 14 | 23.8 | 23.6 | 22.5 | 17.2 | 23.1 |
| 15 | 24.6 | 24.5 | 21.9 | 17.3 | 22.9 |
| 16 | 26.4 | 26.2 | 22.0 | 16.8 | 22.9 |
| 17 | 31.3 | 31.3 | 31.3 | 31.3 | 31.3 |
| 18 | 26.5 | 26.6 | 24.1 | 17.3 | 24.1 |
| 19 | 25.4 | 25.3 | 23.6 | 17.3 | 23.9 |
| 20 | 25.2 | 25.0 | 23.0 | 17.0 | 23.7 |
| 21 | 25.9 | 25.7 | 25.0 | 19.2 | 26.4 |
| 22 | 24.0 | 23.8 | 23.0 | 17.1 | 23.2 |
| 23 | 23.2 | 23.1 | 21.5 | 17.1 | 22.0 |
| 24 | 25.2 | 25.1 | 22.1 | 17.0 | 22.5 |

**Table 4.9: PSNR Values of Coded *"PingPong"* Sequence Using the MPEG-1 Coder (Block Size 16 x 8 and p=8)**

Figure 4.9: **Original Frame No. 6 of the Miss America Sequence**

Figure 4.10: **Reconstructed Frame No. 6 of the Miss America Sequence Using FSA-MSE in the MPEG Coder at 1.18 Mbit/s, PSNR = 34.9 dB,** $n = 16$ **and** $p = 8$

Figure 4.11: **Reconstructed Frame No. 6 of the Miss America Sequence Using FSA-MAE in the MPEG Coder at 1.18 Mbit/s, PSNR = 34.8 dB,** $n = 16$ **and** $p = 8$

Figure 4.12: **Reconstructed Frame No. 6 of the Miss America Sequence Using FSA-PDC in the MPEG Coder at 1.18 Mbit/s, PSNR = 35.0 dB,** $n = 16$ **and** $p = 8$

Figure 4.13: **Reconstructed Frame No. 6 of the Miss America Sequence Using PHODS in the MPEG Coder at 1.18 Mbit/s, PSNR = 34.3 dB,** $n = 16$ **and** $p = 8$

Figure 4.14: **Reconstructed Frame No. 6 of the Miss America Sequence Using LSA-MPDC in the MPEG Coder at 1.18 Mbit/s, PSNR = 35.3 dB,** $n = 16$ **and** $p = 8$

# Chapter 5

# Fast Multiresolution Motion Estimation Scheme for Wavelet Transform Coding

We recall from chapter 2 that motion estimation/compensation is used as an efficient temporal prediction in many video coding schemes. However, the residual video signal (i. e. the difference between the motion compensated previous frame $(t-1)$ and the current frame $(t)$) tends to be highly nonstationary [11]. Hence it is necessary to use a smaller block size to reduce the nonstationarity of each block in the motion estimation process which in turn results in an increased number of reference blocks and hence a computationally expensive motion estimation algorithm. Recently, wavelet transform has emerged as a promising technique for image processing applications due to its flexibility in representing nonstationary video signals. The superior performance of wavelet transform compared to the DCT for video compression has been reported in [12], [13]. This is a result of the absence of blocking effects and mosquito noise in wavelet transform coding [14].

We recall from section 2.6.2 that wavelet transform decomposes an image into a pyramid structure of subimages with various resolutions corresponding to the different frequency bands [34], [35], [36]. Each level of the wavelet pyramid consists of three wavelet components, namely, the horizontal component $W^H$, the vertical component $W^V$, and the diagonal component $W^D$ (as shown in Figures 2.10 and 2.11). In other words, the wavelet representation provides a multiresolution/multifrequency representation of a signal with localization in both time and frequency domains. This property of the wavelet transform makes a nonstationary image signal easier to code since the signal is decomposed into a set of multiscaled wavelets, where each component becomes relatively more stationary.

Recently, Zafar *et al.* [11], [15] have proposed a multiresolution motion estimation scheme (MRME) for wavelet transform based video compression (reviewed in section 2.6.2). In the MRME scheme, the motion vectors at the highest level of the wavelet pyramid are first estimated using the conventional block matching based motion estimation algorithm. Then the motion vectors at the next level of the wavelet pyramid are predicted from the motion vectors of the preceding level which are refined at each step. For example, the motion vectors in $W_4^H$, $W_4^V$ and $W_4^D$ are predicted from the motion vectors in $W_8^H$, $W_8^V$ and $W_8^D$, respectively. We note that in

[11], [15] the motion vectors are estimated separately for each wavelet subimage at each level of the wavelet pyramid. Hence the MRME scheme is still a computationally intensive procedure.

In this chapter, we propose a fast multiresolution motion estimation (FMRME) scheme based on the wavelet transform for video compression. The proposed scheme exploits the cross-correlation between the subimages in the wavelet pyramid structure resulting in significant reductions in computational complexity of motion estimation. In FMRME, the set of orientation subimages at each level of the wavelet pyramid are first combined together into a single (all-orientation) subimage, and then the motion estimation is performed on the newly formed subimage. This contrasts with the MRME scheme where motion estimation is separately performed on all the individual wavelet subimages. The motion vectors of an all-orientation subimage at a lower level are predicted from the motion vectors at the higher (preceding) level and are refined at each step. The FMRME scheme reduces the search time for the motion vectors by 66% compared to MRME. In addition, the FMRME scheme has the advantage of significantly reduced side information for the description of the motion vectors. This corresponds to considerable reductions in the bit rate for transmission of motion vectors. Hence the FMRME based wavelet coding results in improvements in coding performance at a reduced complexity compared to the MRME based wavelet coding for video compression.

## 5.1 Fast Multiresolution Motion Estimation (FMRME) Scheme

We note that the orientation subimages (i.e. the horizontal component $W^H$, the vertical component $W^V$, and the diagonal component $W^D$) at the same level of the wavelet pyramid actually represent the same image in the same scale. Hence the motion vectors for different orientation subimages at each level of the wavelet pyramid describe the same part of an object in a scene in the same scale. In other words, the motion activities of the different wavelet subimages at the same pyramid level are highly correlated.

In the proposed scheme (FMRME), the correlations between the motion activities of the different wavelet subimages are exploited to reduce the computational complexity in motion

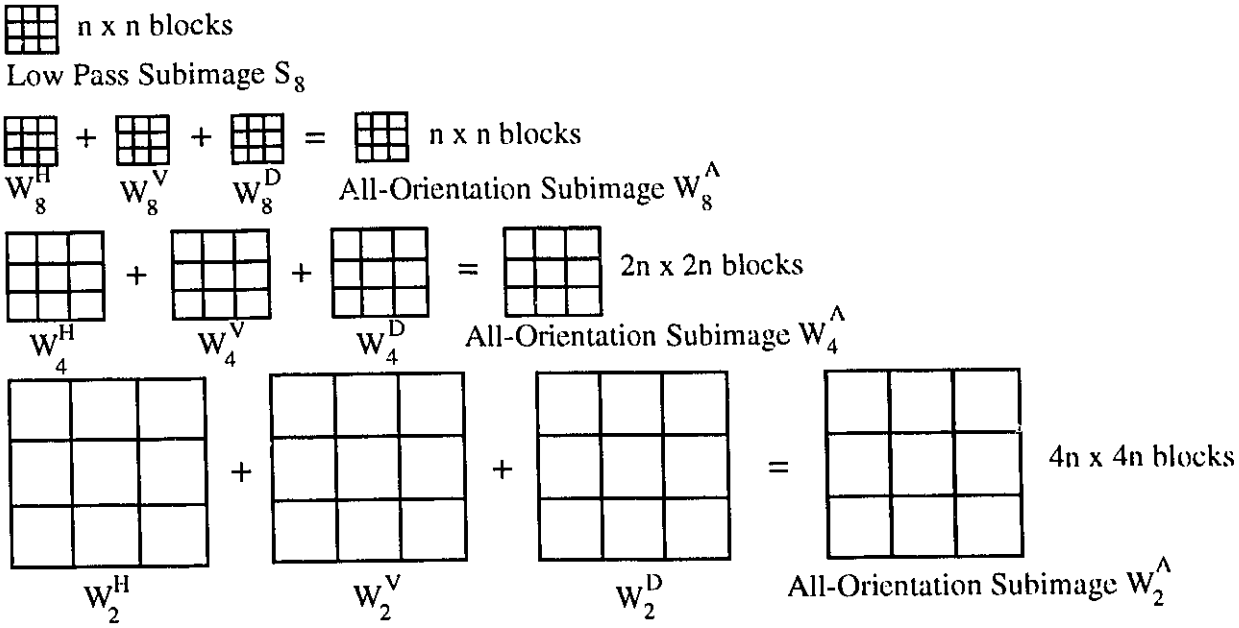



**Figure 5.1: Wavelet All-Orientation Subimages**

estimation. Here, the set of wavelet components at each pyramid level are combined into a single all-orientation subimage, i.e. $W_8^H$, $W_8^V$, $W_8^D$ are combined into $W_8^A$; $W_4^H$, $W_4^V$, $W_4^D$ are combined into $W_4^A$; $W_2^H$, $W_2^V$, $W_2^D$ are combined into $W_2^A$, etc. This process is illustrated in Figure 5.1. We note that the motion estimation is performed only on the all-orientation subimages ($W_8^A$, $W_4^A$, $W_2^A$ etc.). This contrasts the MRME scheme where the motion estimation is separately performed on all the individual wavelet subimages ($W_8^H$, $W_8^V$, $W_8^D$, etc.). Hence the FMRME scheme exploits the correlation between the motion vectors of the subimages in the same pyramid level.

In addition, the motion activities at different levels of the pyramid are also highly correlated since they actually characterize the same motion structure at different scales [11], [15]. Hence, in FMRME, the motion vectors are first determined for the low pass subimage $S_8$ and the all-orientation subimage $W_8^A$. The motion vectors of the all-orientation subimage in the lower levels of the wavelet pyramid are predicted from the motion vectors of the all-orientation subimage at the higher levels and are further refined at each step. In other words, the motion vectors of $W_4^A$ are predicted from the motion vectors of $W_8^A$ while the motion vectors of $W_2^A$ are predicted from those

of $W_4^A$. If a block matching based motion estimation algorithm is used, this process is illustrated by the following equation:

$$V_i(x,y) = 2V_{2i}(x,y) + \Delta_i(x,y) \qquad for\ i = 2,4 \qquad \text{--- (5.1)}$$

where $V_i(x,y)$ represents the motion vector of the reference block centered at $(x,y)$ for the all-orientation subimage $W_i^A$ and $\Delta_i(x,y)$ is the incremental motion vector which is determined within a reduced search area centered at $2V_{2i}(x,y)$. We note that the motion vectors $V_i$ can be determined using a variety of block matching algorithms such as FSA, OS [26], PHODS [28] (reviewed in chapter 2), etc. These motion vectors $V_i$ are then transmitted to the receiver. At the receiver, the motion vectors of the all-orientation subimage $W_i^A$ are assigned as the motion vectors of $W_i^H$, $W_i^V$ and $W_i^D$ in order to reconstruct each wavelet subimage. This process is illustrated as follows:

$$V_i^H(x,y) = V_i^V(x,y) = V_i^D(x,y) = V_i(x,y) \qquad for\ i = 2,4,8 \qquad \text{--- (5.2)}$$

An example of the FMRME scheme is given in Figure 5.2. The motion vectors of the all-orientation subimage $W_8^A$ are first determined using the full search block matching algorithm with a

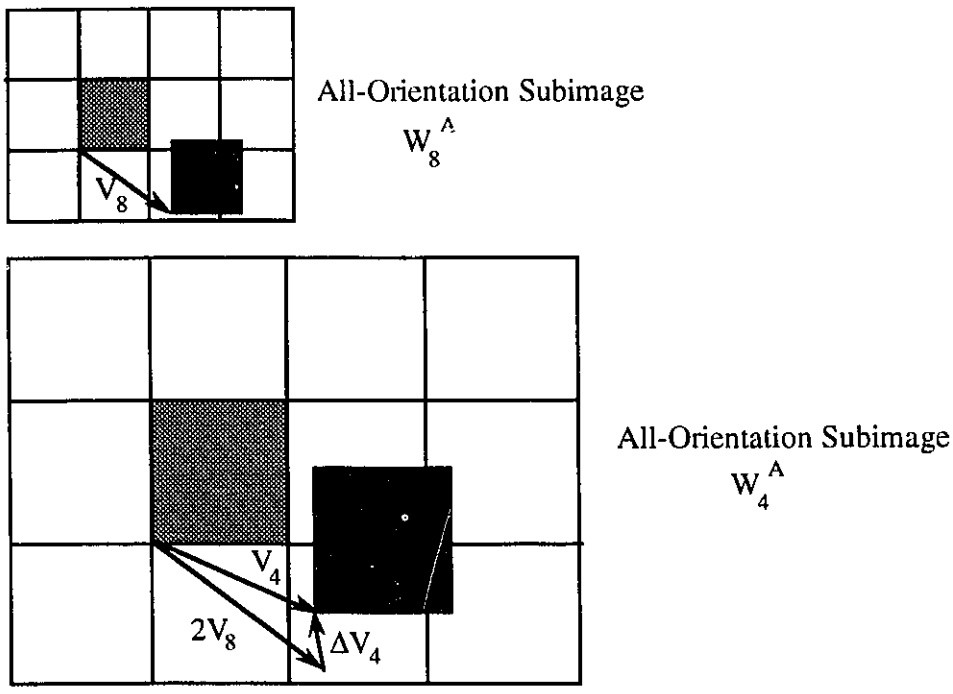



**Figure 5.2: Illustration of the Proposed FMRME Scheme**

maximum allowed displacement $p$. These motion vectors are then scaled appropriately and used as initial estimates for the motion vectors in $W_4^A$. The incremental motion vector $\Delta_4(x,y)$ is then calculated using equation (5.1) where the corresponding (reduced) search area is centered at $2V_8(x,y)$. We note that the size of the blocks are scaled to correspond to the level of the wavelet pyramid, i.e. the blocks in all-orientation subimages $W_8^A$, $W_4^A$ and $W_2^A$ have sizes of $n \times n$, $2n \times 2n$ and $4n \times 4n$, respectively. With this structure, the number of blocks in the all-orientation subimages is a constant and hence there is a one to one correspondence between a block at one resolution level in the wavelet pyramid and a block at the same position (object) at another resolution level.

## 5.2 Performance Analysis

The efficiency of wavelet transform coding can be measured using the entropy [68] of the transformed images which is defined as follows:

$$H(X) = -\frac{1}{MN}\sum_{all\,x} p(x)\log_2 p(x) \quad bits\,/\,pixel \qquad \text{--- (5.3)}$$

where $X$ is a video frame of size $M \times N$ pixels and $p(x)$ is the probability of occurrence of luminance intensity of $x$. This gives the average information of the video frame $X$. In other words, $H(X)$ is the lower bound entropy for the lossless compression (i.e. the video frame can be recovered perfectly) of $X$.

In this thesis, a simple wavelet transform based video coder (Figure 5.3) is used to evaluate the performance of the FMRME and MRME motion estimation schemes. We recall from chapter 2 that this wavelet transform based coder consists of four major modules, namely, wavelet transform, quantization, motion estimation and variable length coding (reviewed in section 2.6.2). We note that the quantized transform coefficients together with the motion vectors are first variable length coded (lossless) and then transmitted to the receiver. Since entropy is the measure of the lower bound of the compression ratio for a lossless compression scheme, the total transmission bit rate can be calculated as the sum of the entropy of the quantized wavelet coefficients (wavelet transformed image) and the motion vectors:

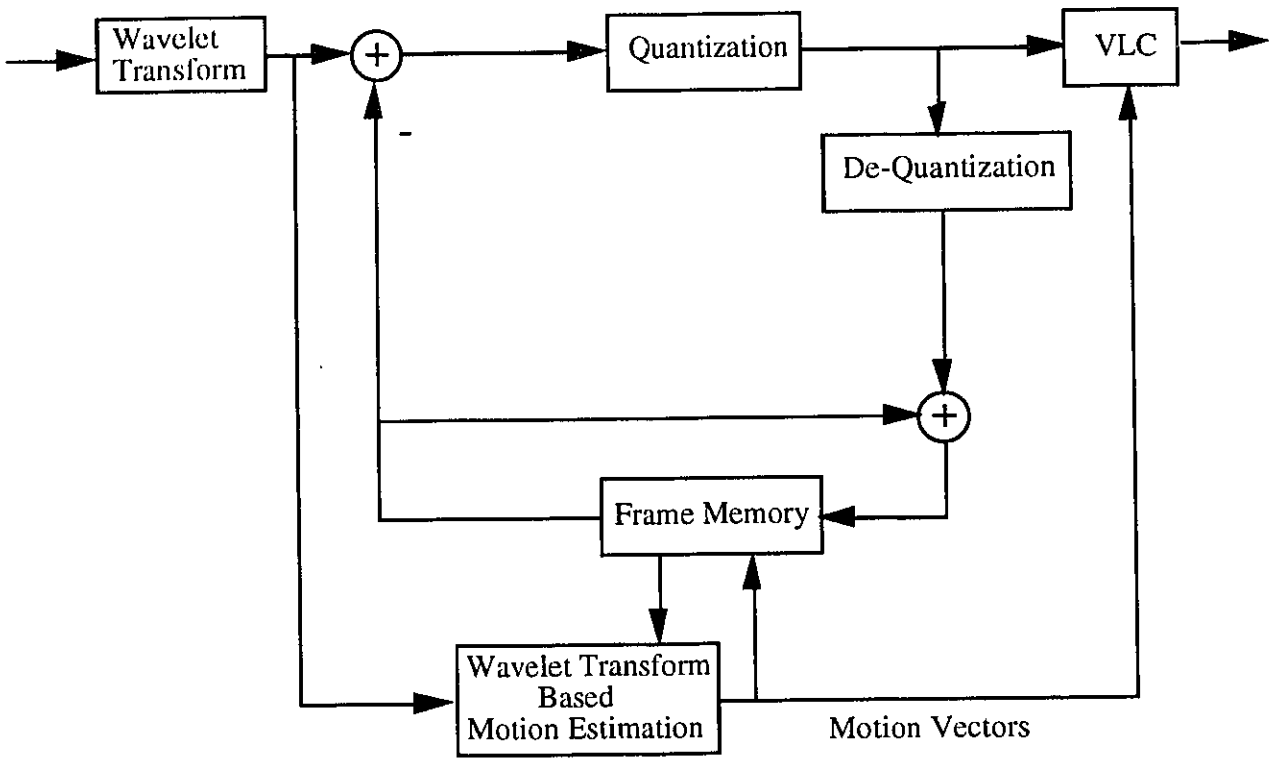



**Figure 5.3: Motion Compensated Wavelet Transform Coder**

Bit Rate(Quantized Wavelet Coefficients) --- (5.4)
= Frame_Rate * M * N * (Average Entropy of Quantized Wavelet Coefficients)

Bit Rate(Motion Vectors) --- (5.5)
= Frame_Rate * (Number of Motion Vectors) * (Average Entropy of Motion Vectors)

and the total bit rate is given as follows:

Total Bit Rate = Bit Rate(Quantized Wavelet Coefficients) --- (5.6).
+ Bit Rate(Motion Vectors)

We note that there is one motion vector associated with each block of each subimage in the MRME scheme. On the other hand, only one motion vector is associated with each block of each all-orientation subimage in the FMRME scheme. The total number of motion vectors required to be transmitted to the receiver for the two schemes are as follows:

For the MRME scheme:

$$number\ of\ motion\ vectors = \frac{M}{2^L * n} * \frac{N}{2^L * n} * (3L+1) \quad \text{--- (5.7)}$$

where $L$ is the level of the wavelet decomposition and $n$ is block size.

For the FMRME scheme:

$$number\ of\ motion\ vectors\ =\ \frac{M}{2^L * n} * \frac{N}{2^L * n} * (L+1) \qquad \text{--- (5.8).}$$

This corresponds to a reduction of number of motion vectors by a factor of $(3L+1)/(L+1)$. For example, for a video frame of size $360 \times 288$ pixels with a 3-level wavelet decomposition ( $L = 3$) and a block size of 3 ( $n = 3$), the total number of motion vectors for MRME and FMRME are 1800 and 720, respectively. Since the bit rate is a function of the number of motion vectors required to be transmitted (equations (5.5) and (5.6)), the FMRME based wavelet transform coding results in significant improvements in coding performance compared to the MRME based wavelet coding for video compression.

In addition, we note that the FMRME scheme performs the motion estimation only on the all-orientation subimages. This contrasts with the MRME scheme where motion estimation is separately performed on all the individual wavelet subimages. The total number of subimages required to be executed for the two schemes are as follows:

For the MRME scheme:

$$number\ of\ subimages = 3L+1 \qquad \text{--- (5.9).}$$

For the FMRME scheme:

$$number\ of\ subimages = L+1 \qquad \text{--- (5.10).}$$

Hence the speed up factor of the FMRME scheme is $(3L+1)/(L+1)$. If $L >> 1$, this corresponds to a reduction of the search time by 66% of that of MRME.

## 5.3 Simulation Results

Simulations have been performed using the standard test sequences, *"Miss America"* and *"PingPong"* (detailed in section 4.2). The full search algorithm using MAE as the matching criterion (FSA-MAE) is used to obtain the motion vectors. The block size for the low pass subimage $S_8$ and the all-orientation subimage $W_8$ are $3 \times 3$ pixels and the corresponding maximum allowed displacement $p$ is 4 pixels. The maximum allowed displacements for all-orientation subimages $W_4$ and $W_2$ are 2 and 1 pixels, respectively. The coding performance is measured

using the peak signal to noise ratio (PSNR) and entropy. Four sets of experiments have been performed which are detailed as follows:

In the first set of experiments, the FMRME and MRME schemes have been simulated to compare their estimation accuracy. The entropy of the motion compensated frame difference for *"Miss America"* and *"PingPong"* are tabulated in Table 5.1. The corresponding entropy vs frame number is plotted in Figures 5.4. It can be seen that the entropy of FMRME is marginally higher than the MRME scheme (approximately 3% and 9% higher for "*Miss America*" and "*PingPong*", respectively). This is mainly a result of the motion estimation performed separately for each wavelet subimage by the MRME scheme at the expense of a significantly higher computational complexity.

We recall that the motion vectors for different orientation subimages at each level of wavelet pyramid are highly correlated since they actually describe the same part of an object in a scene. This can be seen from Table 5.2 where the motion vectors of blocks #85-99 for $W_8^H$, $W_8^V$ and $W_8^D$ for the frame No. 11 of "*Miss America*" using the MRME scheme are tabulated. The corresponding motion vectors for the all-orientation subimage $W_8^A$ using the FMRME scheme are tabulated in Table 5.3. It can be seen from Tables 5.2 and 5.3 that the motion vectors are similar which confirm that the performance of FMRME is comparable to MRME without significant degradation in the estimation accuracy of the motion vectors.

The execution time for determining the motion vectors for frame No. 11 of *"Miss America"* is tabulated in Table 5.4. The actual speed up of the proposed scheme is less than 3 (equations (5.9) and (5.10)) as a result of the additional overheads involved for combining the wavelet components into a single all-orientation subimage.

In experiments #2-5, the motion compensated wavelet transform coder as shown in Figure 5.3 has been simulated using the MRME and FMRME schemes for motion estimation. We note only the first frame of the test sequence is intra-frame coded while the rest of the frames are inter-frame coded. In our experiments, for intra-frame coding, a fixed quantization threshold $T$

(equation (2.12)) of value 0 is used for all subimages while the quantizer step sizes $D_m$'s (equation (2.13)) were 1, 1 and 2 for the wavelet pyramid level 3, 2 and 1, respectively.

In experiment #2, the performance of MRME is compared with that of FMRME for a motion compensated wavelet transform coder. A fixed threshold $T$ of value 2 is used while the $D_m$'s were 2, 4 and 8 for pyramid level of 3, 2 and 1, respectively. The PSNR of the reconstructed frames are tabulated in Tables 5.5 and 5.6 for the "*Miss America*" sequence. The corresponding PSNR vs frame number is plotted in Figure 5.5. We note that the performance of the FMRME based wavelet coding is comparable to that of MRME based wavelet coding (less than 0.5 dB difference). The entropies of the quantized wavelet coefficients (quantized wavelet transformed video frames) and the motion vectors are tabulated in Tables 5.5 and 5.6 (Figures 5.6-5.8). The bit rates are 0.85 Mbit/sec and 0.99 Mbit/sec for FMRME and MRME, respectively (equation (5.6)). This confirms that the FMRME based wavelet coding results in a considerable reduction (15%) in the bit rate compared to the MRME based wavelet coding. In other words, the FMRME based wavelet transform coding for video compression has superior coding performance.

In experiment #3, the performance of the motion compensated wavelet coder using FMRME scheme is compared with the MPEG-1 coder. The PSNR values of the reconstructed frames are tabulated in Table 5.7 for the *"Miss America"* sequence at 0.26 Mbit/sec. The corresponding PSNR vs frame number is plotted in Figure 5.9. It can be seen that the performance of the FMRME based wavelet transform coder is comparable to the MPEG-1 video coder. In addition, we note that the wavelet coder provides an approximately constant quality among all of the inter-coded frames. This contrasts with the MPEG-1 coder where the P frames have a better performance compared to the B frames. The subjective qualities of the reconstructed images (frame No. 7 of *"Miss America"*) are shown in Figures 5.15 and 5.16. It can be seen that the reconstructed image using wavelet transform based coding is free from block effects resulting in a superior subjective performance compared to the MPEG-1 coder.

In experiment #4, the effects of the quantization threshold $T$ on the performance of the FMRME based wavelet coding are examined. We note that any wavelet coefficient is discarded if

its value is smaller than $T$. A larger $T$ ($T = 10$) with the quantizer step sizes $D_m$'s of 2, 4, 8 was used. The entropy of the quantized wavelet coefficients and the motion vectors are tabulated in Table 5.8. The corresponding entropy vs frame number is plotted in Figure 5.10. It can be seen that there is a significant reduction in the entropy of the wavelet coefficients resulting in a smaller bit rate (0.70 Mbit/sec). However, the PSNR of the reconstructed frames (plotted in Figure 5.11) is approximately 2 dB lower than in experiment #2. This is because the slow motion in the "*Miss America*" sequence is captured accurately by the FMRME scheme. Hence the residual (error) frame (i.e. the difference between the current wavelet transformed frame and the motion compensated previous wavelet transformed frame) has smaller values, especially at the lower level of the wavelet pyramid (i.e. the high pass subimages). Hence the choice of the quantization threshold has a significant effect of the coding performance.

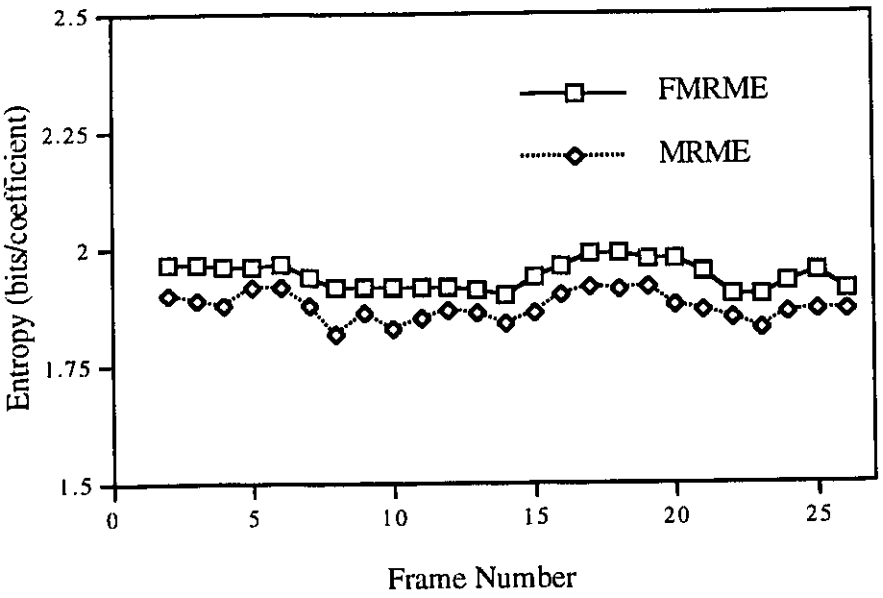
In experiment #5, simulations were performed for the quantizer step sizes $D_m$'s of 2, 4 and 32 for pyramid level 3, 2 and 1, respectively. The change in the quantizer step size is used to examine the effects on the performance of the FMRME based wavelet coding. The entropy of the quantized wavelet coefficients and the motion vectors are tabulated in Table 5.9. The corresponding entropy vs frame number is plotted in Figure 5.10. The bit rate is 0.82 Mbit /sec for the "*Miss America*" sequence. Once again, the slow motion in the *"Miss America"* sequence is captured accurately by the FMRME scheme resulting in small wavelet coefficients in the residual (error) frame. Hence the choice of the quantizer step size for the high pass subimage has a marginal effect on the coding performance. This can be seen from the PSNR of the reconstructed frames (plotted in Figure 5.11) which is similar to that of experiment #2.

## 5.4 Summary

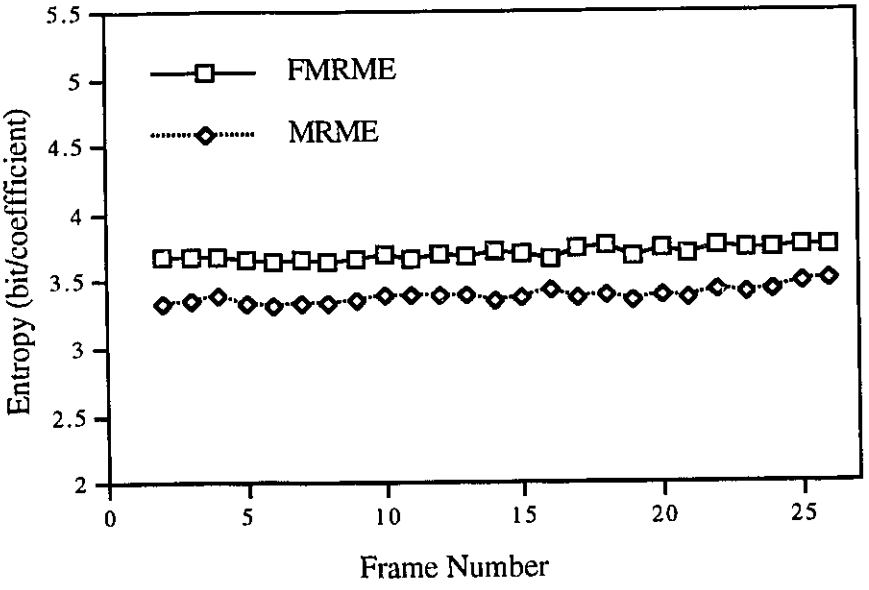
A new and efficient block based multiresolution motion estimation scheme for video compression has been presented. The proposed scheme combines the individual wavelet subimages into a single all-orientation subimage on which the motion estimation is performed. This reduces the search time by 66% compared to the MRME approach where the motion vectors

for each wavelet subimage are determined separately. In addition, the side information for the description of motion vectors are also significantly reduced. Hence the FMRME based wavelet transform coding technique has superior coding performance. Our simulation results show that the FMRME based wavelet coder has a reduction of bit rate by 15% compared to the MRME based wavelet coder. In addition, simulation results demonstrate that the proposed scheme has a comparable accuracy of estimation of motion vectors compared to the MRME at a reduced computationally complexity.

We recall from chapter 2 that the MPEG-1 coder employs the concept of bidirectional motion estimation and bidirectional Interpolated frames (B frames) resulting in a reduction in the bit rate. If we incorporate the same concepts in the motion compensated wavelet transform coder, this should further reduce the bit rate of the coder. In addition, we have only used scalar quantization to compress the wavelet coefficients (wavelet transformed video frames) in our simulations. If we employ a more novel technique such as non-linear interpolative vector quantization [70] to exploit the cross-correlations in the wavelet coefficients, this will further improve the performance (bit rate) of the FMRME based wavelet transform coding for video compression [33].




(a) *"Miss America"*




(b) *"PingPong"*

**Figure 5.4: Entropies of the Motion Compensated Frame Difference of *"Miss America"* and *"PingPong"***

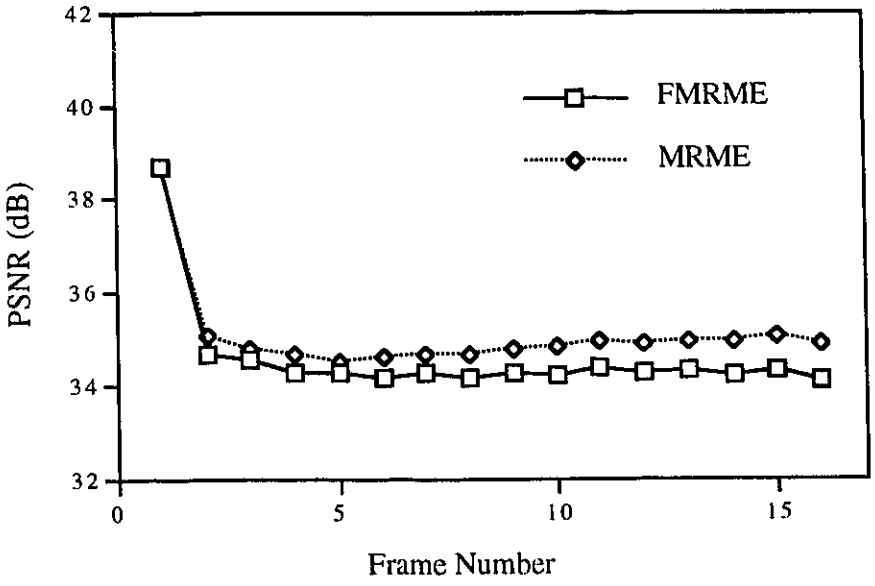



**Figure 5.5: PSNR Values of Coded *"Miss America"* Using the Motion Compensated Wavelet Transform Coder**

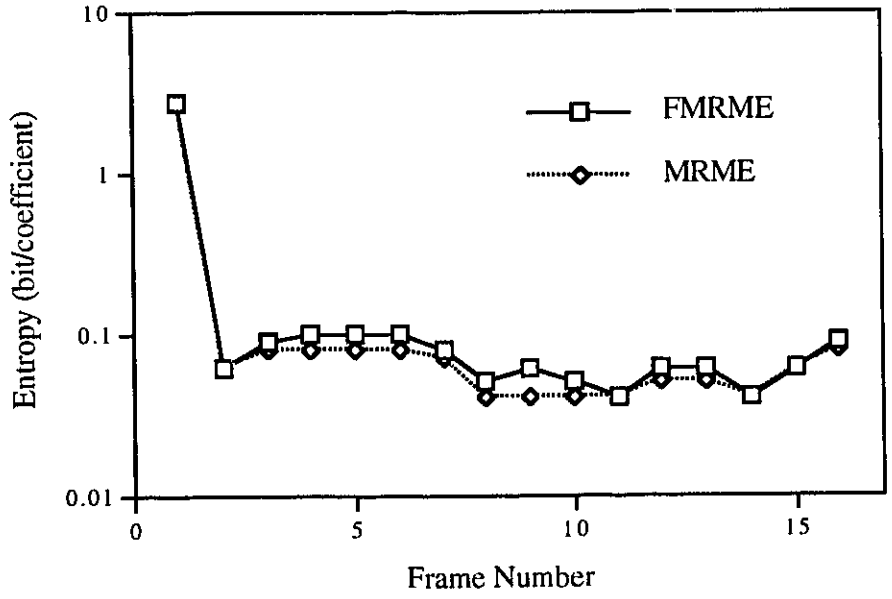



**Figure 5.6: Entropies of the Quantized Wavelet Transform Coefficients of *"Miss America"* Using the Motion Compensated Wavelet Transform Coder**

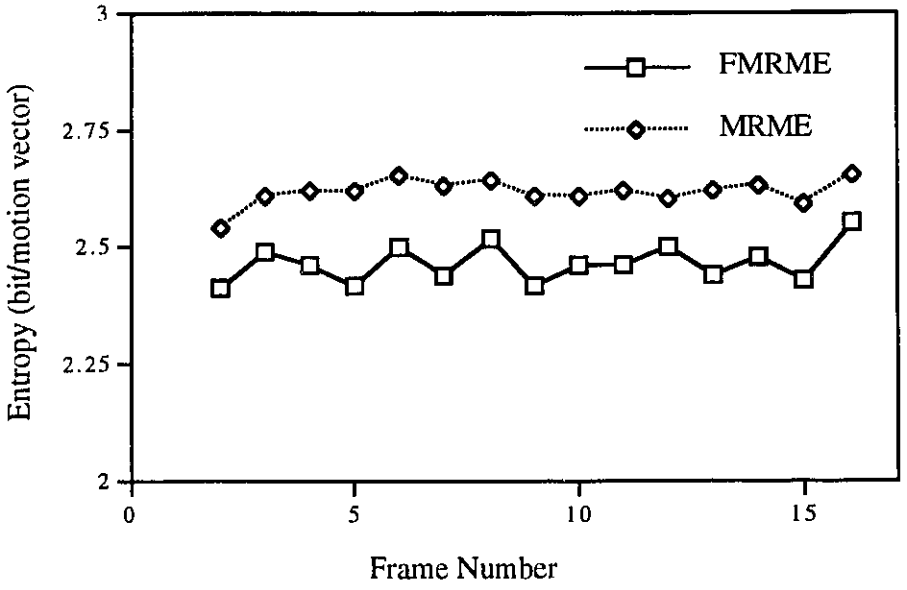



**Figure 5.7: Entropies of the Motion Vectors in the Horizontal Direction of *"Miss America"* Using the Motion Compensated Wavelet Transform Coder**

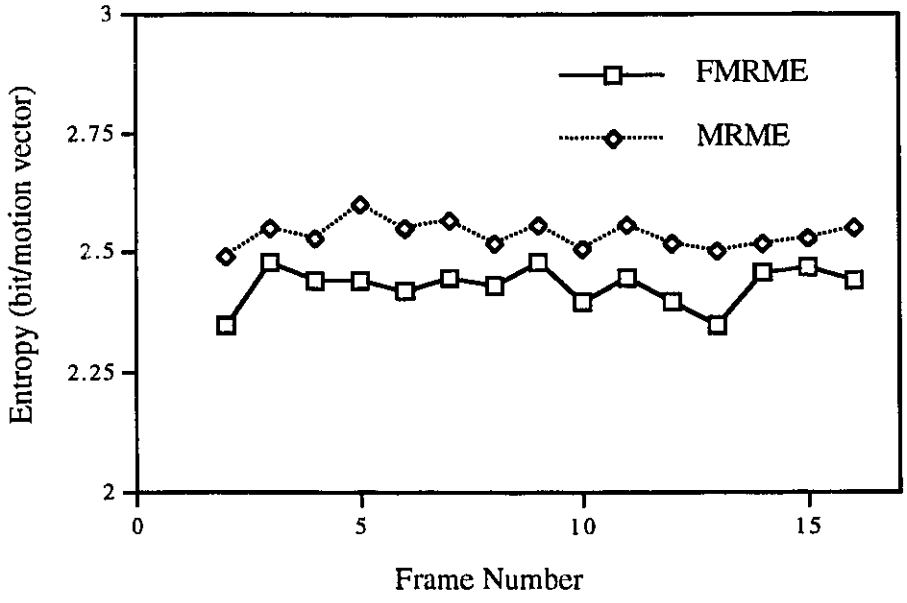



**Figure 5.8: Entropies of the Motion Vectors in the Vertical Direction of *"Miss America"* Using the Motion Compensated Wavelet Transform Coder**

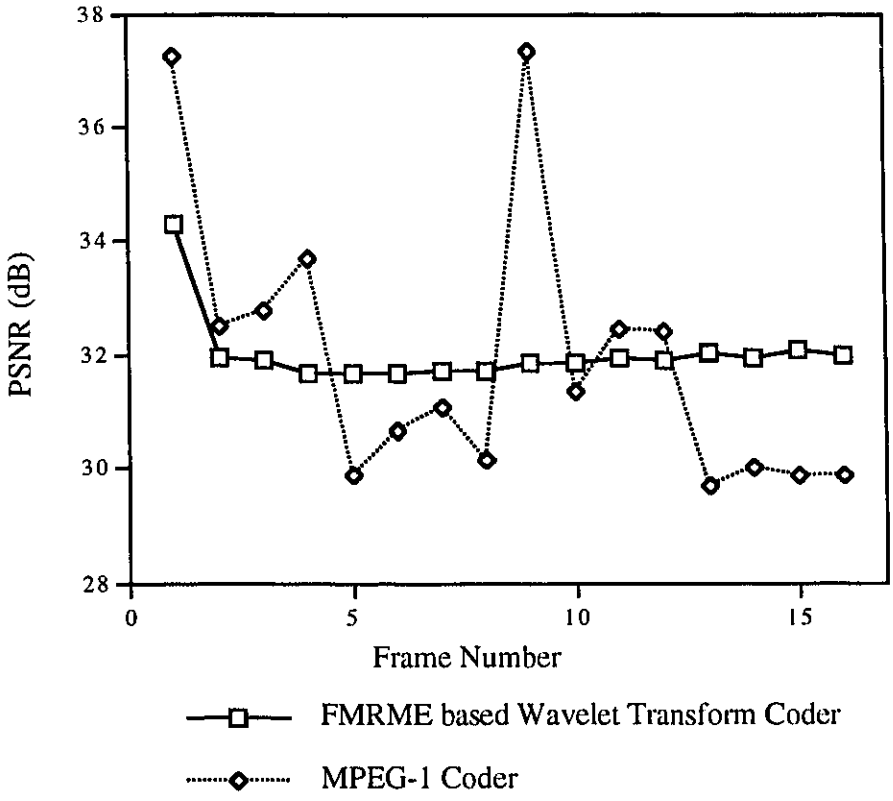



**Figure 5.9: PSNR Values of Coded *"Miss America"* Using FMRME Based Wavelet Transform Coder and MPEG-1 Coder at 0.26 Mbit/sec**

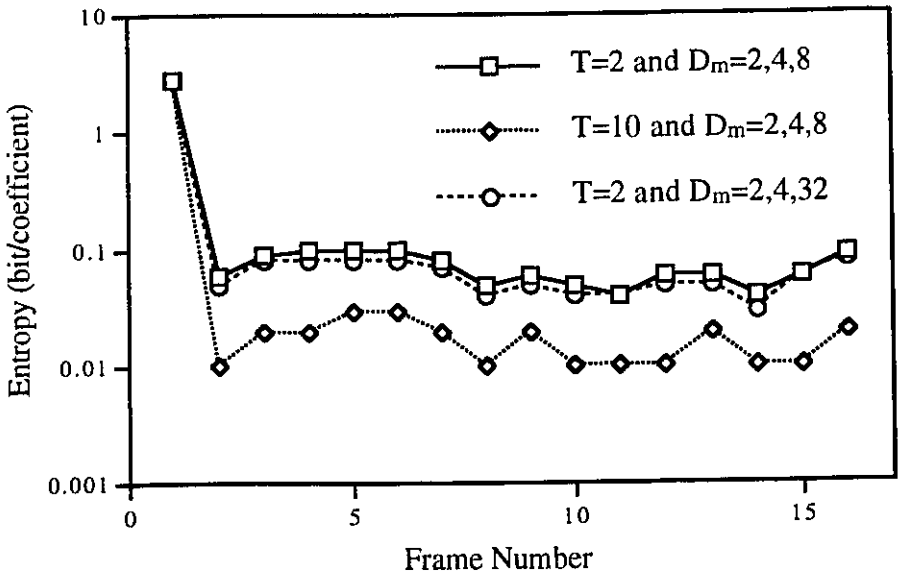



**Figure 5.10: Entropies of the Quantized Wavelet Transform Coefficients Values of *"Miss America"* Using FMRME Based Wavelet Transform Coder with Different Quantization Threshold and Step Size Values**

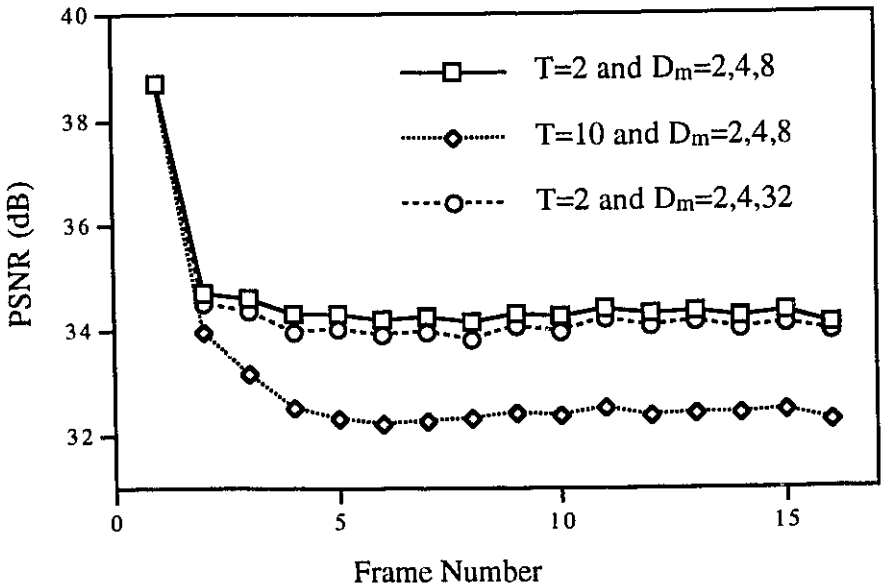



**Figure 5.11: PSNR Values of Coded *"Miss America"* Using FMRME Based Wavelet Transform Coder with Different Quantization Threshold and Step Size Values**

| Frame # | "*Miss America*" Sequence | | "*PingPong*" Sequence | |
|---|---|---|---|---|
| | MRME | FMRME | MRME | FMRME |
| 2 | 1.97 | 1.90 | 3.68 | 3.33 |
| 3 | 1.97 | 1.89 | 3.68 | 3.35 |
| 4 | 1.96 | 1.88 | 3.68 | 3.39 |
| 5 | 1.96 | 1.92 | 3.66 | 3.34 |
| 6 | 1.97 | 1.92 | 3.64 | 3.32 |
| 7 | 1.94 | 1.88 | 3.66 | 3.34 |
| 8 | 1.92 | 1.82 | 3.64 | 3.34 |
| 9 | 1.92 | 1.86 | 3.66 | 3.36 |
| 10 | 1.92 | 1.83 | 3.70 | 3.39 |
| 11 | 1.92 | 1.85 | 3.66 | 3.39 |
| 12 | 1.92 | 1.87 | 3.70 | 3.40 |
| 13 | 1.91 | 1.86 | 3.69 | 3.40 |
| 14 | 1.90 | 1.84 | 3.72 | 3.36 |
| 15 | 1.94 | 1.86 | 3.70 | 3.38 |
| 16 | 1.96 | 1.90 | 3.67 | 3.44 |
| 17 | 1.99 | 1.92 | 3.74 | 3.38 |
| 18 | 1.99 | 1.91 | 3.76 | 3.39 |
| 19 | 1.98 | 1.92 | 3.69 | 3.36 |
| 20 | 1.98 | 1.88 | 3.74 | 3.39 |
| 21 | 1.95 | 1.87 | 3.71 | 3.38 |
| 22 | 1.90 | 1.85 | 3.76 | 3.43 |
| 23 | 1.90 | 1.83 | 3.74 | 3.42 |
| 24 | 1.93 | 1.86 | 3.75 | 3.44 |
| 25 | 1.95 | 1.87 | 3.77 | 3.48 |
| 26 | 1.91 | 1.87 | 3.76 | 3.51 |

**Table 5.1: Entropies of the Motion Compensated Frame Difference of "*Miss America*" and "*PingPong*"**

| | Block Number | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 |
| $W_8^H$ | 0,0 | 0,0 | 4,-4 | 2,-3 | -1,-3 | 3,-1 | 2,3 | -3,3 | -1,-4 | -3,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| $W_8^V$ | 0,0 | 0,0 | 0,2 | 2,-4 | 0,0 | 0,1 | -1,-3 | 0,-4 | 2,-2 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |
| $W_8^D$ | 0,0 | 0,0 | 4,-1 | 0,-4 | -2,-4 | 3,-4 | 0,1 | -1,-3 | 0,-3 | -4,-4 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |

**Table 5.2: Motion Vectors for Frame No. 11 of "*Miss America*" Using the MRME Scheme**

| | Block Number | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 |
| $W_8^\Lambda$ | 0,0 | 0,0 | 1,-2 | 1,-4 | -4,-4 | 0,-4 | 0,-1 | -3,2 | 1,-4 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 | 0,0 |

**Table 5.3: Motion Vectors for Frame No. 11 of "*Miss America*" Using the FMRME Scheme**

| MRME Scheme | FMRME Scheme |
|---|---|
| 39.17 seconds | 17.87 seconds |

**Table 5.4: Execution Time for Frame No. 11 of "*Miss America*"**

| Frame # | PSNR | Entropy Wavelet Coefficients | Entropy Motion Vectors - Horizontal Direction | Entropy Motion Vectors - Vertical Direction |
|---|---|---|---|---|
| 1 | 38.67 | 2.78 | – | – |
| 2 | 34.66 | 0.06 | 2.41 | 2.35 |
| 3 | 34.57 | 0.09 | 2.49 | 2.48 |
| 4 | 34.29 | 0.10 | 2.46 | 2.44 |
| 5 | 34.30 | 0.10 | 2.42 | 2.44 |
| 6 | 34.17 | 0.10 | 2.50 | 2.42 |
| 7 | 34.25 | 0.08 | 2.44 | 2.45 |
| 8 | 34.14 | 0.05 | 2.52 | 2.43 |
| 9 | 34.30 | 0.06 | 2.42 | 2.48 |
| 10 | 34.22 | 0.05 | 2.46 | 2.40 |
| 11 | 34.38 | 0.04 | 2.46 | 2.45 |
| 12 | 34.27 | 0.06 | 2.50 | 2.40 |
| 13 | 34.33 | 0.06 | 2.44 | 2.35 |
| 14 | 34.24 | 0.04 | 2.48 | 2.46 |
| 15 | 34.31 | 0.06 | 2.43 | 2.47 |
| 16 | 34.10 | 0.09 | 2.55 | 2.44 |
| Average | 34.58 | 0.239 | 2.47 | 2.43 |

**Table 5.5: Simulation Results of "*Miss America*" Using FMRME Based Wavelet Transform Video Coder with Quantization Threshold T = 2 and Quantizer Step Sizes $D_m$'s = 2, 4 and 8**

| Frame # | PSNR | Entropy Wavelet Coefficients | Entropy Motion Vectors - Horizontal Direction | Entropy Motion Vectors - Vertical Direction |
|---|---|---|---|---|
| 1 | 38.67 | 2.78 | – | – |
| 2 | 35.05 | 0.06 | 2.54 | 2.49 |
| 3 | 34.79 | 0.08 | 2.61 | 2.55 |
| 4 | 34.64 | 0.08 | 2.62 | 2.53 |
| 5 | 34.50 | 0.08 | 2.62 | 2.60 |
| 6 | 34.60 | 0.08 | 2.65 | 2.55 |
| 7 | 34.64 | 0.07 | 2.63 | 2.57 |
| 8 | 34.68 | 0.04 | 2.64 | 2.52 |
| 9 | 34.80 | 0.04 | 2.61 | 2.56 |
| 10 | 34.82 | 0.04 | 2.61 | 2.51 |
| 11 | 34.93 | 0.04 | 2.62 | 2.56 |
| 12 | 34.87 | 0.05 | 2.60 | 2.52 |
| 13 | 34.92 | 0.05 | 2.62 | 2.50 |
| 14 | 34.96 | 0.04 | 2.63 | 2.52 |
| 15 | 35.04 | 0.06 | 2.59 | 2.53 |
| 16 | 34.88 | 0.08 | 2.65 | 2.55 |
| Average | 35.05 | 0.229 | 2.62 | 2.54 |

**Table 5.6: Simulation Results of "*Miss America*" Using MRME Based Wavelet Transform Video Coder with Quantization Threshold T = 2 and Quantizer Step Sizes $D_m$'s = 2, 4 and 8**

| Frame # | FMRME Based Wavelet Transform Video Coder | MPEG-1 Video Coder |
|---|---|---|
| 1 | 34.3 | 37.24 |
| 2 | 31.93 | 32.51 |
| 3 | 31.90 | 32.76 |
| 4 | 31.66 | 33.71 |
| 5 | 31.68 | 29.85 |
| 6 | 31.68 | 30.66 |
| 7 | 31.72 | 31.06 |
| 8 | 31.71 | 30.15 |
| 9 | 31.87 | 37.35 |
| 10 | 31.84 | 31.36 |
| 11 | 31.96 | 32.46 |
| 12 | 31.92 | 32.39 |
| 13 | 32.02 | 29.70 |
| 14 | 31.93 | 30.02 |
| 15 | 32.06 | 29.89 |
| 16 | 31.98 | 29.89 |

**Table 5.7: PSNR Values of Coded "*Miss America*" Using FMRME Based Wavelet Transform Video Coder and MPEG-1 Coder at 0.26 Mbit/sec**

| Frame # | PSNR | Entropy Wavelet Coefficients | Entropy Motion Vectors - Horizontal Direction | Entropy Motion Vectors - Vertical Direction |
|---|---|---|---|---|
| 1 | 38.67 | 2.78 | – | – |
| 2 | 33.95 | 0.01 | 2.41 | 2.35 |
| 3 | 33.13 | 0.02 | 2.55 | 2.53 |
| 4 | 32.48 | 0.02 | 2.52 | 2.52 |
| 5 | 32.30 | 0.03 | 2.47 | 2.52 |
| 6 | 32.20 | 0.03 | 2.54 | 2.48 |
| 7 | 32.21 | 0.02 | 2.54 | 2.51 |
| 8 | 32.26 | 0.01 | 2.57 | 2.46 |
| 9 | 32.39 | 0.02 | 2.50 | 2.58 |
| 10 | 32.31 | 0.01 | 2.46 | 2.43 |
| 11 | 32.47 | 0.01 | 2.53 | 2.54 |
| 12 | 32.31 | 0.01 | 2.52 | 2.43 |
| 13 | 32.36 | 0.02 | 2.57 | 2.49 |
| 14 | 32.37 | 0.01 | 2.55 | 2.56 |
| 15 | 32.42 | 0.01 | 2.51 | 2.53 |
| 16 | 32.23 | 0.02 | 2.54 | 2.56 |
| Average | 32.88 | 0.189 | 2.52 | 2.50 |

**Table 5.8: Simulation Results of "*Miss America*" Using FMRME Based Wavelet Transform Video Coder with Quantization Threshold T = 10 and Quantizer Step Sizes $D_m$'s = 2, 4 and 8**

| Frame # | PSNR | Entropy Wavelet Coefficients | Entropy Motion Vectors - Horizontal Direction | Entropy Motion Vectors - Vertical Direction |
|---|---|---|---|---|
| 1 | 38.67 | 2.78 | - | - |
| 2 | 34.47 | 0.05 | 2.41 | 2.35 |
| 3 | 34.33 | 0.08 | 2.49 | 2.48 |
| 4 | 33.92 | 0.08 | 2.46 | 2.44 |
| 5 | 34.00 | 0.08 | 2.42 | 2.43 |
| 6 | 33.86 | 0.08 | 2.50 | 2.42 |
| 7 | 33.93 | 0.07 | 2.44 | 2.45 |
| 8 | 33.78 | 0.04 | 2.54 | 2.42 |
| 9 | 34.03 | 0.05 | 2.43 | 2.47 |
| 10 | 33.95 | 0.04 | 2.45 | 2.41 |
| 11 | 34.18 | 0.04 | 2.46 | 2.45 |
| 12 | 34.02 | 0.05 | 2.50 | 2.39 |
| 13 | 34.11 | 0.05 | 2.44 | 2.35 |
| 14 | 34.00 | 0.03 | 2.48 | 2.46 |
| 15 | 34.10 | 0.06 | 2.44 | 2.48 |
| 16 | 33.94 | 0.08 | 2.56 | 2.45 |
| Average | 34.33 | 0.229 | 2.47 | 2.43 |

**Table 5.9: Simulation Results of "*Miss America*" Using FMRME Based Wavelet Transform Video Coder with Quantization Threshold T = 2 and Quantizer Step Sizes $D_m$'s = 2, 4 and 32**

Figure 5.12: **Original Frame No. 7 of the Miss America Sequence**

Figure 5.13: **Reconstructed Frame No. 7 of the Miss America Sequence Using FMRME Based Wavelet Transform Coder at 0.85 Mbit/s, PSNR = 34.25 dB**

Figure 5.14: **Reconstructed Frame No. 7 of the Miss America Sequence Using MRME Based Wavelet Transform Coder at 0.99 Mbit/s, PSNR = 34.64 dB**

Figure 5.15: **Reconstructed Frame No. 7 of the Miss America Sequence Using FMRME Based Wavelet Transform Coder at 0.26 Mbit/s, PSNR = 31.72 dB**

Figure 5.16: **Reconstructed Frame No. 7 of the Miss America Sequence Using MPEG-1 Coder at 0.26 Mbit/s, PSNR = 31.06 dB**

# Chapter 6

# Conclusions and Future Work

## 6.1 Conclusions

In this thesis, the problem of motion estimation has been addressed from two perspectives, namely, hardware architecture and reduced complexity algorithms. First, we have presented a VLSI architecture which implements the full search block matching motion estimation algorithm (FSA) in real time. The architecture consists of a two-dimensional structure of basic cells (BC's) where each BC is capable of computing the mean absolute error (MAE). The interblock dependency is exploited and hence the architecture can meet the real time requirement in various applications. For example, in MPEG standard with block size of $16 \times 16$ and maximum allowed displacement of 8, the proposed architecture has a speed up factor of 90 over a general purpose processor. Most importantly, the architecture is simple, modular and cascadable and hence makes possible computation of a $2n \times 2n$ reference block by a simple cascade of four $n \times n$ chips. The proposed architecture is easily implementable in VLSI as a codec.

Since the FSA requires all possible candidate blocks to be searched, it is a computationally expensive procedure. Hence it is desirable to have reduced complexity motion estimation algorithms which have a comparable performance to FSA and in addition be easily implementable in hardware. In this thesis, we have presented two novel motion estimation algorithms in the spatial and transform domains for video compression. The spatial domain algorithm consists of a layered structure and hence does not converge to the local optimum. Most importantly, it employs a simple matching criterion, namely, a modified pixel difference classification (MPDC), resulting in a reduced computational complexity. In addition, the performance of the MPDC criterion does not depend on the initial threshold value as in the pixel difference classification (PDC) criterion. The MPEG-1 video coder has been simulated using the proposed layered structure MPDC (LSA-MPDC) algorithm and other techniques recently reported in the literature (FSA-MSE, FSA-MAE, FSA-PDC and PHODS). Simulation results indicated that the LSA-MPDC algorithm performs comparable to the FSA-MSE and the FSA-MAE algorithms (less than 0.4 dB in *"Miss America"* and 0.7 dB in *"PingPong"*) at a significantly reduced computational complexity. In addition, the performance of LSA-PDC is better than FSA-PDC and PHODS. We note that the PHODS

algorithm converges to the local optimum and fails to capture the fast motion in the *"PingPong"* sequence. Hence its performance is approximately 3 dB lower than LSA-MPDC. We note that the hardware implementation of the LSA-MPDC algorithm is very simple because of the binary operations used in the matching criteria.

We recall from chapter 2 that in the MPEG-1 video compression standard, discrete cosine transform and motion estimation/compensation are used to exploit the spatial and temporal redundancies, respectively. However, these redundancies are exploited independently. In this thesis, we have presented a fast multiresolution motion estimation (FMRME) scheme based on wavelet transform coding. Here, wavelet transform is used to exploit both spatial and temporal redundancies resulting in an efficient video coder. In FMRME, the set of orientation subimages at each level of the wavelet pyramid structure are first combined together into a single (all-orientation) subimage, and then the motion estimation is applied on the newly formed subimage. The motion vectors of an all-orientation subimage at a lower level are predicted from the motion vectors at the preceding higher level and are refined at each step. The FMRME scheme has the advantages of significantly reduced side information for motion vectors and search time compared to the multiresolution motion estimation (MRME) scheme recently reported in the literature. Simulation results show that the entropy of FMRME is marginally higher than the MRME scheme (approximately 3% higher for "*Miss America*"). Since the bit rate is a function of the number of motion vectors required to be transmitted, the FMRME based wavelet coding has a significant improvement in coding performance compared to the MRME based wavelet coding for video compression. Our simulation results show that the FMRME based wavelet coder has a reduction of bit rate by 15% compared to the MRME based wavelet coder. In addition, the FMRME based wavelet coder has comparable performance to the MPEG-1 coder at a significantly reduced computational complexity.

## 6.2 Suggestions for Future Work

### 6.2.1 Extension of the LSA-MPDC Algorithm

Some modifications to the LSA-MPDC algorithm can be made to improve its performance. For example, additional speed up can be achieved by using the LSA-MPDC algorithm on a pyramidal data structure [55], [56] which is a reorganization of the input video frame with increasing resolutions from the top to bottom level of the pyramid [77]. In other words, a pyramid decomposition provides a layered structure of data which matches the requirements of the LSA-MPDC algorithm resulting in efficient motion estimation. Here, the MPDC criterion can be used at the higher levels of the pyramid while the PDC criterion can be used at the lowest level. The final value of the threshold for a specific block may be used as in initial estimate for the lower levels of the pyramid which may result in a faster convergence.

### 6.2.2 Hardware Implementation of the LSA-MPDC Algorithm

In section 4.3, we presented the basic schematic of the hardware requirements for the LSA-MPDC algorithm. Hence the actual design of the hardware implementation is a possible future research work. The basic cell of the architecture can be identical to the schematic presented in section 4.3. However, a major modification to the control unit design is required to synchronize the flow of data.

### 6.2.3 Extension of the FMRME Based Wavelet Transform Coder

In our current implementation of the FMRME based wavelet transform coder, we have used scalar quantization to compress the wavelet coefficients. However, it has been shown that vector quantization provides a better performance compared to scalar quantization [72]. Hence the extension of the FMRME based wavelet transform coder by employing a more novel technique such as non-linear interpolative vector quantization [70] is a possible future research work. Since this approach exploits the cross-correlations in the wavelet coefficients efficiently, this can further improve the performance (bit rate) of the FMRME based wavelet transform coding for video

compression [33]. In addition, we can also incorporate the MPEG standard's group of picture and bidirectional motion estimation approach in the FMRME based wavelet transform coder which has the potential to reduce the bit rate and improve the accuracy of the motion estimation.

In our simulations, we have used the adaptive truncation scheme to quantize the wavelet coefficients, i. e. each subimage of the wavelet pyramid is uniformly quantized. Hence the design of the specific quantization table for the wavelet coefficients is another possible future research work. This can result in improvements in the coding performance using the wavelet transform.

# Bibliography


[1] N. Jayant, "High Quality Networking of Audio-Visual Information", *IEEE Communications Magazine*, pp. 84-95, September 1993.

[2] E. A. Fox, "Advances in Interactive Digital Multimedia Systems", *IEEE Computer Magazine*, pp. 9-21, October 1991.

[3] P. H. Ang, P. A. Ruetz and D. Auld, "Video Compression Makes Big Gains", *IEEE Spectrum Magazine*, pp. 16-19, October 1991.

[4] A. K. Jain, "Image Data Compression: A Review", *Proceedings of the IEEE*, Vol. 69, No. 3, pp. 349-389, March 1981.

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

[6] A. N. Netravali and B. G. Haskell, *Digital Pictures*, Plenum: New York, 1989.

[7] K. R. Rao and P. Yip, *Discrete Cosine Transform - Algorithms, Advantages, Applications*, Academic Press : San Diego, 1990.

[8] N. M. Nasrabadi and R. A. King, "Image Coding Using Vector Quantization: A Review", *IEEE Transactions on Communications*, Vol. 36, No. 8, pp. 957-971, August 1988.

[9] D. L. Gall, "MPEG: A Video Compression Standard for Multimedia Applications", *Communications of the ACM*, Vol. 34, No. 4, pp. 46-58, April 1991.

[10] C. T. Chen, "Video Compression: Standards and Applications", *Journal of Visual Communication and Image Processing*, Vol. 4, No. 2, pp. 103-111, June 1993.

[11] S. Zafar, Y. Q. Zhang and B. Jabbari, "Multiscale Video Representation Using Multiresolution Motion Compensation and Wavelet Decomposition", *IEEE Journal on Selected Areas in Communications*, Vol. 11, No. 1, pp. 24-35, January 1993.

[12] W. R. Zettles, J. Huffman and D. C. P. Linden, "Application of Compactly Supported Wavelets to Image Processing", *Image Processing Algorithms and Techniques, Proc. SPIE*, Vol. 1244, pp. 150-160, 1990.

[13] S. Mallat, "Multifrequency Channel Decompositions of Images and Wavelet Models", *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 37, No. 12, pp. 2091-2110, December 1989.

[14] M. Ohta, M. Yana and T. Nishitani, "Entropy Coding for Wavelet Transform of Image and Its Application for Motion Picture Coding", *SPIE Visual Communications and Image Processing'91*, pp. 456-466, 1991.

[15] Y. Q. Zhang and S. Zafar, "Motion-Compensated Wavelet Transform Coding for Color Video Compression", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 2, No. 3, pp. 285-296, September 1992.

[16] R. W. Young and N. G. Kingsbury, "Frequency-Domain Motion Estimation Using a Complex Lapped Transform", *IEEE Transactions on Image Processing*, Vol. 2, No. 1, pp. 2-17, January 1993.

[17] A. N. Netravali and J. D. Robbins, "Motion-compensated Television Coding: Part I", *Bell Syst, Tech. J.*, Vol. 58, pp. 631-670, 1979.

[18] J. Biemond, L. Looijenga, D. E. Boekee and R. H. J. M. Plompen, "A Pel-Recursive Wiener-based Displacement Estimation Algorithm", *Signal Processing*, Vol. 13, No. 4, pp. 399-412, 1987.

[19] R. J. Moorhead II, S. A. Rajala and L. W. Cook, "Image Sequence Compression Using a Pel-Recursive Motion-Compensated Technique", *IEEE Journal on Selected Areas in Communications*, Vol. 5, No. 7, pp. 1100-1114, August 1987.

[20] Q. Wang and R. J. Clarke, "Motion Estimation and Compensation for Image Sequence Coding", *Signal Processing: Image Communication*, Vol. 4, No. 2, pp. 161-174, 1992

[21] M. Orchard, "A Comparison of Techniques for Estimating Block Motion in Image Sequence Coding", *SPIE Visual Communications and Image Processing '89*, pp. 248-258, 1989.

[22] 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.

[23] J. R. Jain and A. K. Jain, "Displacement Measurement and Its Application in Interframe Image Coding", *IEEE Transactions on Communications*, Vol. COM-29, No. 12, pp. 1799-1808, December 1981.

[24] T. Koga, K. Iinuma, A. Hirano, Y. Iijima and T. Ishiguro, "Motion-Compensated Interframe Coding for Video Conferencing", *Proceedings of the National Telecommunications Conference '81*, pp. G5.3.1-G5.3.5, 1981.

[25] R. Srinivasan and K. R. Rao, "Predictive Coding Based on Efficient Motion Estimation", *IEEE Transactions on Communications*, Vol. COM-33, No. 8, pp. 888-896, August 1985.

[26] A. Puri, H. M. Hang and D. L. Schilling, "An Efficient Block-Matching Algorithm for Motion-Compensated Coding", *Proceedings of the ICASSP'87*, pp. 1063-1066, 1987.

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

[28] 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, pp. 378-385, December 1991.

[29] K. W. Chun and J. B. Ra, "Fast Block Matching Algorithm by Successive Refinement of Matching Criterion", *SPIE Visual Communications and Image Processing '92*, pp. 552-560, 1992.

[30] C. H. Hsieh, P. C. Lu and J. S. Shyn, "Motion Estimation Using Interblock Correlation", *IEEE International Symposium on Circuits and Systems*, pp. 995-998, 1990.

[31] S. Zafar, Y. Q. Zhang and J. S. Bara, "Predictive Block-Matching Motion Estimation for TV Coding", *IEEE Transactions on Broadcasting*, Vol. 37, No. 3, pp. 97-101, September 1991.

[32] B. Liu and A. Zaccarin, "New Fast Algorithms for the Estimation of Block Motion Vectors", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 3, No. 2, pp. 148-157, April 1993.

[33] X. Wang and S. Panchanathan, "Wavelet Transform Coding Using NIVQ", *SPIE Visual Communications and Image Processing'93*, pp. 999-1009, 1993.

[34] M. Antonini, M. Barlaud, P. Mathieu and I. Daubechies, "Image Coding Using Wavelet Transform", *IEEE Transactions on Image Processing*, Vol. 1, No. 2, pp. 205-220, April 1992.

[35] A. S. Lewis and G. Knowles, "Image Compression Using the 2-D Wavelet Transform", *IEEE Transactions on Image Processing*, Vol. 1, No. 2, pp. 244-250, April 1992.

[36] A. S. Lewis and G. Knowles, "Video Compression Using 3D Wavelet Transforms", *Electronic Letter*, Vol. 26, No. 6, pp. 396-397, 1990.

[37] F. May, "Full Motion 64 kbit/s Video Codec with 8 DSPs", *International Workshop on 64 kbit/s Coding of Moving Video*, Hannover, FRG, June 1988.

[38] C . Hoek, "Using Array Processors for Low Bit Rate Videocoding", *International Workshop on 64 kbit/s Coding of Moving Video*, Hannover, FRG, June 1988.

[39] I. Tamitani, H. Harasaki, T. Nishitani, Y. Endo, M. Yamashina and T. Enomoto, "A Real Time Video Signal Processor Suitable for Motion Picture Coding Applications", *IEEE Transactions on Circuits and Systems*, Vol. 36, No. 10, pp. 1259-1266, October 1989.

[40] K. M. Yang, L. Wu, H. Chong and M. T. Sun, "VLSI Implementation of Motion Compensation Full Search Block Matching Algorithm", *SPIE Visual Communications and Image Processing'88*, pp. 892-899, 1988.

[41] T. Komarek and P. Pirsch, "Array Architecture for Block Matching Algorithms", *IEEE Transactions on Circuits and Systems*, Vol. 36, No. 10, pp. 1301-1308, October 1989.

[42] L. D. Vos and M. Stegherr, "Parameterizable VLSI Architectures for the Full-Search Block Matching Algorithm", *IEEE Transactions on Circuits and Systems*, Vol. 36, No. 10, pp. 1309-1316, October 1989.

[43] K. M. Yang, M. T. Sun and L. Wu, "A Family of VLSI Designs for the Motion Compensation Block-Matching Algorithm", *IEEE Transactions on Circuits and Systems*, Vol. 36, No. 10, pp. 1317-1325, October 1989.

[44] A. Artieri and F. Jutand, "A Versatile and Powerful Chip for Real Time Motion Estimation", *Proceedings of the ICASSP'89*, pp. 2453-2456, 1989.

[45] 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, No. 2, pp. 169-175, June, 1992.

[46] G. Privat and M. Renaudin, "Motion Estimation VLSI Architecture for Image Coding", *IEEE International Conference on Computer Design*, pp. 78-81, 1989.

[47] Y. S. Jehng, L. G. Chen and T. D. Chiueh, "An Efficient and Simple VLSI Tree Architecture for Motion Estimation Algorithms", *IEEE Transactions on Signal Processing*, Vol. 41, No. 2, pp. 889-900, February 1993.

[48] F. M. Idris and S. Panchanathan, "Associative Memory Architecture for Video Compression", submitted for possible publication in the *IEE Proceedings - E: Computers and Digital Techniques.*

[49] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, McGraw Hill: New York, 1985.

[50] H. T. Kung, "Why Systolic Arrays", *IEEE Computer Magazine*, pp. 37-46, January 1982.

[51] J. A. B. Fortes and B. W. Wah, "Systolic Arrays - From Concept to Implementation", *IEEE Computer Magazine*, pp. 12-17, July 1987.

[52] J. P. Hayes, *Computer Architecture and Organization*, McGraw Hill: New York, 1988.

[53] T. Kohonen, *Content Addressable Memories*, Spring-Verlag: Berlin, 1987.

[54] L. Chisvin and R. J. Duckworth, "Content-Addressable and Associative Memory", *IEEE Computer Magazine*, pp. 51-64, July 1989.

[55] F. Glazer, "Scene Matching by Hierarchical Correlation", *IEEE Conference Proceedings*, pp. 432-441, 1983.

[56] M. Bierling and R. Thoma, "Motion Compensating Field Interpolation using a Hierarchically Structured Displaced Estimator", *Signal Processing 11, North Holland*, pp. 387-404, 1986.

[57] M. Goldberg and L. Wang, "Comparative Performance of Pyramid Data Structure for Progressive Image Transmission", *IEEE Transactions on Communications*, Vol. COM-39, No. 4, pp. 540-548, April 1991.

[58] R. Gandhi, "3-Dimensional Pyramids for Video Compression", *M.A.Sc Thesis*, University of Ottawa, Canada, 1993.

[59] N. Jayant, "Signal Compression: Technology Targets and Research Directions", *IEEE Journal on Selected Areas in Communications*, Vol. 10, No. 5, pp. 796-818, June, 1992.

[60] W. K. Pratt, *Digital Image Processing*, John Wiley & Sons, Inc.: New York, 1991.

[61] R. Clarke, *Transform Coding of Images*, Academic Press: New York, 1985.

[62] A. Habibi, "Survey of Adaptive Image Coding Techniques", *IEEE Transactions on Communications*, Vol. COM-25, No. 11, pp. 1275-1284, November 1977.

[63] J. Weiss and D. Schremp, "Putting Data On A Diet", *IEEE Spectrum Magazine*, pp. 36-39, August 1993.

[64] R. E. Ziemer and W. H. Tranter, *Principles of Communications: Systems, Modulation, and Noise*, Houghton Mifflin Company: Boston, 1990.

[65] W. Chen and W. Pratt, "Scene Adaptive Coder", *IEEE Transactions on Communications*, Vol. COM-32, No. 3, pp. 225-232, March 1984.

[66] M. Ziegler, "Hierarchical Motion Estimation Using the Phase Correlation Method in 140 Mbits/s HDTV Coding", in *Signal Processing of HDTV II, Proceeding of the 3rd International Workshop on HDTV*, pp. 131-137, 1990.

[67] M. Liou, "Overview of the p x 64 kbits/s Video Coding Standard", *Communications of the ACM*, Vol. 34, No. 4, pp. 59-63, April 1991.

[68] N. Tavakoli, "Short Communication: Entropy and Image Compression", *Journal of Visual Communication and Image Representation*, Vol. 4, No. 3, pp. 271-278, September 1993.

[69] S. Mallat, "A Theory for Multiresolution Signal Representation: The Wavelet Decomposition", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 7, pp. 674-693, July 1989.

[70] A. Gersho, "Optimal Nonlinear Interpolative Vector Quantization", *IEEE Transactions on Communications*, Vol. 38, No. 9, pp. 1285-1287, September 1990.

[71] W. D. Ray and R. M. Driver, "Further Decomposition of the Karhunen Loeve Series Representation of a Stationary Random Process", *IEEE Transactions on Information Theory*, Vol. IT-16, No. 6, pp. 663-668, November 1970.

[72] R. M . Gray, "Vector Quantization", *IEEE ASSP Magazine*, pp. 4-29, April 1984.

[73] K. A. Prabhu and A. N. Netravali, "Motion Compensated Component Color Coding", *IEEE Transactions on Communications*, Vol. COM-30, No. 12, pp. 2519-2527, December 1982.

[74] H. S. Malvar and D. H. Staelin, "The LOT: Transform Coding Without Blocking Effect", *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 37, No. 4, pp. 553-559, April 1989.

[75] M. J. Smith and D. P. Barnwell, "Exact Reconstruction Techniques For Tree-Structured Subband Coders", *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 34, No. 3, pp. 434-441, June 1986.

[76] A. H. Wong and C. T. Chen, "A Comparison of ISO MPEG1 and MPEG2 Video Coding Standards", *SPIE Visual Communications and Image Processing'93*, pp. 1436-1448, 1993.

[77] R. Gandhi, L. Wang, S. Panchanathan and M. Goldberg, "MPEG-like Pyramidal Video Coder", *SPIE Visual Communications and Image Processing'93*, pp. 707-717, 1993.

[78] F. M. Idris, "An Algorithm and Architecture for Video Compression", *M.A.Sc Thesis*, University of Ottawa, Canada, 1993.

[79] E. Chan and S. Panchanathan, "Motion Estimation Architecture for Video Compression", *Proceedings of the International Conference on Consumer Electronics '93*, pp. 72-73, 1993.

[80] E. Chan and S. Panchanathan, "Motion Estimation Architecture for Video Compression", *IEEE Transactions on Consumer Electronics*, Vol. 39, No. 3, pp. 292-297, August 1993.

[81] E. Chan and S. Panchanathan, "Review of Block Matching based Motion Estimation Algorithms for Video Compression", *Proceedings of the Canadian Conference on Electrical and Computer Engineering '93*, pp. 151-154, 1993.

[82] E. Chan and S. Panchanathan, "A Novel Motion Estimation Algorithm for Video Compression", *Proceedings of the 1993 Asia-Pacific Conference on Communications*, pp. 410-413, 1993.

[83] E. Chan, R. Gandhi and S. Panchanathan, "A Fast Motion Estimation Algorithm for an MPEG Video Coder", accepted for presentation at the *SPIE Conference on Digital Video Compression and Processing on Personal Computers '94*, San Jose, 1994.

[84] E. Chan, X. Wang and S. Panchanathan, "Fast Multiresolution Motion Estimation Scheme for Wavelet Transform Coding", submitted for possible publication in the *IEEE Transactions on Image Processing*.

[85] E. Chan, R. Gandhi and S. Panchanathan, "Reduced Complexity Block-Matching Techniques for an MPEG Video Coder", submitted for possible publication in the *Multimedia Systems Journal*.

[86] S. Panchanathan, E. Chan and X. Wang, "Fast Multiresolution Motion Estimation Scheme for a Wavelet Transform Video Coder", accepted for presentation at the *SPIE Visual Communications and Image Processing'94*, Chicago, 1994.