Low power design of Motion Compensation Module for MPEG-4 Video Transcoder in DCT-domain

Wan Hoi Cheung

A Thesis

In

The Department

Of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Electrical) at

Concordia University

Montreal, Quebec, Canada

August 2006

Your file Votre référence
ISBN: 978-0-494-20740-6
Our file Notre référence
ISBN: 978-0-494-20740-6

ABSTRACT

Low power design of Motion Compensation Module for MPEG-4 Video Transcoder in DCT-domain

Wan Hoi Cheung

For the 21st century, the Internet has matured into the de facto data transport platform that analog media such as videotape, film, and broadcast will be supplanted by a digital media infrastructure. This digital infrastructure will allow data to be transferred between any two computing machines on the planet in a heterogeneous network environment. In the case where data is transmitted from the SONET network to PSTN network, dynamic bit-rate adaptation/reduction at the gateways is then required due to the transmission media has a lower capacity than the capacity required by the bitstream. There are many different approaches to this problem of bit rate conversion. A robust scheme is to implement a transcoder module to perform dynamic adjustments of bit rate of the coded video bitstream to the desired transmission rate.

The emphasis in this thesis is on the study of the transcoder's Motion Compensation module of MPEG-4 compressed video stream in the DCT domain on both algorithmic and implementation level.

For the algorithmic level of Motion Compensation in DCT-domain (MC-DCT) design, the 3-2-1 partial information scheme is integrated into the DCT Coefficient Translation and Truncation Transformation Matrix (DCTTTM)-based algorithm with two bit precision on the element of DCT-Constant Matrix (DCT-CM) to process the MC-DCT operation.

For the VHDL hardware-level design, data-dependent signal processing has been applied to the MC-DCT module to reduce power consumption via various manoeuvres such as zero bypassing, custom handling of internal bandwidth, the implementation of multiplication-free module, the 3-2 Wallace tree propagation-delay-free addition map and the logic-based addition module.

Keywords: MPEG-4, Motion Compensation, transcoder, drift error correction, discrete cosine transform (DCT) domain, requantization, data-dependent logic, low power.

Acknowledgements

The work presented in this thesis would not have been possible without the continuous help and substantial guidelines from my supervisors Dr. Asim J. Al-Khalili and Dr. William E. Lynch. I am so glad finally to be afforded this opportunity to express my deepest gratitude and appreciation toward them, for their patience and exceptional assistance all these "years" during my master study, I mean, second to none. I would also like to thank Mr. Cheung-Yu Pai and Mr. Chen HongQuan for all their "giga-byte" help on, if not 24/7, would be 12/5 basis.

Thanks to National Sciences and Engineering Research Council of Canada (NSERC) and MRC Networks for the sponsorship of the Industrial Post-Graduate Scholarship (IPS) to financial support my work.

Thanks to my mentor Michel Lortie who helps me to mob up all the grammatical errors inherently embedded deep down in the thesis. An extremely talent guy who has been guiding me through all the predicaments at work, indeed, a "lovely" fellow you may only got one chance to meet…

Finally, I would love to dedicate this work to my family for their love and support. Thanks!!!

Table of Contents

List of Figure

List of Tables

List of Abbreviations and Symbols

ATM	Asynchronous Transfer Mode
CPDT	Cascade Pixel Domain Transcoder
DCT	Discrete Cosine Transform
DCT-CM	DCT-Correction Matrix
DCTTTM	DCT Coefficient Translation and Truncation Transformation Matrix
DDT	DCT-Domain Transcoder
DSL	Digital Subscriber Line
FPDT	Fast Pixel Domain Transcoder
GOP	Group of Picture
ISDN	Integrated Service Digital Network
MC	Motion Compensation
MoMuSys	Mobile Multimedia System
MPEG	Moving Picture Experts Group
MSE	Mean Squared Error
POTS	Plain Old Telephone System
PSNR	Peak Signal to Noise Ratio
PSTN	Public Switch Telephone Network
RE	Requantization Error
SDH	Synchronous Digital Hierarchy
SONET	Synchronous Optical Networks

VHDL Very High Speed Integrated Circuit (VHSIC) Hardware Description Language

Chapter 1

INTRODUCTION

1.1. RESEARCH MOTIVATION

Visual information is of vital importance if human beings are to perceive, recognize, and understand the surrounding world. As the Internet matures into the de facto data transport platform for the 21st century, it seems clear that analog media such as videotape, film, and broadcast will be supplanted by a digital media infrastructure built on the Internet and Internet-related technologies. In addition, services like teleconferencing, video on demand, distance learning, and remote surveillance for provide a convenient means of acquiring visual information from remote locations. This digital infrastructure will allow visual data to be readily transferred between any two computing machines on the planet, if so desired. However, the speed at which this data can be transmitted will depend on a number of factors, but primarily on the network infrastructure. Most existing networks such as POTS, ISDN, DSL over PSTN, ATM over SONET (Synchronous optical network) or SDH (Synchronous Digital Hierarchy), and 3G Wireless are interconnected resulting in a heterogeneous network environment. On one hand, POTS over PSTN networks planned with copper wires laid down over a century ago and intended for analog voice communications are used with modem technology to transmit data at speeds as high as 56 kb/s. On the other hand, ATM over SONET via optical fiber technology allows data and/or voice transfers at 51.840 Mbit/s and higher. In the case where data is transmitted from the SONET network to PSTN network, dynamic bit-rate adaptation/reduction at the gateways is required since the

receiving transmission medium has a lower capacity than the capacity required by the originating bitstream [1].

One solution is to use the scalable coding feature inherent in the MPEG-4 standard where the video is coded at two or more layers, a base layer and one or more enhancement layers. The base layer stores the most critical information while the other enhanced layers supplement the video quality. The enhanced layers maybe skipped (dropped) in the case of changing network environments. This approach is nonetheless not sufficiently flexible to handle finer scaling since the scalability in MPEG-2 provides only a limited number of possible transmission bit rates.

A more robust scheme is to implement a transcoder module to perform dynamic adjustments of the bit rate of the coded video bitstream to match the desired downstream network. This can be done by first partial decoding of the bitstream and then requantizing the DCT coefficients coarsely to achieve bit rate reduction. Figure 1.1 shows a diagram of the transmission of a video stream from a server X to end user Y where a transcoder is deployed at the gateway between two heterogeneous networks. One raw uncompressed video at server X is compressed by an encoder at a bit rate $R1$. The encoded video is converted to a bit rate $R2 < R1$ by the transcoder module when the transmission medium has a lower capacity than the bitstream requires. The decoder at the end user Y decompresses the transcoded video bitstream for display.

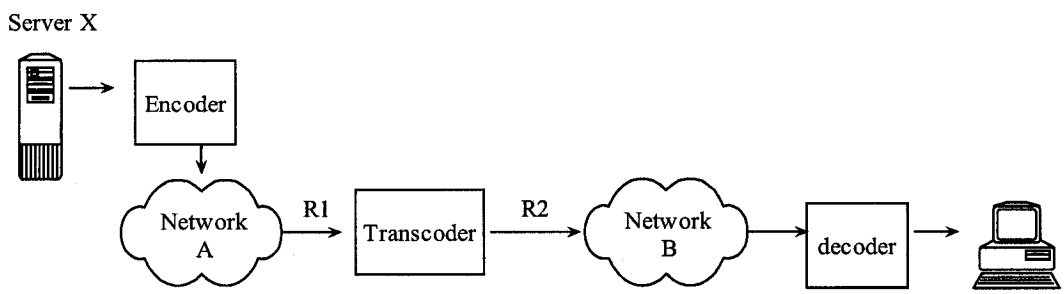


Figure 1.1 A scenario where the transcoder is deployed between two heterogeneous networks

1.2. FIGURE OF MERIT

The process of converting a compressed high bitrate stream into a lower bitrate stream introduces distortion. Conventionally, the distortion can be measured using one of two schemes; subjective assessment and objective assessment.

Subjective assessment requires human perception and/or opinion about a group of degraded target pictures evaluated based on a predefined scale to determine the quality of the degraded pictures as compared to the original undistorted ones. While the result of this scheme is generally more reliable than any objective assessment, the cost remains prohibitively high and the process is extremely time-consuming as in all man-in-the-coop processes. In this thesis, the formal subjective assessment method is not considered; however, informal subjective evaluation is included in deciding the application of the optimization technique.

Objective measurement provides a much cheaper and time-efficient means of simulating human observer grading via mathematical models. The simplest objective measure and yet the most popular method employed within the video/audio and the signal processing community is known as the peak-to-peak signal-to-noise ratio (PSNR) and defined as:

$$PSNR = 10\log_{10}\frac{255^2}{\frac{1}{N}\sum_i\sum_j (Y_{ref}(i,j) - Y_{rec}(i,j))^2}$$

Where $Y_{ref}(i\,j)$, and $Y_{rec}(i\,j)$, are the pixel values of the reference and reconstructed images respectively. N is the total number of pixels in the image. The larger the PSNR logically implies the better the reconstructed image; it is however not necessarily true at all times. A case where image A has lower PSNR than image B with noise spreading all

over might yield a better subjective quality for human perception than an image *B* with higher PSNR but having noise concentrated within a tiny area which can easily be spotted visually.

In general, the gain of better PSNR for a reconstructed picture requires the expense of computation complexity. In chapter 3, several factors to reduce the computational complexity of the core module in the transcoding process are investigated and analyzed in order to achieve a balance between the saving in computational load and the cost in the objective PSNR measure.

1.3. THESIS ORGANIZATION

The organization of this thesis is as follows: in Chapter 2, an overview of video transcoder architecture and MPEG-4 compressed video bitstream is provided. Chapter 3 reviews the pioneering work for Motion-Compensation in Discrete Cosin Transform (MC-DCT) algorithm by Chang and Messerschmitt's [20] and presents an optimization based on a 3-2-1 partial information scheme. Chapter 4 illustrates the VHDL design to implement the MC-DCT module via Xilinx FPGA technology. Chapter 5 summarizes the proposed schemes and the contribution of this thesis. Future work is also discussed.

Chapter 2

AN OVERVIEW OF VIDEO TRANSCODER ARCHITECTURE AND MPEG-4 COMPRESSED VIDEO BITSTREAMS

Transcoders can be broadly classified into two types, open-loop and closed-loop. The first, open-loop type of transcoder uses a straightforward requantization method without compensating for drift errors. The drift error occurs when the reconstructed pictures used for prediction in the encoder are not the same as the reconstructed pictures in the decoder due to the loss of high frequency data. This error will propagate if the reconstructed picture is to be used as reference for future pictures. In MPEG standard, this error is terminated when the I frame is processed. This type of transcoder demands little hardware, requires no frame buffer and is suitable for low cost and/or low delay application [2][3].

The second type of transcoder is built with a feedback loop to compensate for drift error. This approach provides a higher quality transcoding, but comes at the price of more complexity and greater memory requirements.

In this chapter, we will examine each mainstream transcoder's architecture proposed in the literature, Open-Loop transcoder, Cascade Pixel Domain Transcoder (CPDT), Fast Pixel Domain Transcoder (FPDT) and DCT Domain Transcoder (DDT) in Sections 2.1, 2.2, 2.3, 2.4 respectively [4][5][6][7][8]. Section 2.5 discusses the video signal streams compiled to the Moving Picture Experts Group Compression Standard Version 4 (MPEG-4) under the context of transcoding. In this thesis, the MPEG-4 standard is the video compression scheme used for all the simulations.

2.1. OPEN-LOOP TRANSCODER

The open-loop transcoder architecture achieves the bit-rate reduction by implementing the requantization process without feedback path for drift correction. Conceptually, the transcoding process with this structure is done by first inversely quantizing the encoded bitstream by mean of the finer quantizer Q1 and then heavily compressing with the coarse quantization Q2 (i.e. the step size for the quantization of Q2 is larger than the step size of Q1) to achieve the lower target bit rate. The basic structure is shown in Figure 2.1:

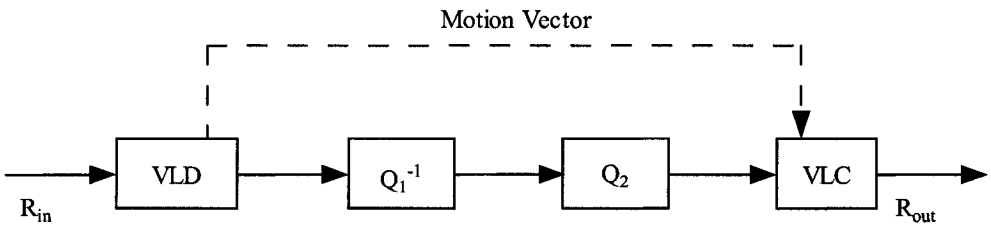


Figure 2.1 Transcoder structure with no drift-error correction

While such arrangements will always lead to increased distortion due to the lossy nature of quantization when the reconstructed values differ from the original ones or when some nonzero coefficients in the input frame become zero after quantization, this simple bit rate-conversion process may introduce an additional error which can be avoided by directly quantizing the DCT coefficients with the same coarser step size. This additional error is called Requantization Error (RE). Figures 2.2 and 2.3 below illustrate the cases where additional requantization error may occur:

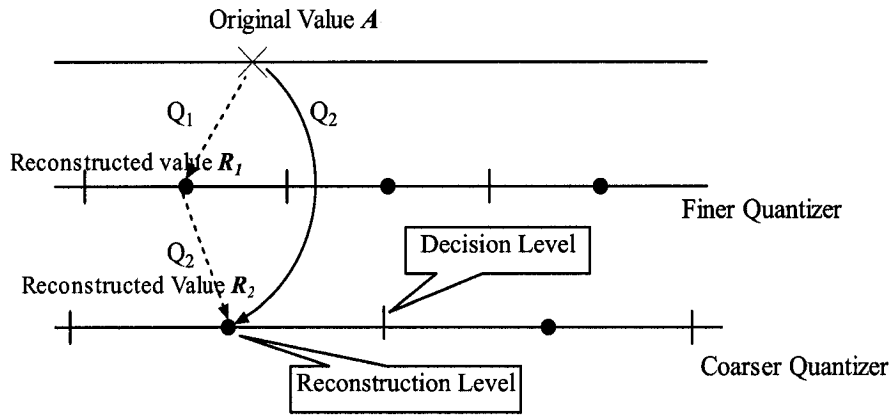


Figure 2.2 Cascaded quantization provides the same result as direct quantization

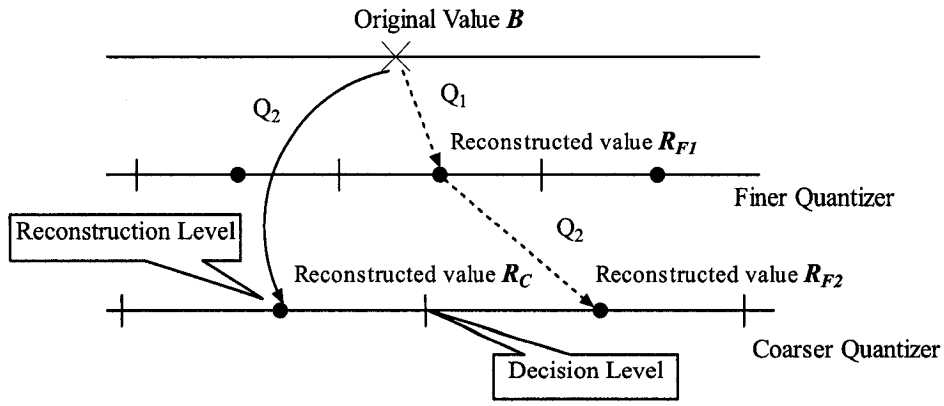


Figure 2.3 Cascaded quantization induces extra distortion

In Figure 2.1, the original value A is first quantized by a finer quantizer Q1, and then heavily quantized with Q2 resulting in the final reconstructed value R_1 (paths indicated by the two dash-line arrows). The same reconstructed R_1 may also be obtained by directly quantizing A by the coarse quantizer Q2 (path indicated by the solid-line arrow). In this case the direct coarse quantization and requantization distortions are equal. However, in Figure 2.3, the reconstructed value R_{F2} of coefficient B resulted from requantization is different from that of direct coarse quantization. This is caused when the

finer interval is completely contained by the coarse interval, no cascading error is introduced. On the other hand, if the finer interval overlaps the decision level of the coarse interval and if the original value falls into a coarse interval different from that of the interval where the coarse quantizer will resolve the value to, the requantization error subsequently becomes larger. In fact, by carefully selecting the ratio of Q2/Q1 to avoid critical ratios of the cascaded quantizations during the transcoding process, the magnitude of the requantization error could be greatly reduced [9][10][11].

The open-loop transcoder structure is straightforward and requires no frame memory; it is recommended for low delay and low cost application in which minor drift error is considered to be tolerable. It should be noted that having video stream with a small group of pictures (GOP) may considerably reduce the effect of drift error.

2.2. CASCADED PIXEL-DOMAIN TRANSCODER (CPDT)

A simple drift-free transcoder can be produced by first fully decoding the video streams into reconstructed pixels and then re-encoding them to a lower bit rate. The structure can be built with a decoder cascading an encoder as in Figure 2.4:

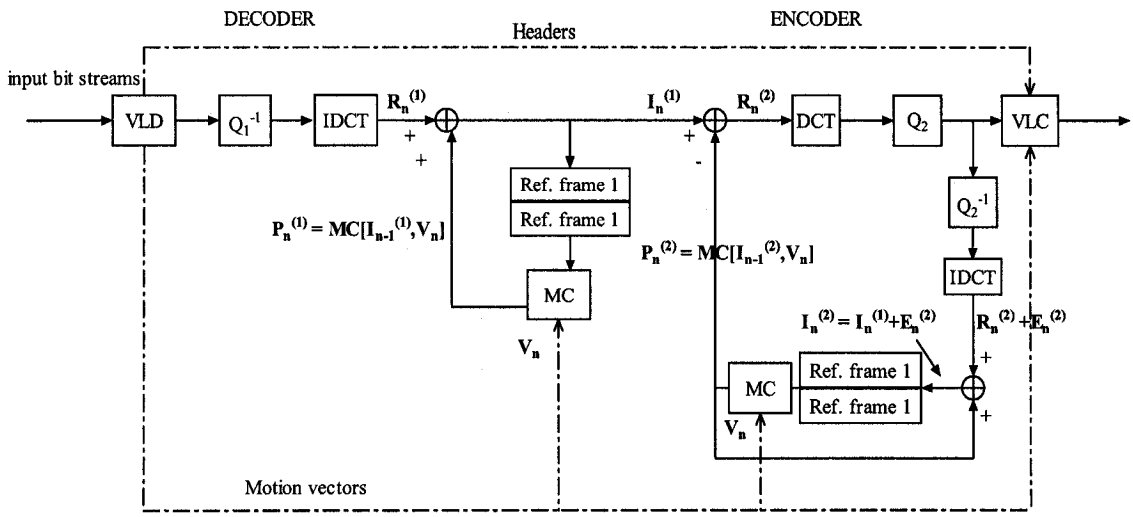


Figure 2.4 Cascade of decoder and encoder as a transcoder

The structure Figure 2.4 is inherently drift-free due to the complete isolation of the encoder loop from the decoder loop. Assuming the ideal environment of no transmission noise, the reconstructed frame resulting from the decoding section of the transcoder would be identical to the one generated by the front-encoder once motion compensation is applied. Likewise, the reconstructed transcoded frame in the encoding part of the transcoder would be the same generated by the end-decoder. Nonetheless, the direct implementation of the CPDT is not attractive due to high complexity, high cost and long delays. Knowing that the motion compensation is undoubtedly the most computationally expensive process in the CPDT structure, Petro in [12] derived a step-by-step simplification to realize the Fast Pixel-Domain Transcoder which reused the motion vector to reduce the overall computational complexity and hence improve the performance by an order of magnitude.

2.3. FAST PIXEL-DOMAIN TRANSCODER (FPDT)

Referring to Figure 2.4, let V_n represent the motion vector field of thencurrent picture n, x be the position of a pixel in the picture, and the motion compensated signal $MC[I_{n-1}^{(1)}, V_n]$ where MC[•] denotes a simple shift operation (in essence, Motion Compensation (MC) can be understood as a space/time-domain operation to form an estimate of the current frame by extracting the MC block from the location specified by the Motion Vector (MV('s)) in the reference frame(s), refer to the detailed discussion in Chapter 3), *n-1* stands for the previous frame, superscript *(1)* indicates the decoding part of the transcoder, and the encoding part of the trancoder is denoted as superscript *(2)*. $I_{n-1}^{(1)}$

for example is the previous decoded signal from the decoder. Therefore, the current predictive frame can be estimated as

$$P_n^{(1)} = MC[I_{n-1}^{(1)}, V_n] \tag{2.1}$$

Alternatively, it can be expressed per pixel as follows:

$$P_n^{(1)}(x) = MC[I_{n-1}^{(1)}, V_n](x) = I_{n-1}^{(1)}(x + V_n(x)) \tag{2.2}$$

where $V_n(x)$ is equal to the displacement for each pixel within a Macroblock (In MPEG standard, Macroblock is defined as a composition of four 8 x 8 blocks).

Since the Motion Compensation process is a linear operation, the predictive frame of the encoding part of the transcoder is to be formulated by the following equation:

$$P_n^{(2)} = MC[I_{n-1}^{(1)} + E_{n-1}^{(2)}, V_n] = MC[I_{n-1}^{(1)}, V_n] + MC[E_{n-1}^{(2)}, V_n] \tag{2.3}$$

where $E_n^{(2)}$ is the requantization error caused by the second quantizer Q2. Thus the residual $R_n^{(2)}$ can be written as:

$$\begin{aligned} R_n^{(2)} &= I_n^{(1)} - MC[I_{n-1}^{(1)} + E_{n-1}^{(2)}, V_n] \\ &= I_n^{(1)} - MC[I_{n-1}^{(1)}, V_n] - MC[E_{n-1}^{(2)}, V_n] \end{aligned} \tag{2.4}$$

And the decoded picture $I_n^{(1)}$ can be written as:

$$I_n^{(1)} = R_n^{(1)} + MC[I_{n-1}^{(1)}, V_n] \tag{2.5}$$

Substituting Equation (2.5) into Equation (2.4), we get:

$$R_n^{(2)} = R_n^{(1)} - MC[E_{n-1}^{(2)}, V_n] \tag{2.6}$$

Equation 2.6 implies that the residual $R_n^{(2)}$ can be computed directly by subtracting the motion-compensated requantization error $E_{n-1}^{(2)}$. Here, $MC[E_{n-1}^{(2)}, V_n]$ is the so called drift error in the transcoding process. Based on the Equation (2.6), only the previous

requantization errors $E_{n-1}^{(2)}$ need to be stored in frame memory and the frame storage for $I_n^{(1)}$ can be discarded. This development leads to a simplified architecture for fast transcoding. Figure 2.5 shows the block diagram of a Fast Pixel-Domain Transcoder (FPDT).

The underlying premise for the simplification to achieve the FPDT architecture makes use of the linearity of the Motion Compensation property such that the requantization\transcoding error can be isolated out for the Motion Compensated operation. The partial decoding methodology of reusing the Motion Vector (MV's) also contributes to simplifying the FPDT architecture since the process of Motion Estimation is no longer required. By doing so, only one reconstruction loop and one DCT/IDCT pair are needed.

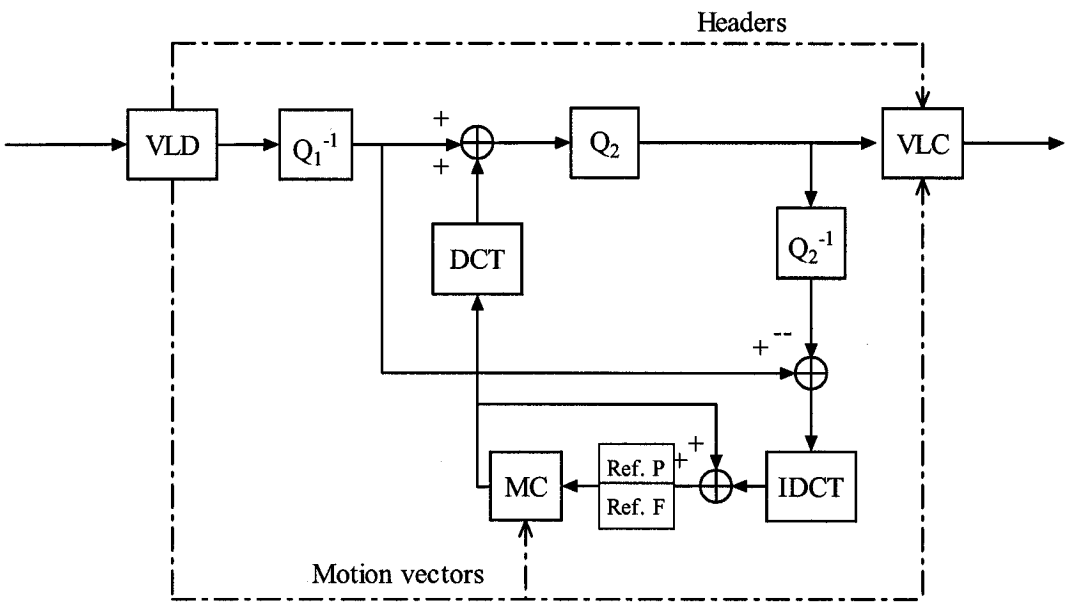


Figure 2.5 Fast Pixel-Domain Transcoder (FPDT)

2.4. DCT-DOMAIN TRANSCODER

The DCT/IDCT modules would no longer required if the bitstream could be transcoded entirely in the DCT domain. Figure 2.6 shows the architecture of this conceptual design:

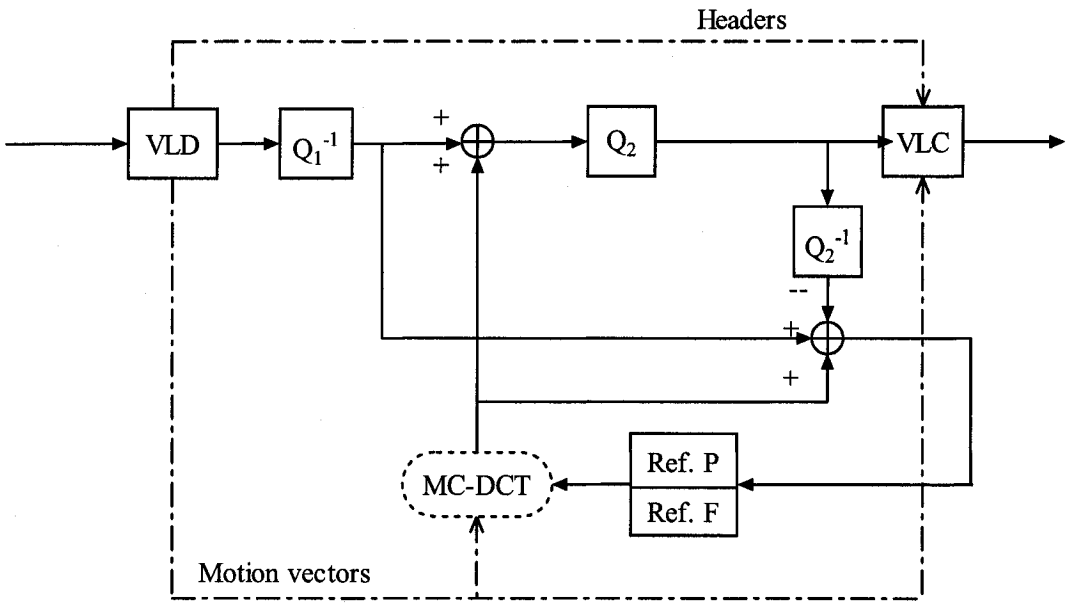


Figure 2.6 DCT-Domain Transcoder (DDT)

From Figure 2.6, the transcoding/requantization errors are accumulated in the DCT domain and compensated in the DCT block of the current picture through the module MC-DCT (Motion Compensation in Discrete Cosine Transform domain). In this thesis, the MC-DCT module is to be investigated thoroughly in later chapters. Chapter 3

will discuss in detail a step-by-step deviation for computing motion compensation in DCT-domain.

2.5. TRANSCODING OF MPEG-4 BITSREAMS

A feature of MPEG-4 in contrast to MPEG-2 or MPEG-1 is its content-based coding of images and video to allow the separate decoding and reconstruction of arbitrarily shaped video objects. This thesis only considers rectangular shaped video objects in the simulations, therefore the shape information is not specified and no shape coding was developed. In essence, only the texture content of MPEG-4 compressed video is transcoded.

For MPEG-4 visual bitstream syntax, similarin concept to MPEG-1 and -2, each GOV (Group of Video Object Planes) may include three types of VOPs (Video Object Plane), Intracoded (I-VOP), predictive-coded (P-VOP) and bi-directionally predictive-coded (B-VOP). Since I-VOPs are self-sufficient with no need of prior information, no Motion Compensation is applied. Given our choice of the transcoder architecture in Figure 2.6, the requantization errors (the differences of DCT coefficients between Quantizing parameters Q1 and Q2) are to be stored in the frame memory for compensating the next P- or B-VOP. P-VOP is motion-compensated by using the same Motion Vector computed by the front-encoder and its requantization errors are subsequently saved for motion compensating the next P- or B-VOP. Since there is no referrence for B-VOP, the requantization error was not calculated [13]. For simplicity, only the IPPP...IPPP GOV structure was used for all the simulations in this thesis.

With regard to transcoding of MPEG-4 compressed video bitstreams using a DDT architecture, motion vectors of the input bitstream are reused to avoid the most computationally intensive operation, motion estimation, of the MPEG-4 encoding algorithm.

2.6. CHAPTER SUMMARY

In this chapter, an overview of video transcoder architectures was presented. Two types of transcoders with different complexity and memory requirements were discussed. The transcoder with no drift correction has a lower complexity and requires no frame buffers. The more complex transcoder is structured with a closed loop to perform drift compensation. A drift-free CPDT transcoder can be designed by cascading a decoder to an encoder. While the CPDT design requires no special customization, this structure suffers from long delay and high complexity. A more versatile DDT design was addressed conceptually where DCT/IDCT modules are removed from the CPDT design to gain substantial reduction in the architectural complexity. The chapter also discussed the properties of MPEG-4 video stream used in the simulations completed as part of the thesis.

Chapter 3

ALGORITHMIC DESIGN CHOICES FOR MC-DCT MODULE

In Chapter 2, the fast DCT-domain transcoder architecture was introduced by removing the Inverse DCT and the DCT modules. The underlying premise is that Motion Compensation (MC) can be done in the DCT-domain which would allow this simplified architecture to be implemented without hampering the transcoder's correct functionality. In this chapter, the problem of MC in the DCT-domain (MC-DCT) is dealt with in full detail.

The practice of manipulating video signals in the compressed/transform domain originated over a decade ago [14][15][16][17][18][19]. With video compression systems incorporating Motion Compensation (MC) such as MPEG, the major challenge in providing common video editing functions in the compressed domain, including overlap, translation, scaling, linear filtering, rotation, and pixel multiplication etc, is the development of a fast algorithm. MC-DCT reconstructs the video in the compressed domain by Inverse Motion Composition without requiring conversion back to the time/spatial domain. In [14], it has been shown that compositing the DCT-compressed images directly in the DCT domain can save computations for many of the compositing operations mentioned above compared to the straightforward spatial-domain approach which composites images pixel by pixel.

Several fast MC-DCT algorithms based on Chang and Messerschmitt's [20] have been proposed in the literature [21][22]. In [21], Neri use a factorization of computing 8-point DCT to achieve 32% computational complexity savings compared to a spatial domain approach., while 81% reduction of complexity was realized in [22] by

approximating the translated matrices to binary numbers with maximum distortion of 1/32 to allow the execution of basic operations such as shift-right and add. In [23], Bovik approached the problem by analyzing the statistical properties of natural video data to avoid processing those DCT coefficients outside the estimated local bandwidth to gain 25%~55% computational improvement over the method proposed in [20]. They also present a Look Up Table (LUT)-based implementation scheme to obtain another 31%~48% improvements in computational complexity. Based on the fact that motion compensation in MPEG is done on a macroblock basis, Song and Yeo [24] exploit the shared information in the predictions of multiple neighboring blocks to speed up the MC-DCT algorithm. Song and Yeo show that their method can be implemented on top of the already fast algorithm proposed in [21] and [22], and provide incrementally about 19% and 13.5% above these techniques respectively..

In Section 3.1, the operation of Motion Compensation in the spatial domain is reviewed. The inherent obstacle to performing motion compensation in the frequency (DCT) domain is then examined. One of the MC-DCT algorithms reported in the literature, Chang and Messerschmitt's algorithm [20], is presented in Section 3.2. Section 3.3 introduces the DCT-Coefficient-Translation-and-Truncation-Transformation-Matrix based (DCTTTM) MC-DCT algorithm [25] which addressed the MC-DCT problem from a different perspective.

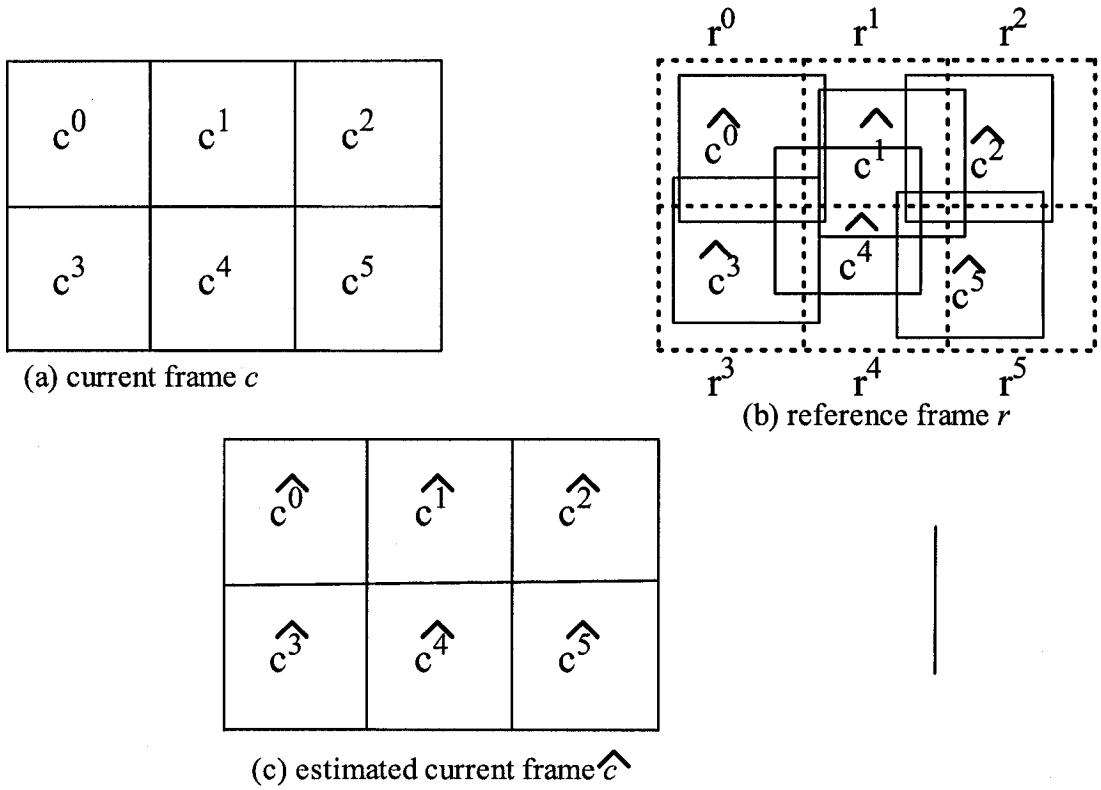
The computational complexity of DCTTTM-based MC-DCT algorithm can be dramatically reduced by considering only partial DCT information and quantizing the constant transformed matrices to finite precision. Section 3.5 studies the impact of applying these two maneuvers to the quality of five video test sequences.

3.1. DCT-Domain Motion Compensation

Motion Compensation (MC) can be understood as a space/time-domain operation to form an estimate of the current frame by extracting a MC block from the location specified by Motion Vector (MV('s)) in the reference frame(s). In general, neither the vertical nor the horizontal component of the MV is an integer multiple of the block size; hence, it is likely that a MC block extracted from a reference frame straddles four adjacent DCT blocks in the reference frame.

For Block Motion Compensation, the frames are partitioned into blocks of pixels in the spatial domain (e.g. macroblocks of 16×16 pixels). Each block is predicted from a block of equal size in the reference frame. These will be called Motion Blocks (MBs) with their size (p = 8)×(q = 8) where p is the number of rows and q is the number of columns. The blocks are not altered in any way other than being shifted to the position of the predicted block. This shift is represented by a motion vector.

Let c be the current frame, r be the reference frame, $\hat{c}$ be the estimated current frame. When a superscript is attached to these symbols it represents one MB. Thus r^5 represents the fifth MB in the reference frame. V is the vector field made up of all of the MVs for one frame. For MB k, its MV has horizontal and vertical components $MV_{row}^{\ k}$ and $MV_{col}^{\ k}$ respectively. The MVs are the result of the Motion Estimation. The position of a pixel in a frame is denoted by x and y. For example, assuming we a frame size of (m=16)×(n=24) where m is the number of rows and n is the number of columns, then, the current frame c, the estimated current frame $\hat{c}$, and the reference frame r can be partitioned into (16×24) / (8×8) = 6 MBs as shown in Figure 3.1:



(a) current frame c

(b) reference frame r

(c) estimated current frame $\hat{c}$

Figure 3.1: Estimated current frame in (c) is motion-compensated from reference frame in (b). Notice that the estimated current MBs are likely mapped to reference frame in a non-block boundary fashion.

Figure 3.1: Estimated current frame in (c) is motion-compensated from reference frame in (b). Notice that the estimated current MBs are likely mapped to reference frame in a non-block boundary fashion.

Each estimated MB $\hat{c}^k$ has a particular $\overrightarrow{MV}^k$ associated to it. Therefore, its $\hat{c}$ can is determined:

$$\hat{c} = MC[r, V] \tag{3.0}$$

Essentially, the MC[•] operation is simply a shift operation for each MB. For an MB, Equation 3.0 can be written explicitly as

$$\hat{c}^k(x,y) = r(i + MV_x^k, j + MV_y^k)\Big|_{\substack{i=[(k \bmod (n/q))*q]+x, \\ j=[\text{floor}(k/(n/q))*p]+y}} \tag{3.1}$$

for $x = 0 \ldots p\text{-}1$, $y = 0 \ldots q\text{-}1$, and $k = 0 \ldots [(m \times n)/(p \times q)] - 1$

where MV_{row}^{k} and MV_{col}^{k} (for $0 \le MV_x^k, MV_y^k \le 7$) are constant for each pixel in the same MB k. Taking the above example with $p = q = 8$, $m = 16$, $n = 24$, $k = 1$, $MV_{row}^{1} = 6$ and $MV_{col}^{1} = 3$, Equation (3.1) becomes,

$$\hat{c}^1(x,y) = r(i+6, j+3)\big|_{\substack{i=8+x,\\ j=y}} \quad \text{for } x = 0\ldots 7 \text{ and } y = 0\ldots 7 \qquad (3.2)$$

which graphically allows the estimated current block $\hat{c}^1$ to be depicted as the following Figure 3.2:

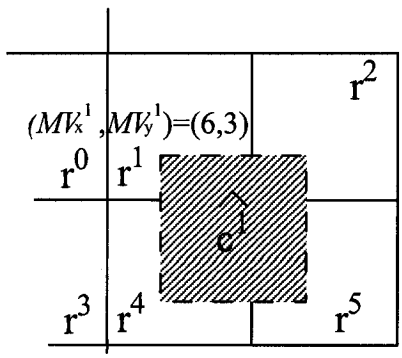


Figure 3.2: Graphical interpretation of Equation (3.2) above where $\hat{c}^1$ straddles four adjacent MBs r^1, r^2, r^4, r^5 in the reference frame.

For MPEG motion-compensated encoding, the previously reconstructed reference frame r (either an I or P frame) is stored, in the spatial domain, in a frame memory for use by the MC operation. In the transcoder proposed in this thesis, however, all previously stored frame data are in the form of DCT-Requantization Error Blocks (REBs). Given that the conventional mechanism of the MC operation only handles blocks in the spatial domain, one way to motion-compensate current DCT-REB from the reference DCT-REB can be achieved in the following manner:

1. Perform Inverse DCT for each DCT-REB to transform it back to the spatial domain. These are the REBs.
2. Motion compensate each REB as defined in Equation (3.1) above.
3. Perform the DCT operation for each REB to transform REB back to DCT domain. These are the DCT-REB's.

Because DCT and inverse DCT are linear operations it is possible to go directly from the DCT-REBs in the reference frame to correction factors for the DCT-REBs in the current frame, via a linear transformation. The particular transformation depends on the Motion Vector for each MB. In the next section, details of the appropriate linear transformations are given.

3.2. Chang and Messerschmitt's algorithm

Chang and Messerschmitt's algorithm [20] proposed that the MC could be performed in the DCT domain by matrix multiplication with an appropriate 8x8 prematrix p_1 and postmatrix p_2 to eliminate the conversion process back and forth between the DCT and the spatial domains.

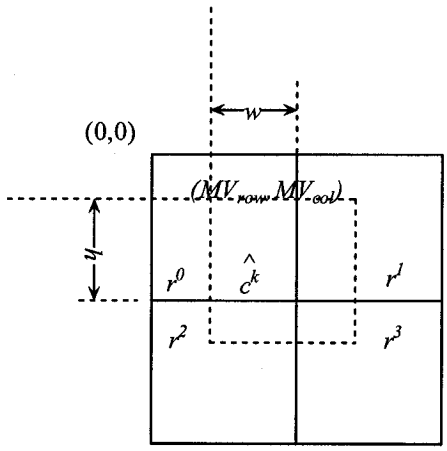


Figure 3.3(a):
In general, estimated $\hat{c}^k$ MB is unlikely to be mapped into fixed block boundary position

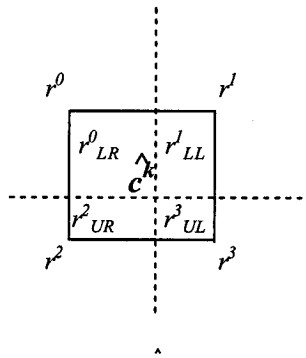


Figure 3.3(b):
$\hat{c}^k$ MB is made up of at most four neighboring MB in reference frame

In the Figure 3.3(a) and 3.3(b) above, the k^{th} estimated block in the current frame, $\hat{c}^k$ is composed of parts of four adjacent 8 x 8 blocks, r^0, r^1, r^2, and r^3 in the reference frame, namely the lower-right corner (r^{0e}_{LR}) *of MB* r^0, the lower-left corner (r^1_{LL}) *of MB* r^1, the upper-right corner (r^2_{UR}) *of MB* r^2 and the upper-left corner (r^3_{UL}) *of MB* r^3. These are exactly the parts of the r^i (i = 0 to 3) that should be used to form $\hat{c}^k$. Notice that the formation of $\hat{c}^k$ requires the translations of r^0_{LR} from lower right corner into upper left corner of $\hat{c}^k$, r^1_{LL} from lower left corner into upper right corner of $\hat{c}^k$, and similarly for r^2_{UR} and r^3_{UL} as shown in the Figure 3.3(b). These translations can be mathematically

achieved by pre and post-multiplying the r^0, r^1, r^2, and r^3, by appropriate matrices, p_{i1} (prematrix) and p_{i2} (postmatrix) where the first subscript designates which r submatrix is being multiplied (i.e. $i = 0...3$) and the second subscript indicates whether this is a pre (1) or post (2) multiply matrix. Note that p_{i1} and p_{i2} also depend on the motion vector; this dependence is suppressed in the notation for readability. These p_{i1} and p_{i2} are defined in Equation (3.3) as follows:

$$\begin{aligned} p_{11} = p_{01} &= \begin{bmatrix} 0 & I_h \\ 0 & 0 \end{bmatrix} \\ p_{02} = p_{22} &= \begin{bmatrix} 0 & 0 \\ I_w & 0 \end{bmatrix} \\ p_{21} = p_{31} &= \begin{bmatrix} 0 & 0 \\ I_{8-h} & 0 \end{bmatrix} \\ p_{12} = p_{32} &= \begin{bmatrix} 0 & I_{8-w} \\ 0 & 0 \end{bmatrix} \end{aligned} \tag{3.3}$$

where I are identity matrices with size specified by their subscript h and w ($1 \le h,w \le 8$ are the number of rows and columns overlapped by part of $\widehat{c^k}$). Also, h and w depend on the motion vector and they are related to MV_{row} and MV_{col} as follows (refer to Figure 3.3):

$$\begin{cases} h = 7 - MV_{row} + 1, & 0 \le MV_{row} \le 7, 1 \le h \le 8 \\ w = 7 - MV_{col} + 1, & 0 \le MV_{col} \le 7, 1 \le w \le 8 \end{cases} \tag{3.4}$$

As an example, for $h = 6$ and $w = 5$, the translations of r^1_{LR} from lower right corner into upper left corner of $\widehat{c^k}$ ($\widehat{c^k_{UL}}$) can be formulated in matrix form as Equation (3.4) below:

$$\widehat{c^k_{UL}} = r^1_{LR} = p_{01} r^1 p_{02} = \begin{bmatrix} 0 & I_6 \\ 0 & 0 \end{bmatrix} r^1 \begin{bmatrix} 0 & 0 \\ I_5 & 0 \end{bmatrix} \tag{3.5}$$

where

$$\begin{bmatrix} 0 & I_6 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \text{ and } \begin{bmatrix} 0 & 0 \\ I_5 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

the effect of premultiplying r^1 by p_{01} is to shift the bottom six rows up vertically and roll in zeros for the two bottom rows while the postmultiplication by p_{02} horizontally translates the rightmost five columns to the left and zeros stuffs the remaining three most right columns. This can be illustrated as in Figure 3.4 below:

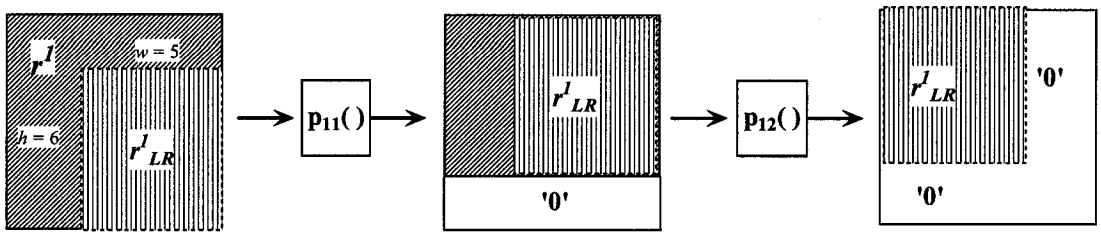


Figure 3.4: Using matrix multiplication to extract r^1_{LR} and translate it to upper right corner

Thus, following the same convention as above, $\widehat{c^k}$ in the spatial domain can be computed by the matrix Equation (3.6)

$$\begin{aligned} \widehat{c^k} &= r^1{}_{LR} + r^2{}_{LL} + r^3{}_{UR} + r^4{}_{UL} = \sum_{i=1}^{4} p_{i1} r^i p_{i2} \\ &= p_{01} r^1 p_{02} + p_{11} r^2 p_{12} + p_{21} r^3 p_{22} + p_{31} r^4 p_{32} \end{aligned} \tag{3.6}$$

For an 8 x 8 matrix $x = x(i,j)\big|_{i,j=0}^{7}$, the 2D Forward DCT (FDCT) transforms x from the spatial domain into the frequency domain X [24]

$$X = X(k,l)\big|_{k,l=0}^{7} \text{ where } X = DCT(x) = DxD^t \tag{3.7}$$

where

$$D = d(e,f) = \frac{a(e)}{2}\cos\left(\frac{2f+1}{16} \bullet e\pi\right)\Big|_{e,f=0}^{7}, \qquad (3.8)$$

$$a(0) = \frac{1}{\sqrt{2}}, a(e) = 1 \quad for \quad e > 0$$

The superscript t denotes matrix transposition. Its counterpart 2D Inverse DCT (IDCT) is then defined as

$$IDCT(X) = x = D^{-1}XD^{-t} = D^{t}XD \qquad (3.9)$$

where the second equality follows from the orthonormality of D (i.e. $D^t=D^{-1}$) [24]. Following from the orthonormal property:

$$DCT(a) \times DCT(b) = DaD^{t}DbD^{t} = DaD^{-1}DbD^{t} = DabD^{t} = DCT(a \times b) \qquad (3.10)$$

The DCT-domain of $\widehat{c^k}$ can then be computed directly from r^1, r^2, r^3, and r^4 by the following,

$$\begin{aligned} \widehat{C^k} = D\widehat{c^k}D^{t} == \sum_{i=0}^{3} D(p_{i1}r^{i}p_{i2})D^{t} = \sum_{i=0}^{3} Dp_{i1}D^{t} \times Dr^{i}D^{t} \times Dp_{i2}D^{t} \\ = \sum_{i=0}^{3} P_{i1}R^{i}P_{i2} \end{aligned} \qquad (3.11)$$

where P_{i1} and P_{i2} are the DCT of p_{i1} and p_{i2} respectively. In the Messerschmidt and Chang paper, they proposed, $DCT(p_{i1})$ and $DCT(p_{i2})$ for $1 \le i \le 4$ can be pre-computed at design time. The straightforward implementation of Equation (3.11) requires $4*2*8^3 = 4096$ multiplications and $4*2*(8-1)*8^2 = 3584$ additions.

In this thesis, we apply this concept to MPEG-4 video transcoding, bit-rate conversion from high bit-rate to low bit-rate in a heterogeneous networking environment.

3.3. DCT-Coefficient-Translation-and-Truncation-Transformation-Matrix (DCTTTM) based Motion Compensation

The development in Section 3.2 was aimed at generating the DCT coefficients of a block from the DCT coefficients of four blocks which it straddled. In contrast, in transcoding the important information from the four straddled blocks is requantization errors, not DCT coefficients. That is, one must use the method of Section 3.2 to take information from the reference DCT-REB's and put it in a form to be used by the current motion compensated DCT-REB. Unlike the algorithms [20] in Section 3.2, DCTTTM motion compensation hides the shifting detail and puts an emphasis on the contribution of each coefficient in the four DCT-REBs of the reference frame to that of one estimated current DCT-REB via a linear operation of matrix multiplication.

Consider a DCT-REB called E. It is an 8 x 8 matrix of DCT Requantization Errors (DCT-RE's). The elements of E are e_{ij}, for $1 \leq i,j \leq 8$ which can be written as:

$$E = \sum_{i=1}^{8} \sum_{j=1}^{8} e_{ij} \times Q_{ij} \qquad (3.12)$$

where Q_{ij} is an 8 x 8 matrix with only one non-zero coefficient '1' designated by the indices ij. For example, for the element in the third row and second column, i=2 and j=1 (index starts from zero):

$$Q_{21} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad (3.13)$$

The IDCT of Equation (3.12), due to the property of linearity, can be written as:

$$IDCT(E) = \sum_{i=0}^{7}\sum_{j=0}^{7} e_{ij} \times IDCT(Q_{ij}) \qquad (3.14)$$

The DCT of the Q_{ij} matrices can be pre-computed at design time.

Similar to the scenario illustrated in the Figure 3.3(a), four 8*8 blocks of DCT-RE's in the reference frame will contribute to the DCT-REB in the current motion compensated block. Call these DCT-RE's $e_{ij}(n)$ where n indicates which of the four blocks the DCT-RE originated from (using the numbering convention of Figure 3.1) and i and j indicate which DCT-RE. Given a specific $\overrightarrow{MV^k}$, the motion compensated DCT-REB, in the current frame (call it E_{MC}), can be written as a linear combination of the 256 DCT-RE's in the four straddled blocks in the reference frame. That is:

$$E_{MC} = \sum_{n=0}^{3}\sum_{i=0}^{7}\sum_{j=0}^{7} e_{ij}(n) \times \hat{Q}_{ij}(n) \qquad (3.15)$$

where $\hat{Q}_{ij}(n)$ is called the DCT-Correction Matrix (DCT-CM).

In deriving the $\hat{Q}_{ij}(n)$, three steps are followed. First, each of the four straddled blocks of DCT-RE's in the reference frame are put in the space domain via an IDCT. (Note that this is a linear operation, and is independent of the motion vector). Second, given the motion vectors, appropriate pieces of the four straddled blocks (now in the space domain) are partitioned off and assembled to form an 8*8 block. This partitioning is a linear operation. Third, the assembled block is returned to the DCT domain. This is, of course, a linear operation.

Mathematically:

Step 1:

$$q_{ij}(n) = IDCT(Q_{ij}(n))\Big|_{\substack{i,j=0..7 \\ n=0..3}} \tag{3.16}$$

where $q_{ij}(n)$ is the space domain representation of the $Q_{ij}(n)$ via an IDCT. As mentioned above, this operation is independent of the motion vector, therefore:

$$q_{ij}(0) = q_{ij}(1) = q_{ij}(2) = q_{ij}(3) \quad \text{i,j=0..7} \tag{3.17}$$

Step 2:

Given the motion vector (MV_{row},MV_{col}) and based on Equations (3.4) and (3.5), the assembled block, call it $q\_shift_{ij}(n)$ can be found as follows:

$$q\_shift_{ij}(n) = p_{(n1)} * q_{ij}(n) * p_{(n2)}\Big|_{\substack{i,j=0...7 \\ n=0...3}} \tag{3.18}$$

As in Equation 3.3 $p_{(n1)}$ is the matrix which when it pre-multiplying matrix $q_{ij}(n)$ effects the appropriate partition for the given motion vector. Thus $p_{(n1)}$ is dependent on the motion vector. Specifically, the motion vector generates h and w from Equation 3.4 and then these are used in Equation 3.3 to generate the appropriate $p_{(n1)}$. Similar comments can be made for the post-multiply matrix $p_{(n2)}$.

Step 3:

The assembled block $q\_shift_{ij}(n)$ is transformed back to the DCT domain via linear DCT operation:

$$\hat{Q}_{ij}(n) = DCT(q\_shift_{ij}(n))\Big|_{\substack{i,j=0..7 \\ n=0..3}} \tag{3.19}$$

Note that there are a total of 256 8 x 8 $\hat{Q}_{ij}(n)$ which is referred to as DCT-CMs.

3.3.1 Example Generation of a DCT-CM $\hat{Q}_{ij}(n)$

Assume $MV_{row} = 0$, $MV_{col} = 4$, $i = 2$ $j = 1$ and $n = 0$, therefore $\hat{Q}_{21}(0)$ can be computed as below:

Step 1: Perform the IDCT on Q_{21}.

$$q_{21}(0) = IDCT(Q_{21}) = IDCT\left(\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0.2265 & 0.1920 & 0.1283 & 0.0451 & -0.0451 & -0.1283 & -0.1920 & -0.2265 \\ 0.0938 & 0.0795 & 0.0532 & 0.0187 & -0.0187 & -0.0532 & -0.0795 & -0.0938 \\ -0.0938 & -0.0795 & -0.0532 & -0.0187 & 0.0187 & 0.0532 & 0.0795 & 0.0938 \\ -0.2265 & -0.1920 & -0.1283 & -0.0451 & 0.0451 & 0.1283 & 0.1920 & -0.2265 \\ -0.2265 & -0.1920 & -0.1283 & -0.0451 & 0.0451 & 0.1283 & 0.1920 & 0.2265 \\ 0.0938 & -0.0795 & -0.0532 & -0.0187 & 0.0187 & 0.0532 & 0.0795 & 0.0938 \\ 0.0938 & 0.0795 & 0.0532 & 0.0187 & -0.0187 & -0.0532 & -0.0795 & -0.0938 \\ 0.2265 & 0.1920 & 0.1283 & -0.0451 & -0.0451 & -0.1283 & -0.1920 & -0.2265 \end{bmatrix} \quad (3.20)$$

Step 2: Translate the spatial-domain $q_{21}(0)$ into the location specified by (MV_{row}, MV_{col}).

In the case of n=0, $MV_{row} = 0$ and $MV_{col} = 4$ (i.e., $h = 8$, w = 4 from Equation 3.4), $q\_shift_{21}(0)$ using Equation 3.18 is:

$$q\_shift_{21}(0) = p_{01} * q_{21}(0) * p_{02} = \begin{bmatrix} 0 & I_h \\ 0 & 0 \end{bmatrix} * q_{21}(0) * \begin{bmatrix} 0 & 0 \\ I_w & 0 \end{bmatrix}$$

$$= \begin{bmatrix} -0.0451 & -0.1283 & -0.1920 & -0.2265 & 0 & 0 & 0 & 0 \\ -0.0187 & -0.0532 & -0.0795 & -0.0938 & 0 & 0 & 0 & 0 \\ 0.0187 & 0.0532 & 0.0795 & 0.0938 & 0 & 0 & 0 & 0 \\ 0.0451 & 0.1283 & 0.1920 & -0.2265 & 0 & 0 & 0 & 0 \\ 0.0451 & 0.1283 & 0.1920 & 0.2265 & 0 & 0 & 0 & 0 \\ 0.0187 & 0.0532 & 0.0795 & 0.0938 & 0 & 0 & 0 & 0 \\ -0.0187 & -0.0532 & -0.0795 & -0.0938 & 0 & 0 & 0 & 0 \\ -0.0451 & -0.1283 & -0.1920 & -0.2265 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (3.21)$$

Step 3) Perform the DCT on $q\_shift_{21}(0)$:

$$\hat{Q}_{21}(0) = DCT(q\_shift_{21}(0))$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.4531 & -0.3266 & 0.2079 & 0.3266 & 0.0373 & -0.1353 & 0.0114 & 0.1353 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (3.22)$$

Thus if the DCT-RE is 1 for the DCT position in the 3rd row (i=2) and 2nd column (j=1) for the DCT Block in the reference frame that forms the upper left portion of the estimated block (n=0), the above $\hat{Q}_{21}(0)$ represents the resulting error in each of the 64 DCT coefficients of the residue. This can be seen mathematically in Equation 3.15. Thus, given the (MV_{row}, MV_{col}) pair, there are a total of 256 8 x 8 DCT-CMs (the $\hat{Q}_{ij}(n)$) generated in a similar fashion as $\hat{Q}_{21}(0)$.

Rearranging each of the 8 x 8 DCT-CMs into 64 x 1 vectors and making these vectors the columns of a new composite matrix W of size 64 x 256. Converting the 8*8 matrix E_{MC} (the motion compensated DCT-REB from Equation 3.15) into a 64 x 1 vector, called $\hat{Z}$, Equation 3.15 can be rewritten as:

$$\hat{Z} = \sum_{i=0}^{255} e_i \times W_i \qquad (3.23)$$

where e_i again for $0 \le i \le 255$ are the DCT-RE's of the four overlapped MB r^0, r^1, r^2, r^3, and W_i for $0 \le i \le 255$ is 64 x 1 and the ith column of the 64 x 256 composite matrix W. For all 64 combinations of MV_{row} and MV_{col} where $0 \le MV_{row}, MV_{col} \le 7$, there is a particular composite matrix W associated and hence 64 such composite matrices are generated. The computational complexity of implementing Equation (3.23) requires 256*64 = 16384 multiplications and (256-1)*64 = 16320 additions for each motion compensated block.

In comparison with Chang and Messerschmitt's algorithm reviewed in Section 3.2 which requires 4096 multiplication and 3584 additions, the multiplication-and-add count of the DCTTTM approach is greater. However, the e_i's are formed by the requantization of already quantized (from the first compression) DCT coefficients. Many of these already quantized DCT coefficients are in fact 0. Note that if a DCT coefficient was quantized to 0 in the first quantization its e_i is also 0. For example if it is known that, all of the DCT coefficient in the 7th row and all of the DCT coefficients in the 7th column were always (or almost always) 0 there would be no need to calculate the corresponding terms in Equation 3.23. Thus we could replace the 256*64 composite matrix by a 256*49 composite matrix. This will result in a reduction of computation.

In the second quantization, higher DCT frequencies will tend to be quantized much more heavily than lower frequencies. This, coupled with the fact that these DCT coefficients are small (even with the addition of the DCT-RE's), means that one is likely to end up with zero in these coefficients after the second quantization. This observation suggests that we may neglect rows in the composite matrix W corresponding to higher frequencies. This further reduces computation.

In the next section an analysis will be done to determine which rows and columns of the composite matrix W may be neglected.

3.4. Analysis of computational complexity for DCTTTM-based MC-DCT using partial DCT information

For Equation (3.17) to render one DCT-REB, all the correction factors of the composite matrix W from every location of four overlapped DCT-REBs in the reference frame are taken into account. Consequently, this approach induces a very high cost in term of computational complexity. However, it is known that typically the low frequency DCT coefficients contain most of the energy of the signal, and also that these frequencies are most important for human perception [33]. Thus one may reduce the computational complexity of the DCTTTM algorithm by using only partial DCT information.

For every coefficient discarded, the quality of the video sequence is expected to be degraded. Consequently, it would be preferable to keep the most significant DCT coefficients, with the others discarded to maximize computational savings.

Simulations were perfeormed on several video sequences in order to make judgments of which coefficients should be kept. Table 3.1, 3.2, and 3.3 show the average

percentage of non-zero DCT-REs for the video sequences of 150 frames, table-tennis, flowergarden and Miss America respectively:

Table 3.1: Average Percentage of Non-Zero Coefficient for Table Tennis
(Grid location below corresponding to actual positioning of a 8x8 block)

71.2	64.1	59.9	58.6	56.4	55.0	51.5	48.6
64.7	58.2	55.3	53.5	52.3	50.1	47.3	44.2
61.3	56.1	52.9	51.4	50.1	48.1	45.4	42.2
60.2	55.6	52.6	51.0	49.3	47.4	44.6	41.5
58.9	55.4	52.4	50.6	48.8	46.6	44.0	40.6
58.2	54.1	51.1	49.3	47.7	45.7	42.6	39.5
56.2	52.9	50.4	48.3	46.2	44.5	41.5	38.3
54.4	51.5	48.3	46.2	44.6	42.7	39.9	36.5

Table 3.2: Average Percentage of Non-Zero Coefficient for Flowergarden
(Grid location below corresponding to actual positioning of a 8x8 block)

86.3	78.9	76.0	74.3	72.4	70.9	68.6	65.9
79.6	75.2	73.5	71.9	70.2	68.7	66.8	63.7
76.1	73.4	72.1	70.6	69.1	67.8	65.7	62.9
74.2	72.0	70.7	69.8	68.2	67.0	64.8	61.8
72.7	70.9	69.6	68.7	67.5	66.2	63.9	61.1
71.2	69.7	68.5	67.5	66.2	65.0	62.8	60.1
68.5	67.6	66.5	65.5	63.9	62.7	60.8	58.2
64.7	64.1	63.2	62.3	60.8	59.8	58.0	55.7

Table 3.3: Average Percentage of Non-Zero Coefficient for Miss America
(Grid location below corresponding to actual positioning of a 8x8 block)

88.6	59.3	49.6	42.2	44.0	42.4	37.9	40.9
60.1	47.3	41.8	35.6	31.4	29.6	29.9	33.0
49.2	42.0	37.6	32.6	28.8	27.1	26.4	27.7
43.1	37.1	33.1	29.0	25.8	24.1	22.7	21.8
37.4	32.7	29.2	26.1	23.3	22.1	20.3	20.6
33.1	28.6	25.5	23.1	20.9	19.6	18.1	17.8
29.6	26.5	23.3	22.2	19.9	19.8	18.3	21.4
24.3	22.1	19.2	18.4	16.2	16.3	15.2	17.7

Notice that the average percentages of non-zero coefficients decrease as one gets further from the top left corner of the DCT matrices. These are of course the low

frequencies. Following on this observation, in this thesis schemes which preserve only these coefficients are studied. Figure 3.5b-e) show four schemes labeled 4-3-2-1, 3-2-1, 2-1 and DC. A dot indicates that the corresponding DCT coefficient will be used.

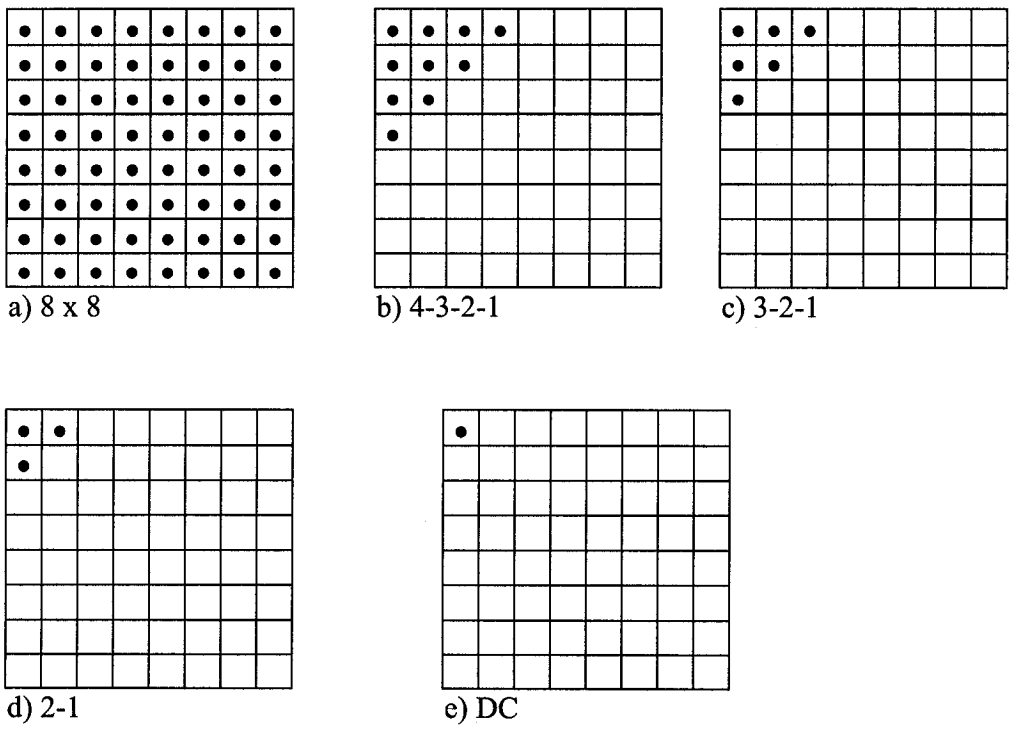


Figure 3.5: Partial DCT-REs information. • refer to the locations of an 8 x 8 DCT Requantization Error Block (DCT-REB) where values are taken into account for the computation of motion compensation and assumed to be non-zero. Non-marked locations are ignored and assumed to be zero-valued.

For example, in the case of Figure 3.5(c), we assume only the six RE's on the top-left corner are nonzero-valued and zero elsewhere and refer to it as the 3-2-1 scheme. For the DCTTTM algorithm, the 3-2-1 scheme implies the computation of one motion-compensated REB from six RE's in each of the four overlapped REBs in the reference frame. Thus, a total of twenty-four RE's are used. Applying this scheme to Equation 3.15, E_{MC} (now call it E_{MC_321}) becomes:

$$E_{MC\_321} = \sum_{n=0}^{3} \sum_{i=0}^{2} \sum_{j=0}^{2-i} e_{ijn} \times \hat{Q}_{ijn} \qquad (3.24)$$

For each RE, there is a $\hat{Q}$ matrix associated to the transformation. Therefore, rather than having a composite-matrix W of size 64 x 256 where all four REBs locations are taken into account, the matrix W is shrunk to 64 x 24. Further computational reductions can be achieved by applying the partial information scheme to the motion-compensated REB; of the 64 REs coefficients, only the six located in the top-left corner designated by the 3-2-1 scheme are assumed to be non-zero value. By doing so, the size of the composite matrix W can be scaled down from its original 64 x 256 to 6 x 24. The computational complexity of one motion-compensated REB is thus reduced to a simple matrix-vector multiplication of 6 x 24 times 24 x 1. The Equation of (3.24) becomes:

$$\hat{Z}_{321} = \sum_{i=0}^{23} e_i \times W_{321i} \qquad (3.25)$$

where e_i again for $0 \le i \le 23$ is the DCT-RE's of the four overlapped MB r^0, r^1, r^2, r^3 as per 3-2-1 scheme, W_{321i} is the 6 x 1 DCT vector derived from the ith column of the 64 x 24 matrix W_{321} and $\hat{Z}_{321}$ is the motion-compensated DCT-REB. The direct implementation of Equation (3.18) matrix-vector multiplication requires 24*6 = 144 mults and 23*6 = 138 adds. Table 3.4 outlines the computational cost for the five schemes:

Table 3.4: Computation cost in term of mult-and-add count for different partial information scheme				
	Size of the composite-matrix (row x col)	**Mult. Count**	**Add Count**	**% of Mult Count reduction relative to 8x8**
8 x 8	64 x 256	16384	16320	N/A
4-3-2-1	10 x 40	400	390	97.56%
3-2-1	6 x 24	144	138	99.12%
2-1	3 x 12	36	33	99.78%
DC	1 x 4	4	3	99.98%

This section has examined the potential savings in computation to be reaped by neglecting the errors in some DCT-REs in the reference frame and in omitting the calculation of the effect of all errors on certain DCT coefficients in the residue of the current frame under different schemes. In the next section, the cost in both subjective and objective quality implied by omitting these coefficients is examined.

3.5. SIMULATION STUDY ON THE IMPACT OF DIFFERENCE SCHEMES OF PARTIAL INFORMATION USED TO VIDEO QUALITY

Both subjective measure and objective measure are used to assess the tradeoff between computational cost and video quality, when one considers only partial DCT-RE information, The video sequence *Table Tennis* is simulated. The MoMuSys MPEG-4 encoder and decoder software V1.0 [26] are used in the simulations. Only integer (full) pixel motion estimation is employed limiting the motion vector to only 64 possible displacements with respect to the top-left corner pixel location of the top-left block of the four straddled blocks. *IPP...IPP* structure with M=1 (Number of B-VOPs [M-1] between two consecutive P-VOPs) and N=15 (Number of P-VOPs [M-1] between two consecutive

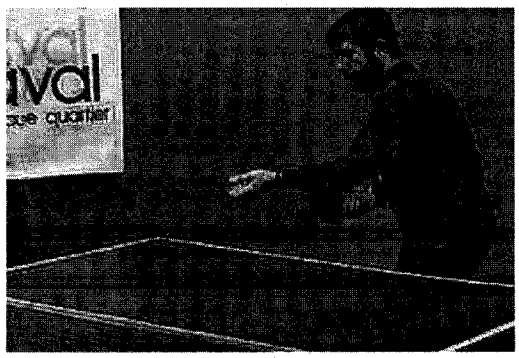
I-VOPs) transcoding from 5Mbps to 256Kbps for various cases. The transcoder software was based on the work of Mr. Hong Quan Chen [25] and was modified to implement different schemes of partial information. The same set of parameters is applied in all the simulations for the rest of this chapter.

(a) Open-Loop (No MC)

(b) DC-only scheme

(c) 2-1 scheme

(d) 3-2-1 scheme



(e) 4-3-2-1 scheme

(f) Full scheme (all 64 coeffs considered)

Figure 3.6 (a)-(e): the 61th motion-compensated frame of video sequence *Table Tennis* using different partial information scheme

Subjectively, the simulation suggests that for an open-loop implementation (Figure 3.6(a)) without drift correction, the blocking artifacts are clearly noticeable, severely degrading the visual quality when compared to full pixel drift error compensation (Figure 3.6 (e)). While there is still noticeable quality improvement by employing the 3-2-1 scheme (Figure 3.6 (d)) over the 2-1 scheme (Figure 3.6(c)), there is virtually no noticeable enhancement in the 4-3-2-1 scheme from the 3-2-1 scheme.

Figure 3.7 shows the average PSNR for *Table Tennis* using different schemes in the transcoding process. The curve in Figure 3.7 exhibits an enhancement of around 1dB as one moves from one scheme to the next.

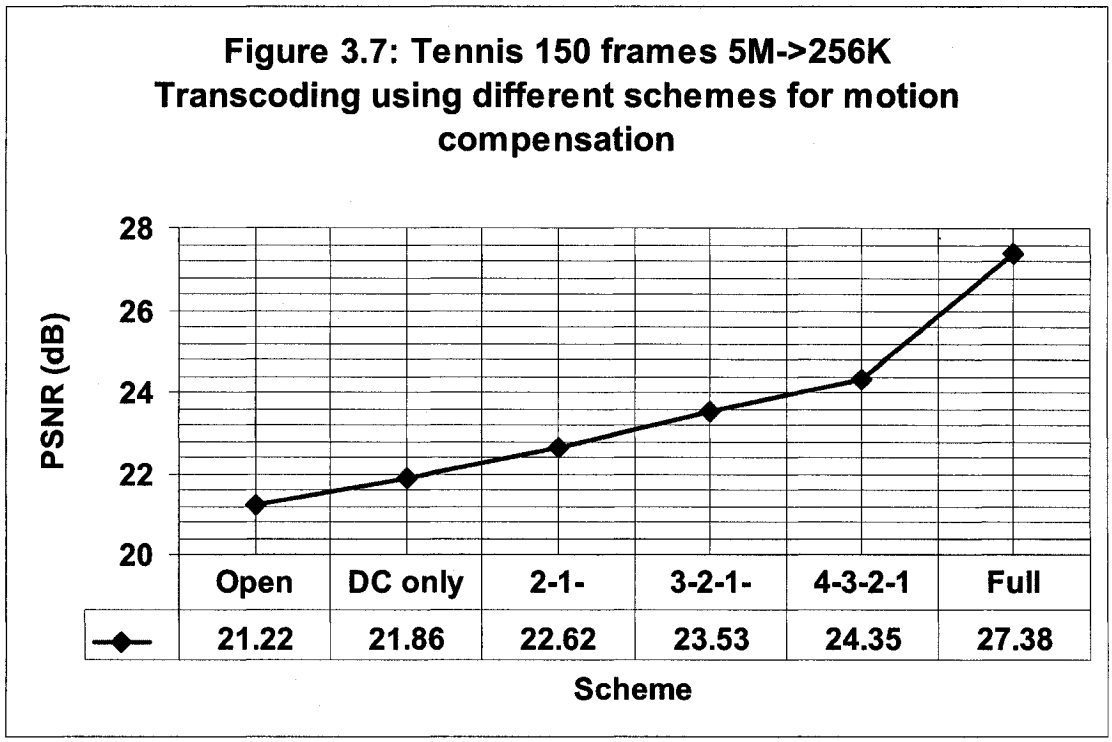


Figure 3.7: Average PSNR for Tennis 150 frames Transcoding using different schemes for motion compensation

This section has examined the cost in quality, measured both subjectively and objectively, in either neglecting the effects of errors in some DCT-RE coefficients in the reference frame, or in omitting the calculation of the effect of all errors on certain DCT coefficients in the residue of the current frame. Given our goal is to reduce the computational cost to the greatest extent possible (99.12% gain in computational saving when comparing with full 8 x 8 full information scheme) while minimizing video quality degradation, the analysis in the previous two sections suggest that the 3-2-1 Scheme provides a fair balance between these conflicting requirements. For the rest of this thesis, the 3-2-1 Scheme is the algorithm used for implementing the MC-DCT module.

In addition to employing the 3-2-1 Scheme, further computational savings may be achieved by reducing the precision of constants in the DCT Correction Matrices (DCT-CMs) $\hat{Q}$. In the next section, an appropriate quantizer design and the effect of the subsequent quantization inherent in transcoding on video quality are examined.

3.6 Quantization over Constant Transformed Coefficients

The precision in bits is a very important parameter for hardware implementation. Each additional bit of precision costs extra resources. To further reduce computations this thesis considers heavily quantizing the constants in the DCT-CMs $\hat{Q}$,

3.6.1. Design of Quantizer for the elements of DCT-CMs

The design of an appropriate quantizer for these constant coefficients should depend on the characteristics of these matrices. First, there are many zeros within these matrices, thus a zero reconstruction level is required. This suggests a mid-step quantizer. However, the drawback to mid-step quantizer is that there is always an odd number of

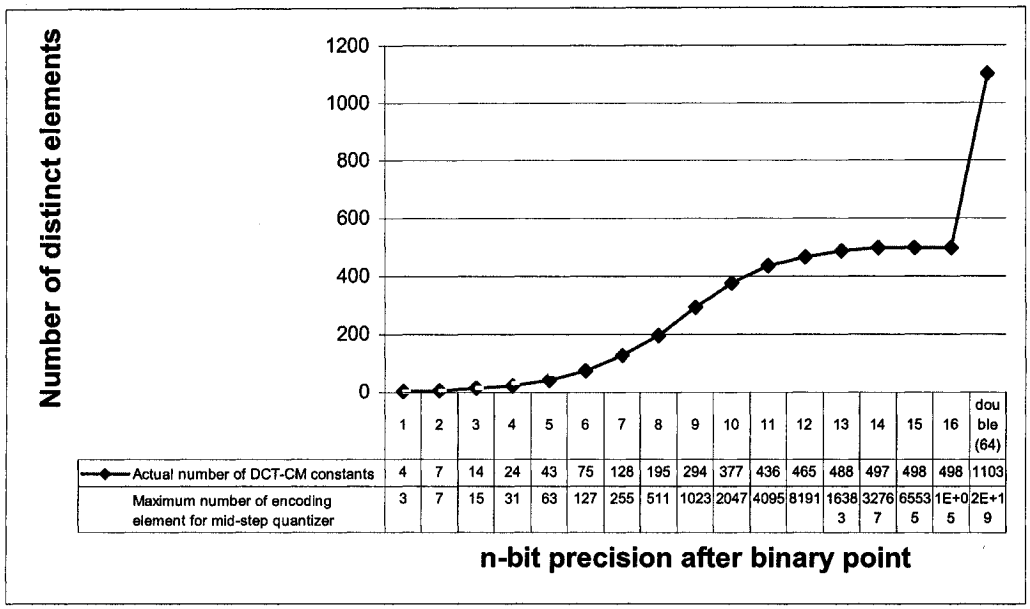
reconstruction levels. Thus, in a hardware implementation, the reconstruction levels can not be economically represented in a base-two numerical system.

In the following, if there are n bits to the right of the binary point we say that we are using n bit precision. Note that in fact, an extra bit is required. Specifically n bit precision requires n+1 bits. For example, if 2-bit precision after binary point is used (ie. Step size of 0.25) to quantize the DCT-CM constants, $7 \leq 2^{(1+2)}=8$ distinct elements result; they are 1, 0, ± 0.25, 0.75, ± 0.5 (notice that -0.75 and -1 never appear in examining all DCT-CMs). In other words, these quantized values can be encoded using 3 bits only. Table 3.5 shows one possible encoding scheme:

Table 3.5 Possible encoding scheme for DCT-CM constants of 2-bit precision with one integer bit

Actual Quantized Value	Encoded Value in binary
0	000
-0.25	001
-0.5	010
0.25	011
0.5	100
0.75	101
1	110

Inside each of the 64 DCT-CM's there are only a finite number of *distinct* values. As these values are more heavily quantized, there will be fewer and fewer *distinct* values. Figure 3.8 shows the result by exhaustively counting the number of distinct values in all 64 DCT-CM's for different bit-precisions.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	double (64)
Actual number of DCT-CM constants	4	7	14	24	43	75	128	195	294	377	436	465	488	497	498	498	1103
Maximum number of encoding element for mid-step quantizer	3	7	15	31	63	127	255	511	1023	2047	4095	8191	16383	32767	65535	1E+05	2E+19

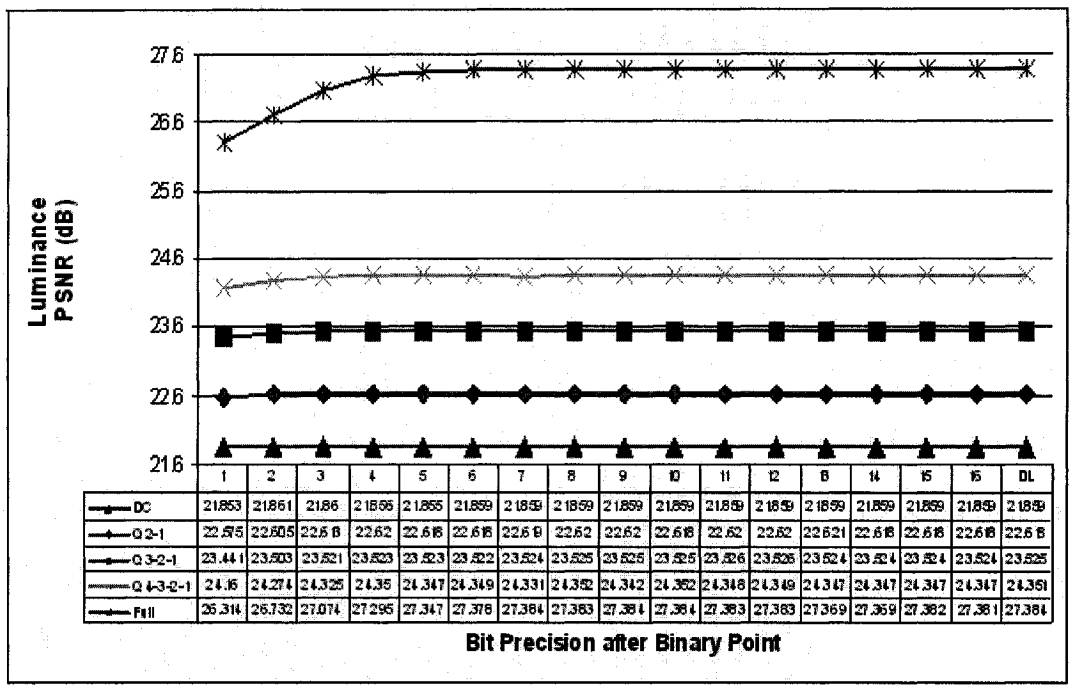
Figure 3.8: The relation between the number of bit-precision (after binary point) used and the number of resulted distinct quantized DCT-CM elements

It should be noted from Figure 3.8, with $2^{(1+n)}$ -1 bit budget, (except for one bit precision scheme), all other precision schemes could safely and economically encode the quantized DCT-CM constants.

In the next section, different precision schemes are employed to run the simulations against five video test sets to assess the impact of PSNR quality versus bit precision.

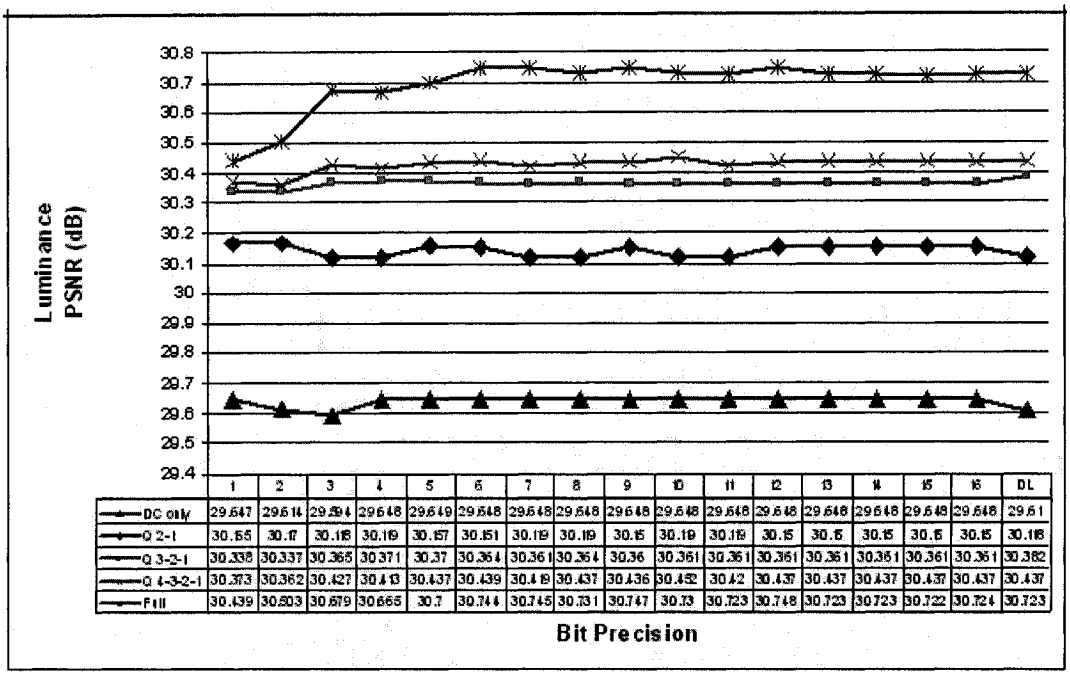
3.7. Simulation results of PSNR performance for DCTTTM-based MC-DCT with n-bit precision after binary point

Here simulation results are given that examine the various partial information schemes (DC only, 2-1, 3-2-1 and 4-3-2-1), as well as different bit precisions. Figure 3.9 to 3.12 shows the PSNR video quality measure for five video sequences, *Table Tennis, Flowergarden, Football, Miss America,* and *Mobile.* Note again the horizontal-axis represents the bit precision used after the binary point.



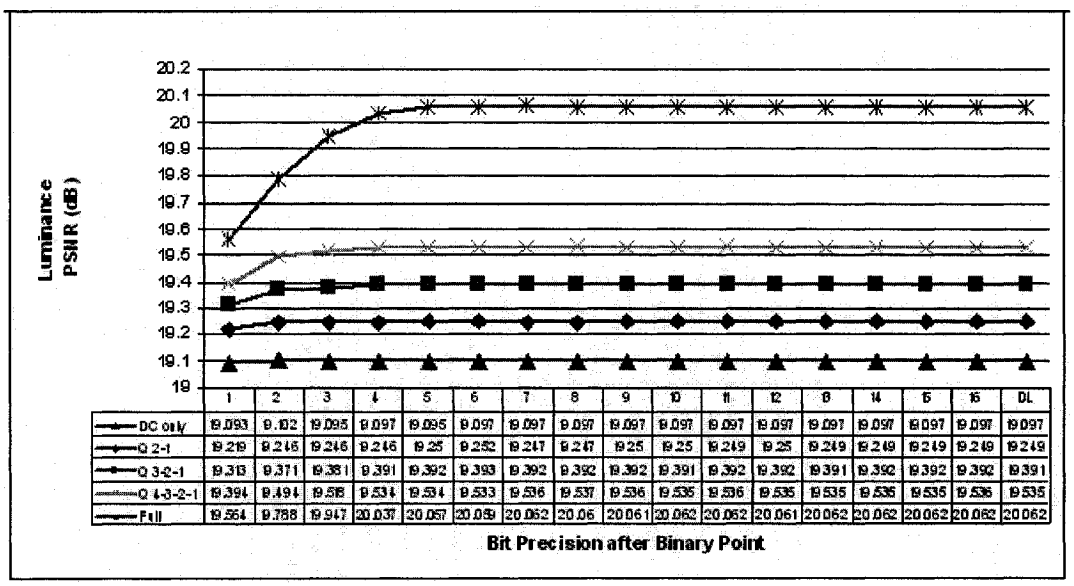
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	DL
DC	21.853	21.861	21.86	21.856	21.855	21.859	21.859	21.859	21.859	21.859	21.859	21.859	21.859	21.859	21.859	21.859	21.859
Q 2-1	22.575	22.605	22.6 8	22.62	22.618	22.618	22.6 9	22.62	22.62	22.618	22.62	22.62	22.621	22.618	22.618	22.618	22.6 8
Q 3-2-1	23.441	23.503	23.521	23.523	23.523	23.522	23.524	23.525	23.525	23.525	23.526	23.526	23.524	23.524	23.524	23.524	23.525
Q 4-3-2-1	24.16	24.274	24.325	24.35	24.347	24.349	24.331	24.352	24.342	24.352	24.348	24.349	24.347	24.347	24.347	24.347	24.351
Full	26.314	26.732	27.074	27.295	27.347	27.376	27.384	27.383	27.384	27.384	27.383	27.383	27.369	27.359	27.382	27.381	27.384

Figure 3.9 PSNR measure on transcoding *Table Tennis* using different partial information schemes for motion compensation



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	DL
DC only	29.647	29.64	29.694	29.648	29.649	29.648	29.648	29.648	29.648	29.648	29.648	29.648	29.648	29.648	29.648	29.648	29.61
Q 2-1	30.165	30.17	30.168	30.169	30.167	30.151	30.169	30.169	30.15	30.169	30.169	30.15	30.15	30.15	30.15	30.15	30.168
Q 3-2-1	30.338	30.337	30.365	30.371	30.37	30.364	30.361	30.364	30.36	30.361	30.361	30.361	30.361	30.361	30.361	30.361	30.362
Q 4-3-2-1	30.373	30.362	30.427	30.413	30.437	30.439	30.419	30.437	30.436	30.432	30.42	30.437	30.437	30.437	30.437	30.437	30.437
Full	30.439	30.503	30.679	30.665	30.7	30.744	30.745	30.731	30.747	30.73	30.723	30.748	30.723	30.723	30.722	30.724	30.723

Figure 3.10 PSNR measure on transcoding *Miss America* using different partial information schemes for motion compensation



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	DL
DC only	19.093	19.102	19.095	19.097	19.095	19.097	19.097	19.097	19.097	19.097	19.097	19.097	19.097	19.097	19.097	19.097	19.097
Q 2-1	19.219	19.216	19.216	19.216	19.25	19.252	19.247	19.247	19.25	19.25	19.249	19.25	19.249	19.249	19.249	19.249	19.249
Q 3-2-1	19.313	19.371	19.381	19.391	19.392	19.393	19.392	19.392	19.392	19.391	19.392	19.392	19.391	19.392	19.392	19.392	19.391
Q 4-3-2-1	19.394	19.494	19.518	19.534	19.534	19.533	19.536	19.537	19.536	19.535	19.536	19.535	19.535	19.536	19.535	19.536	19.535
Full	19.564	19.788	19.947	20.037	20.057	20.059	20.062	20.06	20.061	20.062	20.062	20.061	20.062	20.062	20.062	20.062	20.062

Figure 3.11 PSNR measure on transcoding *Football* using different partial information schemes for motion compensation

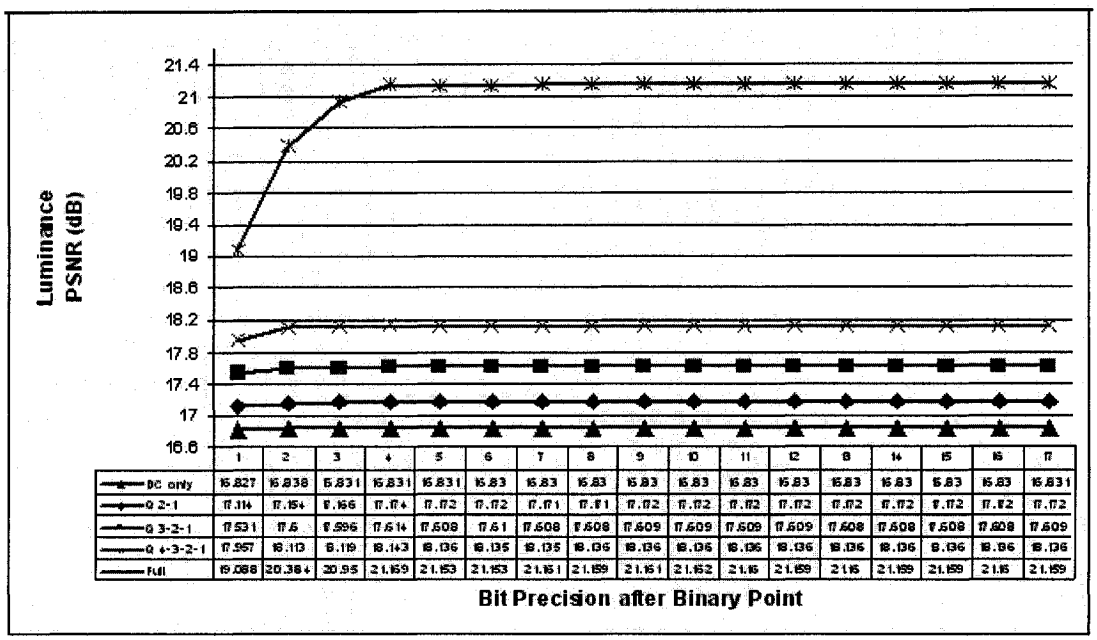


Figure 3.12 PSNR measure on transcoding *Flowergarden* using different partial information schemes for motion compensation

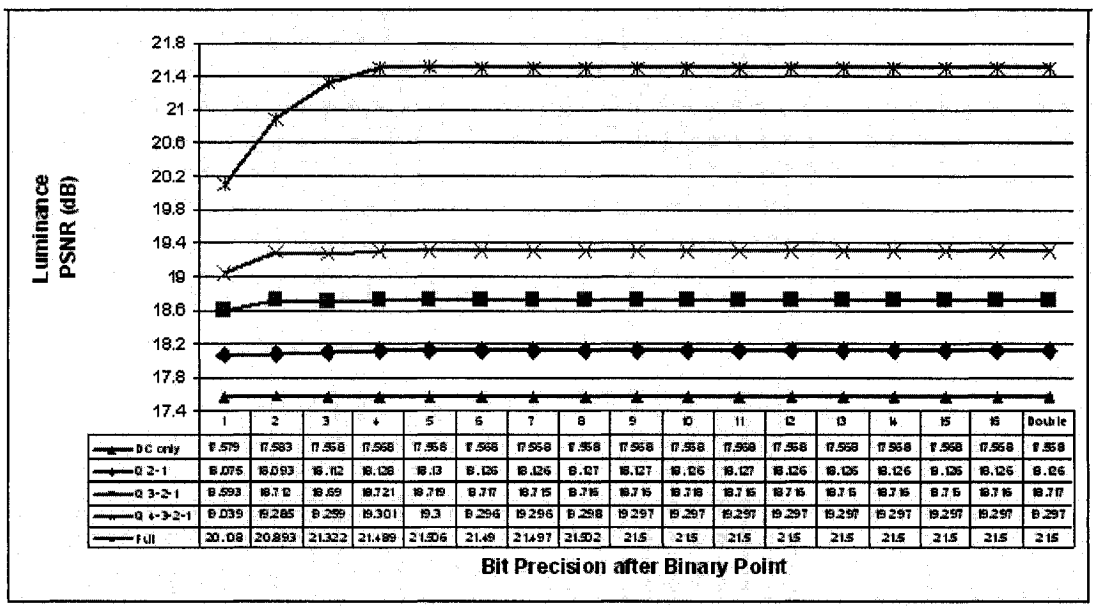


Figure 3.13 PSNR measure on transcoding *Mobile* using different partial information schemes for motion compensation

Surprisingly the reduction in bit precision for the DCT-CMs do not severely affect the PSNR measure among the five test sets. In fact, the only noticeable gain (around 0.1 to 0.2 dB improvement in PSNR) occurs in the most significant digits (up to three digits after binary point) of the DCT-CMs constants. Further increases of the bit precision do not yield any significant improvement in PSNR measure.

These observations construct a solid grounding to achieve considerable computational savings in the hardware implementation by quantizing the DCT-CMs constant to only the most significant digits.

On the basis of the analyses and simulation results from Section 3.4, 3.5, 3.6 and 3.7, the 3-2-1 partial information scheme along with the 2-bit precision for quantized DCT-CMs constants will be used for the hardware implemetation of the MC-DCT module in the next chapter.

3.8. CHAPTER SUMMARY

In this chapter, the MC-DCT algorithm introduced in the literature by Chang and Messerschmitt's was reviewed; a different approach, DCTTTM-based MC-DCT algorithm by Hong Quan Chen [25], was also described. Based on the assumed sparse nature of the Requantization Error Block (REB), the 3-2-1 partial information scheme was integrated into the DCTTTM-based algorithm to process the MC-DCT operation. Further reductions in computing power were achieved by the quantization of the DCT-CM constants into two bit precision.

In the next chapter, details are given on the hardware design of the MC-DCT module.

Chapter 4

IMPLEMENTATION OF MC-DCT MODULE

In this chapter, the data-dependent processing concept is applied to implement the MC-DCT module for the 3-2-1 scheme with the DCT-CM constants quantized to two bit precision after binary point. Further reduction of power consumption can be achieved by carefully designing the width of the data bus of the incoming coefficients - twenty-four reference DCT-REs (DCT-Requantization Error). Since our focus is on the arithmetic level and implementation level, no optimization on the circuit level or technology level is made. The optimization of the implementation level is done on the code-level via the VHDL hardware programming language and the Synopsys compiler was chosen as the tools for ultimately synthesizing the module into a Xilinx FPGA Virtex II XC2V3000 model [27]. Notice that the choice of FPGA model is of little importance as the design can be applied to different FPGA technologies.

Section 4.1 outlines the MC-DCT block diagram with the interconnectivity of all the sub-modules. Detailed discussion of each component of the MC-DCT module takes place in Section 4.2; optimization techniques applied to these components in order to reduce power consumption are studied. In Section 4.3, the synthesized results of the custom design with Xilinx FPGA device of the MC-DCT module are presented. These results are to be compared to the standard design where the same MC-DCT unit is implemented in a conventional multiplier and adder.

4.1. Block Diagram of MC-DCT module

The core function of MC-DCT is to perform Motion Compensation in the DCT domain. By applying the 3-2-1 scheme with the DCT-CM constants quantized to two bit precision after binary point to the MC-DCT module proposed in Chapter 3, the implementation can be modeled mathematically by Equation (4.1) as follow:

$$\hat{Z}_{321} = \sum_{i=0}^{23} e_i \times W_{321i} \qquad (4.1)$$

where e_i for $0 \le i \le 23$ is the DCT-RE's of the four overlapped MB r^0, r^1, r^2, r^3 as per the 3-2-1 scheme and W_{321i} is the 6 x 1 DCT vector derived from the ith column of the 64 x 24 matrix W_{321}. Note that the derivation of Equation (4.1) can be referenced to Section 3.4. Alternatively, Equation (4.1) in its vector form can be expressed as

$$\hat{Z}_{321} = W_{321} \times E \qquad (4.2)$$

where E is a vector of size 24 x 1 composed of twenty-four DCT-REs from the four overlapped MB and W_{321} is the DCT-CM matrix of size 6 x 24. The $\hat{Z}_{321}$ is a MC-DCT vector of size 6 x 1 corresponding to the six motion-compensated DCT-RE elements from the top-left corner of a single block designated by the 3-2-1 scheme. Essentially, Equation (4.2) is a matrix-vector multiplication and its execution involves two simple steps. First, the elements in each row (from left to right) of W_{321} get multiplied to the vector E (from top to bottom). Since the multiplication taking place from a given row does not depends on the result of any other neighbouring rows, the multiplication for each row can be carried out simultaneously. The second step is to generate the resulting vector by summing all the intermediate products previously computed for each row. In our case, such multiplication yields a resulting vector of size 6 x 1. It should be noted that

this vector is essentially a single block of DCT-RE with the top left corner of six coefficients and all others zero.

The hardware design of MC-DCT module follows closely these two execution steps illustrated for Equation (4.2). Figure 4.1 shows the block diagram of the design. For concurrent processing, the number of the MC-DCT modules needed to be deployed depends on the size of the video frame. For example, a frame of 350x288 can be partitioned into 1584 8 x 8 blocks, therefore a minimum of 1584 MC-DCT modules are required. In addition, the design is intended to operate in synchronous clock-driven mode to prevent any race condition from happening.

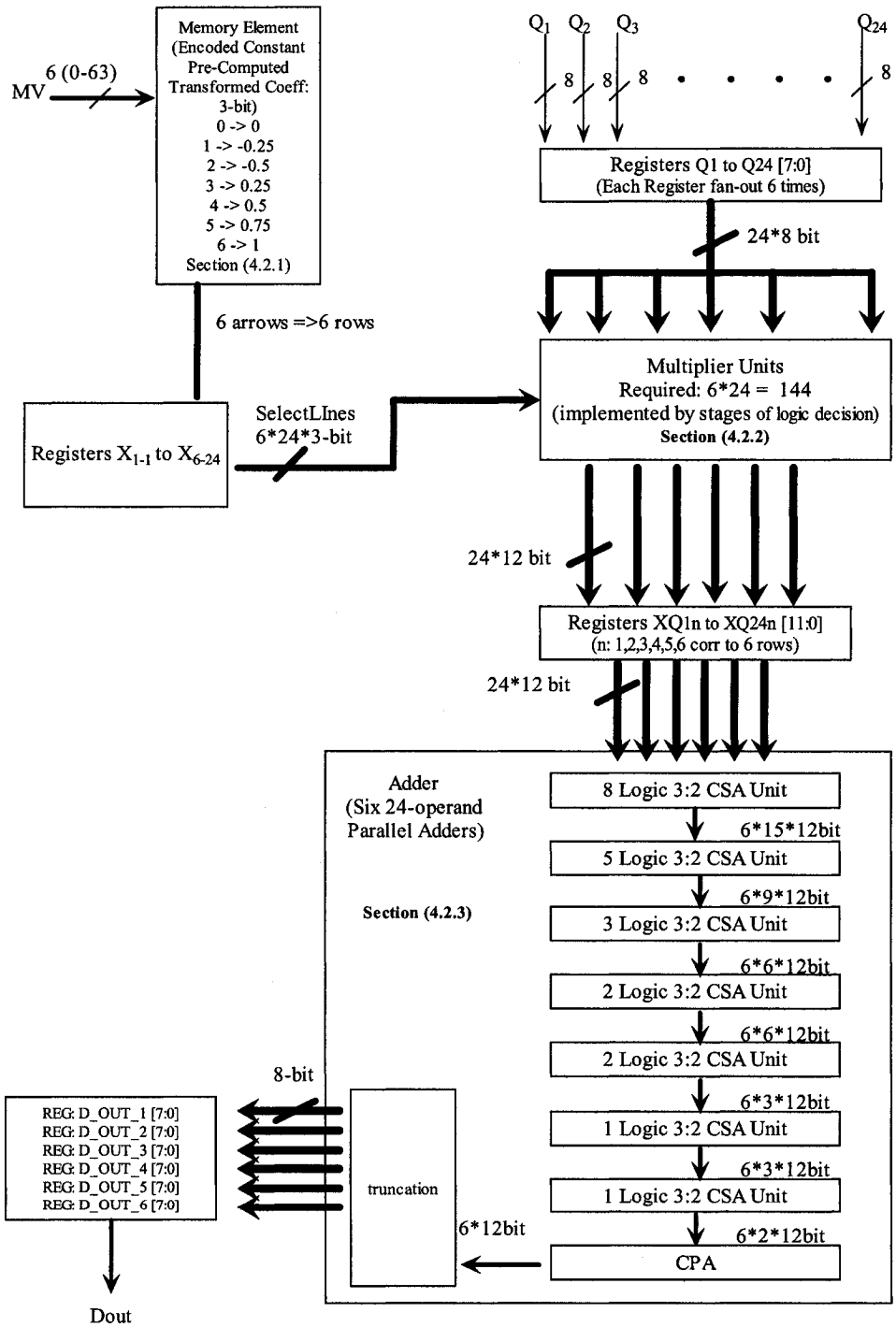


Figure 4.1 Overall architecture of MC-DCT

Based on Figure 4.1, the MC-DCT hardware design can be divided into three major units; they are:

1. Encoded DCT-CM Storage Unit (Section 4.2.1.)
2. Logic Multiplication Unit (Section 4.2.2.)
3. Logic Addition Unit (Section 4.2.3.)

These three units are discussed in more detail in the next Sections. Beside the attempt at applying different optimizing techniques to these three primary units to reduce power consumption, the width of the data bus for the sub-module interconnection is also carefully designed to achiece resource savings. The first concern is to minimize the width of the data bus used to route the twenty-four DCT-REs data inputs (Q_1 to Q_{24} in Figure 4.1), in order to reduce the workload on the Logic Multiplication Unit and later Logic Addition Unit and hence, to attain the goal of reduced power consumption.

The size needed to store a single DCT-RE's is governed by the quantization error, introduced from the Q_2 module due to a coarse quantization procedure. By exhaustively going through all possible input values within the range from 0 to 2048 and extracting the largest value (in an absolute sense) for each of step size *QP* from 1 to 31 using the H.263 quantization scheme from the transcoder software code introduced in Chapter 3, the correlation between the maximum value of DCT-RE and corresponding *QP* can be obtained. Figure 4.2 shows this relationship.

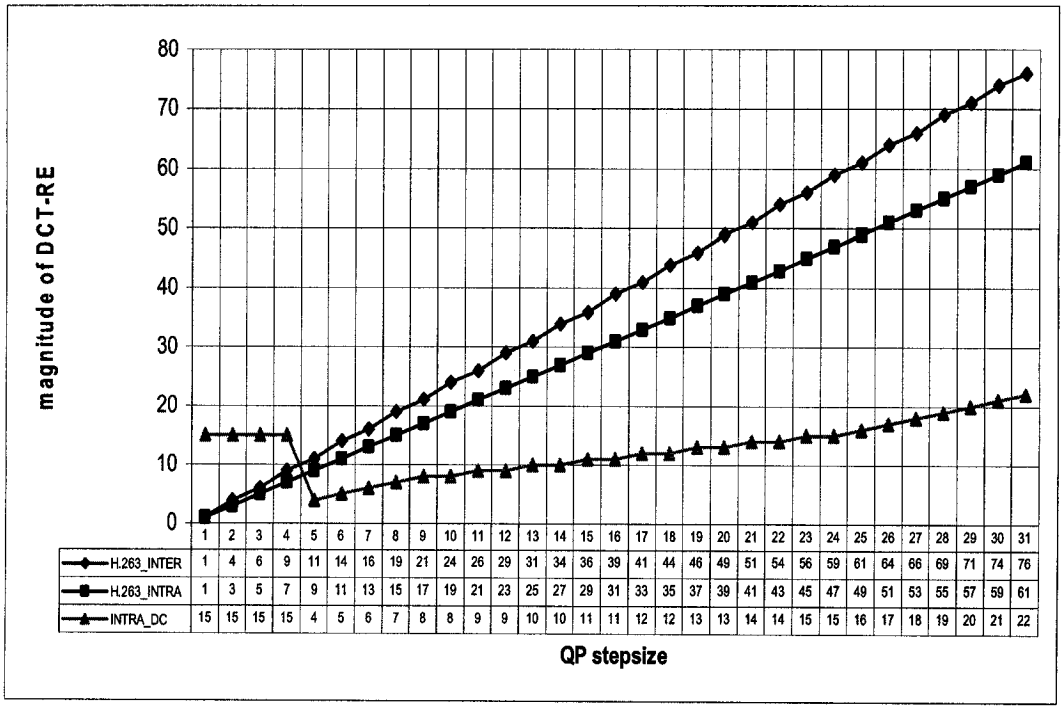


Figure 4.2 Max DCT- ReQuantization Error vs StepSize QP for H.263 Quantization Scheme

Neglecting the overloading noise, Figure 4.2 suggests that the size of DCT-RE is upper-bounded by 76. Table 4.1 shows the maximum magnitude of DCT-RE simulated from the video sequences Football, Flower, Miss American, Tennis, Mobile and Container respectively.

Table 4.1 Maximum DCT-RE for H.263 scheme

	H.263	
	Intra	**Inter**
Football	45	***76***
Flower	27	76
Missa	21	39
Tennis	16	***76***
Mobile	16	***76***
Container	45	31
Bit Requirement:	*7 bit required+1sign bit*	

This analysis shows that a data bus width of 8 (7 magnitude bit with 1 sign bit) is sufficient to ensure that no loss of DCT-RE precision occurs.

4.2. CUSTOMIZED COMPONENT FOR LOW POWER MC-DCT DESIGN

To achieve a low power implementation of the MC-DCT module, the design of each component is targeted to minimize the number of dynamic operations as possible.

4.2.1. Encoded DCT-CM Storage Unit

In Section 3.6, it is shown that we can safely and economically use a 3-bit budget to encode a single DCT-CM constant with the scheme of 2-bit precision after binary point which normally requires a 4-bit budget. Table 4.2 shows the pre-defined map between the actual value of the DCT-CM elements and the encoded one:

Table 4.2: Element of DCT-CM (DCT-Constant Matrix) in 2-bit precision after binary point map

Decimal	2-complement binary	Encoded in Decimal	Encoded in 2-complement binary
0	0.00	0	000
-0.25	1.11	1	001
-0.5	1.10	2	010
0.25	0.01	3	011
0.5	0.10	4	100
0.75	0.11	5	101
1	N/A	6	110

The encoded DCT-CM elements are to be hard-wired in the internal RAM of the device and loaded into registers. Following the steps outlined in Section 3.6, one of the

DCT-CM matrix, assuming MV_{row} = 0, MV_{col} = 2, is composed of elements shown in Table 4.3

Table 4.3: DCT-CM matrix for MVrow =0 and MVcol = 2

1	0	0	-0.25	0	-0.25	0	0	0	0	0	0	0.25	0	0	0.25	0	0.25	0	0	0	0	0	0
0	0.75	0	0	-0.25	0	0	0	0	0	0	0	0	0	0	0	0.25	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0.25	0	0	0	0	0	0	0	0	0
0.25	0	0	0.75	0	-0.5	0	0	0	0	0	0	-0.25	0	0	-0.25	0	-0.25	0	0	0	0	0	0
0	0.25	0	0	0.75	0	0	0	0	0	0	0	0	-0.25	0	0	-0.25	0	0	0	0	0	0	0
-0.25	0	0	0.5	0	0.5	0	0	0	0	0	0	0.25	0	0	0.25	0	0.25	0	0	0	0	0	0

In VHDL, these elements are defined as a constant array of standard logic vector type. As an example, the second row of the above matrix based on Table 4.2 map is encoded as

```
constant TCM_1_1 : ty24_3 :=
    ("000","101","000","000","001","000","000","000",
     "000","000","000","000","000","000","000","000",
     "011","000","000","000","000","000","000","000");
```

according to Table 4.2 where `ty24_3` is defined as

```
type ty24_3 is array (0 to 23) of std_logic_vector(2 downto 0);
```

The selection of a single DCT-CM (contain 6*24 elements) among a total of sixty-four DCT-CMs is driven by the MV(motion vector). Since the subsequent multipliers and adders are customized to work with the encoded values, therefore there is no performance penalty incurred for later the decoding process.

4.2.2. Logic Multiplication Unit

Multiplication is indisputably one of the most basic yet power-demanding arithmetic operations. Optimizations subjected to different aspects of performance improvement (i.e. area, speed) to implement multiplier modules have been researched for decades [28][29][30][31]. In this thesis, the multiplier unit is tailored specifically to the nature of the operands (both DCT-CM elements and DCT-REBs) which are either in a finite range (one out of seven possibility for the encode DCT-CM elements) or contain substantial zero elements (DCT-REs). This data-dependent design concept is applied to the multiplier unit by means of logical data routing as opposed to performing the conventional shift-and-add multiplication and thus supporting the power reduction goal.

In Section 3.4, it was found that the average percentages of non-zero coefficients decrease as the location of the coefficients moves away from the top left corner of the DCT-REB. With this anticipation, several data analyses for both DCT-CM's and DCT-REB are conducted to quantify the occurrence of zero coefficients. Figure 4.3, 4.4, 4.5 shows the histograms of DCT-CM's constants, the intra frame and the inter frame of DCT-RE's coefficients from the video *Table Tennis* respectively. Table 4.3, 4.4, and 4.5 tabulate the result to show the percentage of occurrence of the associate values from Figure 4.3, 4.4 and 4.5 correspondingly.

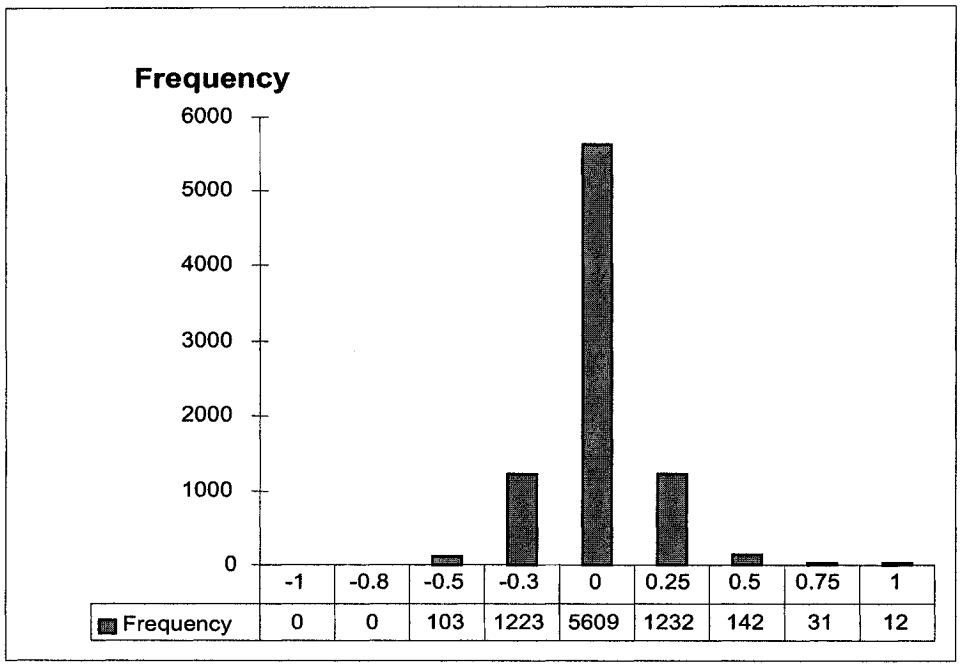


Figure 4.3 Histogram of DCT-CM's Constant

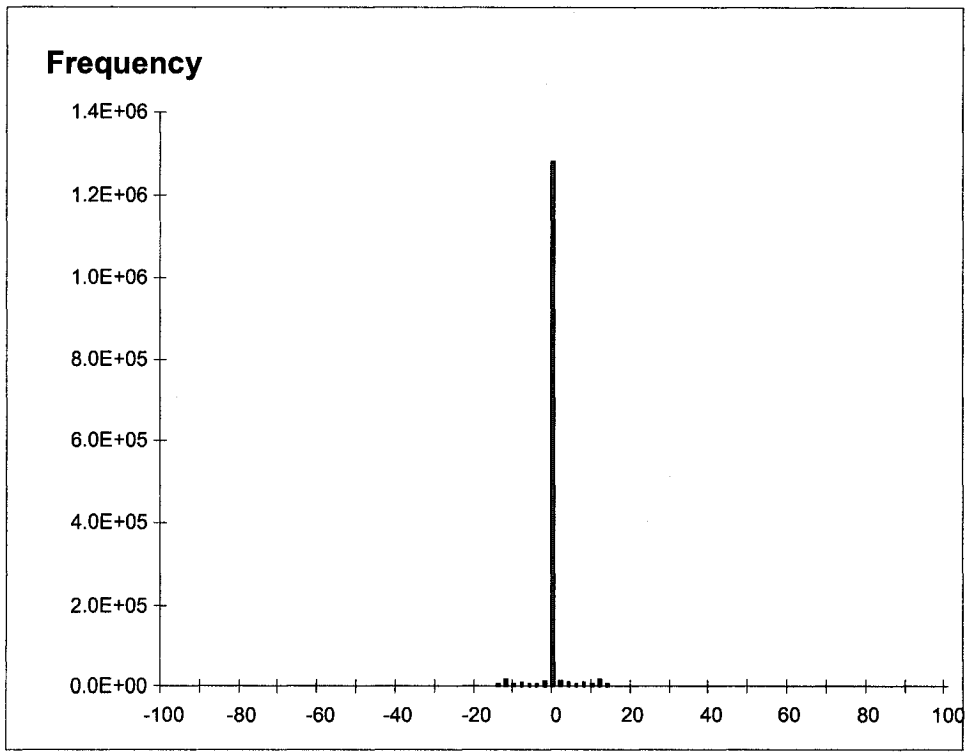


Figure 4.4 Histogram of Intra DCT-RE's for video Table Tennis

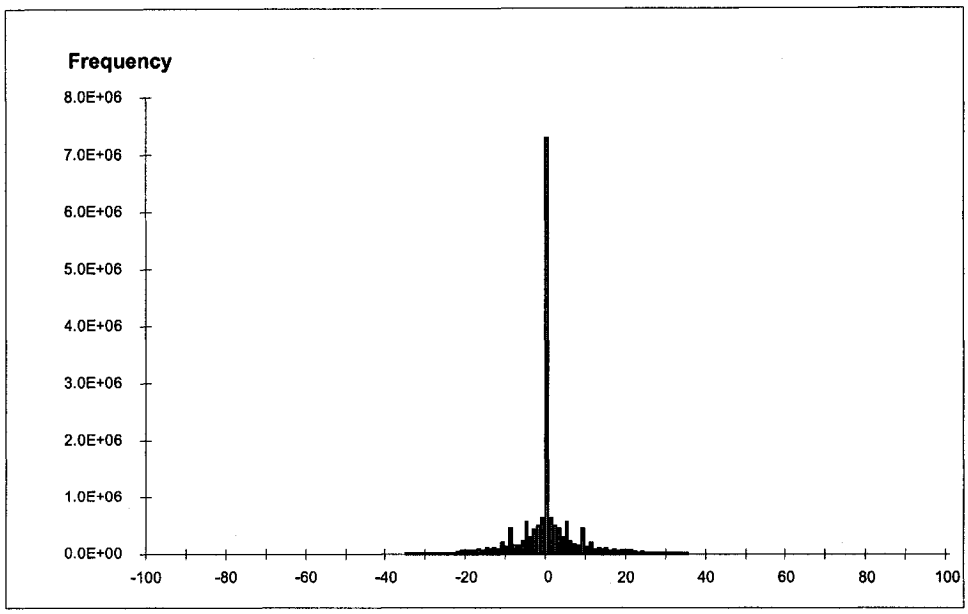


Figure 4.5 Histogram of Inter DCT-RE's for video Table Tennis

Table 4.4 Percentage of occurrence of the associated values for DCT-CM's constants		
VALUE	FREQUENCY	% OF FREQUENCY (%)
-1	0	0
-.75	0	0
-0.5	103	1.23
-0.25	1223	14.64
0	**5609**	**67.16**
0.25	1232	14.75
0.5	142	1.7
0.75	31	0.37
1	12	0.144

Table 4.5 Percentage of occurrence of the associated values for DCT-RE's coefficients Intra frame

VALUE	FREQ	% FREQ	VALUE	FREQ	% FREQ
-16	305	0.022	***0***	**1282594**	**92.013**
-15	0	0.000	*1*	403	0.029
-14	5564	0.399	*2*	10506	0.754
-13	0	0.000	*3*	892	0.064
-12	14931	1.071	*4*	5861	0.420
-11	152	0.011	*5*	233	0.017
-10	2680	0.192	*6*	4921	0.353
-9	433	0.031	*7*	595	0.043
-8	8578	0.615	*8*	8701	0.624
-7	131	0.009	*9*	666	0.048
-6	4113	0.295	*10*	2729	0.196
-5	366	0.026	*11*	56	0.004
-4	5205	0.373	*12*	15943	1.144
-3	1028	0.074	*13*	0	0.000
-2	10203	0.732	*14*	5455	0.391
-1	369	0.026	*15*	0	0.000
			16	307	0.022

Table 4.6 Percentage of occurrence of the associated values for DCT-RE's coefficients Inter frame

VALUE	FREQ	% FREQ	VALUE	FREQ	% FREQ	VALUE	FREQ	%FREQ	VALUE	FREQ	% FREQ
-76	816	0.0047	*-38*	8996	0.052	***0***	**7287865**	**42.057**	*39*	8615	0.050
-75	877	0.0051	*-37*	9587	0.055	*1*	623957	3.601	*40*	7797	0.045
-74	927	0.0053	*-36*	10428	0.060	*2*	483491	2.790	*41*	7483	0.043
-73	997	0.0058	*-35*	13461	0.078	*3*	436051	2.516	*42*	6863	0.040
-72	1090	0.0063	*-34*	11415	0.066	*4*	286076	1.651	*43*	6533	0.038
-71	1080	0.0062	*-33*	13586	0.078	*5*	562096	3.244	*44*	5943	0.034
-70	1147	0.0066	*-32*	13724	0.079	*6*	215329	1.243	*45*	7035	0.041
-69	1269	0.0073	*-31*	15119	0.087	*7*	170363	0.983	*46*	5247	0.030
-68	1280	0.0074	*-30*	17946	0.104	*8*	161655	0.933	*47*	4880	0.028
-67	1380	0.0080	*-29*	19677	0.114	*9*	451864	2.608	*48*	4606	0.027
-66	1565	0.0090	*-28*	21315	0.123	*10*	131764	0.760	*49*	4246	0.025
-65	1534	0.0089	*-27*	30227	0.174	*11*	204189	1.178	*50*	4102	0.024
-64	1713	0.0099	*-26*	28065	0.162	*12*	99957	0.577	*51*	4083	0.024
-63	1800	0.0104	*-25*	27079	0.156	*13*	112959	0.652	*52*	3494	0.020
-62	1890	0.0109	*-24*	33426	0.193	*14*	92798	0.536	*53*	3327	0.019
-61	2019	0.0117	*-23*	32344	0.187	*15*	107415	0.620	*54*	3164	0.018
-60	2139	0.0123	*-22*	38862	0.224	*16*	62118	0.358	*55*	2946	0.017
-59	2352	0.0136	*-21*	64754	0.374	*17*	79157	0.457	*56*	2785	0.016
-58	2280	0.0132	*-20*	60436	0.349	*18*	75839	0.438	*57*	2636	0.015
-57	2617	0.0151	*-19*	71739	0.414	*19*	72498	0.418	*58*	2507	0.014
-56	2754	0.0159	*-18*	74612	0.431	*20*	59564	0.344	*59*	2352	0.014
-55	3050	0.0176	*-17*	79862	0.461	*21*	65664	0.379	*60*	2170	0.013
-54	2993	0.0173	*-16*	61721	0.356	*22*	39360	0.227	*61*	2061	0.012
-53	3322	0.0192	*-15*	106087	0.612	*23*	32675	0.189	*62*	1851	0.011
-52	3527	0.0204	*-14*	92815	0.536	*24*	33955	0.196	*63*	1848	0.011
-51	3940	0.0227	*-13*	113671	0.656	*25*	27627	0.159	*64*	1701	0.010
-50	4215	0.0243	*-12*	98560	0.569	*26*	28742	0.166	*65*	1611	0.009
-49	4081	0.0236	*-11*	202918	1.171	*27*	31559	0.182	*66*	1508	0.009
-48	4560	0.0263	*-10*	130399	0.753	*28*	22031	0.127	*67*	1359	0.008
-47	4907	0.0283	*-9*	451746	2.607	*29*	20141	0.116	*68*	1370	0.008
-46	5259	0.0303	*-8*	161807	0.934	*30*	17729	0.102	*69*	1296	0.007
-45	6817	0.0393	*-7*	166815	0.963	*31*	15076	0.087	*70*	1157	0.007
-44	6010	0.0347	*-6*	214862	1.240	*32*	13789	0.080	*71*	1131	0.007
-43	6502	0.0375	*-5*	562754	3.248	*33*	13904	0.080	*72*	1035	0.006
-42	6759	0.0390	*-4*	284761	1.643	*34*	11686	0.067	*73*	991	0.006
-41	7282	0.0420	*-3*	434884	2.510	*35*	13590	0.078	*74*	981	0.006
-40	7730	0.0446	*-2*	482793	2.786	*36*	10656	0.061	*75*	920	0.005
-39	8668	0.0500	*-1*	623719	3.599	*37*	9513	0.055	*76*	828	0.005
						38	9100	0.053			

From Table 4.3, 4.4 and 4.5, it should be noted that the percentages of zero coefficients occurrence are 67.16%, 92.013% and 42.057% for DCT-CM elements, Intra-frame, Inter-frame of *Table Tennis* respectively. From these results, we may infer that there is a near 50% probability that one of the operands, either the incoming DCT-RE's coefficients or the DCT-CM's constants, is zero. Thus, the first power consumption optimization of the Logic Multiplier sub-module is to implement zero bypassing logic to detect occurrences of the zero coefficients for both operands. In VHDL, the zero detector is performed in bit-wise gate-level comparison as shown below:

```
is_zero <= (((Q(0) NOR Q(1)) AND (Q(2) NOR Q(3))) AND ((Q(4) NOR
Q(5)) AND (Q(6) NOR Q(7)))) OR ( NOT(sel(0) OR sel(1) OR sel(2)));
```

where `is_zero` is of type std_logic to signify no multiplication should be preformed if one or both of the operands is/are zero. `Q` is the eight-bit incoming DCT-RE of type std_logic_vector (7 down to 0) and `sel` is a single three-bit DCT-CM element of type std_logic_vector(2 downto 0).

The second optimization for the Logic Multiplier sub-module, as its name suggests, is to implement a multiplication-free logic based multiplier. The premise of such a design comes from the fact that the DCT-CM's constants are within a finite set of seven elements according to the 3-2-1 2-bit precision scheme. That is, the incoming DCT-RE's coefficients, call it the multiplier, are to multiply with one of the seven DCT-CM's elements, call it the multiplicand, fed from the storage unit. Also notice that the seven elements are, in fact, the reconstruction levels of 0, 0.25, 0.5, 0.75, -0.25, -0.5 and 1 and the multiplication of such operands requires no more than three basic operations; shift, two-complement and add.

The multiplication of an n-bit binary number with an m-bit binary number results in a product that is up to m + n bits in length for both signed and unsigned representation. In Section 4.1, it was shown that at least eight bits are required to store a single DCT-RE's element. Thus, with the two-bit precision scheme used for a DCT-CM element, a data bus width of 8+3=11 bits is needed. The width of the data-bus for the product has, however, been designed to be twenty bits wide by appending an extra bit, call it *is_zero* bit, at the MSB of the product to signify whether the result product is zero or not. This manoeuvre is intended to bypass any dynamic operation and reduce the dynamic power usage in the expense of maintaining the state of an extra bit for later logic addition processing. Table 4.6 indicates the operations required to achieve the multiplication in bit-level. Figure 4.6 outlines the implementation detail of the logic multiplier graphically.

Table 4.7 Operations required to achieve the multiplication for different DCT-CM Multiplicand

Let the multiplier be $A = a_8 a_7 a_6 a_5 a_4 a_3 a_2 a_1$ [8 bits]and the product be $P = p_{10} p_9 p_8 p_7 p_6 p_5 p_4 p_3 p_2 p_1 \bullet p_{-1} p_{-2}$ [11 bits + 1 *is_zero* bit]
where • is the binary point

Multiplicand [3 bits]	Operations required	Result Product
0	None	$p_{10} = p_9 = p_8 = p_7 = p_6 = p_5 =$ $= p_4 = p_3 = p_2 = p_1 = p_{-1} = p_{-2} = 0$
0.25	Shift A right by two	$p_{10} = 1 \quad p_9 = 0 \quad p_8 = 0 \quad p_7 = 0$ $p_6 = a_8 \quad p_5 = a_7 \quad p_4 = a_6 \quad p_3 = a_5$ $p_2 = a_4 \quad p_1 = a_3 \quad p_{-1} = a_2 \quad p_{-2} = a_1$
0.5	Shift A right by one	$p_{10} = 1 \quad p_9 = 0 \quad p_8 = 0 \quad p_7 = a_8$ $p_6 = a_7 \quad p_5 = a_6 \quad p_4 = a_5 \quad p_3 = a_4$ $p_2 = a_3 \quad p_1 = a_2 \quad p_{-1} = a_1 \quad p_{-2} = 0$
0.75	Shift A right by two plus Shift A right by one	Not immediate available
-0.25	Shift A right by two, then XOR each bit and ADD ONE	Not immediate available
-0.5	Shift A right by one, then XOR each bit and ADD ONE	Not immediate available
1	None	$p_{10} = 1 \quad p_9 = 0 \quad p_8 = a_8 \quad p_7 = a_7$ $p_6 = a_6 \quad p_5 = a_5 \quad p_4 = a_4 \quad p_3 = a_3$ $p_2 = a_2 \quad p_1 = a_1 \quad p_{-1} = 0 \quad p_{-2} = 0$

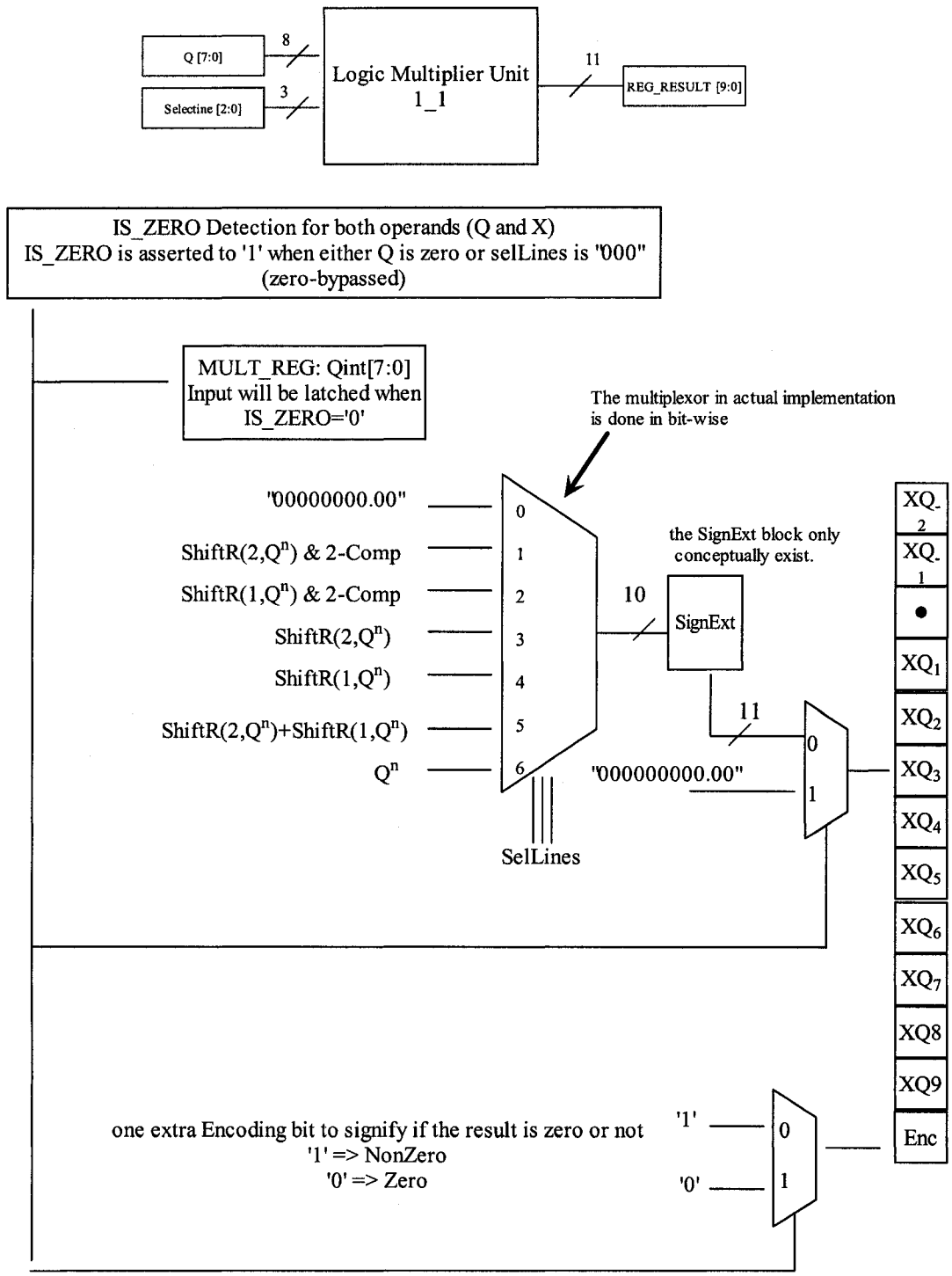


Figure 4.6 Overall design for the Logic Multiplier module

As we can see in Table 4.6, the multiplication is trivial when the multiplicand is either 0, 0.25, 0.5, or 1. No real operation is required except to "wire" the value properly to the output register. In the case where the multiplicand is -0.25 or -0.5, the multiplier needs to be two-complemented by negation and then add one. The negation is achieved by XORing every bit of the multiplier with '1' and the add-one operation uses the carry look-ahead algorithm with the prior knowledge that one of the operands is 1. The implementation of this adder can be found in Appendix A. For the multiplicand of 0.75, the operation involves the addition of half of the multiplier and a quarter of the multiplier. This addition is performed with a ripple-carry adder.

4.2.3 Logic Addition Unit (LAU)

For power efficiency, an effective strategy is to minimize the amount of hardware. This implies spreading the computation over a longer time interval through serialization in order to reduce the power consumption. On the other hand, a reasonable time performance is also required since the carry propagations are critical in the performance of the overall architecture. To gain a balance between these constraints, we decide to opt for the carry-save methodology in performing the arithmetic addition. This methodology regulates all carry propagations, sums three or more operands in a redundant and carry propagate free manner, and delivers a very short critical path delay, thus rendering a considerable high limit for the operating frequency. By applying the carry save adders (CSAs also called as full adders or 3-2 counters) in a Wallace tree structure [32], any number of operands can be added and reduced to 2 numbers without carry propagate adder. A single carry propagate addition is needed only in the final stage to sum two numbers into a final result.

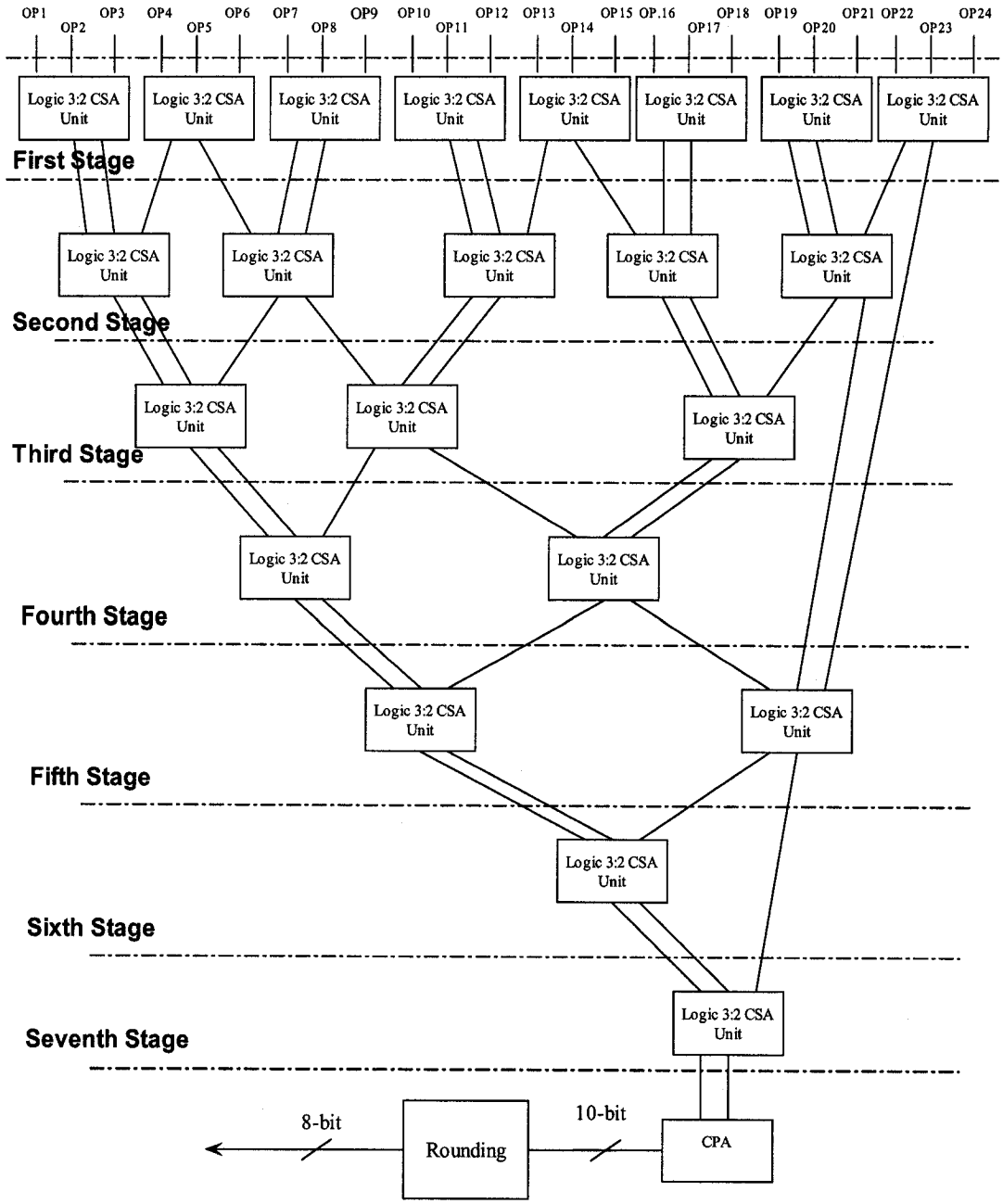


Figure 4.7 The deployment of summing 24 operands via customized 3:2 CSAs in a carry propagate free manner

From Figure 4.7, notice that seven stages of 3-2 compressor is required to produce the final two operands for the last summation. The final two operands are then added via

the pseudo 10-bit carry-propagation adder to render the result. The name 'pseudo' come from the analysis done in Section 3.7 indicating that the range of a single DCT-RE is limited to a maximum value of 76, hence, the MSB possibly resulting from the addition is neglected. Since the operation of MC-DCT is targeted to full-pixel resolution for motion compensation, the fractional part is truncated from the final result to yield an 8-bit outcome.

Logically, the probability of hitting a zero operand is reduced as CSA compresses the operands into lower stages. Under this premise, Logic Addition Units are only deployed on the first stage to maximize the diminishing return.

A Logic Addition Unit is composed of one 3-2 counter, two zero-detection units and two 8-to-2 multiplexers. Figure 4.8 shows the LAU design.

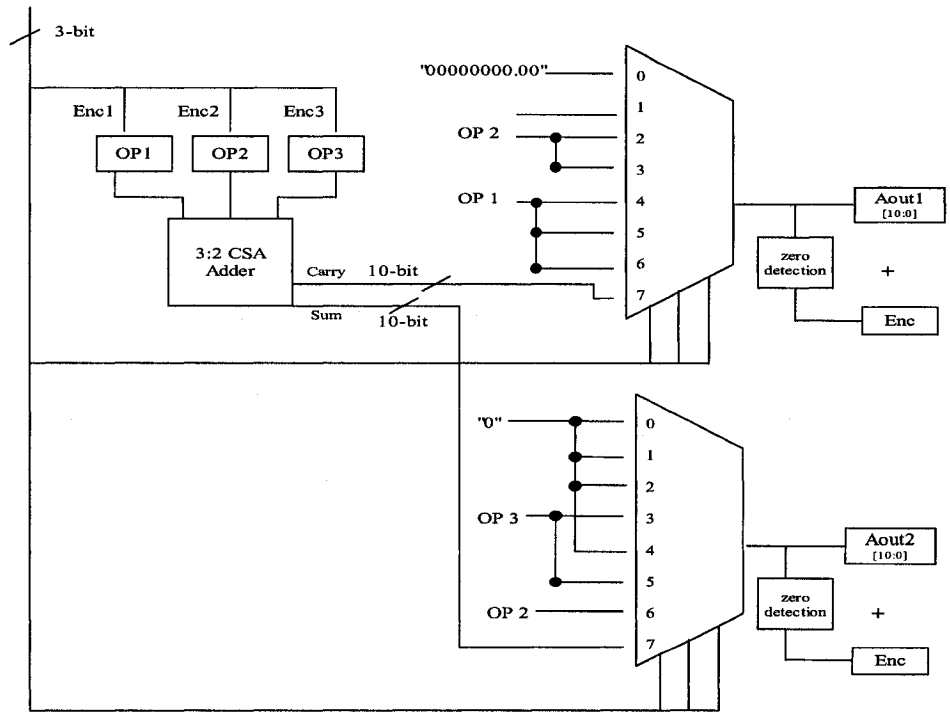


Figure 4.8 Logic Addition Unit design

The deployment of the two multiplexers is to implement the designs zero-bypass logic as of the Logic Multiplication Unit. Table 4.7 shows the function table of the MUX design with 3-bit select line, eight inputs and two output registers, *AOut1*, *AOut2*. The MSB of *AOut1* and *AOut2* are the encoded *is_zero* bit.

Table 4.8 The function mapping for the two MUX inside the LAU design

is_zero bit			AOut1	AOut2
Operand #1	Operand #2	Operand #3		
0	0	0	000000000.00	000000000.00
0	0	1	000000000.00	Operand #3
0	1	0	Operand #2	000000000.00
0	1	1	Operand #2	Operand #3
1	0	0	Operand #1	000000000.00
1	0	1	Operand #1	Operand #3
1	1	0	Operand #1	Operand #2
1	1	1	Carry of (Op#1+Op#2+Op#3)	Sum of (Op#1+Op#2+Op#3)

The formation of the *is_zero* bit is again constructed by ORing all the bits of the Aout1 and Aout2 respectively. The carry-propagation paradigm is not only implemented in the operand level as shown in Figure 4.7, but also employed to carry out the 3:2 CSA adder within each LAU in bit level. By using a dot diagram, Figure 4.8 shows the graphical representation of compressing three 10-bit inputs into two via ten full adders with no carry-prorogation delay.

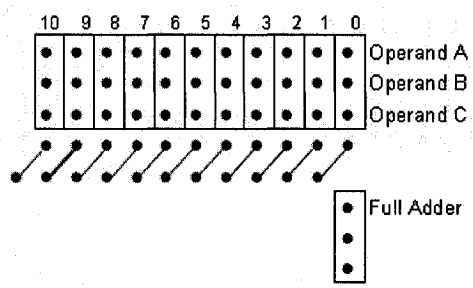


Figure 4.9 dot-diagram of 10-bit 3-2 CSA

4.3. SYNTHESIS RESULTS OF MC-DCT COMPONENTS

The VHDL code of the proposed MC-DCT module was synthesized using Synopsys in Xilinx 5.2 source environment with the target device Xilinx VirtexII 3000 series. It is always difficult to compute the dynamic power consumption accurately based on solely the synthesis tools, the attempt here is to attain some level of understanding and to obtain the quantitative measurement for the design. Future work may include actually setting up a testing environment by downloading the design to a Xilinx board and using an appropriate device to probe the current usage so as to compute the power consumption.

In order to obtain some comparative measurement, the MC-DCT sub-modules are also implemented in a conventional standard approach. Under this approach, the multiplication is carried out by the shift-and-add logic combined with the signed detection. In the case one or both two operands being are negative, they will be two-complemented to carry an unsigned multiplication and the final result is to be complemented to obtain the right sign. As to the addition part of the MC-DCT unit, the twenty-four-twelve-bit operands are to be summed via ripple-carry adder in five stages in which each stage reduces half of the operands from the previous stage in succession, the final output is truncated to render an 8-bit final result.

The synthesis result indicates that the proposed MC-DCT module consumes 545.34mW at 20MHz. The specification and power consumption reported by the Synposys tool after the synthesis process for the sub-modules of MC-DCT design is tabulated in Table 4.9.

Table 4.9 Specification and power consumption of MC-DCT design	
Specification	
Operating Voltage	4.75 Volt
Synthesis Library	xdc_virtex2-6
Proposed data-dependent design	
Logic Multiplication Unit Count	144
Logic Addition Unit Count	8
Carry Save Adder Count	14
Power Consumption	545.34mW
Data Arrival Time	13.10ns
Number of Cells	1275
Standard Conventional design	
Unsigned Multiplier Count	144
Ripple Carry Adder Count	23
Power Consumption	621.57mW
Data Arrival Time	14.41ns
Number of Cells	1451

From Table 4.9, the proposed design for the multiplication consumes 76.23mW less power than the conventional scheme, in other word, a 12.26% reduction in power saving. The proposed design also outpreforms the conventional approach in area as well as time delay reduction, 9.9% and 12.13% respectively.

The other factor that can affect the power consumption of the module is inevitably the operational frequency. Table 4.9 illustrates the direct proportionality with the operational frequency and power consumption for the proposed MC-DCT design.

Table 4.10 Power consumption of MC-DCT module under different operational frequency		
Clock Period (ns)	Operational Frequency (MHz)	Power Consumption (mW)
50	20	545.34
25	40	1090.7
10	100	2743.7

4.4. CHAPTER SUMMARY

In this chapter, different strategies for minimzing the power consumption of the MC-DCT module were investigated. Detail design procedures for all the sub-modules were described. Based on the data-dependant process concept, the zero-bypassing logic is inserted into the sub-modules of the MC-DCT to reduce the volume of data to be processed, thereby also reducing the power consumption. Further power saving is achieved by careful planning of the data-bus width required to ensure the integrity of the module connectivity while reducing the hardware.

The synthesized result of VHDL code using Synopsys shows the hardware optimization used in the proposed MC-DCT design is superior compared to the conventional implementation in all three critical aspects of power, area and time delay, when both designs are based on the optimized algorithmic approach studied in Section 3.

5. CONCLUSION AND FUTURE WORK

In this thesis, a data-dependent low-power MC-DCT design is presented. Low power is achieved by performing optimization on both algorithmic and architectural levels.

The MC-DCT design is built based on the Hong algorithm for the DDT transcoder structure with special treatment of the selection of information included to perform the Motion-Compensation in the DCT-domain. In addition, the design of the quantizer for the elements of DCT-CMs is also considered to reduce the bit-budget to perform the MC-DCT operation.

The MC-DCT design is coded using VHDL, and synthesized using Synopsis. No transistor-level circuit optimization is made. Operating at 4.75V and 20MHz, the MC-DCT design consumes 545.34mW. Low-power operation is achieved through the customized design of the internal databus, the implementation of a multiplication-free module, the 3-2 Wallace tree propagation-delay-free addition map and the logic-based addition module.

5.1. CONCLUSION

From the analysis and simulation results, the following conclusion can be drawn about this thesis:

The reduction of computation complexity for the MC-DCT operation can be achieved by using only partial DCT-REs information via the DCT-Coeffficient-Translation-and-Truncation-Transformation-Matrix (DCTTTM) based Motion Compensation algorithm. The employment of the 3-2-1 partial information scheme,

which takes into account only the top left corner of six coefficients (three from first row, two from second row and one from third row of a 8 by 8 Requantization Error Block (REB)) for both the input (the four straddling REBs) and the output (the resulting motion-compensated REB), appears to render a fair compromise between loss of video quality and increasing computational load.

The reduction in the bit precision of the constant transform matrix DCT-CM does not severely degrade the PSNR measurement. Simulations among the five video test sets show there is virtually no loss in PSNR improvement except when the reduction in the bit budget occurs in the most significant digits of the DCT-CMs constants. Considerable computational savings in hardware implementation is achieved by heavily quantizing the DCT-CMs constant to only the most two significant bits.

Low-power design of the MC-DCT module is achieved through both design and run time optimization. The integration of the 3-2-1 partial information scheme along with the 2-bit precision for quantized DCT-CMs constant into the DCTTTM-based algorithm in processing the MC-DCT operation renders a fair enhancement toward the power reduction strategy at design time. The run-time optimization is achieved by implemeting data-dependent bypass logic coupled to a customized logic module to reduce the number of operations, which in turn reduces the power consumption.

By comparing the data-dependent design with the standard conventional approach in implementing the MC-DCT module, the synthesized results indicate the proposed design outperforms the conventional approach by achieving 12.26%, 9.9% and 12.13% reductions in power, time, and area respectively.

5.2. POSSIBLE IMPROVEMENTS FOR FUTURE RESEARCH

The following lists some of the directions and future work that are considered and recommended to supplement this research paper.

- *Study the effect of the selection scheme for the partial information*: The 3-2-1 partial information scheme was chosen based on the study of average percentages of no-zero coefficients among the five video test sets. In practice, this selection may well be sub-optimal for any given type of motion picture; a dynamic scheme selection mechanism would provide the design a wider applicability.
- *Setup up a study board to analyze the dynamic power consumption using data-dependent logic*: As discussed in Section 4.3, the accuracy of the power consumption for a data-dependent design can rarely be obtained through simulation and synthesis tools. A more representative result should be collected via a study board with real time data load-in.
- *Study possible alternative technology for hosting the proposed MC-DCT design*: Among all the possible strategies to implement a hardware circuit design, the mainstream remains implementations based on ASICs or FPGAs. In this thesis, FPGA technology was selected mainly due to its availability. It would be worthwhile to contrast the design implemented in ASICs to gain a better understanding of the correlation between the choice of technology and the power consumption of the design.

REFERENCES

[1] S. F. Chang, A. Vetro, "Video Adaptation: Concepts, Technologies, and Open Issues", Proceeding of IEEE, Vol. 93, No. 1, Jan. 2005, pp. 148-158.

[2] A. Vetro, C. Christopoulos, and H. Sun, "Video Transcoding Architectures and Techniques: An Overview", IEEE Signal Processing Magazine, Mar. 2003, pp18-29.

[3] S. Notebaert, J. D. Cock, "Bitrate Transcoding Architectures for H.264/AVC Bitstreams", Sixth FirW PhD Symposium, Faculty of Engineering, Ghent University, Nov. 2005, pp. 108-109.

[4] G. Keesman, R. Hellinghuizen, F. Hoeksema, and G. Heideman, "Transcoding of MPEG bitstreams", Signal Processing: Image Commun., vol.8, Sept. 1996, pp. 481-500.

[5] J. Xin, C.W. Lin, and Ming-Ting Sun, "Digital Video Transcoding", Proceedings of the IEEE, Vol. 93, No. 1, Jan. 2005, pp. 84-97.

[6] N. Bjork and C. Christopoulos, "Acoustics, Speech, and Signal Processing, Vol 5, May 1998 pp. 2813 – 2816.

[7] J. Youn, M.-T. Sun, and J. Xin, "Video transcoder architectures for bit rate scaling of H.263 bit stream", Proceedings of ACM Multimedia, Nov. 1999, pp. 243-250.

[8] L. Yuan, E. Wu, Q. Chen, S. Li and W. Gao, "The fast close-loop video transcoder with limited drifting error", ISCAS '04 Proceedings of the 2004 International Symposium on Circuits and Systems, Vol. 3, May 2004, pp23-26.

[9] H. Sorial, "Thesis Paper: Transcoding of MPEG Compressed Video", Concordia University, Montreal, 2001.

[10] Y. Nakajima, H. Hori and T. Kanoh, "Rate conversion of MPEG coded video by re-quantization process", IEEE Int. Conf. Image Processing, vol. 3, pp. 408-411, 1995.

[11] O. Werner, "Requantization for transcoding of MPEG-2 intraframes", , IEEE Transaction on Image Processing, Vol. 8, Issue 2, Feb. 1999 pp. 179-191.

[12] P. Assuncao and M. Ghanbari, "Post-processing of MPEG2 coded video for transmission at lower bit rates", IEEE Int. Conf. Acoust., Speech, Signal Processing, ICASSP'96, vol. 4, May 1996, pp. 1998-2001

[13] "MPEG-4 Version 2 Visual Working Draft Rev2.0", ISO/IEC JTC1/SC29/WG11 N1993 Feb. 1998.

[14] J.B. Lee and B.G. Lee, "Transform Domain Filtering Based on Pipelined Structure," IEEE Trans. On Signal Processing, pp.2061-4, Vol. 40, No. 8, Aug. 1992

[15] S. F. Chang and D. G. Messerschmitt, "Compositing motion-compensated video within the network", in IEEE 4th Workship Multimedia Commun, Monterey, CA, Apr. 1992

[16] S.F. Chang, W.L. Chen, and D.G. Messerschmitt, "Video Compositing in the DCT Domain", IEEE Intern. Workshop on Visual Signal Processing and Communications, Raleigh, North Carolina, Sept. 1992.

[17] S. Wee and B. Vasudev, "Compressed-domain reverse play of MPEG video streams," SPIE International Symposium on Voice, Video, and Data Communications, Nov. 1998, pp. 237–248.

[18] S. Wee and V. Bhaskaran, "Splicing MPEG video streams in the compressed-domain", IEEE Workshop on Multimedia Signal Processing, Jun. 1997.

[19] K. Wang and J. W. Woods, "Compressed Domain MPEG-2 Video Editing", Proceedings 2000 International Conference on Image Processing, Vol. 1, Sept. 2000 pp. 1016-1019.

[20] S.-F. Chang and D. G. Messerschmitt, "Manipulation and Compositing of MC-DCT compressed video", IEEE J. Select. Area Commun., vol 13 Jan. 1995. pp. 1-11

[21] N. Merhav and V. Bhaskaran, "A fast algorithm for DCT-domain inverse motion compensation," IEEE Int. Conf. Acoust., Speech, Signal Processing, Atlanta, GA, May 1996, vol. 4, pp. 2307–2310.

[22] P. Assuncao and M. Ghanbari, "A frequency-domain video transcoder for dynamic bit rate reduction of MPEG-2 bit streams", Trans. On Circuits Syst. Video Technol., vol. 8, no. 8, Dec 1998, pp. 953-967.

[23] S. Liu and A. C. Bovik, "Local Bandwidth Constrainted Fast Inverse Motion Compensation for DCT-Domain Video Transcoding", IEEE Trans. on Circuit Systems and Video Technology, No.5, Vol. 12, May 2002.

[24] J. Song and Bo. L. Yeo, "A Fast Algorithm for DCT-Domain Inverse Motion Compensation Based on Shared Information in a Macroblock", IEEE Transaction on circuits and system for video technology, Vol. 10, no. 5, August 2000

[25] H. Q. Chen, "Thesis Paper: Transcoding of MPEG-4 Compressed Video", Concordia University, November, 2003.

[26] MoMuSys Codec, "MPEG4 Verification Model", ISO/IEC JTC1/SC29/WG11 Coding of Moving Pictures and Associated Audio MPEG 97, March 1997.

[27] Xilinx Cooperation, "Virtex-II Complete Data Sheet", Version 3.4, Mar. 2005.

[28] Gary.W.Bewick, "Fast Multiplication: Algorithms and Implementation"., *PhD Thesis*, Department of Electrical Engineering: Stanford University, February 1994.

[29] M. Ito, D. Chinnery, and K. Keutzer, "Lower Power Multiplication Algorithm for Switching Activity Reduction through Operand Decomposition", Proceedings of the 21st International Conference on Computer Design, 2003, pp. 21-27.

[30] P.-M. Seidel, "Dynamic Operand Modification for Reduced Power Multiplication", Proceedings of the 36th Asilomar Conference on Signals, Systems and Computers, 2002, pp.52-56.

[31] A. A. Fayed and M. A. Bayoumi, "A Novel Architecture for Low-Power Design of Parallel Multipliers", Proceedings of the IEEE Computer Society Workshop on VLSI, 2001, pp.149-154.

[32] C. S. Wallace. "A Suggestion for a Fast Multiplier," IEEE Transactions on Electronic Computers, EC-13:14–17, February 1964.

[33] N. Jayant and P. Noll, "Digital Coding of Waveforms: Principles and Applications to Speech and Video", Englewood Cliffs, NJ: Prentice-Hall, 1984.

Appendix A: Implementation of Incrementor using Carry Look-Ahead (CLA) scheme

The truth table of a full adder is:

x	y	$c_{in}(c_i)$	$c_{out}(c_{i+1})$	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

From the truth table, the sum of product function for sum s and carry-out c_{out} by using distributive property can be expressed as follows:

$$\begin{aligned} s &= x'y'c_{in} + x'yc_{in}' + xyc_{in} + xy'c_{in}' \\ &= (x'y' + xy)c_{in} + (xy' + x'y)c_{in}' = (x \oplus y)'c_{in} + (x \oplus y)c_{in}' \\ &= x \oplus y \oplus c_{in} \end{aligned}$$

and

$$\begin{aligned} c_{out} &= x'yc_{in} + xy'c_{in} + xyc_{in}' + xyc_{in} \\ &= (c_{in} + c_{in}')xy + (x'y + xy')c_{in} \\ &= xy + c_{in}(x \oplus y) \end{aligned}$$

If we define two new binary variables:

$$p_i = x_i \oplus y_i \qquad g_i = x_i y_i$$

then the output sum s and carry c_{out} become

$$s_i = p_i \oplus c_i \qquad c_{i+1} = g_i + p_i c_i$$

where g_i is called a carry generate and it produces an output carry when both x_i and y_i are one, regardless of the input carry. p_i is called a carry propagate because it is the term associated with the propagation of the carry from c_i to c_{i+1}. The output sum and the carry-

out equations in term of carry generate and carry propagate form the basis to construct n-bit carry look-ahead adder.

Consider the special case of addition of $X+Y=S$ in which Y is always one, the operands in n-bit binary numbers are expressed as, $x_{n-1}...x_0$ to $y_{n-1}=0 y_n=0...y_0=1$ which results in $s_{n-1}...s_0$. By using the basic equations $s_i=p_i \oplus c_i$ & $c_{i+1}=g_i+p_i c_i$, we have

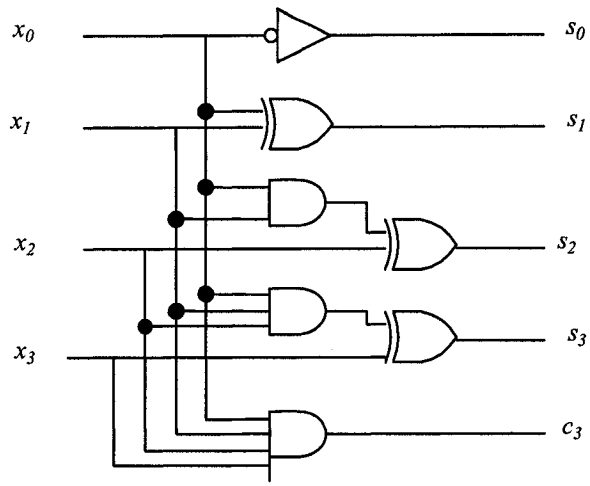
$$\left.\begin{aligned} p_0 &= x_0 \oplus 1 = x_0' \\ g_0 &= x_0 \cdot 1 = x_0 \\ c_0 &= g_0 + c_{-1} \cdot p_0 = g_0 = x_0 \\ s_0 &= c_{-1} \oplus x_0' = 0 \oplus x_0' = x_0' \end{aligned}\right\} \textit{First Bit}$$

$$\left.\begin{aligned} p_1 &= x_1 \oplus 0 = x_1 \\ g_1 &= x_1 \cdot 0 = 0 \\ c_1 &= g_1 + c_0 \cdot p_1 = x_0 x_1 \\ s_1 &= c_0 \oplus p_1 = x_0 \oplus x_1 \end{aligned}\right\} \textit{Second Bit}$$

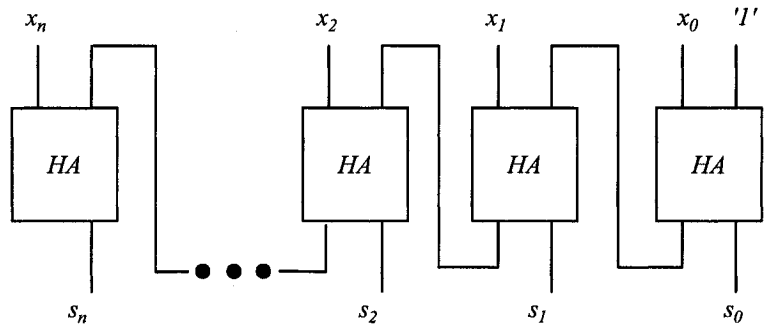
$$\left.\begin{aligned} p_2 &= x_2 \oplus 0 = x_2 \\ g_2 &= x_2 \cdot 0 = 0 \\ c_2 &= g_2 + c_1 \cdot p_2 = x_0 x_1 x_2 \\ s_2 &= c_1 \oplus p_2 = (x_0 x_1) \oplus x_2 \end{aligned}\right\} \textit{Third Bit}$$

$$\left.\begin{aligned} p_n &= x_n \oplus 0 = x_n \\ g_n &= x_n \cdot 0 = 0 \\ c_n &= g_n + c_{n-1} \cdot p_n = x_0 x_1 ... x_n \\ s_n &= c_{n-1} \oplus p_n = (x_0 x_1 ... x_{n-1}) \oplus x_n \end{aligned}\right\} n \textit{ Bit}$$

Depend on the implementation goal, the incrementor can be realized straight from the deviation above if time is critical



Or cascading half-adder one after the other if area is more important.



In the thesis, our target is to optimize power consumption, thus the regular structure of cascading half-adder design is more favourable due to the structure rendering a constant gate fan-out of two.