

Project Report

Title: Low Power Motion Estimation Architecture
for MPEG-4 AVC

Aviral Mittal

Project Report 2006

MSc In System level Integration



Principal academic supervisor: Mr. Asral Bahari

Second academic supervisor: Dr. Ahmet Erdogan

Declaration

I affirm that the submission of the final report
is all my own work in accordance with ISLI policy on
assessment and the University of Edinburgh Regulations

Name : Aviral Mittal

Signature :

Date : 22 Sep 2006

Table of CONTENTS:

| Topic Name | Page # |
|---|--------|
| 1. Alphabetic List of abbreviations | 5 |
| 2. Description of the project | 6 |
| 3. Abstract | 7 |
| <u>SECTION 1 Visual Information Representation and Compression</u> | |
| 1.1 What is MPEG? | 8 |
| 1.2 Motivation for encoding/compression | 8 |
| 1.3 Types of Compression: | 11 |
| 1.4 Introduction to Image compression: | 12 |
| 1.5 Video Compression/MPEG 4 AVC | 15 |
| <u>SECTION 2 MOTION ESTIMATION</u> | |
| 2.1 Motion Estimation: An Introduction | 19 |
| 2.2 Variable Block Size Motion Estimation | 21 |
| 2.3 Sub Pel Motion Estimation | 22 |
| 2.4 Primary Research | 23 |
| 2.4.1 Power Consumption in MPEG Encoders | 24 |
| 2.4.2 Power figures from Prior Art of ME | 25 |
| 2.5 Popular Motion Estimation Algorithms | 25 |
| <u>SECTION 3 Software Modelling and Analytical Analysis</u> | |
| 3.1 PSNR | 30 |
| 3.2 Analysis 1: SAD vs MSE | 32 |
| 3.3 Algorithm Comparison ES vs MDS | 33 |
| 3.3.1 Number of Candidate Blocks. | 34 |
| 3.3.2 PSNR Analysis ES vs MDS | 35 |
| 3.3.3. Frame Distance | 39 |
| 3.4 Conclusion: MDS vs ES | 42 |
| 3.5. Effect of block size on PSNR | 44 |
| <u>SECTION 4: Hardware Design and Testing.</u> | |
| 4.1 The Methodology | 46 |

| | |
|---|----|
| 4.2 Processing Elements for Hardware | 46 |
| 4.3 Designing for Low Power | 49 |
| 4.4 Low Power Fifo Design | 50 |
| 4.5 Design of Address Calculating Unit | 52 |
| 4.6 Verification | 56 |
| 4.7 Synthesis | 60 |
| 4.8 Power Analysis | 60 |
| <u>SECTION 5: Project Automation</u> | 61 |
| <u>SECTION 6 : Conclusions</u> | 64 |
| <u>SECTION 7 : Critical Evaluation/Recommendations/Future Work</u> | 65 |
| <u>SECTION 8 : References</u> | 67 |
| <u>SECTION 9 : Appendix</u> | 70 |
| A How to run/execute/use this project | |
| B Code listings for Scripts in Section 5 | |
| C Deliverables in CD-ROM (directory called ‘ <i>delivery</i> ’ in CD ROM) | |
| D All the development work carried out as a part of this project in a CD-ROM (directory called ‘ <i>work</i> ’ on CD ROM) | |
| <i>Note: Appendix C and D are only available as soft copy on a CD ROM, to conserve paper.</i> | |

Alphabetic List of abbreviations

| | |
|---------|---|
| ACU | : Address Calculation Unit |
| AVC | : Advanced Video Coding |
| CIF | : Common Intermediate Format |
| DCI | : Discrete Cosine Transform |
| DS | : Diamond Search |
| ES | : Exhaustive Search |
| FPS/fps | : Frames Per Second |
| FS | : Full Search |
| FSS | : Four Step Search |
| HDL | : Hardware Description Language |
| HEXBS | : Hexagon Based Search |
| Hz | : Hertz |
| IEEE | : Institute of Electronics And Electrical Engineers |
| IP | : Intellectual Property |
| JPEG | : Joint Photographic Experts Group |
| MDS | : Modified Diamond Search |
| ME | : Motion Estimation |
| MHz | : Mega Hertz |
| MPEG | : Motion Pictures Experts Group |
| MSE | : Mean Squared Error |
| PSNR | : Peak Signal to Noise Ratio |
| QCIF | : Quarter Common Intermediate Format |
| RGB | : Red Green Blue |
| RTL | : Register Transfer Level |
| SAD | : Sum of Absolute Difference |
| TMS | : Techno Mathematical Company |
| TSS | : Three Step Search |
| UMC | : United Microelectronics Corporation |
| Vs | : Versus |

DESCRIPTION OF PROJECT

The digital video application has become increasingly popular in mobile terminals such as cellular phones and personal digital assistance. However, due to its inherent data intensity of video sequences, storing and transmitting raw video data become impractical. With the limited storage and bandwidth capacity, this data must be compressed to a transportable size. For this purpose, MPEG-4 has become the dominant video codec algorithm in streaming and distribution of video content at low bandwidth across wireless media. The latest approved video compression standard, known as MPEG-4 Part 10 (Advance Video Coding-AVC), has shown a 50% compression improvement compared to the previous standard. This makes it the best choice of video compression for the next 5-10 years. However, this improvement comes at the cost of increase in computational complexity, which results in higher power dissipation. This will be a major drawback for mobile terminals with limited battery lifetime. Several sources of major power dissipation have been identified, such as motion estimation. Thus, this project will focus on minimizing power dissipation in MPEG-4 motion estimation block.

Objective & deliverable

1. Study the motion estimation algorithm used in MPEG-4 AVC
2. Study the existing motion estimation architecture
3. Designing low power architecture for the motion estimation
4. Analyze the performance against power, speed and area.

ABSTRACT

MPEG 4 is one of the most popular encoding schemes used for digital visual motion information representation on mobile terminals, which rely on battery power. There has been ever increasing demand to reduce the power consumed by applications running on these mobile terminals.

Motion Estimation is the basis of inter coding of image frames in MPEG 4 AVC, which exploits temporal redundancy between the video frames, to scope massive visual information compression. While it is the basis of compression, It is also the primary source of power consumption in MPEG 4 encoding, and accounts for up to 30-40% power consumed by the encoder itself.

Hardware design of a unit capable of performing Motion Estimation, with low power consumption has been accomplished in this project, using fast Modified Diamond Search Algorithm. Other various power saving strategies are also employed.

Existing algorithms for motion estimation are studied, modelled, analyzed, and as a result, Diamond Search Algorithm is selected, modified and implemented in hardware. The design was implemented using verilog and was synthesized and mapped to 0.18u UMC library. Power consumed by the unit (pre layout, at netlist level) was found to be 4mW, when processing QCIF video at the rate of 30 frames per second

SECTION 1. Visual Information Representation and Compression

1.1 What is MPEG?

In order to understand what is ‘Motion Estimation’, it is essential to first have an overview of what is called as ‘MPEG’, of which motion estimation is a part.

Moving Pictures Expert Group (MPEG) is a body developing a suitable encoding scheme or what are called standards for transmitting moving pictures and sound over various broadcast links and recording them in standard digital storage media such as DVD, CD, Flash Memory etc. The term is used synonymously to the family of digital video/audio compression standards and file formats developed by the group. There are various standards released so far:

MPEG-1

MPEG-2

MPEG-4.

This report would essentially focus on MPEG-4. MPEG-1 and MPEG-2 are beyond the scope of this document. MPEG-4 will henceforth be referred to as MPEG.

1.2 Motivation for encoding/compression

This section provides an introduction to digital images and digital video. It also analytically determines the size in bits/bytes needed to store un-compressed or raw digital visual information i.e. images and video, and hence the need of compression of such information.

Moving Pictures/Video:

Motion video is consist of a series of ‘still images’ which are made to display at a given rate, to produce ‘motion like’ effect for human eye. The rate at which these still images are displayed is one of the major factors on which quality of video largely depends. Higher is the rate better is the quality. Nothing is gratis, higher rate is characterized by higher bandwidth requirements if the video is meant to be transmitted, and more storage space requirements if the video is to be stored. The typical rates are from 15 Hz to 30 Hz.

‘Still Images’:

What are images in digital world?

An image in digital world is a collection of what are called pixels. A pixel is a unit of information which makes an image. A pixel may simply be imagined as a very small ‘dot’ of a given color. Several pixels when collected together in an ordered fashion to produce visual information, is

called an image. For example consider figure 1.1, a collection of pixels are shown, put in order to produce a 'smily face'.

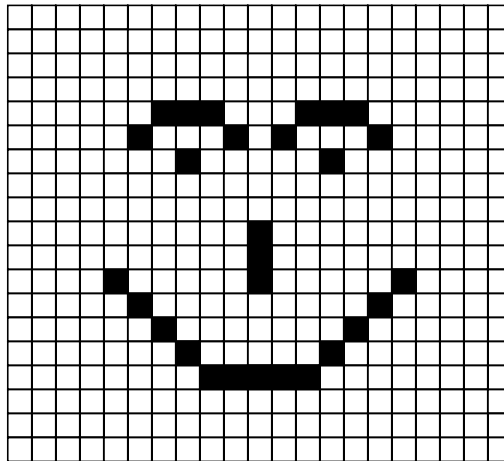


Figure 1.1 A digital Image (pixel size is highly exaggerated)

However the following points are worth noting.

There are only 2 colored pixels used i.e. black and white

The size of pixel is very large, or exaggerated as compared to the size of pixels in digital world

The quality of the image is very poor.

Nevertheless, it serves the purpose of defining what is an 'image' in digital world is and how it is represented.

Bits required to represent the above image:

Total number of pixels = $19 \times 19 = 361$

Bits per pixel = 1

Total number of bits required = 361.

Grey Scale Image:

The shown in Fig 1.1 has only 2 colors i.e. black and white, however a greyscale image may have any color whose intensity lies in between black and white. The range between white and black is divided into 'discrete' levels, with total number of possible levels increases exponentially with the number of bits chosen to represent a single pixel. If 2 bits per pixel are chosen, there can be 4

shades, similarly if 8 bits per pixel are chosen, 256 different shades or ‘discrete levels’ can be expressed and so on and so forth. The number of discrete levels/shades are given by 2^N , where N is the number of bits per pixel.

Size of a grey scale Image:

Number of bits needed to represent a grey scale image, which has ‘row’, ‘col’ pixels, with each pixel represented by ‘N’ bits is given by

$$\text{row} * \text{col} * 2^N \quad \text{-- Equation (1)}$$

Colored Image:

The concept of colored image is the same as that of a greyscale image, i.e. a colored image is composed of pixels. However the pixel of a colored image is subdivided into 3 primary colored pixels, i.e red, green and blue pixel. These 3 colored pixels are placed so close to each other that human eye perceives it has a single pixel. Changing the intensity of each color in a pixel will produce the desired color for a given pixel. Figure 1.2 below shows such a pixel



Figure 1.2 A colored Pixel

Size of a colored Image:

Number of bits needed to represent a colored image, which has ‘row’ x ‘col’ pixels, with each pixel represented by ‘N’ bits is given by

$$3 \times \text{row} \times \text{col} \times 2^N \quad \text{-- Equation (2)}$$

Example 1. Given that row = 144, col = 196, bits/pixel = 8. Calculate the size of un-compressed (raw) image:

Using Eq(2),

$$\begin{aligned} \text{Size} &= 3 \times 144 \times 196 \times 2^8 \\ &= 21676032 \text{ bits} \sim 2.7\text{Mbytes} \end{aligned}$$

Motion Picture/Video:

If un-compressed raw images are used to make a motion video at say ‘F’ Frames (still images) per

second, then the disk space needed to store T seconds of Video is given by

$$(3 \times \text{row} \times \text{col} \times 2^N \times F \times T)/8 \text{ bytes} \quad \text{-- Equation (3)}$$

Example 2. Given row = 144, col = 196, bits/pixel = 8, Calculate the size of un-compressed (raw) video file, which plays for 10 seconds, at 25 Frame per Second.

Using Equation 3,

$$\text{Size} = 3 \times 144 \times 196 \times 2^8 \times 25 \times 10/8 \sim 677.376 \text{ MBytes.}$$

Using no compression/coding technique, the above specified video needs more than half a GB of storage space. The video spec is not even high definition. Imagine if N increases to 16, and the video is a 90 minute film, how much storage space would be needed, and what would the cost of that storage. Not only storage cost is important, its also desired to transmit video using communication channels. Again using no compression/coding technique, it is practically impossible to transmit such visual information, using current or future communication systems.

Also it is practically impossible to commercialize such a digital video format.

Hence Mbpd (Mega bits per dollar) i.e. digital data storage cost, and Mbps (Mega bits per second) i.e. bandwidth famine are two main motivations to have audio/video data compressed.

1.3 Types of Compression:

This section introduces two basic types of compressions, followed by technique to compress a digital image, and then introducing generic video compression/coding.

Lossy and Lossless

Introduction to Image Compression:

Introduction to Video Compression.

Lossy Vs. Lossless Compression:

Compression techniques are broadly classified into 2 types i.e. lossless and lossy. Lossless compression being where the original data can be reconstructed exactly as it was before compression. In digital world a lossless compression data when decompressed should match the original data bit by bit. These kind of compression are generally used for computer files/programs where each bit has its importance. It may not achieve a very high degree of compression, as it is generally desired for audio/visual data. The other class of compression i.e. lossy, boasts high degree of com-

pression usually in compressing audio/visual data, but the reconstructed data is not the same as the original data. This project is all about Motion Estimation in MPEG compression which uses this lossy class of compression techniques which helps achieve compression ratios between 50:1 and 200:1 [2]

1.4 Introduction to Image compression:

Although Image compression is a bit off the main scope of this project, but its worth having an overview of it, taking into account that motion video is a collection of still images.

This seemingly off focus section will be exploited to introduce what are called luminance (grey-scale measure) commonly referred luma, Chrominance (Blue Color Difference commonly referred to as Cr, and Red Color Difference commonly referred to as Cr), which are very important terms in digital representation of colored visual information.

Characteristics of image relevant for image compression.

An image may have useful information and some not so useful information. The useful information being is the variety of colors/brightness in a small area, the not-so-useful information being the 'flat' areas of image showing a uniform color and or brightness. This is one of the characteristic of the image which helps in compression. The other useful characteristic being that image may have very high frequency areas which are redundant for human eye.

Therefore to process the image, an image is divided into a number of blocks. Each block containing a number of pixels. A typical block consist of 'row' rows of pixels and 'col' columns of pixels, where there may be any number of 'rows' and 'cols'. Some typical combinations may be (16,16), (16,8), (8,16). The combinations supported by H264/MPEG will be detailed later in this report.

As mentioned earlier in section 1.2 of this report, an colored image is a collection of pixels, where each pixel has a red (R), green (G) and Blue (B) component. Well that is what humans see and make out of. Where as in electronics world this RGB is usually converted into YCbCr, where Y is luminance or luma, Cb is Chrominance (blue) and Cr is Chrominance (red). The conversion format is very simple and is given by the following standard equations.

$$Y = 0.299R + 0.587G + 0.114B$$

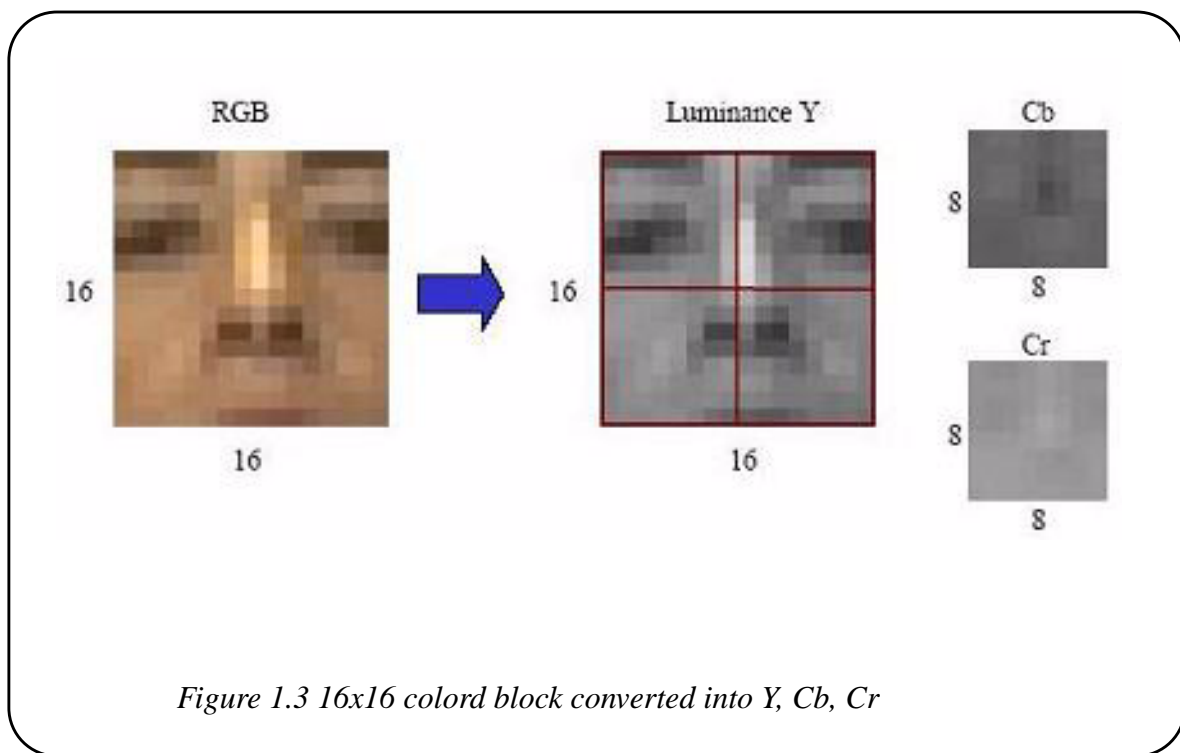
$$Cb = 0.492 (B-Y)$$

$$Cr = 0.877 (R - Y)$$

The reason for using YCbCr format dates back to 1950s when colored TVs were introduced and it was considered important to have a compatibility between colored signal transmission and B&W TVs. B&W TVs only decode the luma component of the colored signal.

Since research suggests that human eye is more sensitive to luma than chrominance (chroma), the chroma signals/components i.e. Cb and Cr, are only half the resolution of luma. Again H264/MPEG may support different resolution combinations of YCbCr,.

Following figure 1.3 shows an example of a 16x16 block from RGB image being converted into YCbCr components. Note that Cb and Cr components are 8x8 i.e. half of the resolution of the luma (Y) component.



Y, Cb, Cr blocks are processed all in the same way. Henceforth in this report a 'block' would be a collection pixel values, which represents an image block to be processed. It wouldn't matter whether the block is a Y block, Cb block or a Cr block, as further down the process of compression each is treated in exactly the same way.

Frequency Transform:

Further down the image processing pipeline, each block usually in a 8x8 format is frequency transformed using a transform called DCT[20], which produces a distribution of energy in the block in frequency domain. The following figure suggests that most of the energy resides in low frequency components.

Following Figure 1.4 shows that the significant energy tends to be concentrated into first few “low frequency” components.

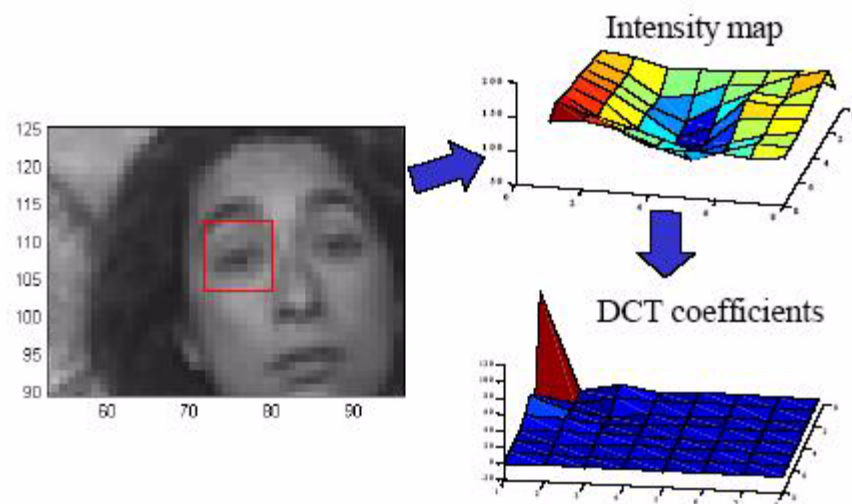


Figure 1.4: Most of the energy is concentrated around first few low frequency components

These components or weights are then quantized, which results in a very few coefficients with a non-zero value. These residual coefficients are then entropy encoded to form a bit string. Usually a very few coefficients are sufficient to be able to give a acceptable quality reconstructed image. Figure 1.5 shows the quality of reconstructed image with 1,3,6 and all coefficients respectively

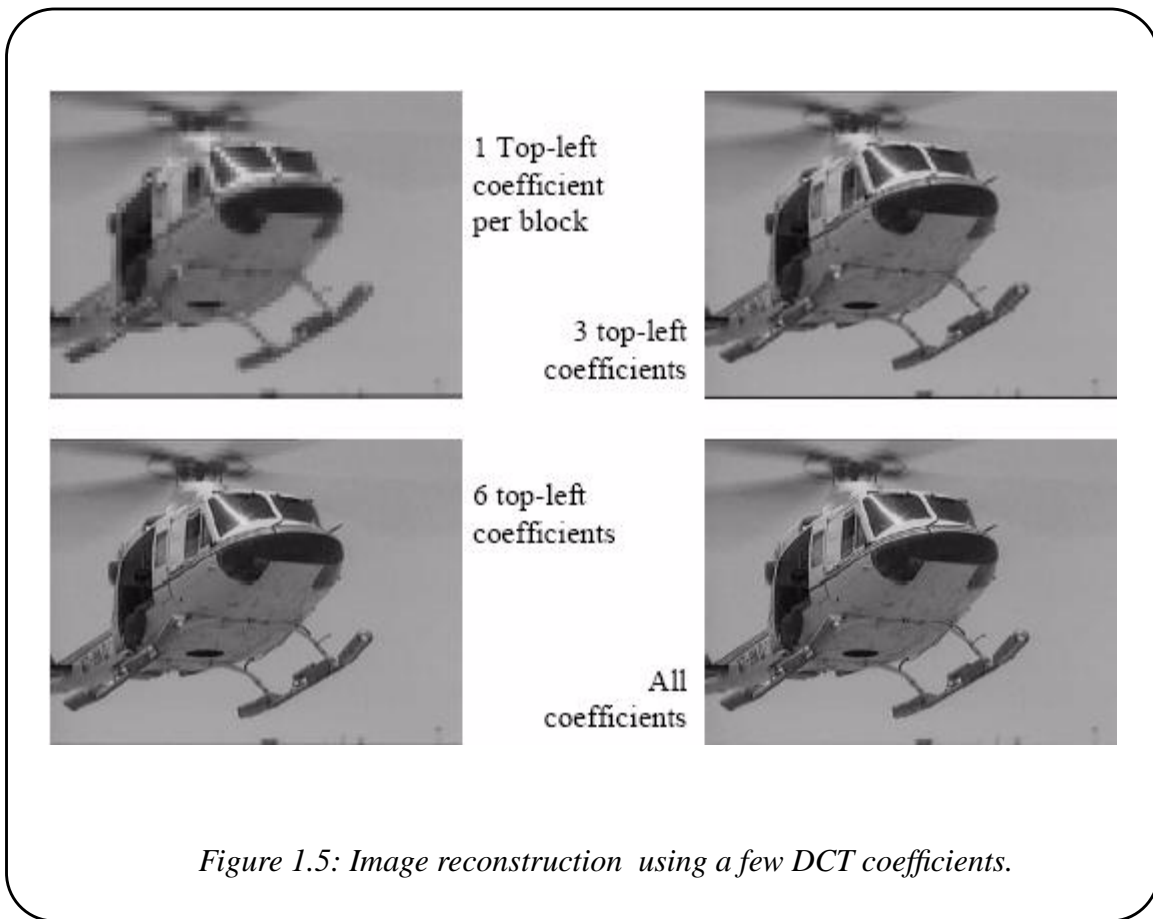


Figure 1.5: Image reconstruction using a few DCT coefficients.

1.5 Video Compression

Now that the principles of compressing an image have been learnt, it will be easier to understand basic principles of video compression or MPEG-4 AVC. The report shall limit video compression to MPEG-4 AVC standard only. Henceforth Video coding/Video compression/ Video decoding will all refer to those functions as done in MPEG-4 AVC. Motion Estimation (ME) will be shown to be an integral part of it. Once basic principles of Video Compression have been discussed, the report shall focus on the main topic i.e Motion Estimation.

Motion video is consist of a series of ‘still images’ or what are called ‘frames’ which are usually made to display at a rate between 10Hz and 30Hz, to produce a ‘motion’ effect for a human eye. It is known that how an image can be compressed, which suggests an intuitive way of compressing

video. Compress all image frames, and that would make an compressed video file. Where video encoding techniques have known to achieve compression ratios between 50:1 and 200:1 [2], the above technique will only produce a compression ratio up 30:1[21], as in JPEG. Well it is evident that a lot more is to be done. But how?

Figure 1.6 shows 2 adjacent frames from a video file. Figure 1.6 (a) is first frame and Figure 1.6 (b) is the following frame in time, or adjacent frame, or simply second frame. What is the visual difference between them? Its indeed very difficult to tell. In a sophisticated way it is said that video frames represent a lot of ‘temporal redundancy’. That is to say that pixel values in the second frame are quite similar to those in the first frame, and therefore the pixel values of frame 2 may be predicted, using the pixel values in frame 1 or visa versa. Exploitation of this class of redundancy to encode video is also termed as Intra Coding or Intra prediction. Motion Estimation is an integral part of intra coding. Much more on Motion Estimation will follow very soon.

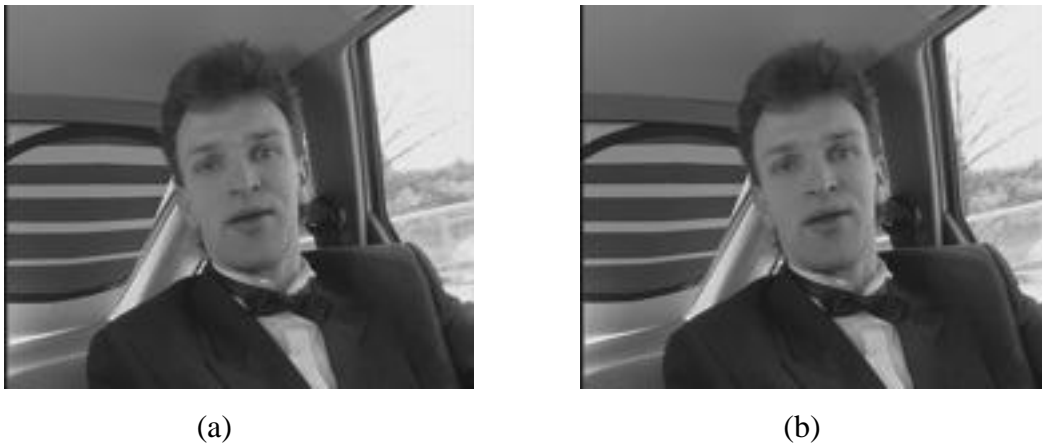


Figure 1.6: Two Adjacent Frames from a Video File.

There is yet another redundancy which exists in a single frame itself. It is called spatial redundancy. Uniform blocks in an image may be predicted from neighboring blocks. It is also termed as Inter prediction. Details of Intra prediction or intra coding is beyond the scope of this report. More can be found on Inter prediction in [22].

To compress video, we have lot more than just image compression i.e. Intra coding and Inter coding.

MPEG-4 AVC thus has the following scopes of compression:

- 1). Intra coding
- 2). Inter coding
- 3). Image compression
- 4). Entropy Encoding.

Figure 1.7 (a) gives a block diagram of an MPEG-4 AVC encoder and Figure 1.7 (b) gives the block diagram of MPEG-4 AVC decoder. Note Entropy encoder/decoder is not shown.

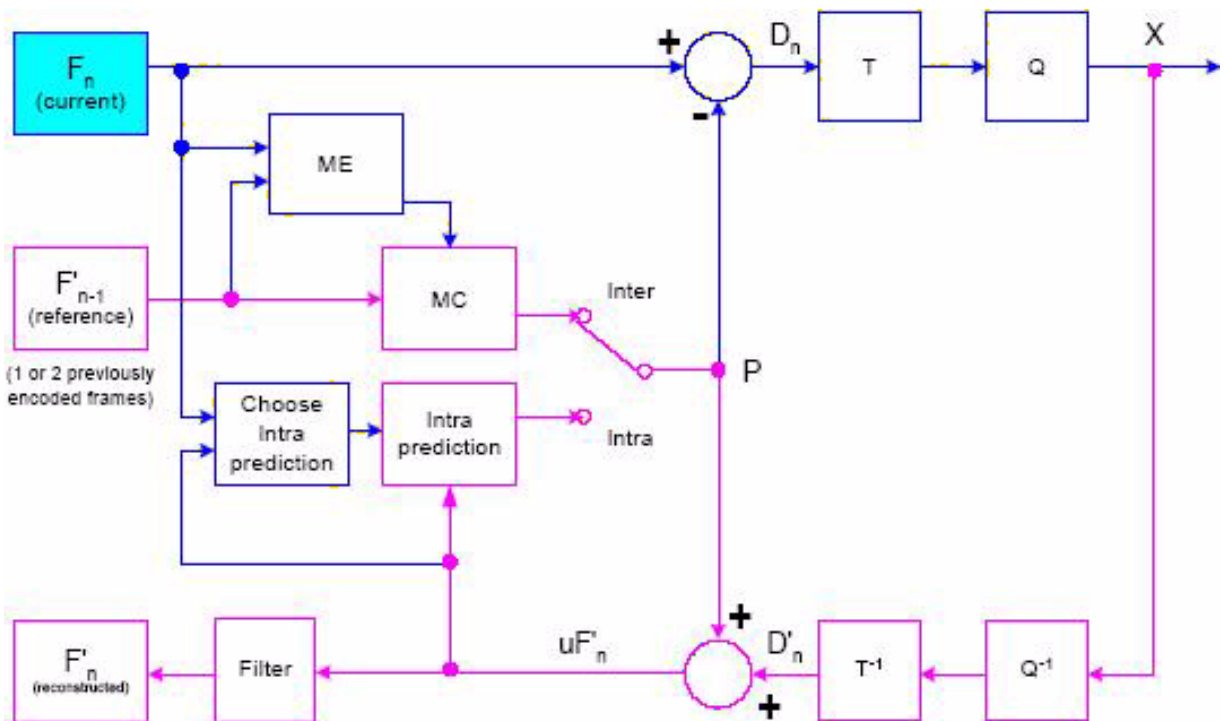


Figure 1.7 (a) : MPEG 4 AVC Encoder Block Diagram

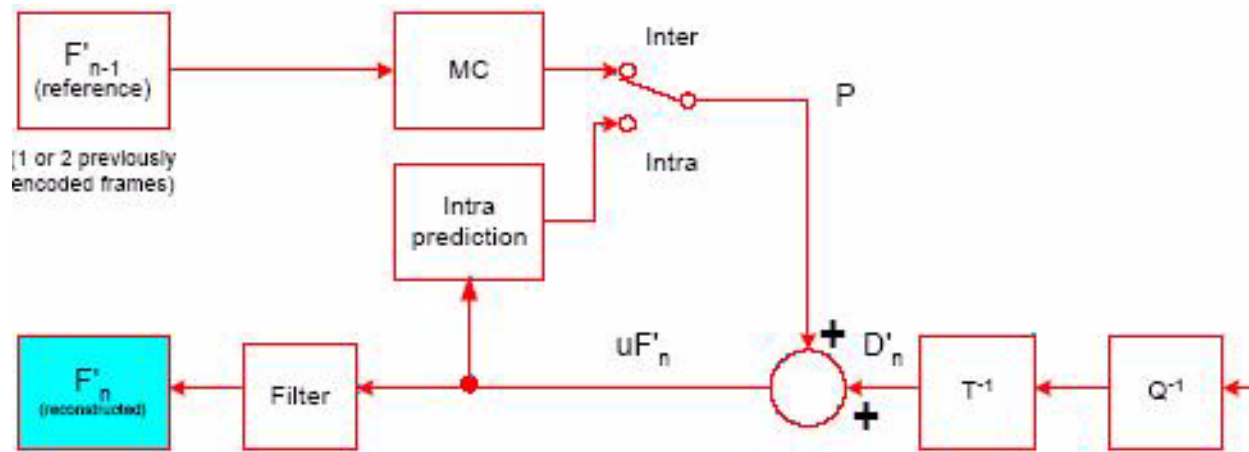


Figure 1.7 (b) MPEG 4 AVC Decoder Block Diagram

Having introduced image, video, image compression and video compression in digital world, it will now make sense to talk about Motion Estimation, what it is, where it stands in the whole process, what is its importance, and how it is done.

SECTION 2 MOTION ESTIMATION

2.1 Motion Estimation: An Introduction

Motion Estimation is a part of ‘inter coding’ technique. Inter coding refers to a mechanism of finding ‘co-relation’ between two frames (still images), which are not far away from each other as far as the order of occurrence is concerned, one called the reference frame and the other called current frame, and then encoding the information which is a function of this ‘co-relation’ instead of the frame itself. Motion Estimation is the basis of inter coding, which exploits temporal redundancy between the video frames, to scope massive visual information compression

Inter Coding:

As we have seen two adjacent frames in a video sequence, does exhibit a lot of temporal redundancy. In order to achieve coding efficiency, this temporal redundancy is exploited. Consider two hypothetical adjacent frames as shown in Fig 2.1, and 2.3 The grid squares in the frames make the figures more elaborative, note that these squares are not pixels, but in reality these grid squares may represent a number of pixels. Figure 2.2 shows a block of pixels which is termed as ‘macroblock’. This macro block is arbitrarily chosen from Figure 2.1. The size of ‘macroblock’ is not fixed in MPEG specs. In fact a number of size and combination of aspect ratio are allowed as per the MPEG specs such as 4x4, 4x8, 8x4, 8x8, 8x16, 16x8, 16x16.

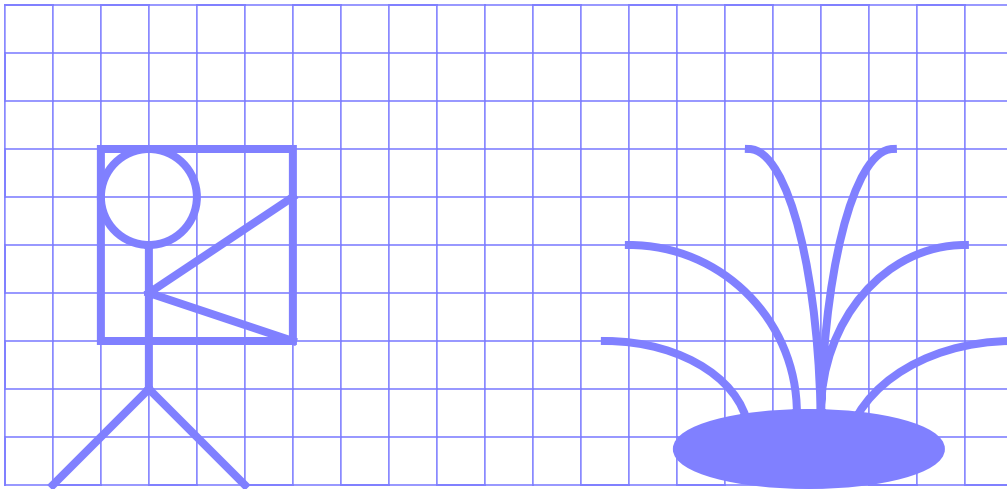


Figure 2.1 Reference frame or 'I' frame

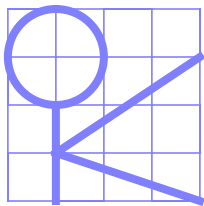


Figure 2.2 Macroblock

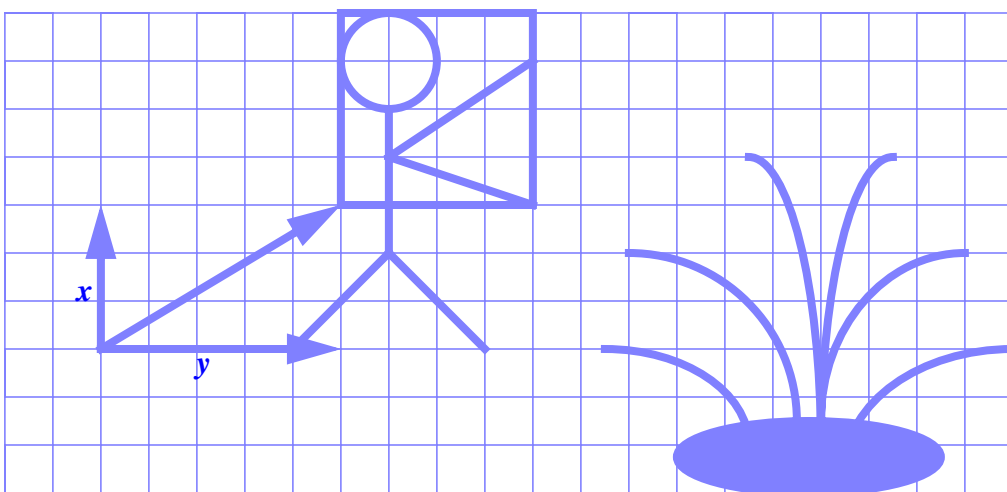


Figure 2.3 Current Frame 'P' frame with motion vectors with respect to Reference Frame

The objective here is to find how much in 'x' direction and 'y' direction has the chosen macroblock (shown separately in figure 2.2) has moved from its position in reference frame 'I' shown in figure 2.1 to a new position in current frame 'P' shown in figure 2.3. To evaluate these 'x' and 'y' values, each possible macroblock in a 'search window' in frame I may be compared to the shown macroblock in frame 'P'. A 'cost' corresponding to each comparison is evaluated. The minimum cost corresponds to a 'match'. The 'x' and 'y' values corresponding to this 'matched' macroblock are recorded and called 'motion vectors' for the 'macroblock' which was being 'searched' in reference frame 'I'. This process of finding 'motion vectors' for each block in the current frame is termed as '**Motion Estimation**'. In this way motion vectors corresponding to each and every macroblock in current frame are evaluated. The way in which the 'search' of the minimum 'cost' macroblock in frame 'I' is done is also not fixed by the specs. The 'search' mechanism where a macroblock from the current frame 'P' is compared with every possible macroblock in a defined 'search window' in reference frame I, to find motion vectors, is called **exhaustive search**. It is the most accurate method to perform search, but as it is evident it is very compute intensive.

It can now be said that instead of coding a frame it is possible to code its 'motion vectors' to bring about efficiency in coding. The method described above where redundancy in adjacent frames are exploited is called inter coding.

Inter coding requirements

Well, it is not always possible to perform inter coding. Some of the requirements for inter coding to be successful are:

- No panning
- No zoom
- No changes in luminance
- No rotational motion

2.2 Variable Block Size Motion Estimation:

The size of the macroblock is required to be variable, so that it can be tuned as per the requirements. For example a smaller macroblock may be required for the area in a frame which is quite detailed, whereas larger macroblocks are suited for homogenous areas of a frame. Figure 2.4 from [3] shows the sizes of selected macroblocks superimposed on the frame itself. Note the size

gets smaller in detailed region of the frame. These variable size macroblocks are also sometimes referred to as 'partitions' in some literature

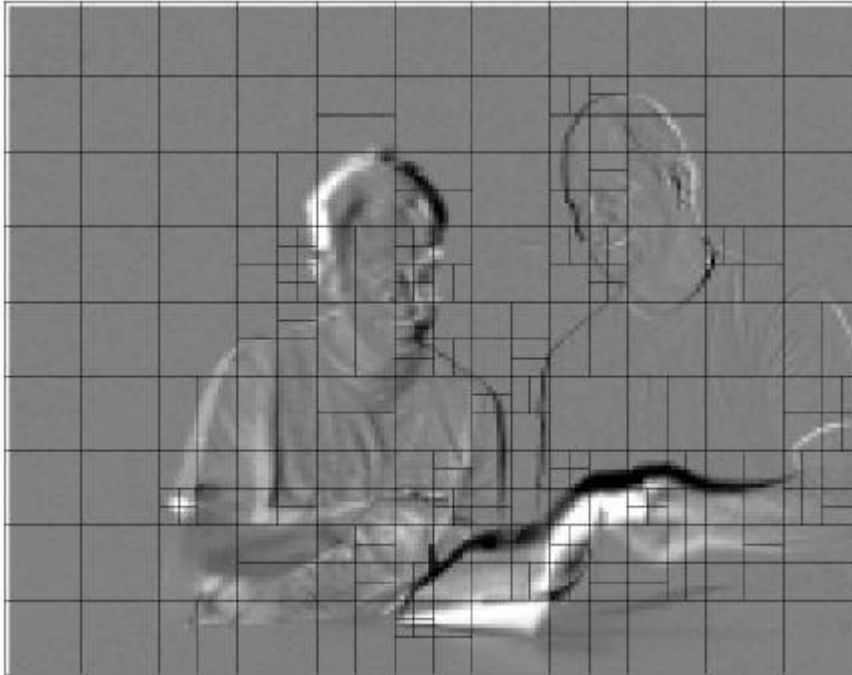
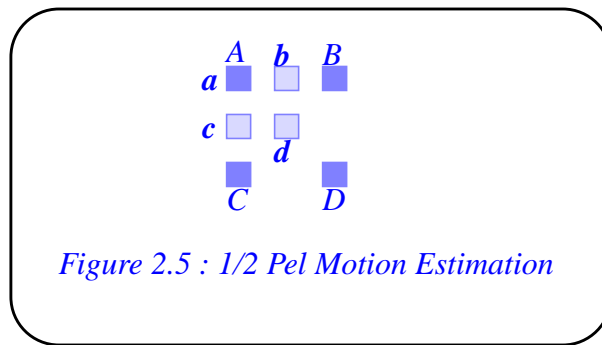


Figure 2.4. Variable size macroblocks shown as per the 'fine-ness' of image
Smaller size macroblocks are featured for detailed areas where as
larger size macroblocks are featured for homogenous areas

2.3 Sub Pel Motion Estimation

To improve quality of matching, sometimes a scheme called sub-pel motion estimation is used. Sub-pel refinement is performed in a small local window around previously found full-pel motion vector. This local pixels are interpolated values of pixels around them. Fig 2.5 shows the location of 1/2 pel pixels in a block.



A, B, C and D are integer or actual pixels of reference frame, where as a,b,c and d are 1/2 pel values. a,b,c and d may be constructed using the following equations.

$$b = (A+B+1)/2$$

$$d = (A+B +C +D +2) /4$$

MPEG 4 uses the FIR filter to compute half-pel samples, it then uses bilinear interpolation to calculate the quarter-pel samples

Sub Pel Motion Estimation is a good idea to improve coding efficiency and to code very low frequency video signals, that comes of course at the cost of computation complexity. Since the objective here is to build a low power ME block, sub-pel motion estimation will not be implemented.

Now that it is understood that what ‘motion estimation’ is, the objective of the project is briefly re-stated, to co-relate it with the term ‘motion-estimation’.

The objective of this project is to be able to design, develop and test a hardware module capable of performing a selected ‘motion estimation’ scheme.

The input to the module will be reference frame ‘I’, current frame ‘P’, in ‘CIF’ or ‘QCIF’ format and the output would be motion vectors corresponding to each and every selected macroblock and the size of the macroblock or partition.

It is required that the design is carried out keeping hand held applications in mind, i.e it should consume as less power as possible.

It is also required that the design supports various macroblock sizes, or various partitions as they are sometimes referred to.

2.4 Primary Research:

Motion estimation is one of the most talked about subject within MPEG implementation. It is the most compute intensive blocks in an MPEG encoder, which consumes nearly 30% of the total compute power needed by the encoder[1]. A wealth of information is available over the world wide web on this topic. The search of words ‘motion estimation’ in ‘TITLE’ field at USPTO patents itself lists about 250 patents on this topic. Going through this endless sea of information is clearly beyond the scope of this project. However some most common algorithms will be studied, evaluated to come up either with a new algorithm or with new architecture of motion estimation block, aimed for reduction in power consumption with variable size macroblocks.

2.4.1 Power Consumption of MPEG Encoders

Power reduction can be brought about using variety of low power digital design techniques. But the main problem is the lack of available reference power figures from the ME block of commercially available MPEG encoders. It is relatively easy to find out the power consumption of the whole encoder chip, but it is very difficult to find out how much power is consumed by the ‘motion estimation’ block sitting inside the encoders. Some of these power figures for H264/MPEG-4 encoder chips are given in table 1 below. The source of this information is individual data sheets available from the company’s web site

Table 1: Power Consumption figures of various MPEG encoders

| Company | Video | Audio | fps | power | Source | part name | Process |
|------------|-------|-------|-----|-------|-------------|-----------|---------|
| Broadcom | yes | yes | 30 | 38mW | [Broadcom] | BCM2702 | 130nm |
| TMC | Yes | no | 30 | 55mW | [TMC] | IP | - |
| Fujitsu | yes | No | 30 | 70mW | [Fujitsu] | IP | 180nm |
| Mobilygen | yes | yes | 30 | 185mW | [mobilygen] | MG1264 | 130nm |
| Toshiba | yes | yes | 15 | 80mW | [Toshiba] | TC35273 | 180nm |
| Freescall* | yes | yes | 30 | 210mW | [Freescall] | MCIMX31 | 90nm |

* MCIMX31 from Freescall is not a standalone MPEG-4 Encoder. Its a Mobile Platform, which includes a lot more functionality than just MPEG encoding.

From the table above it seems that Broadcom’s engine is the lowest power consumption MPEG encoder. However it is to be noted that there has been no quality comparison done.

If it is assumed, based on the research paper [1], that ME block is the dominant power consumption module in an MPEG encoder, then the power consumption by the ME blocks of the above mentioned encoders will show the same trend. By that logic it is clear that the target power consumption of the ME block to be designed should be comparable to the above shown figures.

2.4.2 Power figures from Prior Art:

Research was conducted on how much power ‘Motion Estimation’ would consume from inside the MPEG-4 AVC encoder, but as said, its very difficult to have actual power figures from ME block inside the encoder from a commercially available Encoder, so nothing could be successfully returned from the industry. However research papers from IEEE, did give valuable results which are tabulated in Table 2 below.

The table shows 5 research papers each giving its features and power figures. As from the table it looks like that [12] is the most power efficient Motion Estimation block consuming 15-42mW depending upon the search window and the block size chosen. But these results do not have a common ground to make a proper comparison. There is no common set of features/technology/architecture/quality. Hence these values will be used as a very rough estimate of how much power is consumed by motion estimation block.

Table 2: Power Figures from ME blocks

| Research Paper | Power | fps | technology | features |
|----------------|-----------|---------------|------------|---|
| [7] | 95mW | 30 (QCIF) | 0.18u | sub pel ME up to 1/2 pel |
| [8] | 23.76mW | 30 (QCIF) | 0.13u TSMC | Full Search |
| [10] | 423mW | 36 (CIF) | 0.6 TSMC | Full Search, sub pel up to 1/2 pel |
| [11] | 247.04mW | Not Available | 0.18 TSMC | Full Search sub pel up to 1/2 pel |
| [12] | 15mw-42mW | 30 (QCIF) | 0.25u | Enhanced Full Search, sub pel up to 1/2 pel |

2.5 Popular Motion Estimation Algorithms: Existing Literature/Work on the topic

Apart from exhaustive search algorithm for motion estimation scheme described in section 2.1 the

following algorithms are identified.

1. DS: Diamond Search
2. HEXBS: Hexagonal Search
3. TSS: Three Step Search
4. FSS: Four Step Search
5. MDS: Modified Diamond Search.

1. DS: Diamond Search: Diamond Search is known to be one of the faster Motion Estimation Algorithms [9]. Its an iterative method, where in each iteration there are 'n' searches to be made. For first iteration 'n' = 9, and for subsequent iterations 'n' is either '4' or '5', depending upon the location of minimum SAD block. The block in question, for which motion vectors are required is compared with the superimposed block in the reference frame, and then 8 blocks surrounding it in a diamond shaped pattern, or Large Diamond as it is called as shown in the figure 2.6 below.

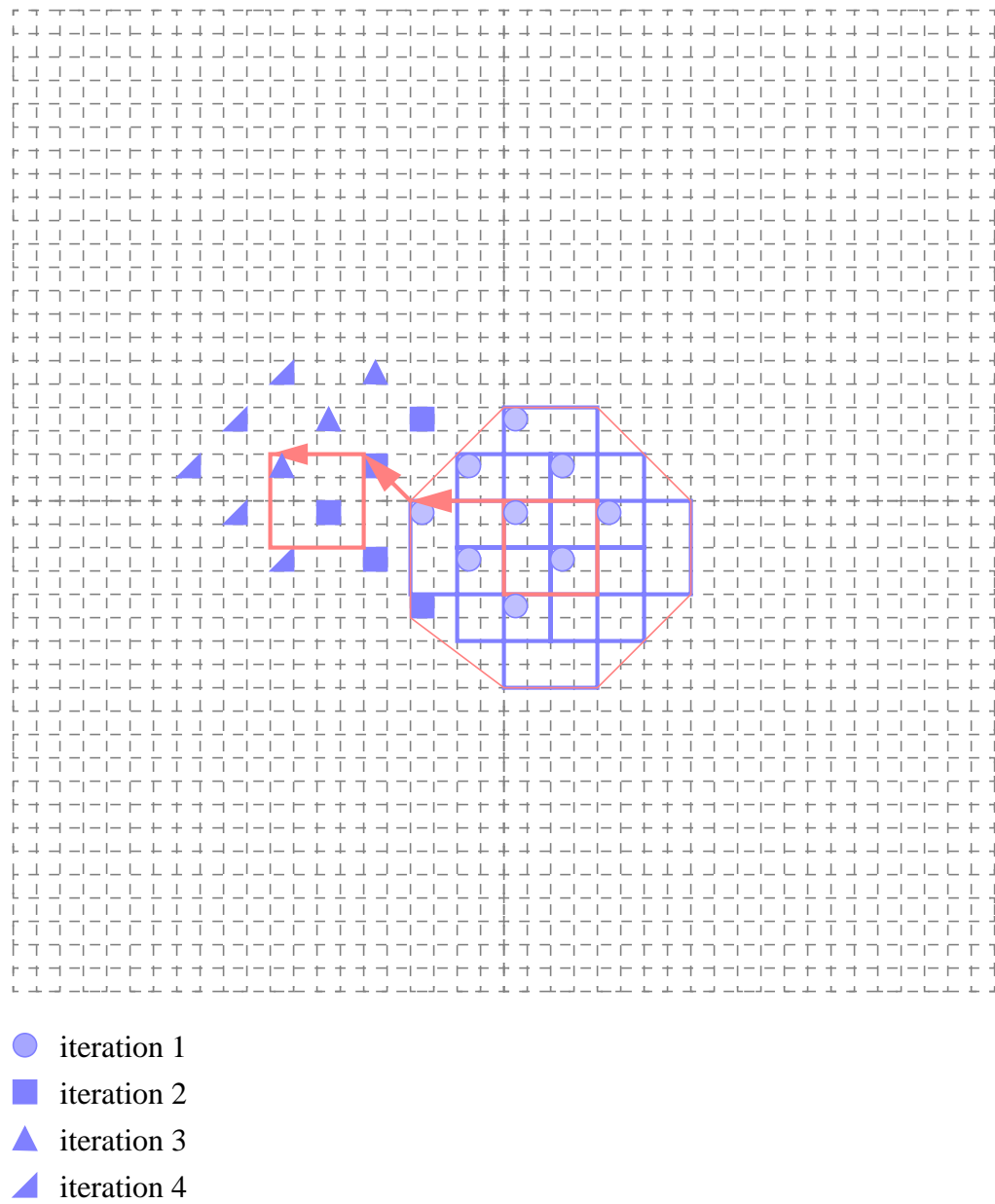


Figure 2.6 Diamond Search Algorithm

From the 9 searches made in the first iteration, if it turns out to be that the Min. SAD block is at the centre of the search window, i.e. the superimposed block from current frame over reference frame, then the algorithm stops. Where as if the Min. SAD block was not at the centre of the search window, then this MinSAD block is taken as a new centre and a diamond search is performed around it. This process is repeated for the number of iterations.

HEXBS: Hexagonal Search

Hexagonal Search is quite like the diamond search, but it reduces the number of searches per iteration. Figure 2.7 (a) below shows the search pattern, for both v-HEXBS fig (a) and h-HEXBS fig (b) patterns. Note that the number of searches in the first iteration in both case is 7 as compared to 9 in DS.

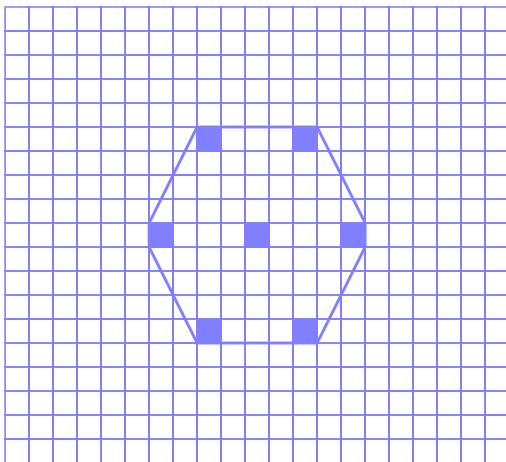


Figure 2.7 (a) v-HEXBS

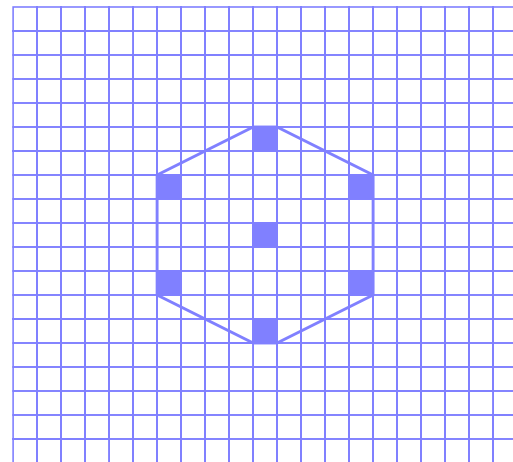


Figure 2.7 (b) h-HEXBS

TSS: Three Step Search

It starts with the search location at the center and sets the step size $S = \text{brow}/2, \text{bcol}/2$, where *brow* and *bcol* is the size of search window. It then searches at eight locations $\pm S$ pixels around location (0,0). From these nine locations searched so far it picks the one giving least cost and makes it the new search origin. It then sets the new step size $S = S/2$, and repeats similar search for two

more iterations until $S = 1$. At that point it finds the location with the least cost function and the macro block at that location is the best match. The calculated motion vector is then saved for transmission. It gives a flat reduction in computation by a factor of 9. So that for $(brow, bcol) = (8, 8)$, ES will perform 225 searches whereas TSS will perform 25 searches.

Four Step Search FSS:

FSS sets a fixed pattern size of $S = 2$ for the first step, no matter what the size of search window is. Thus it looks at 9 locations in a 5×5 window. If the least weight is found at the center of search window the search jumps to fourth step. If the least weight is at one of the eight locations except the center, then we make it the search origin and move to the second step. The search window is still maintained as 5×5 pixels wide. Depending on where the least weight location was, we might end up checking weights at 3 locations or 5 locations. Once again if the least weight location is at the center of the 5×5 search window we jump to fourth step or else we move on to third step. The third is exactly the same as the second step. IN the fourth step the window size is dropped to 3×3 , i.e. $S = 1$. The location with the least weight is the best matching macro block and the motion vector is set to point o that location. A sample procedure is shown in Fig 8. This search algorithm has the best case of 17 checking points and worst case of 27 checking points.

MDS: Modified Diamond Search

While DS gave acceptable PSNR results, it was seen from analytical analysis (details of which are in next section) that DS PSNR was falling for videos having very detailed areas such as ‘fore-man.qcif’ so that it became close to the cut off 30dB mark. **DS search algorithm was then modified to improve PSNR values for such cases by including 9 more searches around the centre of search window at a distance of 1 pixel in each direction.** This improved the PSNR of detailed video sequences, giving a better quality than regular Diamond Search Algorithm. **Henceforth in this report, DS term will be used for what is actually MDS, which is developed as a part of this project work.**

SECTION 3 Software modelling and Analytical analysis

What algorithm, what block/partition size, what frame depth, what search window size, what accuracy, what quality. There are fundamental questions which must be answered before making any attempt to build hardware. Answering these questions is a vast research area, and more than a thousand research papers/patents exists related to one or more of these questions. However a generous attempt is made to answer analytically most of the questions, and justifying the choices made to build hardware.

Due to limited time scale one fast algorithm is selected, which is compared against exhaustive search algorithm taking into account various comparison matrices such as quality, complexity, power, area, speed. Research [9], [13], [14]. [15], [16], [17] suggests that Diamond Search (DS) and Hexagonal search (HEXBS) or modifications of these are two popular, industry/academia adopted fast search algorithms. This project selects Diamond Search as the fast algorithm for analysis, since it is being used by industry, i.e. at Philips, 3DLabs, Ittiam Software and also because its less complex to design, and yields better results than HEXBS in terms of PSNR [13]. HEXBS however is faster than DS.

For this purpose following C-models were written, listings thereof can be found in the appendix.

- 1). *cal_sad_psnr_es.c*: For calculating psnr values using Exhaustive Search algorithm, using Sum of Absolute Squares.
- 2). *cal_mse_psnr_es.c*: For calculating psnr values using Exhaustive Search algorithm using Mean Square Error
- 3). *cal_sad_psnr_ds.c*: For calculating psnr values using Diamond Search algorithm, using Sum of Absolute Squares
- 4). *cal_mse_psnr_ds.c*: For calculating psnr values using Diamond Search algorithm using Mean Square Error

Several Unix scripts were written to automatically run these programs over 10 qcif files, with various variable options such as block size and frame depth, and collect the analyzed data into a matlab readable format. Matlab was used to graph the results from C/Unix analysis.

3.1 PSNR

Most of the questions will need some reliable metric which is a measure of quality of image which is reconstructed using the produced motion vectors and a reference frame. **PSNR Peak Signal to Noise Ratio** is the most widely used metric to measure the quality of image. It is a mea-

sure of how good is the reconstructed image frame R is when compared to the corresponding original frame I. The following equation gives a mathematical expression of PSNR

$$PSNR = 10 \times \log \left[\frac{\{V_{max} - V_{min}\}^2}{\frac{1}{row \times col} \times \sum_{i=0}^{row} \sum_{j=0}^{col} |R_{ij} - I_{ij}|} \right] \quad \text{----- Equation (3.1)}$$

Where, V_{max} is the maximum value a pixel can take, V_{min} is the minimum value a pixel can take, typically if a pixel is represented by 8 bits, then $V_{max} = 255$, $V_{min} = 0$.

row = total number of rows of pixels in an image

col = total number of columns of pixels in an image

R_{ij} = pixel from reconstructed frame R, whose quality is of interest

I_{ij} = corresponding pixel from original frame I.

Log is usually taken because the dynamic range of PSNR without log is very large.

A PSNR value which is greater than 30dB is generally said to have acceptable visual quality.[23]

Motion estimation involves comparison of blocks/partitions from current frame P and reference frame I. How it is comparison done? Well, there are two very popular metric to perform block comparison.

1). SAD: Sum of absolute differences

2). MSE: Mean Squared Error.

SAD is the accumulation the absolute value of difference of each pixel value from a current block and the corresponding pixel value from the reference block. The following equation 3.1 gives a mathematical expression for SAD

$$SAD = \sum_{i=0}^{row} \sum_{j=0}^{col} |P_{ij} - I_{ij}| \quad \text{Equation -----(3.2)}$$

where P_{ij} is pixel from current block, I_{ij} is the corresponding pixel from reference block, 'row' is the number of pixels in a row of the block/partition, 'col' is the number of pixels in one column of the block/partition.

MSE is the accumulation the squared difference of each pixel value from a current block and the corresponding pixel value from the reference block. The following equation gives a mathematical expression for MSE, averaged by total number of pixels in the block. The following equation gives the mathematical expression of MSE

$$MSE = \frac{1}{row \times col} \times \sum_{i=0}^{row} \sum_{j=0}^{col} |P_{ij} - I_{ij}|$$

There is enough evidence on the internet that both are widely used as a metric to measure block match. Larger is the value of either, more is the difference between blocks, less is the value of either, less is the difference between blocks. For two identical blocks, SAD=MSE=0.

3.2 Analysis 1: SAD vs. MSE

Now there is a choice, what to choose, SAD or MSE for this project. To be able to make a judicious choice, a number of video files were analyzed. Two frames from each video were taken, Reference frame I, and Current frame P. Motion estimation was performed, using exhaustive search method and diamond search method to get motion vectors for the current frame P. A frame R was then reconstructed using those motion vectors and Reference frame I. PSNR was then found between reconstructed frame R, and current frame P. Results from the analysis are shown below:

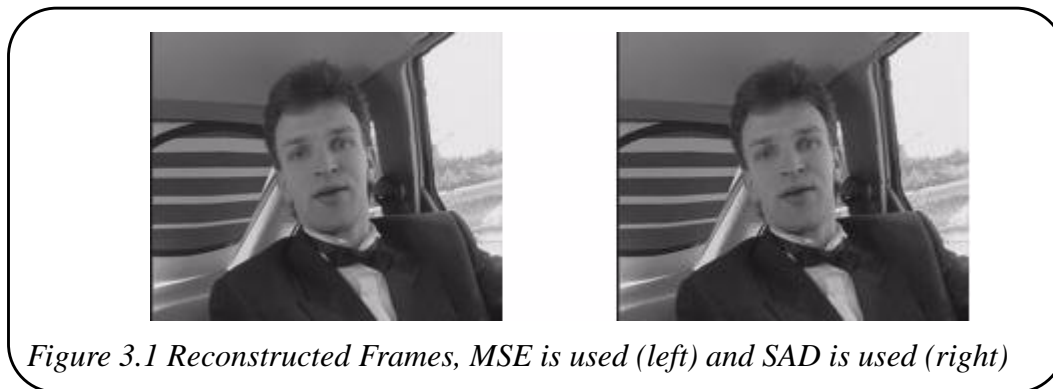
Table 3: SAD PSNR and MSE PSNR Comparison: Exhaustive Search

| File Name | Block Size | Search Window Size | MSE PSNR | SAD PSNR | Delta(mse psnr - sad psnr) |
|----------------|------------|--------------------|-----------|-----------|----------------------------|
| carphone.qcif | 4x4 | 8x8 | 35.704185 | 35.466197 | 0.237988 |
| claire.qcif | 4x4 | 8x8 | 41.943333 | 41.777398 | 0.165935 |
| container.qcif | 4x4 | 8x8 | 43.597497 | 43.590082 | 0.007415 |
| foreman.qcif | 4x4 | 8x8 | 35.781010 | 35.570175 | 0.210835 |
| silent.qcif | 4x4 | 8x8 | 38.080121 | 37.943870 | 0.136251 |

Table 4: SAD PSNR and MSE PSNR Comparison: Diamond Search

| File Name | Block Size | Search Window Size | MSE PSNR | SAD PSNR | Delta(mse psnr - sad psnr) |
|----------------|------------|--------------------|-----------|-----------|----------------------------|
| carphone.qcif | 4x4 | 8x8 | 30.781979 | 30.629255 | 0.152724 |
| claire.qcif | 4x4 | 8x8 | 40.247171 | 40.22338 | 0.023786 |
| container.qcif | 4x4 | 8x8 | 43.568812 | 43.565037 | 0.003775 |
| foreman.qcif | 4x4 | 8x8 | 34.068561 | 33.892309 | 0.176252 |
| silent.qcif | 4x4 | 8x8 | 36.403965 | 36.062570 | 0.341395 |

These results show very little difference between MSE PSNR and SAD PSNR. The order of difference is always less than 1%. To a human eye, these difference are certainly immaterial as also depicted by the two reconstructed frames shown in figure 3.1 below, left one being reconstructed when MSE was used, with PSNR = 35.704185, the right one being reconstructed image while SAD was used with PSNR = 35.466197.



Given that low power is one of the main requirements for this project, SAD will be chosen to make block comparison because its just a subtraction, where MSE involves a square operation which is very expensive both in terms of area and power.

3.3 Algorithm Comparison ES vs. MDS

Sec. 2.5 gives some of the popular algorithms on Motion Estimation. A detailed analysis of all those algorithms is beyond the scope of this project due to tight time scales. However since Dia-

mond Search is quite widely used in industry/academia as suggested by [9] [16] [17], it is studied and analyzed, and modified, along with exhaustive search algorithm. Since exhaustive search is best as far as the quality (PSNR) is concerned, results from exhaustive search algorithm will be taken as a reference to compare diamond search algorithm results with.

3.3.1 Number of candidate blocks.

The number of candidate blocks searched to motion estimate one block from a frame depends upon the block size, and the search window. The following table summarizes the number of candidate blocks required to be searched to motion estimate one block from a given frame using ES. The calculation of these numbers is done for a centrally located block, with all candidate blocks found in the reference frame. The blocks on the edges will not have all possible candidate blocks existing in reference frame. For example the top left corner block will only have 4 candidate blocks instead of 9 in first iteration in MDS algorithm.

Table 5:

| Algorithm | Block Size | Search Window size | Number of Candidate blocks Searched |
|-----------------------------------|------------|--------------------|-------------------------------------|
| ES | 4x4 | 4x4 | 81 |
| ES | 4x4 | 8x8 | 289 |
| ES | 8x8 | 8x8 | 289 |
| ES | 4x8 | 8x16 | 561 |
| ES | 8x8 | 16x16 | 1089 |
| ES | 16x16 | 32x32 | 4225 |
| Average Number of Searches for ES | | | 461.8(Excluding 16x16) |

While for the ES algorithm, the number of searches rises exponentially with increase in block size, the worst number of searches in MDS are fixed and that being equal to 9 (First Iteration)+8 (Second Iteration at 1 pixel distance from centre)+5 (3rd iteration which takes MinSAD block from prev iteration as the Centre)+8(Again 8 searches around the MinSAD block at a distance of

1 pixel from the centre)=30 for 4 iterations and 25 for 3 iterations. Therefore as compared to ES, MDS certainly the algorithm of choice as far as low power implementation is concerned. But nothing is free in engineering, with that massive savings in computations, there is a quality compromise. The next section details what is quality in video frames, how it is measured, and how much MDS effects it.

3.3.2 PSNR Analysis ES vs. MDS

Well it is seen that MDS is clearly a winner in reducing the number of computations, what remains is how much of a quality hit it is as compared to ES. To be able to answer that question, a detailed analysis is done, taking 10 qcif files, and PSNR values were obtained using both the algorithms. The following graphs i.e figure 3.2 and figure 3.3 shows the 2 PSNRs plotted for 2 the qcif files.

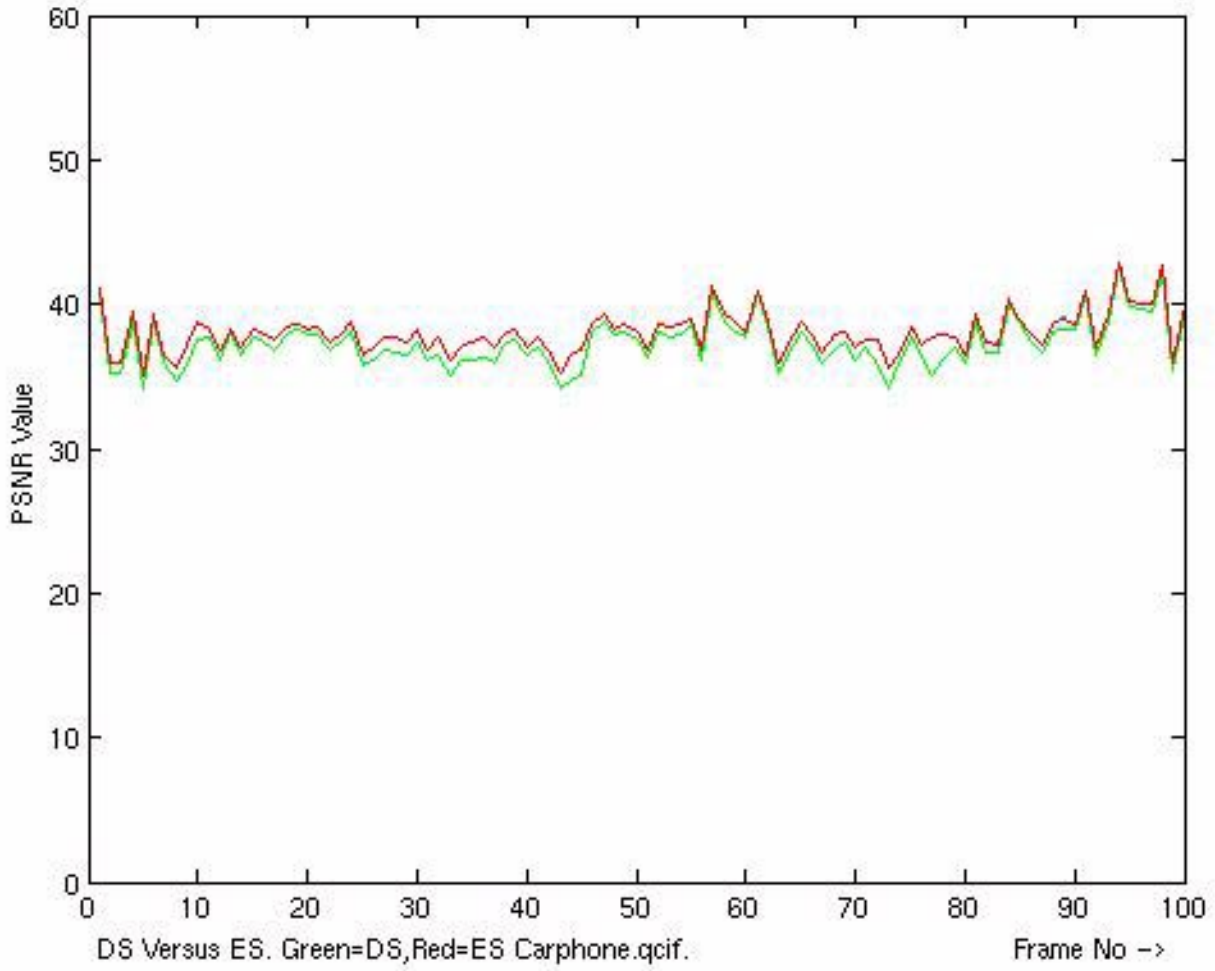


Figure 3.2 Worst MDS PSNR and ES PSNR value difference obtained with carphone.qcif

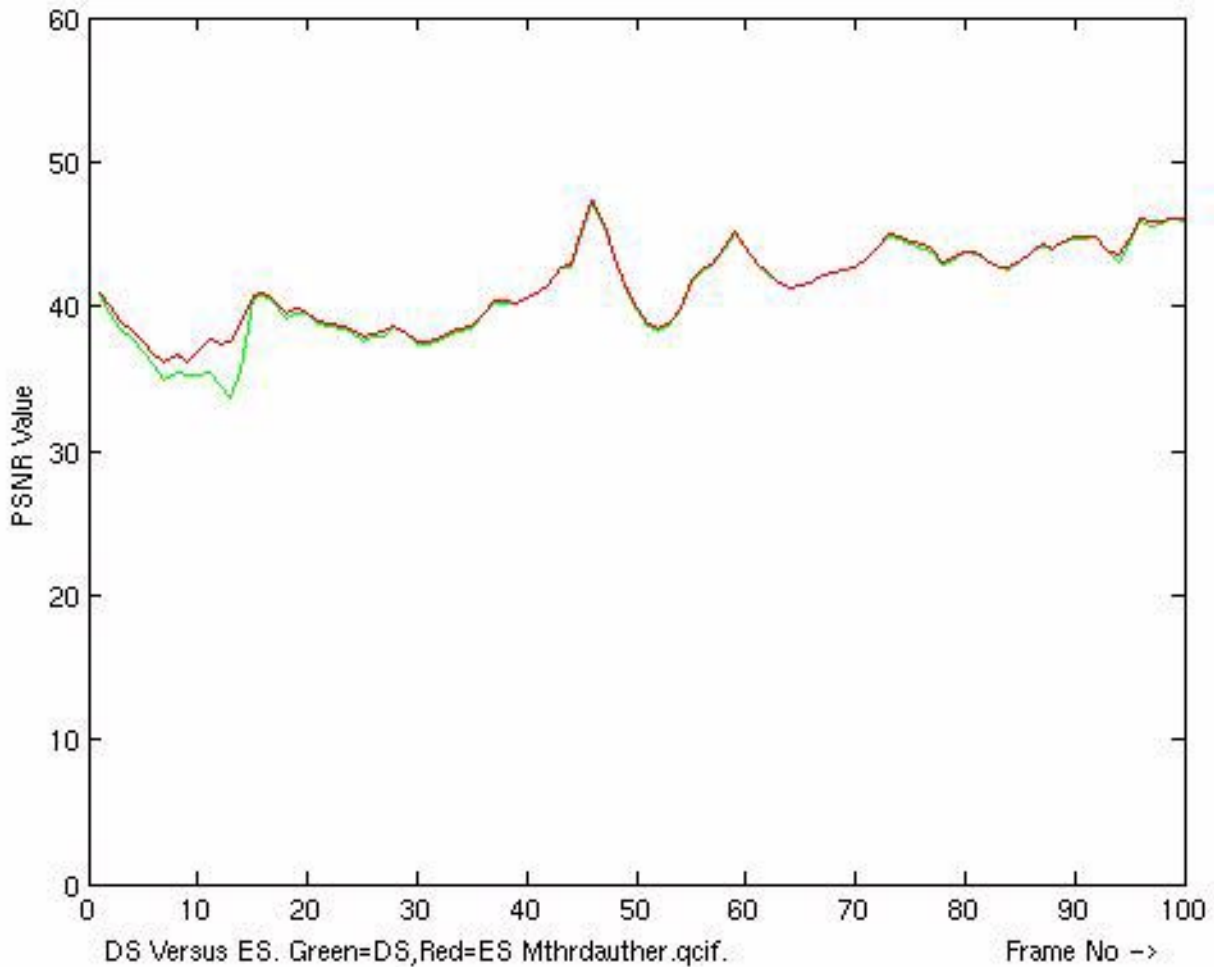


Figure 3.3 : Best MDS PSNR and ES PSNR difference was obtained with mthr_dotr.qcif

Figure 3.2 is one of the worst cases of difference between MDS PSNR and ES PSNR, whereas figure 3.3 is one of the best cases of difference between MDS PSNR and ES PSNR, as is evident from the fact that 2 lines in figure 3.2 are almost coinciding. It is to be noted that delta i.e. the difference between the reference frame and the current frame is 1 in these graphs.

The Results shows that the difference between PSNR values when Diamond Search Algorithm (DS) is used and when Exhaustive Search (ES) is not significant. In some qcif files the two graphs almost coincided, where as in some there was a minor difference. Analytically the maximum difference between PSNR values i.e. error from two algorithm was found to be 3dB. The following

graph shown in figure 3.4 shows all the error values from 10 qcif files, 100 samples taken from each of the 10 qcif files.

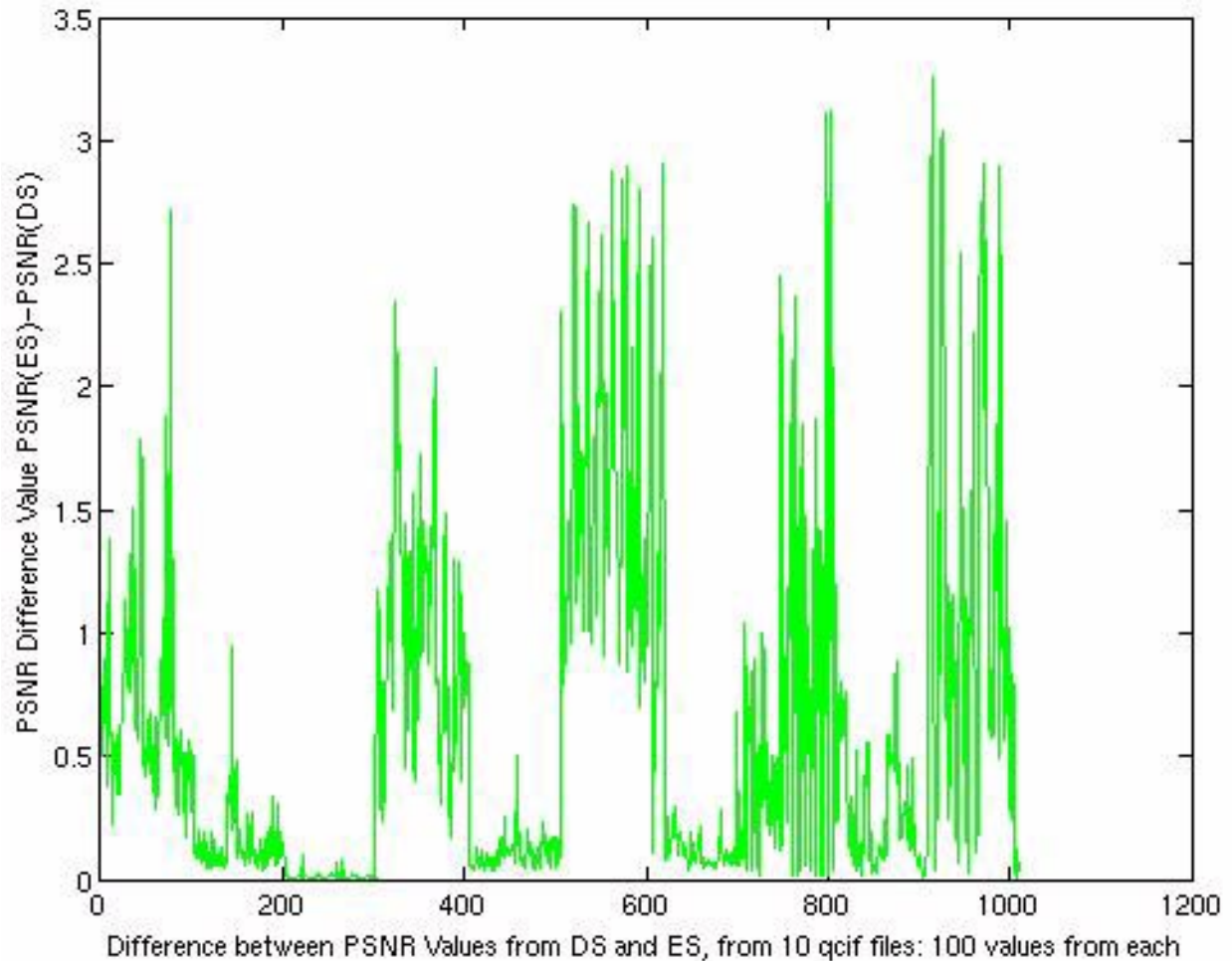


Figure 3.4 Error function. $PSNR(ES) - (PSNR MDS)$ for 10 qcif files, 100 samples from each

It was also observed that none of the PSNR values from DS algorithm was below 30dB. Which tells that all the images produced by using DS algorithm corresponding to a video is of acceptable quality. **There was found no evidence from 1000 image samples collected from 10 qcif files, that by using a diamond search algorithm for motion estimation, and reconstructing a**

image from so produced motion vector has dropped the PSNR value below 30 dB, what otherwise would have been greater than 30 dB if exhaustive search algorithm would have been used instead. Hence it is concluded that the quality hit using a diamond search algorithm would not affect the visual quality of the reconstructed video.

3.3.3. Frame Distance

Number of frames between a current frame and reference frame, which is also called Frame Distance. While having less number of frames between the current frame and reference frame helps in reducing PSNR, it has a disadvantage of making the encoded bitstream larger. The following graphs shows the variation in PSNR as the frame distance 'N' are increased from 1 to 5. Figure 3.5 (a) shows when ES is used figure 3.5 (b) shows when DS is used.

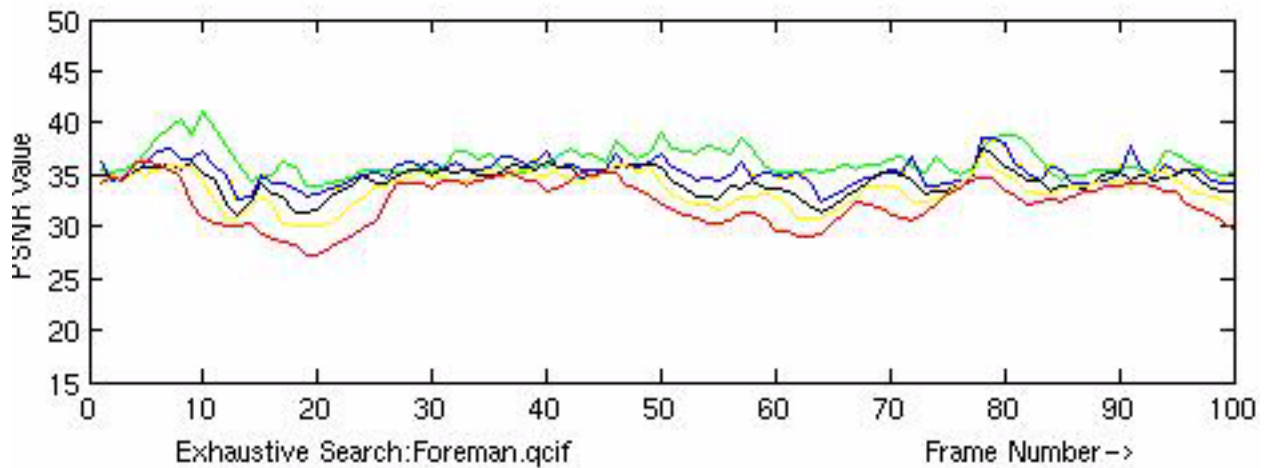


Figure 3.5 (a) PSNR values with Frames Distance of 1,2,3,4,and 5 using ES for foreman.qcif

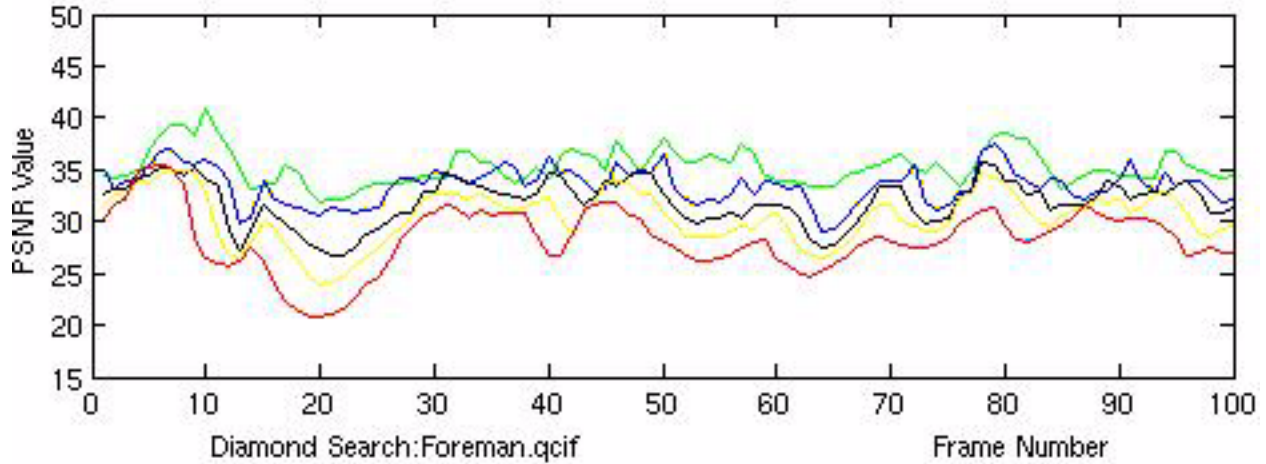


Figure 3.5 (b) PSNR values with Frames Distance of 1,2,3,4,and 5 using MDS for foreman.qcif

The above figures shows that while using ES, there is a scope of using the same reference for 4 forthcoming frames, we can only use the same reference frame for 2 forthcoming frames if DS is to be used. This conclusion is again based on the fact that 30 dB is considered to be an acceptable PSNR value for visual purposes, as suggested in [23], [24]

The following graphs in figure 3.6 shows the same analysis when done on a file called 'highway.qcif', using MDS.. It can easily be seen that even with the difference being 5 frames, PSNR always remains well above the 30dB mark.

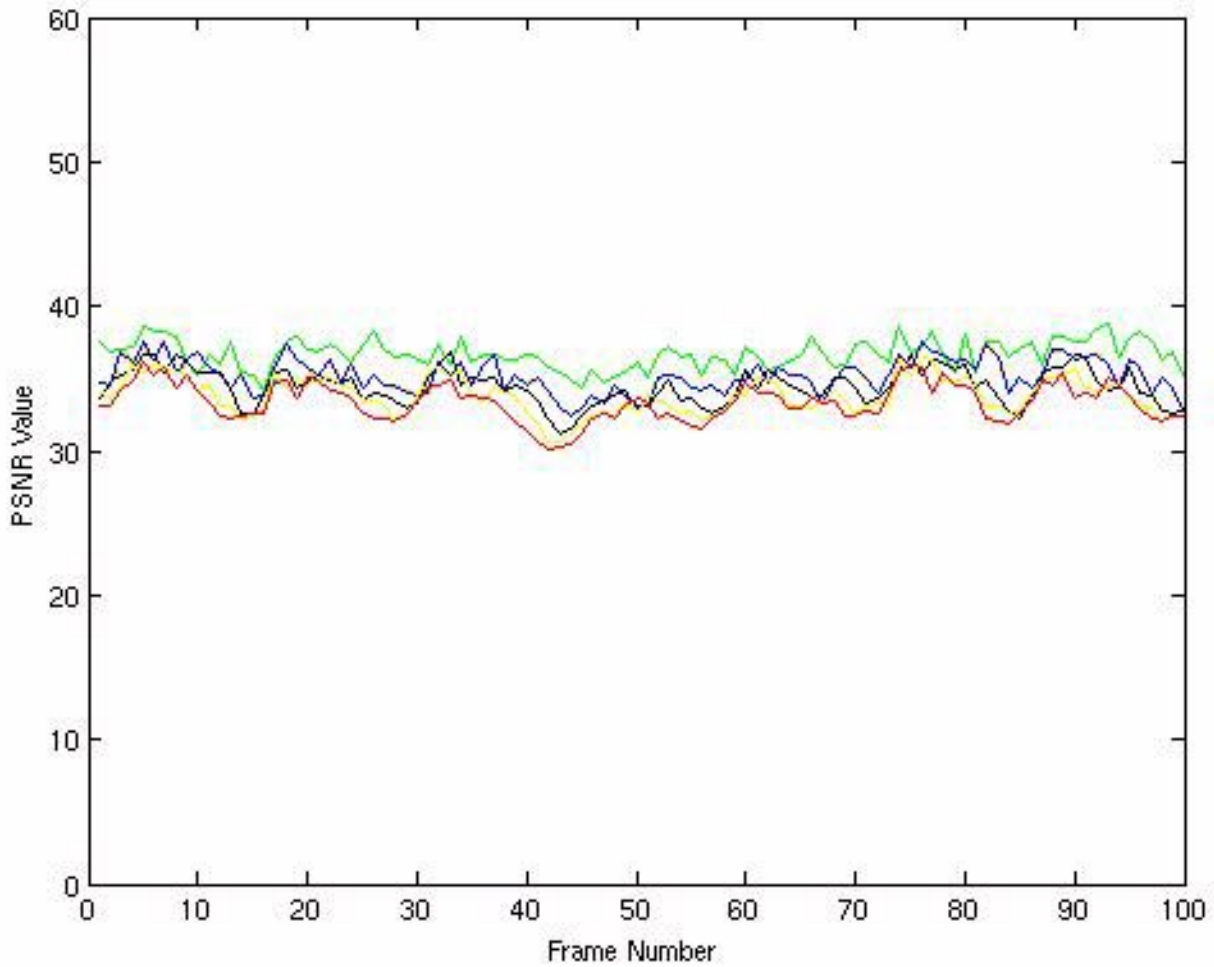


Figure 3.6 PSNR values with Frames Distance of 1,2,3,4,and 5 using MDS.for highway.qcif

So it can be seen how using DS can be disadvantageous in cases, where there are rapid temporal changes in video file.

This analysis was done for all 10 qcif files and the results were found to lie in between the 2 cases shown above by the means of graphs.

3.4 Conclusion: DS vs. ES

The following table gives a concluded comparison of DS vs. ES algorithm for motion estimation.

Table 6:

| Exhaustive Search | Diamond Search. |
|--|---|
| Best Quality, Max PSNR | Good Quality, Reasonable PSNR |
| Very Compute intensive | Needs to do very little computations. |
| Can be used to intercode distant frames from current frame | Not so good for the purpose of inter coding of distant frames, as PSNR drops below acceptable level sooner. |
| Very easy to implement in hardware or software. | Relatively complex Hardware. |
| Verification effort will be less | Verification will be a relatively complex |
| Power Hungary Hardware | Low Power Hardware |

Solved Example 3.1:

What is the estimated frequency of operation of a hardware motion estimation block using both ES Algorithm, and DS algorithm with the following parameters:

Block Size 8x8

Search Window 16x16

Input video format QCIF (144 rows x 174 cols)

Bits/pixel: 8

Memory bus Data width 8x4

Assuming one arithmetic operation /clock cycle.

Frames/Sec. 25

Solution: for ES algorithm:

If Nbpf is number of blocks per frame, Nbmo is the number of arithmetic operations required for motion estimating one block, Npinb is the number of pixels in a block and Fps is the number of frames per second in the video file then the total number of arithmetic operations needed in one second will be:

$$\text{Opers} = \text{Nbpf} \times \text{Nbmo} \times \text{Fps} \times \text{Npinb}$$

$$= (\text{row} \times \text{col}) / (\text{brow} \times \text{bcol}) \times \text{Nbmo} \times \text{Fps} \times \text{brow} \times \text{bcol},$$

where

row = total number of rows in a frame

col = total number of columns in a frame

brow = total number of rows in a block

bcol = total number of columns in a block

Putting row = 144, col = 176, brow = 8, bcol = 8, Nbmo = 1089(from table 4), Fps = 25, we get
Opers = 10781100.

Assuming one operation per clock cycle, the required frequency of the design would be 689.99 MHz

If 'p' bytes can be read from the memory in one clock cycle, a parallel hardware can be built to do 'p' operations in one clock cycle, reducing the frequency to

$$\{(\text{row} \times \text{col}) \times \text{Nbmo} \times \text{fps}\} / p$$

In the above example, the new required frequency

$$\text{fes} = 689.99 / p. \text{ Given that } p = 4,$$

$$\text{fes} \sim 172.49 \text{ MHz}$$

Solution for DS Algorithm:

All the parameters remain the same for DS, only thing which would change is Nbmo which would be 30 for 4 iterations instead of 1089

Therefore

$$\text{fds} \sim 172.49 / (1089/30) \sim 4.75 \text{ MHz}$$

It is shown that in the above example the frequency required for DS is about 30 times less than that of ES. **Since the power consumption is directly proportional to the frequency of operations, the hardware with DS algorithm is likely to consume 10-20 times less power than the hardware with ES algorithm for the given example.**

Similarly it can be calculated for a number of block sizes and a number of search window sizes. Situation is likely to worsen if CIF is used in the example instead of QCIF, where a search window can go up to 32x32.

The Decision:

Based upon the above table of conclusions, the conclusion derived from the solved example above, and given that the objective of this project is a Low Power design, **Modified Diamond Search Algorithm is selected for hardware implementation.**

3.5. Effect of block size on PSNR:

This is yet another area which needs study and analysis. It is intuitive to think that smaller block size will give fine quality. But smaller block sizes will have more number of blocks, and hence more number of motion vectors, thereby increasing the size of encoded bit stream. Some areas in an image can be very detailed others can be uniform. To co-op up with this situation, variable block size support is required. The size of the block will be smaller for detailed areas and larger for uniform areas of the image. As it is also shown in Section 2.4 earlier. The following figure 3.7 shows how the PSNR varies with the block size.

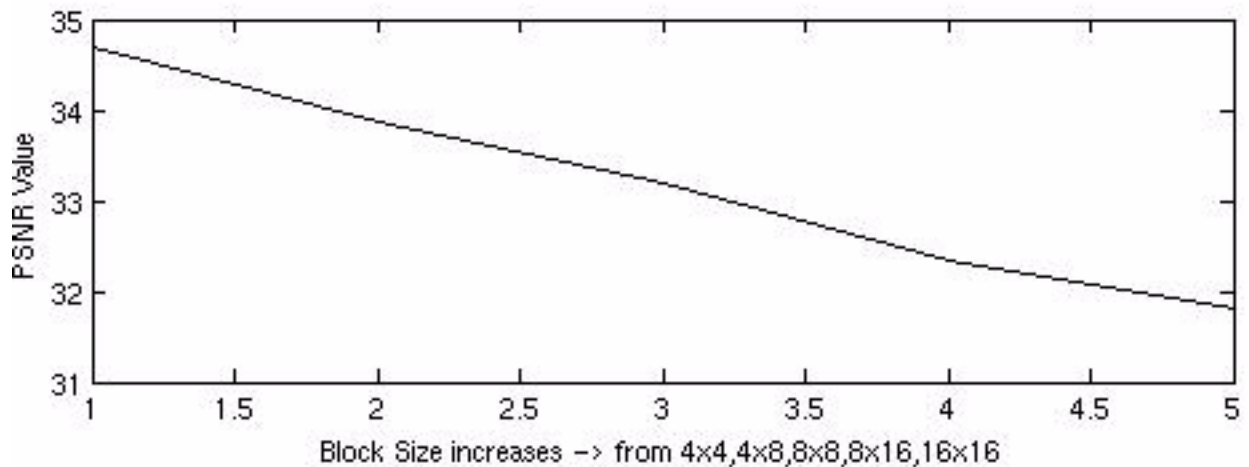


Figure 3.7 PSNR (or quality of image) decreases as the block size increases.

The figure 3.7 above shows clearly that as we increase the block size, the image quality metric i.e PSNR falls.

Final conclusions from the Analysis and inputs to hardware design

- 1). Algorithm to be used 3 iterations Modified Diamond Search Algorithm described in Section 2.5
- 2). If possible Variable block size will be used.
- 3). Frame Depth: The number of frames to be searched would be kept to 3. In case the PSNR drops below 30 for this value, the frame Depth will be reduced to 2.

SECTION 4: HARDWARE DESIGN

4.1 The Methodology:

How to design a hardware? Well, any method which may be employed to design hardware must be very structured. Unlike software, hardware design is characterized by massive parallelism. That means, at a single point of time, there can be multiple parts of the design which are active. which in turns makes hardware design tedious and difficult to debug as well A well known structured approach for building hardware is based upon classic control+datapath method, where the design requirements are analyzed and synthesized manually into two interacting subsystems, Datapath and Control. Apart from these two subsystems, Memories may be required to hold the data to be processed by the design unit, and to store the processed data. Datapath can be defined as a collection of various hardware elements required to execute a desired function. A Control Logic is a block which generates appropriate timing signals, to schedule and sequence functions of the datapath. Algorithm state machine charts will be used to elaborate upon the desired functionality. If needed a timing diagram will also be made to simulate the control logic with a paper pencil. Once ASM is ready, the datapath is ready, and the control logic is ready, it will be relatively easy to code and subsequently debug the design for the desired functionality.

4.2 Processing Elements for Hardware:

The C-reference model helps in identifying the processing elements used to build hardware datapath. Following is the list of PEs required

1. ACUs Address calculation Units for
Reference Frame Memory
Current Frame Memory
Reconstructed Frame Memory
2. Accumulator: for accumulating the difference in pixels from Reference frame and Current Frame
3. MinSAD Unit: To record Minimum SAD value. It will calculate minimum value on the fly. If the current delta(i.e RefPixelValue-CurrentPixelValue) is less than the current MinSAD Reg, MinSAD Reg will be updated with the current SAD value, and Motion Vectors will be recorded.

4. VecGenUnit: Vector Generation Unit: As a new MinSAD value is updated, using the current memory address pointer and reference memory address pointer, motion vectors are calculated, recorded and sent to output. This will be done by VecGenUnit.
5. AddressErrUnit: At times the address calculated may fall outside the *row x col* frame. For example, the upper left corner block, instead of 9 searches only 4 searches would be possible, because both at the left of this block and above this block there doesn't exist any blocks to be compared. In such cases the AddressErrUnit will flag that the current address is not useable.

Apart from these Datapath Units, the design would need

1. Control Unit to control the Datapath
2. Memories to store Current Frame (CurMem), Reference Frame (RefMem), and Reconstructed Frame (RecMem).
3. Fifo: To store the current reference block which is being compared. After the comparison is over, if the block is found to have new MinSAD, this block will be written into RecMem.

While the design of majority of design elements is straight forward, ACUs are not that simple. Also the design of the Fifo will be discussed to make it low power.

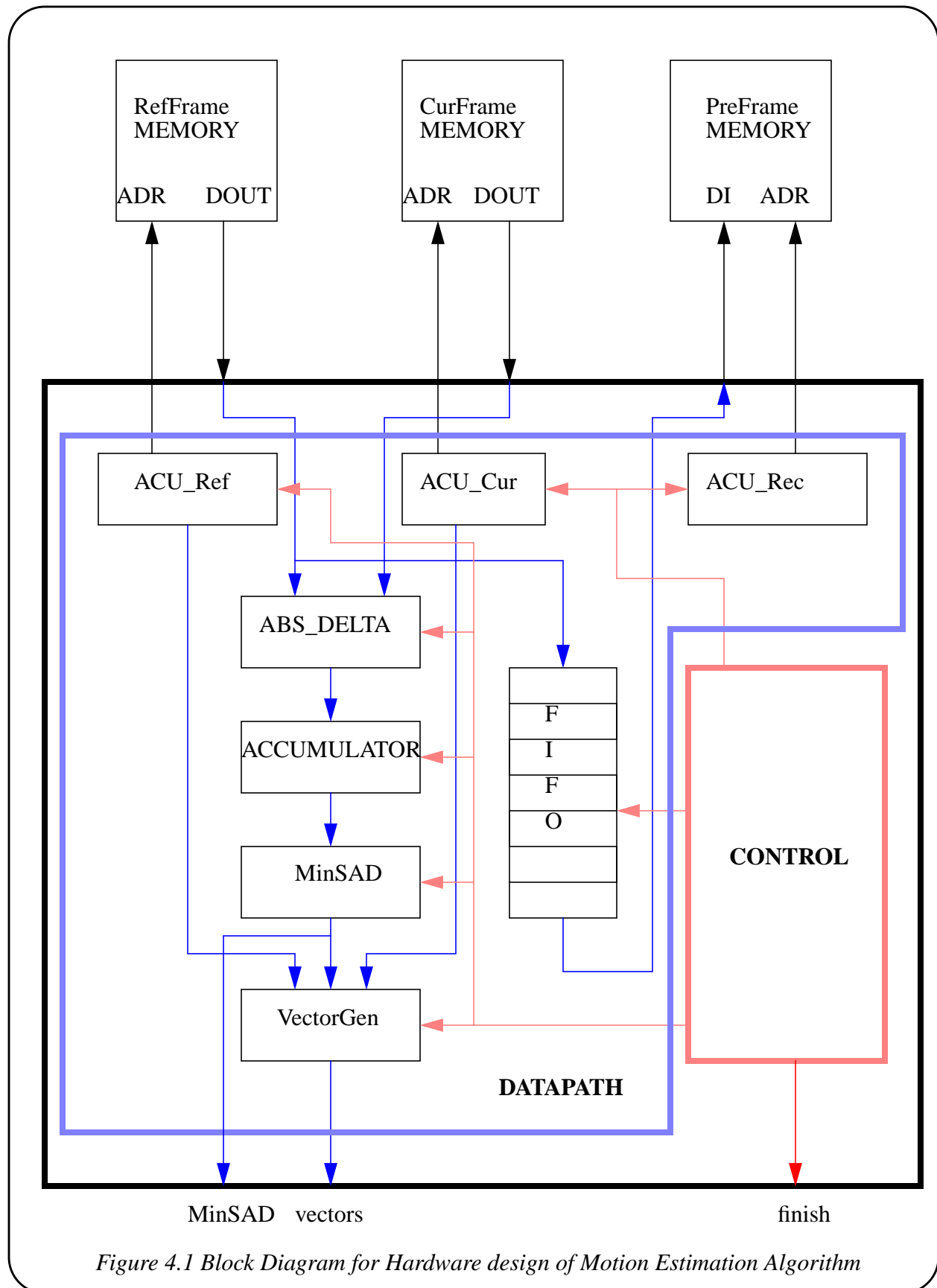


Figure 4.1 Block Diagram for Hardware design of Motion Estimation Algorithm

4.3 Designing for Low Power:

To be able to make a low power design, sources of power consumption are identified. With the help of already existing research, factors contributing to power consumption are also stated. It is then investigated, what factors can be played with in this project.

Power consumption in digital design has following 3 components [18], [19]

1. Switching Power, modelled as

$$P_{sw} = 1/2 CV^2fA$$

where C is the total circuit capacitance,

V is the operating voltage of the circuit

f is the clock frequency of the circuit

A is called activity factor. More on A will be discussed shortly.

2. Short-Circuit current Power, modelled as

$$P_{sc} = t_{sc} * V * I_{peak}, \text{ where}$$

t_{sc} is the slope of input signal

V is operating voltage

I_{peak} is the peak short circuit current.

3. Leakage Current power modelled as

$$P_{leak} = V * I_{leak}, \text{ where}$$

V is the operating voltage

I_{leak} is the leakage current.

Since the design methodology for this project is essentially semi-custom, where RTL level design would be mapped to already existing ASIC libraries, the following factors are identified as factors which cannot be played with.

- Voltage,
- Leakage Current
- Short Circuit Current
- Input Slopes

This makes Short-Circuit power and Leakage Power, something which cannot be controlled easily, unless there is a scope for shutting down the power to the design, or there is a scope for multi voltage design. These two factors are beyond the scope of this project. However, Switching power which may account up to 80% of power consumption in ASICs [18], does have following factors which may be controlled

C, Circuit Capacitance can be controlled by reducing the total area

f, Frequency of operation can be controlled by design.

A, Switching Activity can be controlled again by design

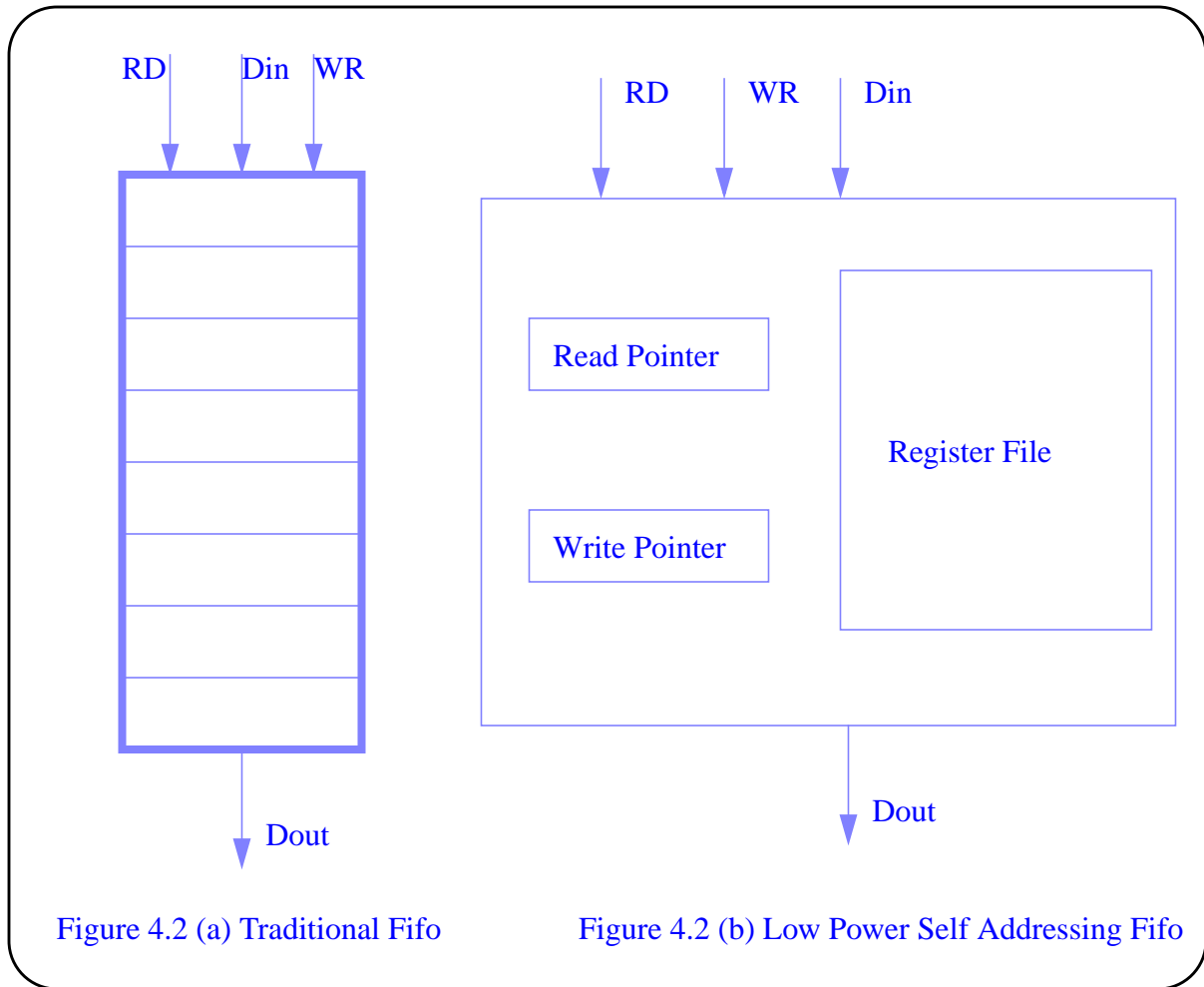
The focus of this project will be on reducing switching activity A, and f, frequency of operation. Since capacitance is a function of area, and area will be a function of architecture chosen, and architecture will be chosen to reduce frequency and switching activity factor, the resulting area (capacitance) may not be negotiated.

Switching Activity Reduction

1. The counters are designed as grey counters as opposed to binary counters to reduce switching activity. Grey codes are characterized by hamming distance of 1.
2. Clock gating has been used actively to reduce the switching activity on registers.
3. Provision to shut the memory clock off is provided in the circuit to save power consumption by the memory. Since for each current block, there will be multiple blocks fetched from reference frame memory, the clock to current frame memory CurMem, can be shut off once, a block to be compared is fetched.
4. Pipe lined architecture has been used, which means the input is continuously processed, and there are no wait states in the design, giving a 20% reduction in operating frequency.
5. Last but not the least, selection of Modified Diamond Search (MDS) as opposed to Exhaustive Search (ES) has been used giving massive reduction in frequency. **From the table 5, it can be seen that the average number of block comparisons for ES is 461, where as the number of block comparisons in MDS (3 iteration) is 25, which effectively reduces the operating frequency by a factor of $461/25 \sim 18$.** So the algorithm chosen for the purpose of this project is a major contribution in reducing the power consumption of the design.

4.4 Low Power Fifo Design.

The design is using a fifo. Traditionally a fifo is designed as shown in figure 4.2 (a), where at each read or write operation, the whole fifo shifts by one place. This kind of fifo obviously produces a lot of switching activity, where in each clock, ‘n’ registers change values, where ‘n’ is the depth of the fifo. The fifo depth depends upon the block size chosen, and as the design as the provision for choosing a variable block size, the fifo can be as large as 32x32x8 bits. Hence the fifo was design using a self-addressing scheme, where ‘read-pointers’ and ‘write-pointers’ were introduced to single out read and write positions in a fifo, as shown in figure 4.2 (b).



Since every word of data which is fetched from the RefMem is written in the fifo, for a QCIF frame, the total number of words, ‘w’ written to the fifo, for motion estimating one frame is given by

$$w = N_c \times b_{row} \times b_{col} \times N_b$$

where,

N_c is the number of candidate blocks searched for each block in the current frame,

b_{row} is the number of pixels in a row of a block

b_{col} is the number of pixels in a column of a block

N_b is the total number of blocks in the Current Frame

since $N_b = \text{row}/b_{row} * \text{col}/b_{col}$, where $\text{row} = 144$ for qcif, and 288 for CIF, and $\text{col} = 176$ for QCIF, and 352 for CIF,

$$w = N_c \times \text{row} \times \text{col}$$

Using traditional design of fifo, that would mean, for a single Frame, there would be

$$N_c \times \text{row} \times \text{col} \times d$$

registers changing value, per clock edge (since the pipelined architecture writes one word every clock cycle), where 'd' is the fifo depth, whereas using the self-addressing fifo, the number of words 'w' written per clock will be

$$N_c \times \text{row} \times \text{col} \times 1.$$

Which reduces the activity factor of the fifo by 'd'. Since fifo has to be designed so that it may be able to contain the largest block supported, 'd' has a value of $32 \times 32 = 1024$. So it can be seen that there is a considerable amount of switching activity reduction by using a self-addressed fifo.

4.5 Design of ACUs

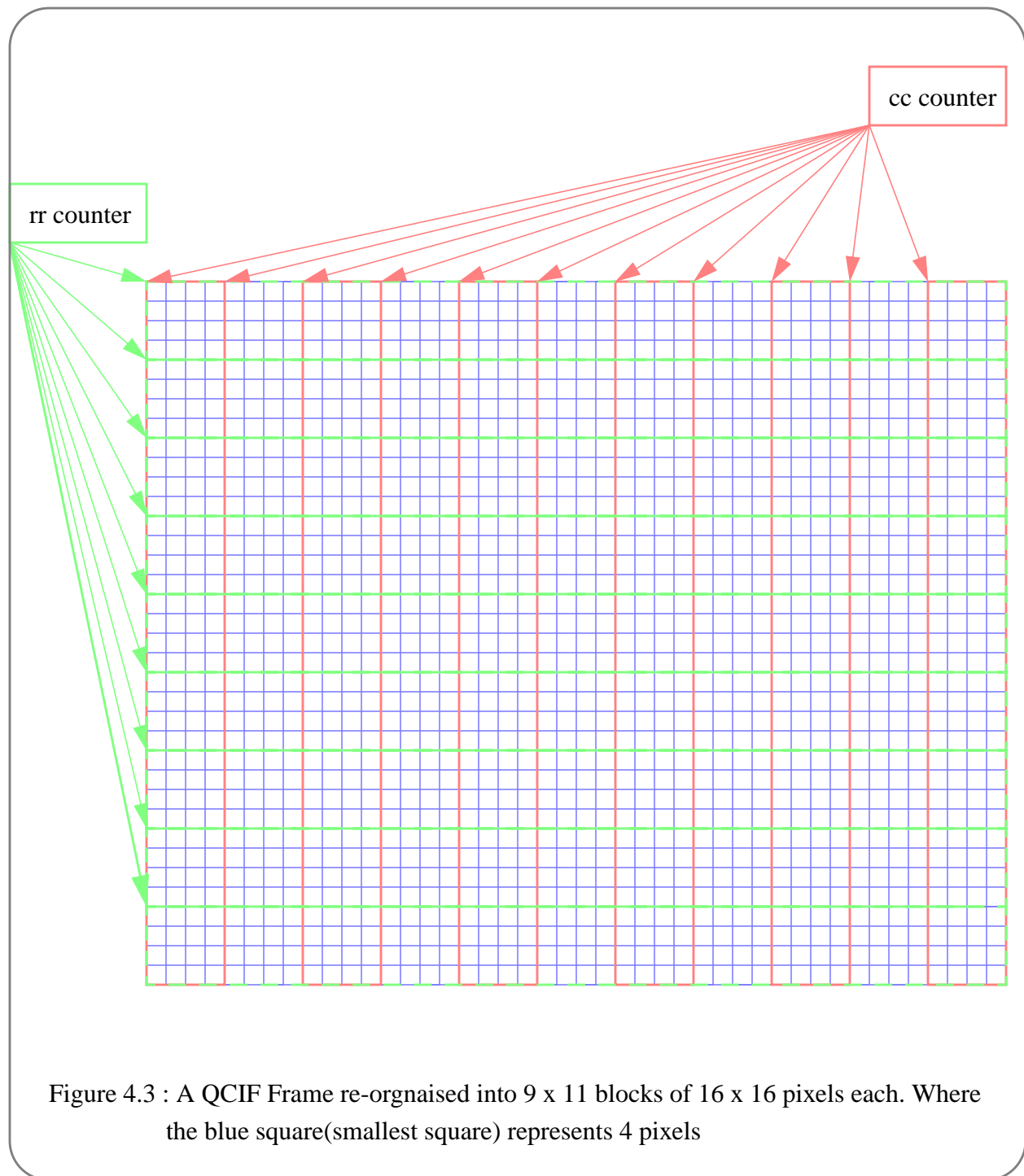
The storage of a Frame/Image in the memory will be in raster scan format. To be able to fetch a block of data from a frame, which do not have a fixed size, base+index addressing scheme is employed. The base address gives the location of top left corner of the block to be searched, and the index register would give the address of the pixel desired within that block.

The whole Frame is logically divided into $\text{row}/b_{row} \times \text{col}/b_{col}$ number of blocks, where 'brow' is the number of pixels in a row of a block, and 'bcol' is the number of pixels in a column of a block, or in other words 'row/brow' number of block rows and 'col/bcol' number of block columns. For example a QCIF frame with 16x16 block size will have $144/16 = 9$ rows of blocks and $176/16 = 11$ columns of blocks, or in other words a QCIF Frame with 16x16 block size can be divided into 9x11 blocks each having 16x16 pixels.

A row pointer register 'rr' is used to point to the rows of blocks and a pointer register 'cc' is used to point to columns of blocks, as shown in Figure 4.3 below. Clearly 'rr' will count from 0 to (row/brow)-1, and cc will count from 0 to (col/bcol)-1, to spawn the entire Frame.

$$\text{BaseAddr} = (\text{rr} \times \text{brow}) \times \text{col} + \text{cc} \times \text{bcol}.$$

Within a block, there would be brow pixels in a row, and bcol number of pixels in a column. Two pointer registers called 'iir' and 'iic' are used to do index addressing within a block. Clearly 'iir' will count from 0 to brow-1 and 'iic' will count from 0 to bcol-1, to spawn the entire block



2 sub addressing counters called 'iir' and 'iic' are used to point to a pixel within a block. Clearly 'iir' will count from 0 to brow-1 and 'iic' will count from 0 to bcol-1, to spawn the entire block, where

brow is the number of pixels in a row of a block

bcol is the number of pixels in a column of a block.

The location of a pixel within the block is therefore

$$\text{IndexAddr} = \text{iir} \times \text{bcol} + \text{iic}.$$

So, Index address is given by

$$\text{IndexAddr} = \text{iir} \times \text{bcol} + \text{iic}$$

Therefore the absolute address of the pixel being referred to is given by

$$\text{PixelAddr} = (\text{rr} \times \text{brow}) \times \text{col} + \text{cc} \times \text{bcol} + (\text{iir} \times \text{bcol}) \times \text{col} + \text{iic}$$

$$\text{PixelAddr} = (\text{rr} \times \text{brow} + \text{iir}) \times \text{col} + \text{cc} \times \text{bcol} + \text{iic} \text{ ----- Equation 4.1}$$

In hardware terms, since we know ‘brow’ and ‘bcol’ can only be a multiple of 2, multiplication with ‘brow’ or ‘bcol’ will be done by shifting the multiplicand by $\log_2(x)$, where x is the multiplier, can be ‘brow’ or ‘bcol’.

There is yet another multiplier in equation 4.1 above. That is ‘col’.

$\text{col} = \text{basecol} \times 2^{n+\text{cif_qcif}}$, where $\text{basecol} = 11$, $n = 3$, and $\text{cif_qcif} = 1$ for CIF, and 0 for QCIF.

$$\text{col} = (8 + 2 + 1) \times 2^{\text{cif_qcif}}$$

Using the above transformations, equation 4.1 can be re-written in hardware terms as

$$\begin{aligned} \text{PixAddr} = & (((\text{rr} \ll \text{mlog}(\text{brow})) | \text{iir}) + \\ & (((\text{rr} \ll \text{mlog}(\text{brow})) | \text{iir}) \ll 1) + (((\text{rr} \ll \text{mlog}(\text{brow})) | \text{iir}) \ll 3)) \ll \text{cif_qcif}) + \\ & ((\text{cc} \ll \text{mlog}(\text{bcol})) | \text{iic}) \text{ ----- Equation 4.2} \end{aligned}$$

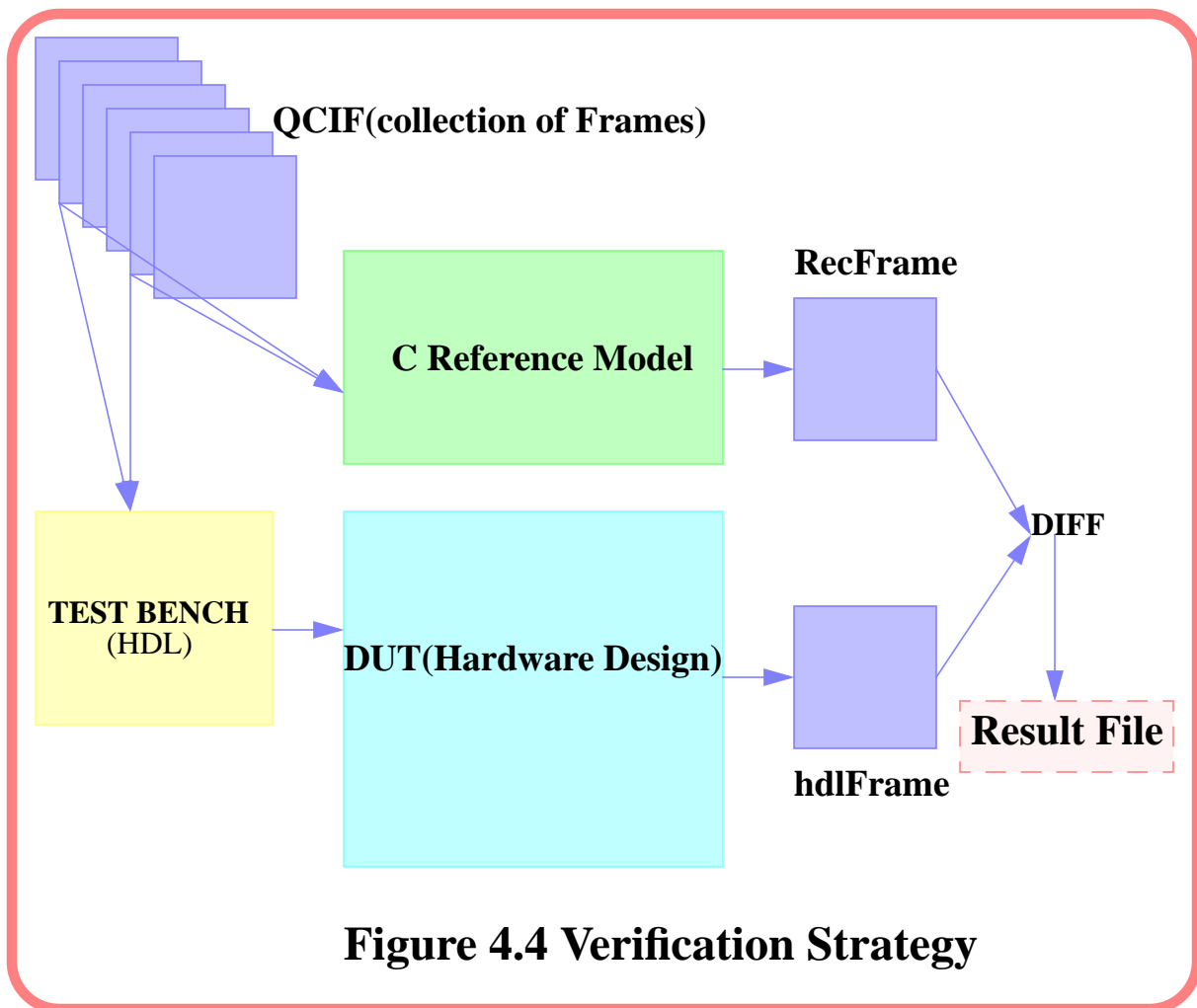
where ‘mlog’ is a function written to convert $x=4,8,16$ or 32 to corresponding $\log_2(x)$ values, using a simple look up table.

This hardware equation uses only adders/shifters/logical operators to give the final pixel address.

The advantage here is that there are no ‘multipliers’ required in the design.

4.6 Verification

Since C-Models already exists, which serve as a reference to design Hardware, these models will also be exploited to verify Hardware results. The following diagram gives a diagramatic representation of verification strategy



Named reference frame I and current frame B are extracted from a named qcif file, and motion estimation is performed over them to get motion vectors and reconstructed frame P. All three frames i.e. I, B and P are also saved as ascii pgm files, to make them human readable, and also to make them readable by any image viewer.

The C reference program output is called 'RecFrame.pgm', while a copy of current frame is kept safe and is named 'cFrame.pgm'. The C-Model run also produces PSNR value for the current run.

A second C program is then run, to extract the same frames as above, to convert them into verilog readable format. RTL/Netlist simulation is then run, which produces motion vectors and reconstructed frame. This reconstructed frame is dumped by the verilog code into a text file, and some unix/C formatting is done to convert it in an ascii pgm file. This file is named as 'hdl.pgm'. A unix diff is performed between 'hdl.pgm' and 'RecFrame.pgm'. This diff should return a null. If it does return something, it would mean that there are problems in HDL output, and there exists a need for debugging the HDL design.

Since 'hdl.pgm' is a reconstructed frame, it can only be correct, if the motion vectors are correct. Because reconstructed frame is a function of reference frame and motion vectors.

Since the test data comes from text file, and output data is dumped into a text file, the design/code of test bench is very simple. The test bench uses verilog functions '*readmemh*' and '*writememh*' to read and dump text files. The read data is then injected into the design on each clock edge. A signal '*start*' is generated by the test bench, and must be asserted, to keep the design working. If '*start*' is deserted, the design is stalled. Ideally '*start*' once asserted should only be de-asserted when the design asserts '*finish*', but if the need be '*start*' can also be deserted any time during the frame processing. The design is made to co-op up with such situations. The design 'stalls' as soon as '*start*' is deserted, and resumes from exactly the same point when '*start*' is asserted again.

Provision is made so that any named frames from any named video sequence can be used on the fly for verification purpose. The whole verification procedure is highly automated using unix/c/perl. A single top level script is capable of running through the whole verification flow automatically. Section 5 'Project Automation' describes in detail about automation and how to run automatic verification. The script takes the name and path of video file as input along with the named frames on the command line itself.

Results from Simulation:

A simulation is run on 'foreman.qcif' with frame numbers 45 46, 45 being the reference frame, and 46 being the current frame. Figure 4.5 (a) is the current frame i.e. frame 46 as such. Figure 4.5 (b) is the reconstructed frame by C model, Figure 4.5 (c) is the reconstructed frame by hardware design. The PSNR for this run happens to be 35.709333



*Figure 4.5 (a) Current Frame to be
Motion Estimated
cFrame.pgm*



*Figure 4.5 (b) Reconstructed Frame using
Motion motion vectors. C Model Output
RecFrame.pgm*



*Figure 4.5 (c) Reconstructed Frame using
Motion motion vectors. Hardware Output
hdl.pgm*

To visually appreciate the degradation in PSNR, a simulation is run with a frame distance of 10. i.e. reference frame is 45, current frame is 55. Figure 4.6 (a) (b) and (c) shows the results. The PSNR happens to be 24.566316. Note the poor visual quality of reconstructed frames.



*Figure 4.6 (a) Current Frame to be Motion Estimated
cFrame.pgm*



Figure 4.5 (b) Reconstructed Frame using Motion motion vectors. C Model Output



*Figure 4.5 (c) Reconstructed Frame using Motion motion vectors. Hardware Output
hdl.pgm*

4.7 Synthesis

Synopsys' Design Compiler is used for synthesis. The target library used is UMC on 0.18u process technology. Again the synthesis is automatic and is complemented by automatic netlist simulation. Following are the results from synthesis, RTL is written has highly parameterised module and can easily be mapped to the target library of choice.

Results from Synthesis:

Operating Frequency 20 Mhz for 8 bit memory I/F

Area : 74930.640625 lib units for 8 bit memory I/F

4.8 Power Analysis:

Synopsys' Design Power is used to perform power analysis at netlist level.

The inputs to the tool are

1. Netlist of the Design
2. Technology Library to which the design has been mapped to.
3. Voltage at which the design is intended to work
4. Capacitance
5. Frequency at which power analysis will be performed
6. Switching Activity Factor

Out of these inputs, the only difficult one is (6) the switching activity factor. Its very easy with Synopsys' Design Power tool when used in conjunction with Synopsys's VSS simulation tool to get the switching activity factor, perhaps because both tools are offered by the same vendor. But with ModelSim as HDL simulator it is not straight forward.

ModelSim has the capability of creating a toggle file, which is essentially a measure of number of 0->1 and 1-> 0 transitions a net/signal would do in a simulation. This switching activity then can be converted into activity factor for the net/signal when expressed as a percentage of number of clock edges occurred in the simulation time.

The Synopsys' 'set_switching_activity' command can be used, once for each and every net in the design, a switching activity is known, and hence the power will be calculated for the whole design.

Each simulation gives slightly different power value, with the average power consumed by the design being 4mW.

SECTION 5 Design Automation

Due to the nature of this project, huge amounts of data generation, collection and processing is required. The project is based upon analytical analysis done in Section 3 of this report. Analytical analysis results are reproduced in this report in the form of graphs which summarises the results, and also makes the reader understand how and why and what decisions were taken based on that. Without automation, it will take weeks/months to collect the data and put it in a concise manner. Hence there is a inevitable need of automation.

Some of the important scripts are described below, code listings of the same can be found in Appendix.

Automation Script 1: A unix script '*top.scr*' was written to re cursively run C Models to calculate PSNR values from 10 qcif files, 100 frames taken from each of them, with frame distance of 1,2,3,4 and 5.

This script was used to run both C models, i.e ES C Model and MDS C Model, to collect respective data.

The output of this script was a file named `psnr.rep.$brow.$bcol.$rscope.$scope`, where

`$brow`: a variable which can be modified to modify the block row size

`$bcol`: a variable which can be modified to modify the block column size

`$rscope`: a variable which can be modified to modify the search window row size

`$scope`: a variable which can be modified to modify the search window column size

The output of these scripts were processed by unix again to collect PNSR values and write a matlab 'm' file to automatically plot its graph vs the frame number. This 'm' file was run in 'matlab' to draw the required graph, which was saved as a JPEG file.

Automation Script 2: A unix script '*compare_psnr.csh*' was written to compare PSNR values collected from '*top.scr*' corresponding to ES and MDS. The PSNR comparison for done for each frame distance 1,2,3,4 and 5. Results from this were again converted into an 'm' file, which was read in 'matlab' to plot a graph. The resulting graphs are shown in Section 3 'Analytical Analysis' of this report.

Automation Script 3: A perl script '*vlogtb.pl*' was written to automatically generate a template test bench corresponding to a given verilog RTL/Netlist Code. This was used to generate testbenches for every module in the Project.

Automation Script 4: A tcl script '*dc.tcl*' was written which automated the synthesis of Design using UMC 0.18u technology Library. It is responsible for applying constraints to the block, synthesize the block and report area, timing and power. (with switching activity factor of 1)

Automation Script 5: A unix script '*make_netlist.csh*' was written to automate the netlist generation. It invokes Synopsys' design compiler in tcl mode and executes '*dc.tcl*' to produce netlist.

Automation Script 6: A unix script '*prj.csh*' was written to automatically run simulations either using RTL or Netlist, in gui or no gui mode. This script is the main script of the project. It is responsible for the following

1. Compiling the reference C models
2. Running the C models to generate reference output, in the form of ascii pgm file.
3. Reporting the current PSNR value
4. Generate input stimulus files readable by verilog testbench,
5. Compiling the whole HDL design, and required library verilog models
6. Running HDL simulation both RTL and Netlist as required
7. Dumping the HDL produced frame into a file and converting it into ascii pgm format
8. Comparing the results of HDL and C model.
9. Producing a toggle file from ModelSim, which may be converted into Synopsys' activity file for true power analysis at netlist level.
10. Starting image viewer program to display, Current Frame, Reconstructed C model Frame, Reconstructed HDL frame.

The ultimate beauty of this script is that it is capable of generating a new ordered stimulus for every simulation if required. It takes qcif file path/name and 2 numbers, which corresponds to reference frame number and current frame number respectively, as input arguments on the command line, and then first runs a C model on it, to produce fresh reference data for HDL, following which it generates corresponding stimulus file for the HDL simulation, and as the HDL simulation gets over, it compares the results. It has optional switches which can be given on command line to turn on/off gui mode and also for selecting RTL or Netlist mode simulations.

More about how to use the scripts is given in Appendix A

Automation Script 7: A unix script '*pwr.csh*' was written to translate the toggle file generated by ModelSim into a Synopsys' readable switching activity file.

To demonstrate the complexity of unix commands used in above script, a few of those commands are quoted below:

```
foreach i ( `grep mes net_list | grep -v Design | awk '{print $2}'
| grep -v ^$ | uniq | sed -e "s/\[/#s/g" -e "s/\]/#c/g" -e "s/\*/
#star#/g" -e "s///#l/g" ` )
    grep $i toggle.rpt | awk '{print $1" " ($2+$3)/700000}' | sed
"s/#lmes_tb#lmes#l//g" >> pwr1
end
```

SECTION 6 Conclusion:

The project successfully implemented a low power architecture for Motion Estimation block. Detailed analysis was performed using Exhaustive Search and Diamond Search algorithm, results were collected and used in various decisions in Hardware Design. Analytical analysis showed that the average number of candidate blocks searched in ES was 461, as compared to 30 in 4 iteration Modified Diamond Search. Experiments showed while DS was able to reduce the number of candidate blocks to be searched by a factor of 15-20, it did introduced noise, which in some cases was pulling down the PSNR of reconstructed image close to 30dB cutoff mark. Diamond Search Algorithm was then modified to increase the PSNR by up to 2dB, on files with high frequency areas. It was also concluded that ES may allow the same reference frame to be used to inter code 4-5 future frames, without dropping the PSNR below the acceptable limit of 30dB. But MDS only allowed 2-3 future frames to be inter coded. This was one of the drawbacks of using MDS other than a slight decrease in PSNR. the average drop in PSNR as the result of using MDS was found to be 1.5 dB. Without modifying the DS, and using DS as such gave this figure to be 3.5dB. An statistical estimate also showed that the low power techniques implemented in hardware to reduce the switching activity saved as much as 30% power with a design which would not implement any switching activity control scheme.

C reference models were build for ES and MDS, and results from MDS C model were used to verify hardware. The design was mapped onto 0.18u technology from UMC foundry. Power analysis was then performed, for a variety of frame types, and the power consumption was found to be 4mW. The gate count was extremely low and was less than 3K gates.

The most difficult part of the project was to collect, summarise and use huge amounts of data generated by experiments which were done on 10 standard video sequences, 100 frames for each video sequence, and Motion estimation performed using frame distance of 1,2,3,4 and 5. Experiment results gave PSNR value for the following variables for ES, DS and MDS

1. Block Size
2. Search Window size
3. Number of iterations in diamond algorithm
4. Frame Distance
5. Cost function. both MSE and SAD were analysed.

These variables resulted in results file which were greater than a GB in some cases. There exists a huge challenge to process this data to derive useful conclusions. There was also a challenge how to convert this data into graphs, which can easily summarise the findings and will give the reader a chance to understand those findings. These challenges were overcome by automation scripts, which are described in SECTION 5.

SECTION 7 Critical Evaluation/Recommendations/Future Work

While the design is capable of performing what it was supposed to perform, it lacks features, which are present in almost all Motion Estimation designs done in academia/Industry. Features like sub-pel motion estimation, dynamic modification on block size depending upon the frame frequency contents are very common, and are recommended to be implemented should this work is carried further.

The analytical analysis has been done using a single fast algorithm. It does not analyse a popular algorithm called HEXBS, which can further reduce the number of candidate blocks to be searched.

Following are some recommendations for future work if any is to be carried out.

1. Implementation of 1/2 pel and 1/4 pel motion estimation along with integer motion estimation
2. Analysis of HEXBS
3. Statistical analysis of the value of motion vectors. During this project, it was found that a lot of blocks have exactly same motion vectors. Especially the ones which are close to each other. Hence a statistical analysis may be performed to find out the probability that a block will have the same motion vector as its neighbouring block. if it is found to be too high, then the search will start from the candidate block obtained by applying the same motion vectors as were found for the neighbouring block. If the 'cost' of this search is comparable to the already calculated min. cost of the neighbouring block, then motion estimation should stop and return the same motion vectors as its neighbouring block has. This is a very serious recommendation, and proper investigation may result in massive reduction in average number of candidate blocks to be searched, and may be worth filing a patent for. Due to the limited time scales, it was not possible to conduct this research in this project.

SECTION 8. References:

- [1] Zhihai He, Yongfang Liang, Lulin Chen, Ishfaq Ahmad, and Dapeng Wu, *Power-Rate-Distortion Analysis for Wireless Video Communication Under Energy Constraints*.
IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY,
VOL. 15, NO. 5, MAY 2005
- [2] Ang, P.H.; Ruetz, P.A.; Auld, D. *Video compression makes big gains*
Spectrum, IEEE Volume 28, Issue 10, Oct. 1991 Page(s):16 - 19
- [3] Iain E G Richardson, H.264 / MPEG-4 Part 10 *White Paper. Prediction of Inter Macroblocks in P-slices*.<http://www.vcodex.com>
- [4] ISO/IEC 15444-1 *Information technology -- JPEG 2000 image coding system: Core coding system*
- [5] ISO/IEC 14496-10 *Information technology -- Coding of audio-visual objects -- Part 10: Advanced Video Coding*
- [6] Yang, S.; Wolf, W.; Vijaykrishnan, N.; *Power and performance analysis of motion estimation based on hardware and software realizations*
Computers, IEEE Transactions on Volume 54, Issue 6, Jun 2005 Page(s):714 - 726
- [7] Y. Kuroda, J. Miyakoshi, M. Miyama, K. Imamura, H. Hashimoto, M. Yoshimoto, Kana zawa University, Japan .*A Sub-mW MPEG-4 Motion Estimation Processor Core for Mobile Video Application*
Asia and South Pacific Design Automation Conference 2004 (ASP-DAC'04) pp. 527-528
- [8] Swee Yeow Yap and John V. McCanny, Fellow, IEEE, *A VLSI Architecture for Variable Block Size Video Motion Estimation*
IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMSII: EXPRESS BRIEFS, VOL. 51, NO. 7, JULY 2004
- [9] Aroh Barjatya, Student Member, IEEE; *Block Matching Algorithms For Motion Estimation*
Utah State University DIP 6620 Spring 2004 Final Project Paper
- [10] Jun-Fu Shen, Tu-Chih Wang, and Liang-Gee Chen; *A Novel Low-Power Full-Search Block-Matching Motion-Estimation Design for H.263+*
IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY,
VOL. 11, NO. 7, JULY 2001

- [11] Mohammed Sayed and Wael Badawy; *A HALF-PEL MOTION ESTIMATION ARCHITECTURE FOR MPEG-4 APPLICATIONS*
Microelectronics, The 14th International Conference on 2002 - ICM 11-13 Dec. 2002
Page(s):24 - 27
- [12] Fanucci, L.; Bertini, L.; Saponara, S.; *Programmable and Low Power VLSI Architectures for Full Search Motion Estimation in Multimedia Communications*
Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on Volume 3, 30 July-2 Aug. 2000 Page(s):1395 - 1398 vol.3
- [13] Hongjun Jia; Li Zhang; *Directional diamond search pattern for fast block motion estimation*
Electronics Letters Volume 39, Issue 22, 30 Oct. 2003 Page(s):1581 - 1583
- [14] Ce Zhu; Xiao Lin; Lap-Pui Chau; *Hexagon-based search pattern for fast block motion estimation*
Circuits and Systems for Video Technology, IEEE Transactions on Volume 12, Issue 5, May 2002 Page(s):349 - 355
Digital Object Identifier 10.1109/TCSVT.2002.1003474
- [15] Ce Zhu; Xiao Lin; Lap-Pui Chau; Keng-Pang Lim; Hock-Ann Ang; Choo-Yin Ong; *A novel hexagon-based search algorithm for fast block motion estimation*
Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference on Volume 3, 7-11 May 2001 Page(s):1593 - 1596 vol.3
Digital Object Identifier 10.1109/ICASSP.2001.941239
- [16] Xiaoquan Yi; Nam Ling; *Rapid block-matching motion estimation using modified diamond search algorithm* *Circuits and Systems*, 2005. ISCAS 2005. IEEE International Symposium on 23-26 May 2005 Page(s):5489 - 5492 Vol. 6
Digital Object Identifier 10.1109/ISCAS.2005.1465879
- [17] Shan Zhu; Kai-Kuang Ma; *A new diamond search algorithm for fast block-matching motion estimation*
Image Processing, IEEE Transactions on Volume 9, Issue 2, Feb. 2000 Page(s):287 - 290
Digital Object Identifier 10.1109/83.821744
- [18] Prof Tughrul Arslan; *Sources of Power Consumption*
University of Edinburgh. http://www.see.ed.ac.uk/~SLIg/low_pwr_introduction.html

- [19] N. Kim, T. Austin, T. Mudge, and D. Grunwald. *Challenges for Architectural Level Power Modeling. In Power Aware Computing*, Kluwer Academic Publishers, Norwell, MA, 2002
- [20] Aburdene, M.F.; Jianqing Zheng; Kozick, R.J.; *Computation of discrete cosine transform using Clenshaw's recurrence formula*
Signal Processing Letters, IEEE Volume 2, Issue 8, Aug. 1995 Page(s):155 - 156
Digital Object Identifier 10.1109/97.404131
- [21] Cavarro-Menard, C.; Le Duff, A.; Balzer, P.; Denizot, B.; Morel, O.; Jallet, P.; Le Jeune, J.-J.; *Quality assessment of compressed cardiac MRI. Effect of lossy compression on computerized physiological parameters*
Image Analysis and Processing, 1999. Proceedings. International Conference on 27-29 Sept. 1999 Page(s):1034 - 1037
- [22] Hoi-Ming Wong; Au, O.C.; Chang, A.; *Fast sub-pixel inter-prediction - based on texture direction analysis (FSIP-BTDA) [video coding applications]*
Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on 23-26 May 2005 Page(s):5477 - 5480 Vol. 6
Digital Object Identifier 10.1109/ISCAS.2005.1465876
- [23] Nicolaos B. Karayiannis and Yiding Li; *A Replenishment Technique for Low Bit-Rate Video Compression Based on Wavelets and Vector Quantization*
IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 11, NO. 5, MAY 2001
- [24] Michael Searles; *Probe Based Dynamic Server Selection for Multimedia Quality of Service*
Master of Computer Science thesis National University of Ireland, Dublin Department of Computer Science Feb 2004

SECTION 9 APPENDIX

Appendix A How to use/run the project.

This is a guide on how to use the automated flow developed as a part of this project. The scripts written are not hardened against unexpected inputs, and any attempt to use the script in a way other than given in this section may produce funny looking results. There is a ‘gotchas’ section where common known problems and solutions are stated. **Any script/C program may just be executed without any arguments, to get a help message, if no help message is returned, then this script/C program is not be used on its own, and is a part of other main scripts/C program.**

For example, the main project script i.e ‘prj.csh’ when typed without any args returns the following message

```
#####
ERROR (prj.csh) :Invalid number of arguments;At least 3 expected
USAGE: unix> prj.csh <qcif_file> <st_fr> <end_fr> [gui/net]
where stfr is the ref Frame No. and end_fr is Cur Frame No.
where option gui/net is optional, and will stat gui or net sims
Example: unix> prj.csh ../video/carphone.qcif 34 36
#####
```

The arguments in diamond braces are mandatory, where as the arguments in square brackets are optional.

Following are the functions that may be done using the scripts. It is recommended that the order in which they are written, should be the order in which they must be used.

1. Run RTL simulation on the project with any video sequence of choice, selecting any 2 frames from the selected video files
2. Synthesize the design
3. Run netlist Simulation on the design
4. Annotate switching activity from the design, to perform power analysis.
5. It is also possible to run just the C reference model as well, stand alone to do analysis if any required.

To be able to do one or more selected tasks from the above list, it is important that the reader understands the directory structure.

The CD-ROM included with this project report contains a single tar gzipped file:

MscPrj.tar.gz with following main directories.

- work
- delivery

Copy the file MscPrjV2.tar.gz from CD-ROM to the work area.

Unzip and untar it:

```
gzip -d MscPrjV2.tar.gz
```

```
tar -xvf MscPrjV2.tar
```

This will create the required directory tree.

Directory 'work' has been included just to demonstrate the amount and kind of development work which has been done as a part of this project. This is an undocumented directory, and if the reader wants to make use of it, its not very easy.

Directory named 'delivery' as the name suggests, contains the deliverables for this project work. It is very well documented and it is expected that the reader will be able to exploit it as per his requirements.

delivery contains the following sub-directories

scripts: contains scripts to run the project.

rtl: contains all rtl source for the design

tb: contains testbench and verilog memory models

sim: this directory is used for ModelSim RTL as well as Netlist simulations.

netlist: contains the final netlist for the design

synopsys: Synopsys' Design Compiler runs in this directory and performs synthesis

c_src: contains all c source files used by the project.

bin: contains all compiled c executables used by the project.

video: contains video sequences

1. Running RTL simulations:

1 (a) Go inside <base_dir>/scripts directory

cd scripts

1 (b) edit the file '*source_file*' for Solaris, or '*source_file_linux*' for linux. This file must contain a way/command to set paths for

ModelSim

Synopsys' Design Compiler

1 (c) run RTL simulation by using the following command

./prj.csh <video_file> <reference_frame_no> <current_frame_number> [gui/net]

Examples:

./prj.csh ../video/carphone.qcif 27 28

./prj.csh ../video/carphone.qcif 27 28 gui

each of them will produce the following files as results in 'results' directory

cFrame.pgm an ascii image file, which represents the current frame which is desired to be motion estimated, i.e frame 28 in above examples. This will be generated on the fly by C reference model following the issuance of command shown above.

RecFrame.pgm: an ascii image file, which represents the reconstructed frame generated by the reference C program, which is constructed my motion vectors and the reference frame to be used to motion estimate the current frame i.e. frame number 27 in above examples. Again this is generated by the C program on the fly after issuance of the command.

hdl.pgm: an ascii image file generated by RTL simulations. This should exactly match RecFrame.pgm, which is the output generated my C reference model. A simple unix diff between Recframe.pgm and hdl.pgm would ensure if the RTL simulation has been successful.

At the end of simulations, 3 images, i.e cFrame.pgm, RecFrame.pgm and hdl.pgm should pop up, if the image viewing program is set up correctly on the system. The project uses 'xv' for Solaris and 'xview' for Linux.

2. Running Synthesis:

Go in <base_dir>/delivery/synopsys directory.

Edit the '*.synopsys_dc.setup*' file to include the target technology file and path.

Go in <base_dir>/delivery/scripts directory.

Run the following command:

```
./make_netlist.csh
```

this will generate the desired netlist in <base_dir>/delivery/synopsys directory and a copy of this in <base_dir>/delivery/netlist directory.

It will also produce following reports in <base_dir>/delivery/synopsys directory

area.rpt: a report on area

time.rpt: a report on timing

pwr.rpt: a report on power. Note that at this point of time, the switching activity is set to '1' for all nets in the netlist. So it is a rough estimate of power at this point of time.

3. Run Netlist Simulation.(Gate Level simulations)

Go in the <base_dir>/delivery/sim directory.

Edit the file '*run_net.scr*' to point to correct verilog library models corresponding to your technology.

Go in the <base_dir>/delivery/scripts directory.

Now run the scripts '*prj.csh*' with '*net*' option.

```
./prj.csh ../video/carphone.qcif 45 46 net
```

This will run netlist simulations and the results files produced will be identical to the results files described in RTL run procedure.

4. Power Analysis:

Go in the <base_dir>/delivery/scripts directory.

Run the following command to annotate the switching activity from netlist simulation run to Synopsys' Design Power to perform accurate netlist level power analysis.

```
./pwr.csh
```

This will write a power report file *pwr_accurate.rpt* in <base_dir>/delivery/synopsys directory. It might be interesting to compare rustles from *pwr.rpt*(which puts the switching activity of 1 on every net) and *pwr_accurate.rpt*(which annotates the correct switching activity on each net)

5. C model run: The directory <base_dir>/delivery/c_src has the reference C models.

Go to the directory: compile the file mes.c using the following command:

```
gcc mes.c -lm -o ../bin/mes
```

go to the <base_dir>/delivery/bin directory and run the C model as

```
./mes <video file> <n1-n2>
```

Example:

```
./mes ../video/carphone.qcif 57 59.
```

This will produce cFrame.pgm, the current frame, 'RecFrame.pgm' the motion estimated frame, out.dat, a file containing the motion vectors, in the same directory as you are running your C program and will also give a PSNR number.

Gotchas:

1). If the resulting pgm file from hdl i.e 'hdl.pgm' file is black, and does not have a viewable image file:

Solution: Look if ModelSim version being used supports the function 'writememh'. If it does not support it, use the one which supports it.

2). The way in which the start frame number and end frame number is given to C program and to prj.csh differ

To C program 'mes.c'

```
mes ../video/carphone.qcif 23-24
```

To prj.csh

```
prj.csh ../video/caprhone.qcif 23 24
```

Notice in C program a '-' is used between frame numbers whereas in prj.csh a space character is required.

3). prj.csh does not accept absolute paths of video files. i.e it only works on relative paths

Example:

```
prj.csh ../video/carphone.qcif 2 5 will work
```

whereas

```
prj.csh /homes/vmittal/MscPrj/delivery/video/carphone.qcif 25 will NOT work.
```

Appendix B: Listing of *prj.csh*

```
#!/bin/csh -f
set title = `basename $0`;
#####
# FOR ARGUMENTS
#####
if ($#argv < 3) then
    set error = "Invalid number of arguments;At least 3 expected"
    echo "#####"
    echo "ERROR ($title) :"$error;
    echo "USAGE: unix> prj.csh <qcif_file> <st_fr> <end_fr> [gui] "
    echo "where stfr is the ref Frame No. and end_fr is Cur Frame No."
    echo "Example: unix> prj.csh ../video/carphone.qcif 34 36"
    echo "#####"
    exit
endif

set C_SRC = "../c_src"
set BIN = "../bin"
set SIM = "../sim"
set VIDEO_SRC = "../video"
set qcif_file = $argv[1]
set RTL = ../rtl
set TB = ../tb
set RESULTS = ../results
set SYN = ../synopsys
set SCRIPTS = ../scripts
set NETLIST = ../netlist
set POWER = ../power

rm -rf $SIM/mes.log
rm -rf $SIM/qcif2hdlinp.log

# start frame, servers as reference frame
set st_fr = $argv[2]
# end frame, servers as current frame to be encoded
```

```

set end_fr = $argv[3]
if(`uname` == "Linux") then
    source $SCRIPTS/source_file
else
    source $SCRIPTS/source_file_solaris
endif

#####
# Check and compile c file to convert qcif/cif file to rtl readable format
#####
if(-r $C_SRC/qcif2hdlinp.c) then
    gcc $C_SRC/qcif2hdlinp.c -lm -o $BIN/qcif2hdlinp
else
    echo "ERROR! Source File qcif2hdlinp.c Not found in #$C_SRC# ....Exiting"
    exit(0)
endif

#####

#####
# Check and compile c file to which performs ME on a givne qcif/cif file
#####
if(-r $C_SRC/mes.c) then
    gcc $C_SRC/mes.c -lm -o $BIN/mes
else
    echo "ERROR! Source File mes.c Not found in #$C_SRC# ....Exiting"
    exit(0)
endif

#####

#####
# Run C program mes to perform motion estimation on a qcif/cif file
# To generate reference motion vectors to be used for RTL verification
#####
if(-r $VIDEO_SRC/$qcif_file) then
else
    echo "ERROR! # $qcif_file # Not found in #$VIDEO_SRC# ....Exiting"
    exit(0)

```

```

endif

if(-r $BIN/mes) then
    $BIN/mes $VIDEO_SRC/$qcif_file $st_fr-$end_fr
    echo "Running C Program for Motion Estimaion ....."
    mv ./out.dat $SIM
else
    echo "ERROR! #mes# Executable NOT found #$BIN# ....Exiting"
    exit(0)
endif

#####

#####
# Run C program to generate input data for RTL Simulation
# This is effectivly to produce 2 frames froma qcif file
#####

if(-r $BIN/qcif2hdlinp) then
    $BIN/qcif2hdlinp $qcif_file $st_fr-$end_fr > $SIM/qcif2hdlinp.log
    echo "Generating Simulation Data ....."
    mv cFrame.pgm $SIM
    mv rFrame.pgm $SIM
else
    echo "ERROR! #qcif2hdlinp# Executable NOT found #$BIN# ....Exiting"
    exit(0)
endif

#####

#####
# Run HDL SiM
#####

if(-r $RTL/mes.v && -r $TB/mes_tb.v) then
    echo "Runnign Simulation ....."
    rm -rf $SIM/work
    pushd $SIM
    vlib work

```

```

vlog $RTL/mes.v $TB/mes_tb.v $TB/RamRef.v $TB/RamCur.v $RTL/fifo_lp.v $TB/
RamRec.v
if($#argv > 3) then
    if($argv[4] == "gui") then
        vsim -do "add wave sim:/mes_tb/mes/*" mes_tb
    else if($argv[4] == "net") then
        echo "NETLIST SIMULATION"
        source $SCRIPTS/run_net.scr
    else
        vsim -c -do "run 7 ms; force -freeze sim:/mes_tb/R64X32M4_Rec/dump 1 0;
run 100ns; quit" mes_tb
    endif
else
    vsim -c -do "run 7 ms; force -freeze sim:/mes_tb/R64X32M4_Rec/dump 1 0;
run 100ns; quit" mes_tb
    endif
popd
else
    echo "ERROR! #Missing mse.v or mse_tb.v# in #$RTL $TB res#....Exiting"
    exit(0)
endif

#####

rm -rf $SIM/hdl_sad $SIM/c_sad sad_diff
pushd $SIM
grep MinSAD transcript | awk '{print $3}' > hdl_sad
grep minMSE out.dat | awk '{print $4}' > c_sad
diff hdl_sad c_sad > $RESULTS/sad_diff
popd

echo
"#####"
echo "# RTL vs C results are in $RESULTS/sad_diff"
echo
"#####"

```

```

rm -rf $RESULTS/hdl.pgm $RESULTS/temp $SIM/diff
sed -n "4,$ p" $SIM/RecFrame.pgm | sed "s/z//g" > $RESULTS/temp
gcc $C_SRC/hex2int.c -o $BIN/hex2int
$BIN/hex2int > $RESULTS/hdl.pgm
gcc $C_SRC/cal_sad_psnr.c -o $BIN/sad_psnr -lm
$BIN/sad_psnr $VIDEO_SRC/$qcif_file $st_fr-$end_fr
mv *.pgm $RESULTS/
mv out.dat $RESULTS/
diff $RESULTS/hdl.pgm $RESULTS/RecFrame.pgm > $RESULTS/recFr_diff

echo
"#####"
echo "# RTL vs C results are in $RESULTS/recFr_diff"
echo
"#####"

cp $SYN/netlist.v $NETLIST
cp $SYN/power.rep $POWER

set uname = `uname`

if($uname == "Linux") then
    xview $RESULTS/hdl.pgm &
    xview $RESULTS/RecFrame.pgm &
    xview $RESULTS/cFrame.pgm &
else
    xv $RESULTS/hdl.pgm &
    xv $RESULTS/RecFrame.pgm &
    xv $RESULTS/cFrame.pgm &
endif

```