LUNDS UNIVERSITET

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Master Thesis Project
Winter 2006

MPEG-4 Low Delay Design for HDTV with Multi-stream Approach

Guided by
Dr. Marco Mattavelli
EPFL/STI-ITS-LTS-3
Dr. Renzo Posega
Live Tools Technology S.A

Submitted by
Harish Reddy Chinta Reddy
Rajesh Babu Chunduri
Department of Electroscience
Lund University

Swiss Federal Institute of Technology, Lausanne (EPFL)
Lausanne, 28 February 2007

EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE LAUSANNE
POLITECNICO FEDERALE DI LOSANA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY LAUSANNE

SCHOOL OF ENGINEERING – STI
MICROELECTRONIC SYSTEM LABORATORY (LSM)

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

EPFL-STI-LSM
ELB 140
Station 11
CH-1015 Lausanne
Site web : http://ltswww.epfl.ch

Phone : + 4121 693 6984
Fax : + 4121 693 4663
E-mail : mattavelli.marco@epfl.ch

PROJECT DE DIPLOMA, Lausanne, July 2006

Study and design of FPGA multistream MPEG video compression architectures for HDTV low delay applications

1. Administrative data

Students:	Harish Reddy Chinta Reddy Rajesh Babu Chunduri
Section/Semester:	Masters
Number of Months:	Master Thesis, Minimum 6 months
Supervision done by:	Dr. Marco Mattavelli, Dr. Renzo Posega

2. Project description

The design of HDTV video compression systems is a very challenging problem for the high throughput that each codec block has to support for the different progressive an interlaced high definition formats. The problem is that some elements of the MPEG processing such as parsing, VLD, and IDCT using ACDC prediction cannot be easily mapped on parallel implementations. One solution of the problem is to implement a multi-stream codec based on multiple parallel standard resolution codecs. Such solution presents also the advantage that appropriate buffer control strategies achieving very low end-to-end delay can be developed, while at the same time keeping efficient bit allocations in constant bitrate transmission.

The project consists of the following steps:

- Specifications of image partitioning and coding settings that do not present visually perceivable boundaries discontinuities for multi stream coding by using an existing C reference model of an MPEG-4 codec
- Development of appropriate buffer control strategies for multi-stream coding that maximize the visual quality by using the reference C MPEG-4 codec
- Definition of the buffer control strategies that maximize image quality and enable to achieve end-to end delays lower than 2 frame

3. Project development

The main points to achieve the multi-stream low delay solution with MPEG-4 can be grouped in six modules:

1. Actual software architecture study:
   - Study of the SP MPEG-4 software and understanding of the existing routines and functions.
   - Input/output data processing.
   - Statistics generation.
   - Buffer control method.
2. Video sequence splitting
   - Band scan.
   - Square scan.
   - Output sequences merging.
3. TS generation
   - Inner TS packing/de-packing functions.
   - Multi-stream multiplexing/de- multiplexing strategy.
4. Decoding process
   - Decoding compliant architecture.
5. Study of the minimum buffer size
   - OF (Over Flow) buffer detection.
   - Global buffer control strategy.
   - Simulations using different HD video sequences.
6. Statistics
   - Buffer fullness.
   - Quality parameter.
   - PSNR.
   - Impact of PIC splitting on coding efficiency.
   - Matlab curves

Abstract

The different MPEG standards are well known for their media compression capabilities. MPEG-4 evolved in effort by JVT to develop a codec that maximize quality and minimize bit rates. MPEG-4 Version2 Simple Profile which we choose contain several novel tools for the low-bitrate coding of audio and video. However, in typical conditions, the latency needed to compress/decompress a digital video signal at Standard Definition (SD) resolution is of the order of 15 frames, which corresponds to 0.5 sec. This delay would be more in case of HDTV because of its high resolution and bigger video size.

A way to solve this problem is to provide a low delay digital processing scheme based on a video-coding algorithm, which avoids massive intermediate data storage. The first effort of this work is to study the MPEG standard to understand its limit from both the coding delay and implementation complexity points of views. To remove the drift and other visible artifact in the encoded data an intra refresh was done. For implementation the single encoder is extended to reliable multi-stream encoders using forks and pipes for the exchange of critical HDTV content.

This thesis also investigates an alternative solution for buffer control model. A step-by-step implementation of the strategy is done which is simulated and is modified to achieve the required model. Then, the thesis proposes a design scheme for the basic hardware structure of the encoding and decoding system based on delay.

Key words: *digital, video, algorithm, MPEG, MPEG-4, codec, delay, latency, complexity, multi-stream, pipes and forks, global buffer, FPGA.*

ACKNOWLEDGEMENTS

We are very happy to be a part of LUND UNIVERSITY as a Masters Students in System-on-Chip (Electro Science Dept), Sweden. We would like to thank **Dr. Marco Mattavelli** for his guidance, encouragement and the complete help throughout our project. Without his encouragement and guidance, we would not have been able to learn so much. We are grateful to him for sparing his valuable time and giving us an opportunity to work under his guidance in LTS3 lab of EPFL, Switzerland. We would sincerely like to thank our internal supervisor **Mr. Peter Nilsson** for his constant vigil and encouragement during our course of study to successfully completing the project.

We would like to thank Mr. Renzo Posega and Mr. Charles Perdrix from Live Tools, Switzerland for their effort to put up the project in write direction with regular updates through weekly meetings. We would also like to thank Mr.Grazanio Varatto who has provided the required initial boost. Olivier and Serge took a grip on the work later and helped us a lot in supplying the required manuals and support. Mr. Ali also played his part in providing the required backup facilities, software support. We would be great finally in thanking the other Administrative faculty for their paper work and support during our Thesis work in EPFL without any disturbance during specified project period.

CERTIFICATE

This is to certify that **Rajesh Babu Chunduri** and **Harish Reddy Chinta Reddy**, pursuing M.Sc in Electro Science at Lund University, has worked on the project titled "Mpeg-4 Low Delay Design with Multi-stream Approach" in the LTS3 lab of Signal Processing Institute at **Ecole Polytechnique Federale de Lausanne (EPFL)** under the guidance of **Dr. Marco Mattavelli** from 21st July 2006 to 28th Feb 2007. They had completed their project work satisfactorily.

Signal Processing Laboratory,
EPFL, Lausanne, Switzerland.

Motivation and Goals

Motivation:

The day to day needs of man in Video conferencing and live streaming of videos and movies have been increasing drastically. So with the using of newer versions in encoding and decoding processes we can meet the demand of outside digital world. Previously they used to store data and then transmit it with some amount of fixed delay. Later with the many flexible ways of encoding they started to transmit the data with buffering delay of the live stream before they actually display. Now they are asking for live streaming with the increase of bandwidths to transmit the video with less delay during encoding, to achieve extremely low bit-rate video transmission.

This can be possible only when you select higher version and better strategies in implementing better compression of data with new ideas of encoding. So in this arena the MPEG group has released MPEG4 with much higher standard in coding technique than previous versions of coding. In this MPEG4 there was a big delay for live streaming and there would be a greater increase in hardware due to its higher buffer sizes. To over come this there are several ways which cause a great motivation for this project to start on. MPEG 4 codec is having a great demand in days to come because of the demand caused in the Hi-Tech world for the use of advanced cutting edge technologies for HD and DirecTV with more adaptive methods which make it more flexible in competitive markets outside.

Goal:

The implementation of the new ideas and structures has found their way into the normal course of life of people working in the Video Compression. As decided we had stepped into the project by giving a detailed reading about the compression and its standards. Then we had started to understand the code in the existing codec. Some simulation work is done to better understand how the code works and what the existing software is. Then we proceeded with the implementation of intra refresh and then on with the multi-stream approach. In the multi-stream we proceed with the splitting of a frame. As and when we completed the multi-stream approach we had to go on by designing a global buffer control strategy for designing the rate control. This rate control is implemented and analyzed in two different ways. The first simulation work is done to check how it works by collecting all the outputs of the encoders and in the second simulation work we had to take care of the buffer fullness and there by deciding and setting the new QP value for the next frame that should be encoded.

Contents

Chapter 1

1 MPEG - 4 Concepts

1.1 Introduction

As a century ago, it is virtually impossible to imagine what life must have been like for the average man. Where would telecommunications technology be, for example, if it were still as difficult to store information as it was, say, 50 years ago? So where does image coding fit into all this? Throughout history, pictures have always had a high-profile role to play in communication. In the case of television, it was quickly realized that communicating a video image was vastly more expensive in terms of necessary channel capacity than was speech or music transmission; and even from early days, methods of reducing this requirement were sought. Undoubtedly, this change has been fostered by the widespread supplanting of analogue by digital technology, for this has allowed us to do three things much more easily than before: (a) carry out signal-processing operations very rapidly (b) build very complex large-scale systems and (c) store data easily.

With the development of digital services, a growing problem soon appeared in that it was no longer a matter of dealing with a few highly specialized application areas, but rather a wide-ranging spectrum encompassing person-to-person (videophone) communication at one end through videoconference and conventional television to high-definition television at the other. Of course, supply and demand are inextricably linked in this field, as elsewhere. Given the capability of sending images and video efficiently, more people want to make use of the facility, and at the same time, they generate new application areas, and so on.

On the other hand, it is hard to escape the realization that, not with standing past successes, some fresh ideas are now necessary to impart a new direction to coding development and also free us from some of the problems which still bedevil what we have come to call conventional approaches. Again, the enormous bandwidth offered to us by optic fiber may remove the need for video-coding algorithms to be used in fixed service applications, leaving mobile systems as the major user of such techniques in an effort to conserve finite, and thus precious, spectrum space. There are thus many factors, not all of a purely technical nature, which will determine the future course of video-coding research. Two organizations dominate video compression standardization:

- ITU-T Video Coding Experts Group (VCEG)
  International Telecommunications Union – Telecommunications Standardization Sector (ITU-T, a United Nations Organization, formerly CCITT), Study Group 16

- ISO/IEC Moving Picture Experts Group (MPEG)
  International Standardization Organization and International Electro technical Commission, Joint Technical Committee Number 1, Subcommittee 29, Working Group 11

These two Organizations are formed into a single combined research team as the **JVT** (***Joint VideoTeam***) in December 2001 and have standardized the research in compression.

Video compression can be performed using either lossless or lossy compression methods. With lossless compression, each pixel is kept unchanged, resulting in an identical, bit-for-bit image after decompression. With lossy compression, the resulting video sequence will be different from the original, but good enough for use. In most cases, a lossy method can produce a much smaller compressed file than any lossless method, while still meeting the requirements of the application.

Video compression is gaining popularity since storage requirements and bandwidth requirements are reduced with compression. There are many algorithms for video compression each aimed at a different target. Thereafter, we consider the possibilities for new directions in video coding. In this Video compression era the most popular codec are MPEG-4 and H.264 of which we have chosen MPEG-4 version 2 (Simple Profile).

1.2 What is MPEG4?

MPEG-4 is one of the latest *(audio and video)* compression method standardized by MPEG group, designed specially for low-bandwidth *(less than 1.5MBit/sec bitrate)* video/audio encoding purposes.

It was designed to provide standardized ways to:

- Represent units of aural, visual or audio visual content, called "media objects". These media objects can be of natural or synthetic origin; this means they could be recorded with a camera or microphone, or generated with a computer;
- Describe the composition of these objects to create compound media objects that form audiovisual scenes;
- Multiplex and synchronize the data associated with media objects, so that they can be transported over network channels providing a QoS (Quality of Service) appropriate for the nature of the specific media objects; and
- Interact with the audiovisual scene generated at the receiver's end.

MPEG-4 technology was developed around the concept of providing a container (or 'wrapper') for the digital delivery of ALL audiovisual media. Audio and video were not the primary focus, but included in that larger concept. MPEG-4 supports coding of video objects with spatial and temporal scalability. Scalability allows decoding a part of a stream and construct images with reduced decoder complexity (reduced quality), reduced spatial resolution, reduced temporal resolution or with equal temporal and spatial resolution but reduced quality. Scalability is desired when video is sent over heterogeneous networks, or receiver can not display at full resolution (limited power) Robustness in error prone environments is an important issue for mobile communications. MPEG-4 has 3 groups of tools for this. Resynchronization tools enable the resynchronization of the bitstream and the decoder when an error has been detected. After synchronization data recovery tools are used to recover the lost data. These tools are techniques that encode the data in an error resilient way. Error concealment tools are used to conceal the lost data. Efficient resynchronization is the key to good data recovery and error concealment.

1.2.1 Tools, Objects and Profiles

MPEG-4 Visual provides its coding functions through a combination of *tools, objects* and *profiles*.

A "*tool"* is a subset of coding functions to support a specific feature. An "*object"* is a video element (e.g. a sequence of rectangular frames, a sequence of arbitrary-shaped regions, a still image) that is coded using one or more tools. To support the diversity of potential application spaces, MPEG-4 offers a variety of so-called "*profiles*": well-defined subsets from the toolbox, useful for specific applications.

1.3 MPEG-4 visual coding profiles

In natural video coding, the MPEG-4 market has converged on two important and the most commonly used profiles for natural video:

- **MPEG-4 Visual – Simple Profile (SP)** is designed primarily for low processing power coding, low latency and use in less-than-ideal transmission circumstances. Ideal for real-time desktop software encoding, mobile and wireless devices, video telephony and video–conferencing.
- **MPEG-4 visual - Advanced Simple Profile (ASP)** offers the best MPEG-4 coding performance and can be deployed into more demanding environments where video decoding at intermediate and higher bit rates is required. ASP is also ideal for broadcasters wishing to provide wide range of programming qualities suited to a variety of platforms, e.g. Broadcast, Internet and Mobile Phones.

Both profiles are already implemented in a wide range of software players, hardware, silicon, embedded systems and consumer devices: mobile phones, handheld computers, PDAs, personal video jukeboxes, and consumer digital still- and video cameras.

1.3.1 Benefits of profiles

- The scalable choice between MPEG-4 profiles supports the diversity of digital video applications, avoiding costly production of a number of different multimedia formats.
- Service providers can use broadband and narrowband platforms for the distribution of their content, such as wireless networks, digital television and the Internet.
- Profiles allow users to choose tool-sets supporting just the functionality they need.

1.4 Video CODEC

A video CODEC (Figure 1-1) encodes a source image or video sequence into a compressed form and decodes this to produce a copy or approximation of the source sequence. If the decoded video sequence is identical to the original, then the coding process is lossless; if the decoded sequence differs from the original, the process is lossy.

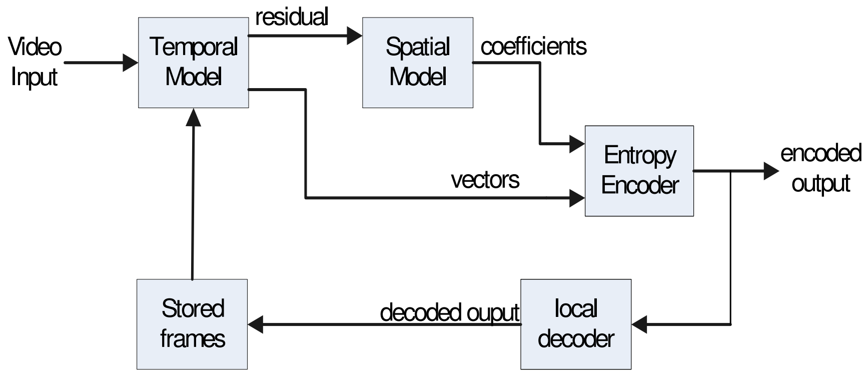


Figure 1-1 video encoding block diagram

The CODEC represents the original video sequence by a *model* (an efficient coded representation that can be used to reconstruct an approximation of the video data). Ideally, the model should represent the sequence using as few bits as possible and with as high a fidelity as possible. These two goals (compression efficiency and high quality) are usually conflicting, because a lower compressed bit rate typically produces reduced image quality at the decoder. This tradeoff between bit rate and quality (the rate-distortion trade off) is found in the following chapter.

A video encoder (Figure 1-1) consists of three main functional units: a *temporal model*, a *spatial model* and an *entropy encoder*. The input to the temporal model is an uncompressed video sequence. The temporal model attempts to reduce temporal redundancy by exploiting the similarities between neighboring video frames, usually by constructing a prediction of the current video frame. In MPEG-4 Visual and H.264, the prediction is formed from one or more previous or future frames and is improved by compensating for differences between the frames (motion compensated prediction). The output of the temporal model is a residual frame (created by subtracting the prediction from the actual current frame) and a set of model parameters, typically a set of motion vectors describing how the motion was compensated.

The residual frame forms the input to the spatial model which makes use of similarities between neighboring samples in the residual frame to reduce spatial redundancy. In MPEG-4 Visual it is achieved by applying a transform to the residual samples and quantizing the results. The transform converts the samples into another domain in which they are represented by transform coefficients. The coefficients are quantized to remove insignificant values, leaving a small number of significant coefficients that provide a more compact representation of the residual frame. The output of the spatial model is a set of quantized transform coefficients.

The parameters of the temporal model (typically motion vectors) and the spatial model (coefficients) are compressed by the entropy encoder. This removes statistical redundancy in the data (for example, representing commonly-occurring vectors and coefficients by short binary codes) and produces a compressed bit stream or file that may be transmitted and/or stored. A compressed sequence consists of coded motion vector parameters, coded residual coefficients and header information.

At the other ends where a video decoder reconstructs a video frame from the compressed bit stream which is the output of the encoder. The coefficients and motion vectors are decoded by an entropy decoder after which the spatial model is decoded to reconstruct a version of the residual frame. The decoder uses the motion vector parameters, together with one or more previously decoded frames, to create a prediction of the current frame and the frame itself is reconstructed by adding the residual frame to this prediction.

1.5 HD format and other types

1.5.1 YCbCr sampling formats

Most video standards use some form of *chroma subsampling*, in which the color information for a frame is sampled at a lower spatial resolution than the luma information. This is reasonable because the human eye itself is less sensitive to color than luminance. So bandwidth can be optimized by storing more luminance detail than color detail At normal viewing distances, there is no perceptible loss incurred by sampling the color detail at a lower rate.

On account of this, we get code words like "4:2:2" and "4:1:1" to describe how the subsampling is done. Roughly, the numbers refer to the ratios of the luma sampling frequency to the sampling frequencies of the two chroma channels (typically C_B and C_R, in digital video). I say "roughly" because this formula doesn't make any sense for things like "4:2:0". The common formats of subsampling are

4:4:4. :- Y, Cb, and Cr are sampled equally along each field or frame line with one Cb sample and one Cr sample for each Y sample. High-end video editing systems use 4:4:4, often together with uncompressed video.

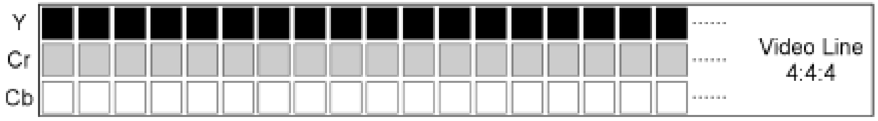


Figure 1-2 4:4:4 YCbCr Sampling format

4:2:2. :- Y is sampled at every pixel, but Cb and Cr color information is only sampled at every other pixel. While this sampling rate significantly reduces the bandwidth requirements, only a slight loss of color fidelity occurs. Many editing systems process this level of color information.

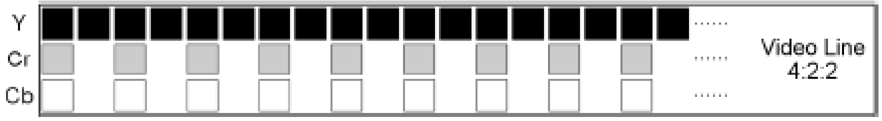


Figure 1-3 4:2:2 YCbCr Sampling

4:1:1. :-Y is sampled at every pixel, but the Cb and Cr color information is only sampled at every fourth pixel, saving even more bandwidth. This sampling rate is used in consumer DV cameras and is currently the default sampling rate for the interlaced mode.

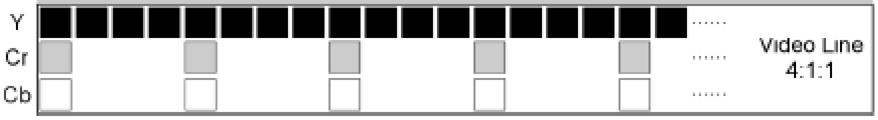


Figure 1-4 4:1:1 YCbCr Sampling format

4:2:0. :- This sampling rate uses a different technique known as a spatial sampling that takes a 2 x 2 square of pixels. 4:2:0 is used by default in the progressive mode of the codec's and by most HD tape formats.

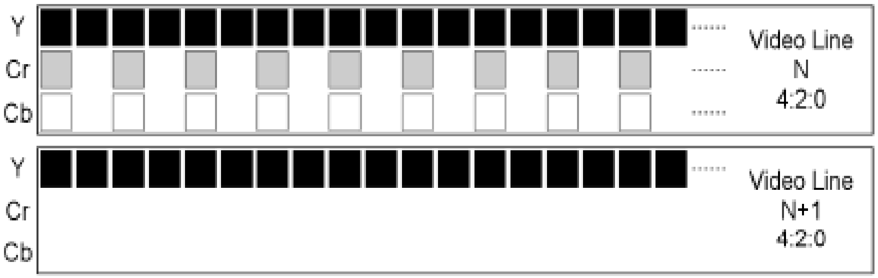


Figure 1-5 4:2:0 YCbCr Sampling format

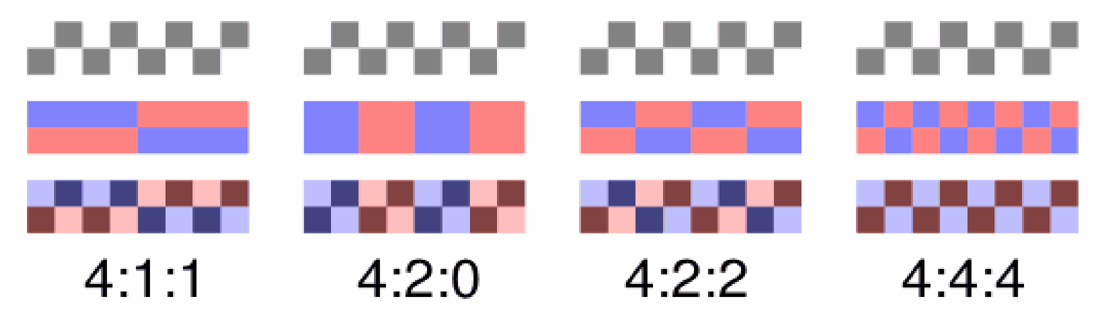


Figure 1-6 all formats together [2]

1.5.2 Video formats

Most of video compression standards support a wide variety of video frame formats. The first step is to capture the video using one of the *intermediate formats* then compression and transmission follow. The Common Interface Format (CIF) is the starting point for a common set of formats widely used in video coding applications. The choice of the frame resolution depends on the particular application and available storage or transmission capacity

- **HD** – It is a high resolution format suitable for HDTV and high quality applications. The active frame size is 1920 × 1080 pixels.

- **4CIF** - 4*times* Common Interface Format. It is a medium resolution format suitable for Standard Definition Television (SDTV) and DVD-Video applications. The active frame size is 704 × 576 pixels.

- **CIF** - Common Interface Format or Common Image Format. The Common Interface Format was developed to support video conference. It has an active resolution of 352× 288.
- **QCIF** - Quarter Common Interface Format. This video format was developed to allow the implementation of cheaper video phones. The QCIF format has a resolution of 176 ×144 active pixels.

- **SQCIF** - Sub Quarter Common Interface Format. As QCIF, this format is appropriate for mobile telephone multimedia applications where the display resolution and the encoding bit-rate are limited. The SQCIF format support a frame size of 128 × 96 active pixels.

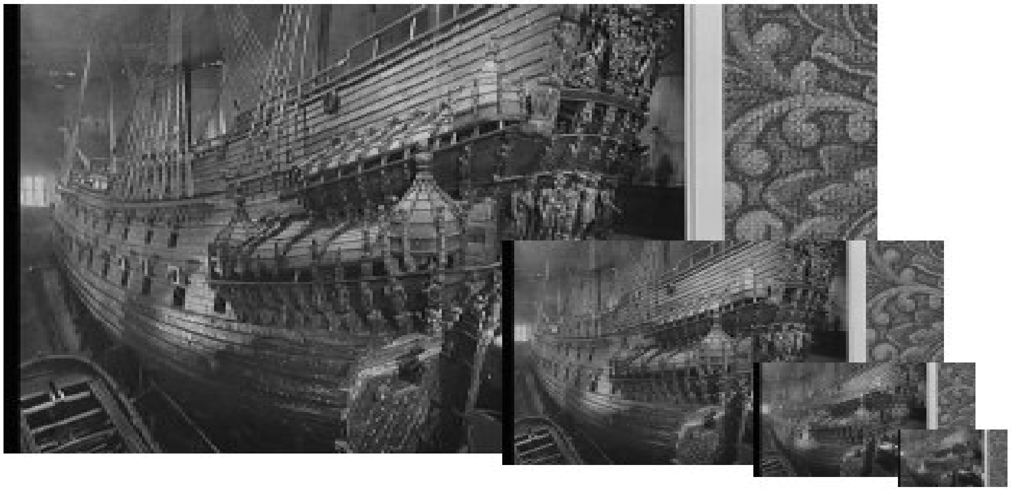


Figure 1-7 luminance video frame at different sampling resolutions

1.6 Existing Codec

In the Standard Model there is a huge delay with big buffer sizes for a single stream. The given encoder codec has a delay of 500ms for the actual buffer size in MPEG4 Ver2. The delay in the system is thus very high for encoding a HDTV input. The size of HDTV input when compared to an ordinary video input is high as well as for a normal encoder it is also very hard to encode. Accordingly we need to either implement a parallel processing inside the FPGA by switching the inputs which is very difficult to implement because of field reordering and buffer occupancy problems. There is no FPGA in market which has a possibility to implement a single stream encoder and decoder for a HDTV application. There is a necessity to implement the design in Multi-stream with a single buffer with a 80ms end to end delay with no 'B' pictures in the video stream. There should be complementary buffers in encoder and decoder to cope up the design for required application. If the design is done in parallel with 4 encoders in Multi-stream inside the FPGA the critical paths and dependency for a compact design are possible. It is very advantageous to select such architecture for implementing a design in FPGA for the required application.

Chapter 2

2 Video Coding Concepts

2.1 Introduction of video coding concepts

The size of digital video data is really huge; despite communication channels have limited transmission bandwidths. However networks bit-rates are continue to increase and the storage capabilities of hard disks, flash memories and optical media (CD, DVD etc's.) is greater than ever before, with the price per transmitted or stored bit continually falling. In this scenario, it is perhaps not immediately obvious why video compression is necessary.

Format	**Resolution**	**Input** **bit rate**	**Remark**
QCIF	176ppl×144lfp× 25fps	10.1 Mbps	Digital video
CIF	352 ppl×288lfp×25fps	40.5Mbps	Digital video
SIF	360ppl×288lfp×25fps	41.4Mbps	Digital video
SDTV	720ppl×576lfp×25fps	165.8Mbps	Digital TV
HDTV	1920ppl×1080lfp×25fps	829.4Mbps	Digital TV

Figure 2-1Table uncompresses digital video sources

Video coding is the process of compressing and decompressing a digital video signal. There are two main reasons why video coding is still used and why there is such a significant effort to make it better. It makes it possible to use the digital video in transmission and storage environment that would not support uncompressed *raw* video because of a lower bitrate bandwidth or storage capability. For example, current internet throughput rates are insufficient to handle a compressed video in real time, even at low bit-rates and resolutions. A DVD (Digital Versatile Disk) can store 30 GB if both layers of 15 GB are used on both disk sides, that means that we could play only 4.56 minutes of video in HDTV raw format, A DVD-Video storage would thus useless without audio/video compression.

Video compression allows exploiting transmission and storage resources in a more efficient way. For example, if a high bit rate transmission channel is available, it is a more attractive solution to send a high-resolution compressed video or multiple compressed video sub-channels rather than sending a single, low resolution, uncompressed bit stream.

Even with constant advances in storage and transmission capacity, compression is likely to be an essential component of multimedia services for many years to come. This chapter introduces concepts such as sampling formats, space-time redundancy reduction and quality metrics which are the base of any video coding algorithm. Then the description is more and more focused on MPEG, starting from its basic coding features until the generic Motion-compensated DPCM/DCT model. Finally, the rate-controlled codec model is explained.

2.1.1 Natural video sequences

A video sequence is a collection of frames sampled at different points in time. Over a rate of 25 frames per second (fps), the Human Visual System (HVS) is not capable to distinguish each single frames and it gives us the illusion of continuity in the movements of objects represented in the scene. PAL and NTSC worldwide TV standards use rates of 25 and 30 frame per second respectively.

A typical *real world* or *natural* scene is composed of multiple objects each one with their own shape, depth, texture and illumination. The color and brightness of a natural video scene changes with varying degrees of smoothness throughout the scene ('continuous tone'). We can identify two main features that are relevant for video processing and compression:

- **Spatial** - Texture variations within the scene, number and shape of objects, colors, brightness etc.

- **Temporal** - Objects motion, changes in illumination, camera movement or viewpoint etc.

Video data sampling

A natural video scene is spatially and temporally continuous. Digital video is the representation of a sampled video scene in digital form, where we perform a spatial-temporal sampling as shown in figure 2.1. Usually a rectangular grid is used (frame) and each spatial-temporal sample (picture element or pixel) is represented as a number or set of numbers that describes the brightness (luminance) and color of the sample. The temporal sampling is performed collecting new still frames at regular intervals in time.

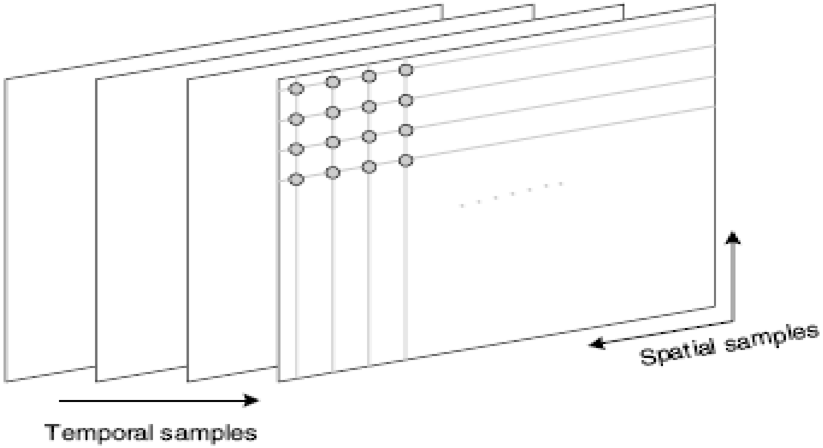


Figure 2-2 spatial and temporal sampling of a natural video sequence

The process of video digitizing involves the three basic operations of filtering, sampling and quantization. The filtering stage is applied to avoid aliasing artifacts to the following up-sampling process. In the case of color image capture, each color component (Red, Green and Blue) is obtained by a separate optical filter and projected onto an image sensor, as for example a Charge Coupled Device (CCD) matrix.

Spatial Sampling

The output of a CCD matrix is an analog video signal, a continuous electrical signal that represents a video image. An anti-aliasing filter is then applied to restrict the spectral characteristic. Sampling the filtered signal means to collect a set of numeric values (samples) corresponding to regular positions on the picture. A rectangle is the most common format for a sampled image, with the sampling points positioned on a grid. The sample rate (number of samples collected per second) determines the grid resolution. Choosing a *coarse* sampling grid produces a low-resolution sampled image, while increasing the sampling rate increases the resolution of the sampled image. Another important parameter is the number of bits used to represent each sample. This latter value is known as the quantization levels, which for consumer video applications is set to 8 bits according to ITU-R Recommendation 601 [8].

Temporal Sampling

A moving video scene is captured by taking rectangular *snapshot* of the signal at continues time intervals. The illusion of motion is given by playing back the series of collected frames. A higher temporal sampling rate (frame rate) gives better results in term of motion fluidity. We can distinguish within four frame rate categories:

- Very low Frames rate below 10 fps are sometimes used for very low bit-rate application as slow motion video. The main advantage is that the amount of data is linearly reduced, but motion is clearly convulsive and unnatural at this rate.
- Low In video conference applications we use a rate between 10 and 20 frames per second. The sequence is fluid, but in the case of fast movements, convulsive and unnatural motion may occur.
- Standard Sampling at 25 or 30 frames per second is the standard for television pictures for PAL and NTSC respectively.
- High in HDTV applications, rates of 50 or 60 frames per second can be used. The motion appears very smooth and fluid.

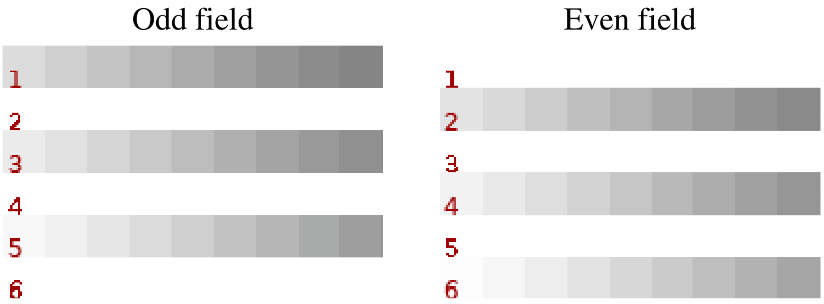


Figure 2-3 odd and even Fields for interlaced image

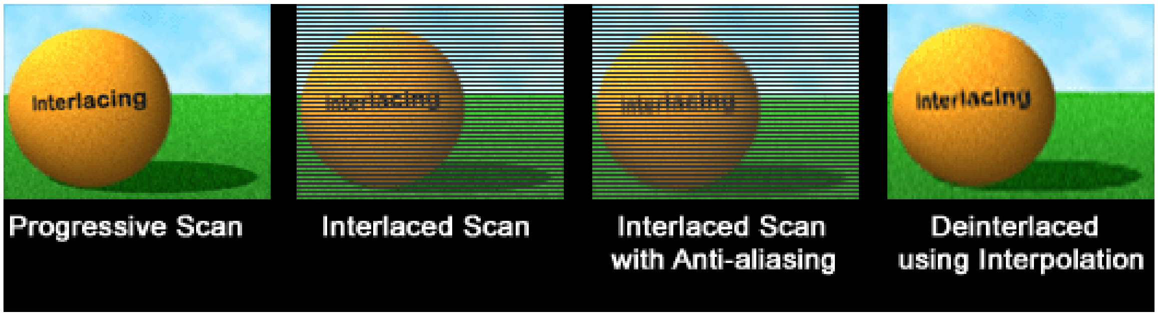


Figure 2-4 interlaced video sequence

A video frame is a collection of lines. Half of the data in a frame, corresponding to odd-numbered lines, are called *top field*, while the remaining even-numbered lines form the *bottom field*. This leads to two different ways of sampling and displaying a video scene:

Progressive: Since video is a series of still images, it makes sense to just display each full image consecutively, one after the other. This is the basic technique of progressive, or non-interlaced, displays which leads to higher vertical resolution. However, the higher refresh time can cause quicker noise in large regions of constant color.

Interlaced: In the early days of television, a technique called *interlacing* was employed to reduce the amount of information sent for each image. By transferring the top field, followed by the bottom field (as shown in figure 2.2 and figure 2.3), the amount of information to capture and send at the same time for each image is halved. For the about same cost as a 25 or 30 Hz progressive display, the interlaced displays can double its refresh rate (to 50 or 60 Hz) in an attempt to avoid flickering.

Frames and fields

A video signal may be sampled as a series of complete frames (*progressive* sampling) or as a sequence of interlaced fields (*interlaced* sampling). In an interlaced video sequence, half of the data in a frame (one field) is sampled at each temporal sampling interval. A field consists of either the odd-numbered or even-numbered lines within a complete video frame and an interlaced video sequence contains a series of fields, each representing half of the information in a complete video frame. The advantage of this sampling method is that it is possible to send twice as many fields per second as the number of frames in an equivalent progressive sequence with the same data rate, giving the appearance of smoother motion. For example, a PAL video sequence consists of 50 fields per second and, when played back, motion can appears smoother than in an equivalent progressive video sequence containing 25 frames per second.

2.2 Video compression

Compression is a process that converts data to a format that requires less space. In the case of video, it is the process of compacting or condensing a digital video sequence into a smaller amount of bits that can be stored or transmitted in a more efficient way.

If *O* is the size of the original or *raw* data and *C* is the size of the data in compressed form, we can define *R* = *C* / *O*, known as the compression ratio. Despite loss compression does not allow the reconstruction of an exact replica of the original data set, a good approximation can be generated, achieving higher compression ratios than lossless techniques. Generally the accuracy of the approximation decreases as the compression ratio increases.

A device (software or hardware) that compresses data is referred to as an *encoder* or *coder*, whereas a device that decompresses data is known as a *decoder*. Furthermore, a device that acts as both coder and decoder is known as ***codec***. The success of data compression largely depends on the data itself and some data types are inherently more compressible than others. Some elements within the data are more common than others and most compression algorithms exploit this property, known as *redundancy*. The greater the redundancy within the data, the more successful the compression of the data is likely to be. A Video compression algorithm is based on three key techniques: Spatial compression, temporal compression and entropy coding:

- Spatial Compression relies upon removing redundancies within areas which own to the same frame (intra-frame).
- Temporal compression exploits similarity between successive frames (inter-frame), per-forming motion estimation and compensation [9].
- Entropy coding groups a set of lossless compression methods developed to increase data- packing efficiency for transmission or storage purposes.

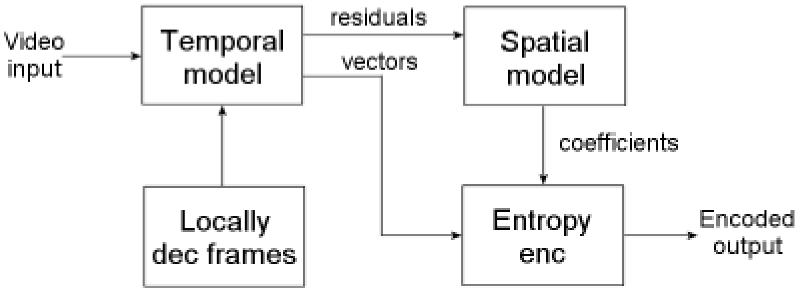
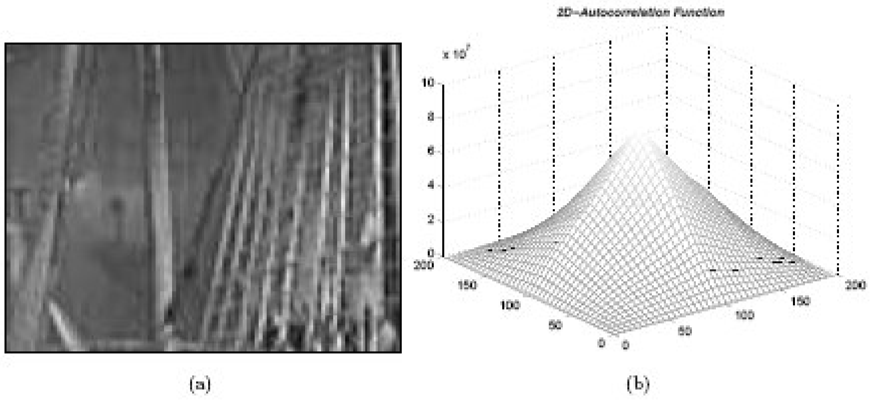


Figure 2-5 video encoder general block diagram

Spatial redundancy

A digitized natural video image consists of a grid of sample values. Figure 2-5(b) shows the similarity between a natural video image (Figure 2-5 (a)) and a spatially-shifted copy of itself (2*D* auto-correlation function). The pyramid-shape graph means that similarity is maximum co-responding to a zero shift and it drops off in the case of any other position of the spatially-shifted copy. The gradual decreasing slope indicates that image samples are locally highly correlated. This means that in natural digital image, neighboring samples on a scanning line are normally similar, leading to a redundancy within the image, known as spatial redundancy.



(a) (b)

Figure 2-6 2D autocorrelation function of a picture sample.

Intra-frame compression techniques (mostly applied to still images) exploit a redundancy reduction within the image and can be applied to individual frames of a video sequence. Practical intra-frame methods typically use four main techniques:

- Prediction - De-correlates data using previously encoded samples.
- Transformation - De-correlates and compacts the data using a transformed domain.
- Quantization - Reduces the precision of the transformed data.
- Reordering - Arranges the data to group together significant values.

Predictive Coding

Spatial prediction is sometimes described as *Differential Pulse Code Modulation* (DPCM), a term borrowed from a method of differentially encoding PCM samples in telecommunication systems. As for motion compensation, DPCM generates a prediction and subtracts this prediction to the current frame to form a residual. If the prediction is successful, the energy in the residual is lower than in the original frame and information can be represented with fewer bits. DPCM is a serial algorithm which exploits a pixel by pixel prediction that is often performed using a square window as in figure 2-6.

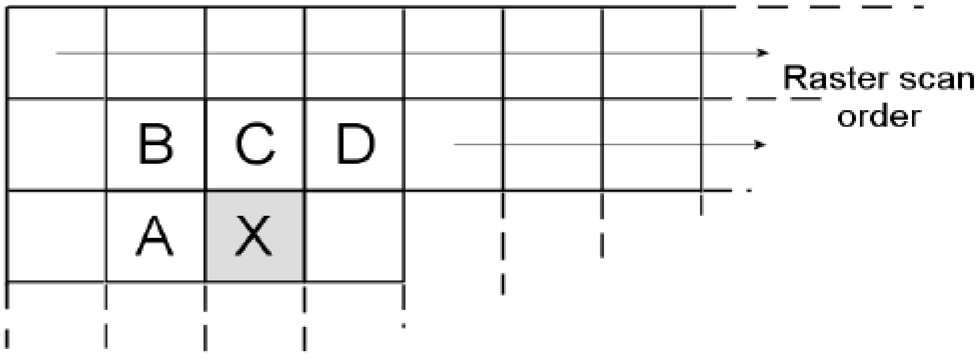


Figure 2-7 Spatial prediction pixel window

The encoder forms a prediction *X* for the current pixel *P* based on some linear combinations of previously-coded pixels: *A, B, C* and *D* (neighboring pixels in present and previous row) then the residual $R = P - X$ is calculated and sent to the decoder. The same prediction *X* is formed on decoder side and the current pixel is simply obtained as

$P = R + X$. If the encoding process is lossy then the decoded pixels may not be identical to the original (*A, B, C, D*) and so the process could lead to a cumulative mismatch or *drift*.

To avoid this problem, the encoder must itself decode the residuals and reconstruct each pixel, computing the prediction on the same pixel values that will be available at decoder side. The efficiency of this method relies on the prediction accuracy. However it is usually not possible to choose a predictor that works well in all conditions and better performance may be obtained by an adaptive prediction scheme, where different predictors are selected depending on the local statistics of the image (flat, vertical, horizontal textures). The disadvantage is that the encoder may have to send additional information to indicate the choice of predictor to the decoder and so there is a trade-off between efficient prediction and the extra bits required.

Transform Coding

The transform stage of a full image or a residual-frame is a reversible process that converts the data into another domain (the transformed domain) where the correlation is lower and compression is easier. We can distinguish between two categories of transform:

Block based - A whole frame or residual-frame is divided into many squared $N \times N$

Blocks each of one is converted into a similar block of coefficients in the transform domain. Block transforms have low memory requirements and are well-suited to be used with block- based motion compensation residuals, although it usually suffers of *blockiness*, an artifact at block edges. Examples of block-based transforms are the Karhunen-Loeve Transform (KLT), Singular Value Decomposition (SVD) and the well known Discrete Cosine Transform (DCT).

Image based - Image-based transform operates at full image or frame level (or a large Section of the image often known as *slice*). Image transforms such as the Discrete Wavelet Transform (DWT) have been shown to out-perform block-based transform for still image compression but they have higher memory requirements and do not *fit* well with block-based motion compensation.

DCT

The most popular transform used in image coding is the Discrete Cosine Transform (DCT, the sampled version of the cosine transform [5, 4]), extensively used in two-dimensional form in many MPEG standards. DCT and inverse DCT equations are shown in 2.6 2.7, where *Xij* is a matrix of video samples and *Yxy* the related matrix of coefficients. Because transformation of large images can be prohibitively complex, it is common to decompose the image into smaller square blocks.

$$Y_{xy} = C_x C_y \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} X_{ij} \cos\frac{(2j+1)y\pi}{2N} \cos\frac{(2i+1)x\pi}{2N}$$

Equation 2.1

$$X_{ij} = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} C_x C_y Y_{xy} \cos\frac{(2j+1)y\pi}{2N} \cos\frac{(2i+1)x\pi}{2N}$$

Equation 2.2

$$C_k = \begin{cases} \sqrt{\frac{1}{N}} & (k = 0) \\ \sqrt{\frac{2}{N}} & (k > 0) \end{cases}$$

Equation 2.3

MPEG exploits decomposition into 8 × 8 pixel blocks, which can be fully represented by a weighted sum of 64 DCT basis functions (Figure 2-7), where the weights are just the corresponding DCT coefficients (Y_{xy})[2]. Since the transform requires multiplication by fractions (Equation 1), higher resolution is needed, resulting in coefficients that have longer word-length than the pixel samples. Typically, each 8-bit samples results in an 11-bit coefficient. Although this process does not in itself result in compression, the DCT coefficients, when read in an appropriate order, tend to be good candidates for compression using run length encoding or predictive coding.

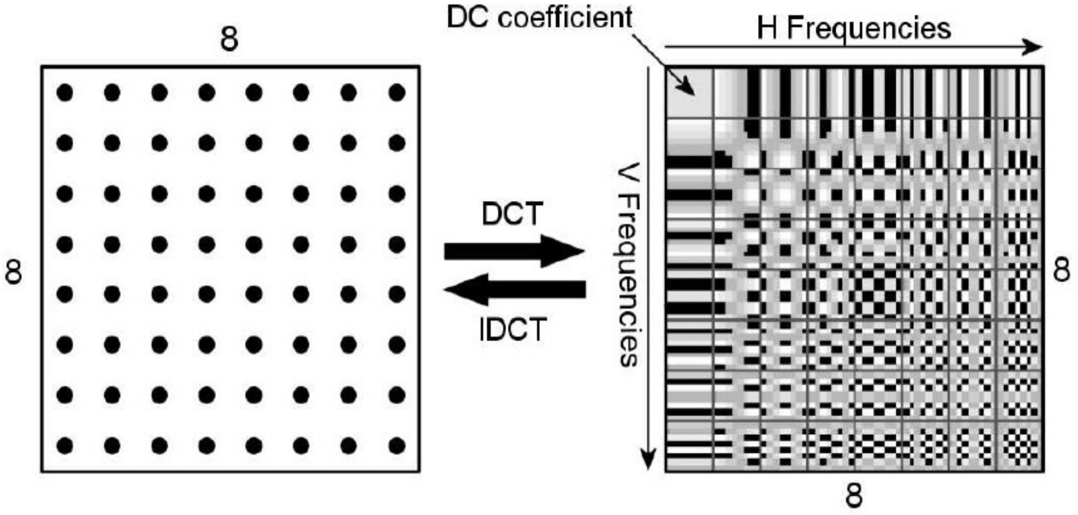


Figure 2-8 8 × 8 DCT basis patterns

Figure 2-7 shows the results of an inverse DCT of each of the individual coefficients of an 8 × 8 transformed block. In the case of the Y signal, the top-left coefficient is the average brightness or DC component of the whole block. Moving across the top row and down the left column, horizontal and vertical spatial frequencies respectively increase. In real pictures, different vertical and horizontal spatial frequencies may occur simultaneously. Using the coefficients in the DCT block, all possible combinations can be achieved. Thus combining the 64 coefficients of the 2-D DCT will result in the original 8 × 8 pixel block. In the case of color pictures, each Y, Cb and Cr component is organized into separated 8 × 8 blocks which are individually transformed.

In much real program material, many of the coefficients will have zero or near-zero values and, therefore, will not be transmitted. This fact results in significant compression that is virtually lossless. If a higher compression factor is needed, then a quantization step is required. The most useful property of DCT coded blocks is that the coefficients can be unrefined quantized without seriously affecting the quality of the image that results from an inverse DCT. However, this quantization will reduce accuracy of the coefficients and will introduce losses into the process. With care, the losses can be introduced in a way that is least visible to the viewer.

DWT

All transforms suffer from uncertainty because the more accurately the frequency domain is known, the less accurately the time domain is known (and vice versa). In most transforms such as Discrete Fourier Transform (DFT) and Discrete Cosine Transform (DCT), the block length is fixed, so the time and frequency resolution is fixed. The frequency coefficients represent evenly spaced values on a linear scale. Unfortunately, because human senses are logarithmic, the even scale of the DFT and DCT gives inadequate frequency resolution at one end and excess resolution at the other.

The wavelet transform is not affected by this problem because its frequency resolution is a fixed fraction of an octave and therefore has a logarithmic characteristic. This is achieved by changing the block length as a function of frequency. As frequency goes down, the block becomes longer. Thus, a characteristic of the wavelet transform is that the basis functions all contain the same number of cycles, and these cycles are simply scaled along the time axis to search for different frequencies. Figure 2-8 contrasts the fixed block size of the DFT/DCT with the variable size of the wavelet.

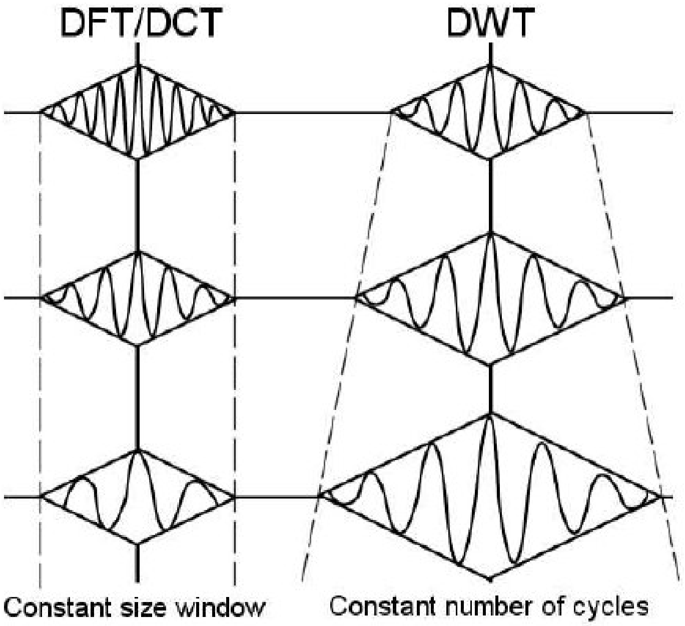


Figure 2-9 DFT,DCT and DWT frequency resolution

Wavelets are especially useful for audio coding because they automatically adapt to the conflicting requirements of the accurate location of transients in time and the accurate assessment of pitch in steady tones. For video coding, wavelets have the advantage of producing resolution scalable signals with almost no extra effort. In moving video, the advantages of wavelets are offset by the difficulty of assigning motion vectors to a variable size block, but for intra-coding applications, this is not an issue. Wavelet coding has shown particular benefits for very-low bit-rate applications, in which the artifacts generated by excessive quantization of wavelet coefficients generally appear as *smearing*, which are much less objectionable than the *blockiness* that results from excessive quantization of DCT coefficients.

Reordering

The most significant DCT coefficients are generally found in or near the top-left corner of the matrix. After weighting, low-value coefficients might be truncated to zero. More efficient transmission can be obtained if the non-zero coefficients are sent first, followed by a code indicating that the remainder are all zero. Scanning is a technique that increases the probability of achieving this result, because it sends coefficients in descending order of magnitude probability. There are two main cases of scanning patterns, depending on whether the video sequence is progressive or interlaced.

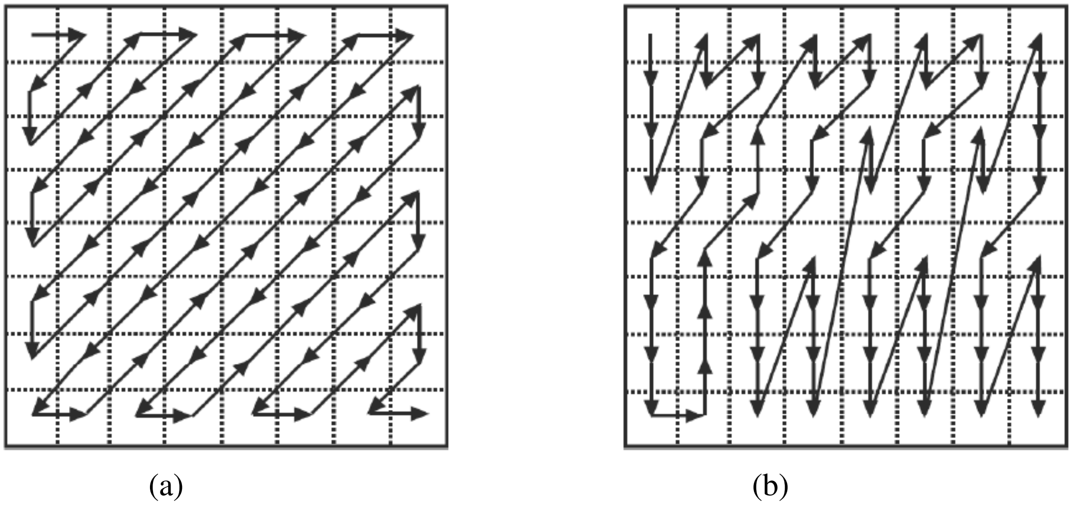

(a) (b)

Figure 2-10 Zigzag scan order (a) Classic, (b) Alternate

In a non-interlaced system, the probability of a coefficient having a high value is highest in the top-left corner and lowest in the bottom-right corner. A 45 degree diagonal zigzag scan (figure 2.19(a)) is the best sequence to use here.

In an interlaced picture, an 8 × 8 DCT block from one field extends over a 16 ×8 pixel screen area, so that for a given picture detail, vertical frequencies will appear to be twice as great as horizontal frequencies. Thus, the ideal scan for an interlaced picture will be on a diagonal that is twice as steep. Figure 2.19(b) shows that a given vertical spatial frequency is scanned before the corresponding horizontal one.

Run-Level Coding

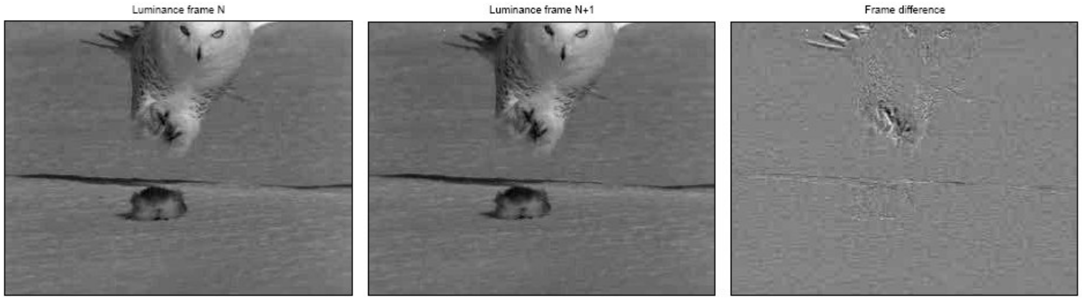
Run-Level Coding (RLC) is an entropy coding technique which is employed in MPEG standards. Higher-frequency DCT coefficients are often quantized to zero and so a reordered block will usually contain one or more clusters of non-zero coefficients near the start, followed by strings of zero coefficients.

The large numbers of zero values are encoded by representing the array as a series of (*run; level*) pairs where *run* represents the number of zero preceding a non-zero coefficient (repeated occurrence of zero) and *level* indicates the magnitude of the non-zero coefficient. Finally, RLC transmits the number of zeros rather than each individual zero sample.

A special case is required to indicate the final non-zero coefficient in a block. In two-dimensional RLC each run-level pair is encoded as in the example and a separate symbol *EOB* (End Of Block) indicates the end of the non-zero values.

2.2.1 Temporal redundancy

There are several inter-frame compression techniques of various degrees of complexity, most of which rely on the general assumption that video frames which are close in time are also high spatially correlated (Figure 2.20). Therefore, when encoding a video frame, it is efficient to make as much use as possible of the information stored in previously encoded frames.



(a) **(b)** **(c)**

Figure 2-11 Example of frame difference (c) = (b) - (a)

Difference Coding

Difference coding, or *conditional replenishment* method, simply considers the difference between the current frame and a previous reference frame, encoding the residual difference. When two frames are very similar, their difference will be simpler to encode than each single frame. Therefore, the reference is used as temporal estimation of the current frame and only pixels that have changed are updated. Information overhead is associated with indicating which pixels are to be updated and if their number is large compression efficiency may be affected. At the cost of introducing some loss, two modifications can alleviate this problem somewhat:

- The intensity of many pixels will change only slightly and when coding is allowed to be lossy, only pixels that change significantly need be updated. Thus, not every changed pixel will be updated.

- Difference coding does not need to operate only at the pixel level, *block* level processing is possible. In the case of 4:2:0 sampling mode, four luminance blocks and two chrominance blocks form a *macro block* (figure 2-11), which corresponds to a 16*x*16-pixel region of a frame.

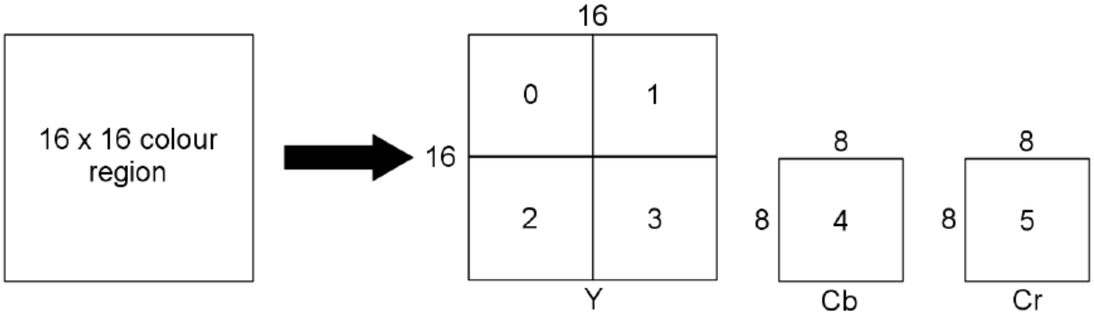


Figure 2-12 Macro block (4:2:0)

If the frames are divided into non-overlapping blocks and each block is compared with its counter part in the previous frame, then only blocks that change significantly need to be updated. For example, if only those blocks of figure 2-10 (b) which contain the bird, its shadow and the prey are updated, the resulting image might be an acceptable substitute for the original. Updating whole blocks of pixels at once reduces the overhead required to specify where updates take place. The 1920 × *1088* pixels in the frame of HD signal can be split into 32640 blocks of 8*x*8 pixels. Significantly fewer bits are required to address one of 32640 blocks than one of 2088960 individual pixels. However, some pixels will be updated unnecessarily and discontinuities might be visible especially if large blocks are used. The suitable block size is a critical parameter which must achieve a trade-off between image quality and compression efficiency.

The obvious problem with difference coding prediction (even if block based) is that only objects that remain stationary within the image can be effectively coded. If there is a lot of motion or indeed if the camera itself is moving (even a very slow pan), then very few pixels will remain unchanged and a lot of energy remains in the residual frame (light and dark areas in above figure 2-10(c)). This means that there is still a significant amount of information to compress after temporal prediction and much of the residual energy is due to object moving. Therefore, a better prediction may be formed by *compensating* for motion between the two frames.

2.2.2 Block based motion estimation and compensation

Considering two consecutive frames divided into uniform non-overlapping blocks, as shown in below figures 2-12 (a)(b), most of the motion that blocks undergo between frames is a translational motion. This assumption forms the based of the Motion Estimation (ME), which attempts to find for each block in the current frame, the best matching block in the previous frame, also known as target block.

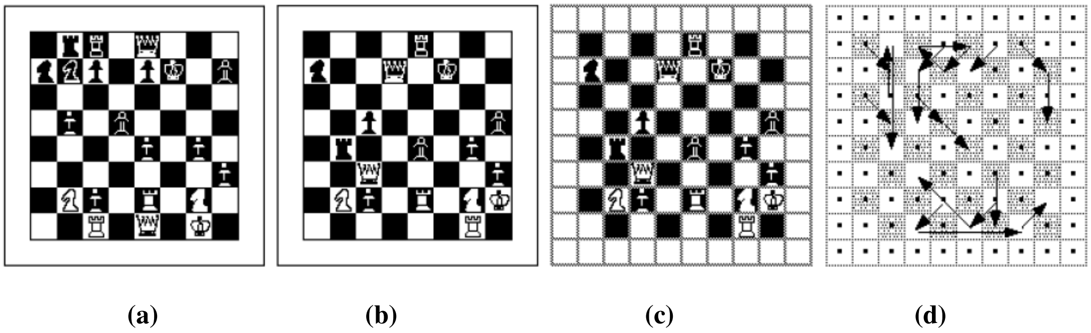


Figure 2-13 Motion vectors estimation

The motion of a block is represented by a Motion Vector (MV) that has two components, the first indicating horizontal displacement, and the second indicating vertical displacement. If the target block and matching block are found at the same location in their respective frames then the motion vector that describes their difference is known as a *zero vector*. The illustration in above figure 2-10(d) shows the motion vectors that describe where blocks in the current frame can be found in past frame. A search window is usually defined and bounds the area within which the encoder can perform the search for the best matching block (figure 2-13). Different criteria (Bidirectional motion compensation) could be used to measure the similarity between two blocks. A common approach to find the best matching block is to evaluate the Sum of Absolute Differences (SAD) at every pixel location within the specified search window. This method is called full search or exhaustive search, and is usually computationally expensive but on the other hand yields good matching results.

A more computationally efficient approach is to restrict the search to only few points in the search area, where there is a high likelihood of finding a good match. These search points are defined by predicted MVs which are calculated on previously encoded blocks.

Once that the each target block has been found, the recovered MVs are used to generate a motion prediction frame by rigidly translating the blocks in the reference one. The difference between the prediction and the actual frame is computed to produce the motion compensated residual to be encoded. This technique is called Motion Compensation (MC)

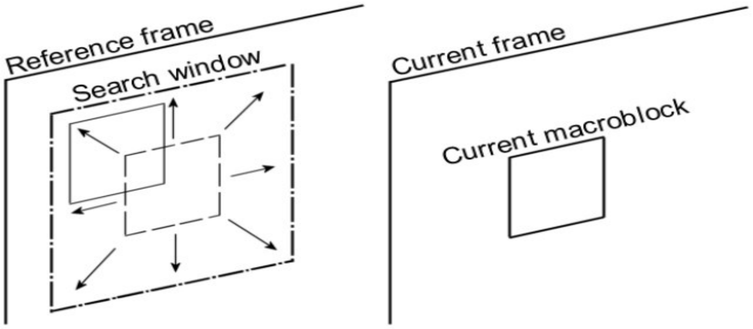


Figure 2-14 Motion estimation search window

Since fewer bits are required to code both residual blocks and MVs than actual blocks, a better compression is achieved. During decoding, the MVs are used to reconstruct the prediction frame (from the reference frame already decoded) which is added to the decoded residual to recover the actual frame [10]. Up to 40% of the bits transmitted by a MPEG encoder might be employed for MVs data. Fortunately, the high correlation between motion vectors and their non-uniform distribution makes them suitable for further lossless compression.

2.2.3 Bidirectional motion compensation

There are many variations on the basic ME and MC process. The reference frame may be a previous frame (in temporal order), a future frame or a combination of predictions from two or more previously encoded frames. When an object moves, it conceals and reveals the background at its leading and trailing edge respectively. The revealed background requires new data to be transmitted no information can be obtained from a previous picture. A similar problem occurs if the camera pans; new areas come into view and nothing is known about them.

Bidirectional motion compensation minimizes this problem by using matching blocks from both a past and future frame to code the current frame. If a background is being revealed, it will be present in a latter picture, and the information can be moved backwards in time to predict part of an earlier picture. Figure 2-14 shows the concept of bidirectional coding. On an individual macro block basis, a bidirectional-coded picture can obtain motion-compensated data from an earlier or later picture, or even use an average of both.

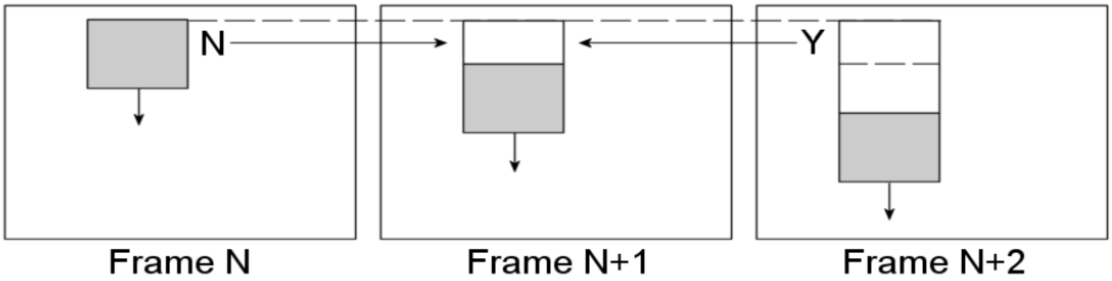


Figure 2-15 Bidirectional coding concept

Bidirectional coding is much more powerful than compression based on single past frame only, reducing the amount of residual data by improving the efficiency of possible prediction, which allows more blocks to be replaced by MVs. However, bidirectional MC requires that frames would be encoded and transmitted in a different order from which they will be displayed. Whether there is a significant change between the reference and current frame (for example a scene change), it may be more efficient to avoid MC. Generally, an encoder may choose between *intra* and *inter* mode (encoding without or with MCP) according to the convenience for each single block. In the case of MPEG-1 and MPEG-2 the ME is performed on luminance component at macro block (16 × 16 pixels) level and furthermore extended at the color component by hypothesis of correlation. For MPEG-4 can perform luminance component at macro block (4 × 4) level.

Moving objects in a natural video scene rarely follow a square 16 × 16-pixel boundaries and so a variable block size for ME and MC can certainly achieve better performances.

2.2.4 Variable block size motion compensation

The ME technique allows to reduce the residual frame energy by compensating each 16×16 macro block. This energy can be further reduced by compensating each 8 × 8 blocks instead, and further more using 4 × 4 blocks. In general, smaller MC block sizes can produces better results.

However, a smaller block size leads to higher complexity (more search operations) and an increase in the number of MVs to transmit. The benefit of reduced residual energy may be outweighed by the overhead of bits necessary to encode the MVs.

In order to exploit the advantages coming from the reduced block size, a trade-off between block sizes and image content has to be found. Large block sizes can be chosen in flat homogeneous zones of a frame, while smaller block sizes can be selected around areas of high detail and complex motion. This technique is the Adaptive Motion Compensation Block Size (AMCBS), which has been introduced in H.264 codec.

2.2.5 Sub-pixel motion compensation:

The motion of objects in the real world is continuous and does not necessarily match the sampling grid points after digitization of the scene. In some cases a better motion compensated prediction may be formed by predicting from interpolated sample positions in the reference frame.

Usually a sub-pixel ME adopts a multi-resolution search approach, which achieves a significant saving in the number of operations over the full search method. The estimated integer-pixel displacement is obtained first through the classical ME method. Then the images are interpolated (in the area of interest only) and a finer search at sub-pixel resolution around this integer position is performed on these interpolated data. In the case of half-pixel resolution, we have to choose the best within 8 different displacements around the integer position.

In general, *finer* interpolation provides better MC (lower residual energy) at the expense of an increased complexity. In fact, sub-pixel ME requires more bits to code the MVs displacement fields, as their resolution is higher. Since a more accurate MC approach requires more bits to encode the MVs and fewer bits for the residual, whereas a less accurate MC requires fewer bits for the MVs and more bits for the residual, a trade-off between MC accuracy and compression efficiency have to be taken into account. Sub-pixel MC is commonly used at *half-pixel* accuracy, as *quarter-pixel* is considered to be the limit of further incremental coding gain.

2.2.6 Entropy coder

Entropy encoding is a reversible lossless process which converts a series of symbols associated to elements of the video sequence into a compressed bit stream suitable for transmission or storage. Input symbols may include quantized transform coefficients (previously run-level coded), MVs (a [*x; y*] displacement vector for each motion compensated block with specified, integer or sub-pixel, resolution), markers (codes used to indicate a synchronization points in the sequence) headers (macro block headers, picture headers, sequence headers, etc.) and supplementary information.

Once that a dictionary of symbols has been defined, then we can study the probability of occurrence of each particular symbol in real video sequences. In practice, some symbols will occur more often than others and this statistical information can be used to achieve further compression using Variable Length Coding (VLC). Frequently occurring symbols are converted to short code words, and infrequent symbols are converted to long code words. However, many methods have been conceived in order to achieve entropy coding and not all of them make use of VLC.

- **Relative Encoding** - Certain symbols are highly correlated in local regions of the picture. For example the DC value of neighboring intra-coded blocks of pixels may be very similar. Neighboring MVs may have similar [*x; y*] displacements and so on. Coding efficiency may be improved by predicting elements of the current block or macro block from previously-encoded data and encoding the difference between the prediction and the actual value [2].

- **Run-Level Coding** - Run-Level Coding (RLC) is an entropy coding technique which is employed in MPEG standards. Higher-frequency DCT coefficients are often quantized to zero and so a reordered block will usually contain one or more clusters of non-zero coefficients near the start, followed by strings of zero coefficients. The large numbers of zero values are encoded by representing the array as a series of (*run; level*) pairs where *run* represents the number of zero preceding a non-zero coefficient (repeated occurrence of zero) and *level* indicates the magnitude of the non-zero coefficient. Finally, RLC transmits the number of zeros rather than each individual zero sample. A special case is required to indicate the final non-zero coefficient in a block. In two-dimensional RLC each run-level pair is encoded as in the example and a separate symbol *EOB* (End Of Block) indicates the end of the non-zero values [2].

- **Huffman Coding** - Huffman coding is a classic compression technique that assigns variable length codes (VLC) to symbols, so that the most frequently occurring symbols have the shortest codes [11]. On decompression the symbols are reassigned their original fixed length codes. For example, in text compression applications, variable length codes are used in place of ASCII codes, and the most common characters, usually *space*, *e* and *t*, are assigned the shortest codes. In this way the total number of bits required to encode the data can be considerably less than what a fixed length representation achieves. Huffman coding is particularly effective where the data are dominated by a small number of symbols [2].

- **Arithmetic Coding** - Although Huffman coding is very efficient, it is only optimal when the symbol probabilities are integral powers of two. Arithmetic coding [12] does not have this restriction and is usually more efficient than the more popular Huffman technique. Although more efficient than Huffman coding, arithmetic coding is more complex [2].

- **Lempel-Ziv Coding** - Lempel-Ziv compressor use a dictionary of symbol sequences. When an occurrence of the sequence is repeated it is replaced by a reference to its position in the dictionary. There are several variations of this coding technique, which are characterized by different dictionary management approaches. The most well known of these techniques is the Lempel-Ziv-Welch variation [2].

2.3 Main causes of delay in MEPG

The most common field where a low latency is needed is video conferencing. However, this application does not require a high resolution (usually CIF or QCIF) and the main constrain is a very low coding bit rate (< 384 Kbps), which limits the image quality. In studio production applications, where the video resolution is bigger (4CIF) and a higher encoding quality is required, the encoding bit rate increases up to 30÷40 Mbps. Quality and bit rate are related variables, in the sense that given a particular set of coding tools, if we increase the bit rate, the quality gets better and vice versa . Using a more efficient coding algorithm, we can obtain higher quality at lower bit rates, but the coding/decoding latency will not necessarily decrease.

Evaluate the coding/decoding delay in a very complex algorithm is a non trivial problem. In a very intuitive way, we can say that latency and complexity are in contrast, because the more complex the algorithm, the more *elaboration time* it requires to finish its job. However, this assumption is not always true. The real trade-off between complexity and latency is the level of parallelization achievable in the algorithm implementation. MPEG standards employ a common set of features in order to assure a basic backward compatibility.

Since MPEG-1, the subsequent evolutions have exploited the hybrid DPCM/DCT video codec model described in section 2.11, which have been more and more enhanced without modifying its basic scheme. In the following sections we analyze a set of basic MPEG features taking into account it's contribute to the processing latency. Most of these features are used by all MPEG standards; therefore, a class of basic coding tools can be defined in order to minimize the coding/decoding delay in MPEG.

2.3.1 Data format

MPEG exploits the hierarchical data structure shown in figure 2-15. The very basic data element is a matrix of 8 × 8 samples called *block* which can be of three types: luminance (Y), red chrominance (Cr), or blue chrominance (Cb). A group of 4 blocks, corresponding to an area of 16 × 16 luminance samples, forms the base of a *macro block*, which is also composed by a number of additional chrominance blocks depending on the sampling format. One or more consecutive macro blocks form a *slice*.

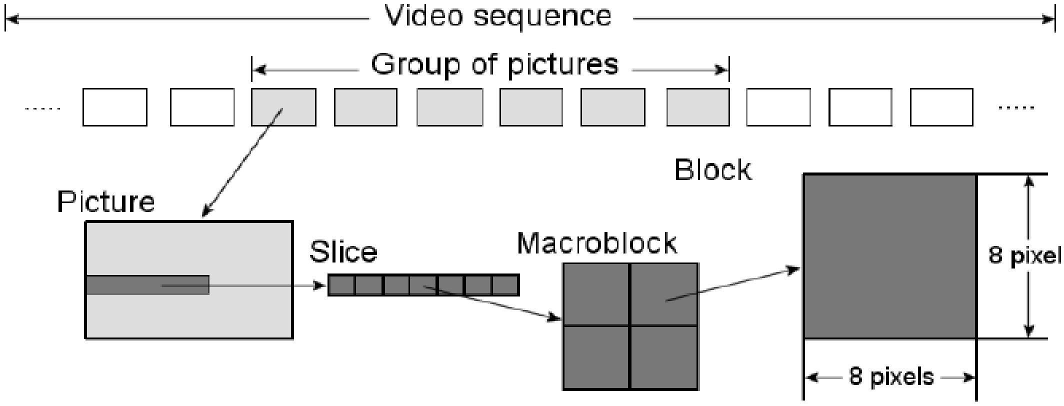


Figure 2-16 MPEG hierarchical data structure

As already mentioned in section 2.8.2, the use of blocks as elementary data suites well the hybrid DPCM/DCT motion compensated basic encoding MPEG scheme. However, from the latency point of view, the encoder cannot start the processing till it has received at least 16 lines of video, which gives a delay of ′ 1 ms. Despite this minimum requirement, in many hardware implementations almost one input frame is stored before to start encoding, which leads to a more important delay of 33 ÷ *40* ms depending whether PAL or NTSC system is employed.

2.3.2 Frame and Field encoding

Many video sources generate interlaced video sequences, where the top and bottom field are acquired and transmitted one after the other. Using a *frame-mode* encoding we have the advantage of a better spatial correlation between consecutive lines (especially in the case of poor movement), but before starting encoding we need to re-interlace fields (which come alone) and store each frame in an input buffer as shown in figure 2-15. Moreover, if at the decoder output we want to generate the same interlaced sequence, we have to take into account an additional latency of half a frame time, which is necessary to store the two fields separately.

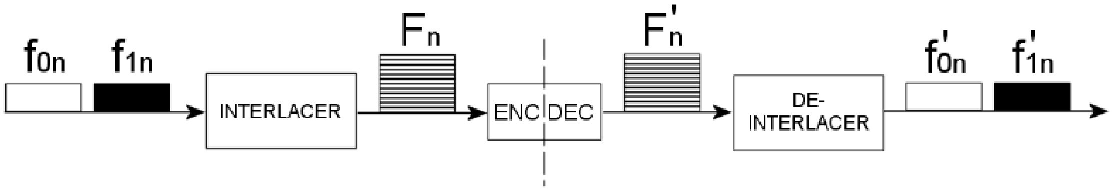


Figure 2-17 Frame encoding of interlaced sequences

On the other hand, the field-encoding mode (figure 2-16) avoids the input latency at encoder side and the output latency at decoder side at the price of a slight decreased vertical resolution. Furthermore, if the encoder exploits temporal redundancy using the MC and ME approach, it can be affected by a reduced efficiency due for example, to any object's pixels which are present in one field but not in the next (It is the case of horizontal lines). However, this effect is strongly dependent on the kind of video scene and in most cases is not very important.

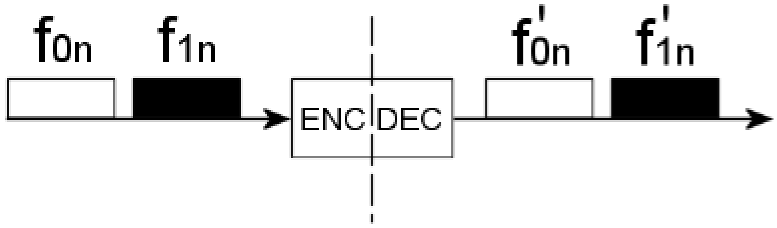


Figure 2-18 Field encoding of interlaced sequences

Using a frame-mode encoding process with interlaced sequences, we introduce a delay at encoder and decoder side that is globally greater than one frame time (40 ms and 33 ms for PAL and NTSC respectively). The picture quality is generally the most appreciated parameter therefore, most of the hard-ware platforms do not care about latency and the field-coding mode is not widely supported.

2.3.3 GOP size and structure

A GOP (Group of Pictures) is a series of one or more coded frames intended to assist in random accessing and editing. A regular GOP can be described with two parameters:

- **N** - The GOP-length is the number of pictures in the GOP.
- **M** - The GOP-structure is the spacing of the P pictures.

The GOP illustrated in figure 2-17 can be described as N = 9 and M = 3. Generally the GOP structure repeats through the sequence, but N and M may be changed at any time during the encoding process. The smaller the GOP, the better the response to movement (since the I frames are closer) and editing capabilities, but the lower the compression. For example, sending I pictures only requires more than twice the bit rate of an IBBP GOP; however each I picture can be edited independently of each other.

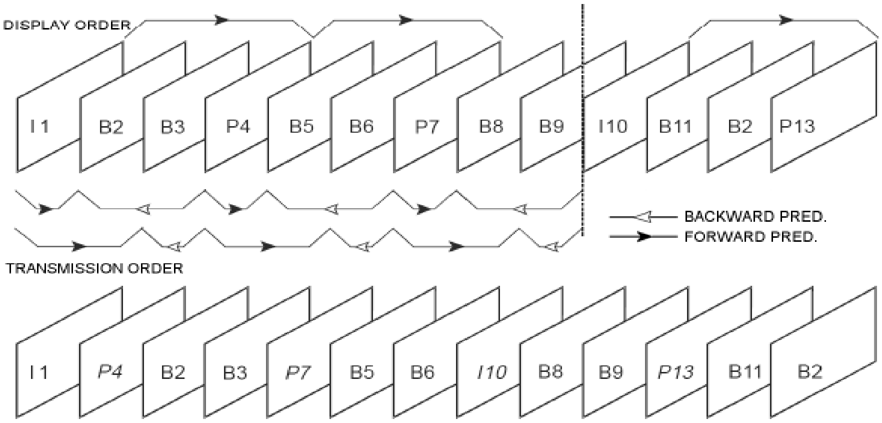


Figure 2-19 Transmission and display picture order

The smallest GOP length is a single I frame, which corresponds to a GOP structure M =1. There are no formal limits on the length of a GOP, but for transmission purposes a typical length of 12 or 15 pictures is employed. A GOP containing B pictures have the further disadvantage that transmission order and display order of the encoded pictures are not the same. In fact, while a P picture naturally follows an I or P picture from which it has been predicted, a B picture cannot be decoded until both of its references (I and P) have been received and decoded. Summarizing, in the coded bit stream, a GOP must start with I frame and may be followed by any number of I, P, or B frames in any order. In display order, a GOP must start with I or B frame and ends with I or P frame. Figure 2-17 shows the same GOP sequence in display order at the top and in transmission order below. Note that, in transmission order B pictures always follow the reference pictures from which they have been predicted.

Despite the better efficiency and flexibility of bidirectional coding, which achieves up to 4 time less data if compared with a typical I pictures occupancy, the use of B pictures have to be restricted or avoided in a low delay coding scheme. Sending pictures out of display order requires additional memory at encoder side and a more complex processing scheme which introduces an important latency. In fact, a GOP-structure of $M \geq 1$ gives a latency contribution of M-1 frames, which correspond to the buffering of the intermediate frames that occurs between I and P references which need to be coded and transmitted in advance.

2.3.4 Video Buffer Verifier

In DPCM / DCT video codec model [5] it was stated that the coded pictures resulting from the MPEG encoding of video are not of constant size. Their length depends on whether they represent an I, P or B picture and on the complexity of the visual scene. A video encoder therefore generates a variable amount of bits per picture and it outputs a VBR bit stream. If the application requires a fixed-bandwidth transmission channel, then the VBR is generally smoothed to a CBR by a First-In First-Out (FIFO) memory known as the coder buffer. On the decoder side a similar buffer is needed in order to reproduce the same VBR outputs by the encoder. This process involves two buffers and introduces a delay that is function of the chosen CBR and the buffer size.

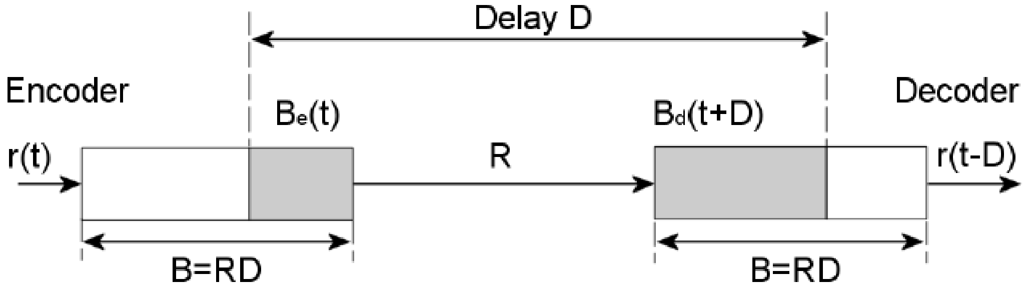


Figure 2-20 Encoder and Decoder buffer delay

If the encoder and decoder buffer are well synchronized, it is possible to prove that the sum of their fullness is a constant *B* equal to the buffer size [27], as shown in figure 2-19 and stated in equation.

$$B_e(t) + B_d(t + D) = B \quad \textbf{Equation 2.4}$$

$$B = RD \qquad \textbf{Equation 2.5}$$

If *R* is the CBR rate then using the equation 2.5 we obtain that the global delay introduced by the rate conversion is $D = B / R$.

The MPEG standard does not specify the strategy to control the encoder buffer, but it gives a table of maximum sizes and encoding bit rates related to each profile and level. Therefore, it is up to the developer to find the best trade-off between delay and buffer size. This delay contribution may be very important at low-medium bit rates however, most implementations employ the maximum sizes suggested by the standard, which means a greater buffer delay contribution whatever the bit rate is.

2.4 MPEG low delay solutions

A low delay mode for real-time video communications, achieving latency 500 ms (default delay of MPEG 4), is described in MPEG-4 standard. This mode has been designed to cope with medium-low quality video applications at low resolution such as: visual telephony, video-conferencing and monitoring.

Three main features have been exploited by MPEG:

- Low buffer occupancy
- No bidirectional coding.
- Intra slice coding.

Low buffer occupancy for both encoder and decoder is required for low delay. However, a constant latency is not automatically performed since the buffer size is kept constant and MPEG delegates to developers to specify a strategy to compensate the dependency on the working bit rate.

The total encoding and decoding delay is kept low by generating a bit stream which does not contain B pictures. Therefore, I and P pictures only are exploited by the coding process, which avoids the high frame reordering latency typical of bidirectional coding approach.

Furthermore, to avoid I pictures, which require a large amount of bits, the intra-slice coding technique is employed.

As 500 ms are not enough to satisfy the requirements of studio production applications, which demands latencies less than 40 ms, the MPEG low delay mode does not provide a valid solution for our purpose. However, this method can be considered a good starting point which can be enhanced and integrated with an efficient VBV buffer control strategy for the case of real low occupancy.

2.4.1 VBV buffer size

The VBV buffer size is a crucial point to control the delay in applications which need a CBR coding. Given a fixed buffer size, the delay is a function of the working bit rate *R* as stated in equation 2.6.

$$D = B / R \quad \textbf{Equation 2.6}$$

$$B = K\ R \quad \textbf{Equation 2.7}$$

If we want *D(delay)* to be a constant then *B(buffer size)* must be linearly dependent on the encoding bit rate *R*, as described in equation 2.7, where *K* is constant and it equals the requested buffer delay *D*. Given the maximum buffer size *VBVmax (variable buffer)*, we can state that:

$$B <= V\ BVmax \qquad \textbf{Equation 2.8}$$

$$K <= V\ BVmax\ R \qquad \textbf{Equation 2.9}$$

Beyond the limitation of equation 2.9 we can make our choice of *K*. We know that the lower the buffer size the lower the delay, but on the other hand, a small buffer is more difficult to control and the risk of overflow and underflow increases with consequent loss of frames. The ideal buffer should be able to compensate the input VBR peaks due to both spatial and temporal complexity variations in the video sequence. A first estimation of the minimum buffer size could be the difference between the size of the biggest and smallest encoded pictures which may occur in the video sequence. However, the information of the maximum possible variation is not enough. We also need to know how much time it takes. In general, three main variables can influence the buffer size:

- Sequence - The performances of a video coding algorithm are data dependent, so the results may change a lot using different input video sequences. For example, using the same average quality, a sequence made by the repetition of a still frame, as for example a bar code test, would require a smaller buffer than a more complicated sequence rich of movements, details and scene changing.

- GOP - The bit occupancy of an I picture is always bigger that a P picture, while B pictures get the lowest sizes. This means that a GOP made of I or P pictures only, can limit the difference between sizes of consecutive frames.

- Control - In a rate-controlled encoding scheme an algorithm adjusts the encoding parameters to achieve a target bit rate. Therefore the controller efficiency can directly influence the required buffer size.

Finding the optimal solution for the minimum buffer size is a non trivial problem which involves many variables. An existing solution will be analyzed in the next sections, for the moment we can suppose that in a given encoding configuration the buffer keeps an average number N € Q of coded frames. If P is the input frame rate (for example 25 Hz for PAL) then the average encoded frame size is $R=P$ and B can be expressed as in equation 2.10 that combined with equation 2.7 gives us a simple expression for K.

$$B = N\,(R / P) \quad \textbf{Equation 2.10}$$

$$K = N / P \quad \textbf{Equation 2.11}$$

In the case of a very regular bit stream, where the average bits per frame is quite constant, the problem to find an optimal buffer size becomes to find the lowest number N which allows an efficient buffer control even in the case of a critical video sequence. In the case of field-mode encoding, we can have N in all previous equations, getting half the delay than the frame-mode configuration.

$$B = (N /2P) \times R \quad \textbf{Equation 2.12}$$

2.4.2 I and P only

In order to minimize the variations between consecutive pictures size, short GOP-length should be avoided. The simplest solution is an I-only scheme (GOP with M = 1). In this case the average bits per encoded-frame are equal to the average intra-coded picture size. As I-picture bit occupancy does not achieve high variance, this working condition agrees with the hypothesis (theory or assumption) of regular bit stream, needed by the buffer model of previous section. On the other hand, intra-coding is not very efficient itself, which means that a higher amount of bits is generated and higher bit rates are required to achieve a target quality.

Despite P pictures requires about half the data of an I picture at the same encoding quality, we cannot use a P pictures only scheme, because the forward prediction method in based on a reference frame which must be intra-coded at least for the first position in the GOP. Using a GOP based on both I and P pictures, we get the following sequence:

IPPP……….. PPPP …….. IPPP…………… PPPP…… IPPP……PPPP…PPPPPP **Equation 2.13**

If the GOP-length is great enough then the average amount of bits per encoded-frame is equal to the average P picture size. However, each time that an I picture occurs, the buffer occupancy rapidly increases and the control feedback will decrease the coding quality to avoid the buffer overflow. This reaction would be visible in the decoded sequence as a periodic flash corresponding to a quick quality change. To avoid this inconvenience a GOP-length=*1* can be employed. However, I pictures repetition represents an effective mechanism to stop the error propagation, which can otherwise really affect the decoding process. Each P picture encodes the difference between two consecutive frames so, if an error occurs during the decoding process, a zone of the decoded picture would be affected and the error would spatially and temporally propagate over all the sequence. To solve this problem without using I pictures, an intra-slice auto fresh encoding strategy can be adopted.

2.4.3 Slice encoding

MPEG defines the slice as a series of consecutive macro blocks. Every slice shall contain at least one macro block and shall not overlap. The position of slices may change from picture to picture but the first and last macro block of a slice shall be in the same horizontal row of macro blocks. Slices shall occur in the bit stream in the order in which they are encountered, starting at the upper-left of the picture and proceeding by raster-scan order from left to right and top to bottom.

The most basic method to achieve spatial localization of errors is to reduce the number of macro blocks in each slice. The increased frequency of re-synchronization points will limit the affected picture area in the event of a loss. It is effective in any transport or storage media, and in any profile since the slice structure is always present in MPEG coded video. The method results in a small loss of coding efficiency due to the increase of overhead information. From the perceived picture quality point of view, the performance of this method is generally dependent on the relative sizes between slice and picture. Therefore, the slice size should be decided by considering the picture size (in unit of macro blocks) and the trade-off between coding efficiency and visual degradation due to errors.

To avoid the disadvantages of intra pictures, some applications requiring low delay can choose to update the sequence by intra-coding only parts of a picture. This may provide the same kind of error resilience as intra pictures. For example, assuming that a constant number of slices per picture from top to bottom are intra-coded, then the whole picture will be completely updated every n frames, where n is a constant which depends on geometrical parameters only. If the intra-coded slice has small size compared to the whole picture, the buffer occupancy does not increase too much and the rate control can keep the quality of the intra-slice close to that of the surrounding non-intra macro blocks. In summary, the intra-slice method makes it possible to use P pictures in a GOP with length=*1*, which represents the best trade-off between coding/decoding delay and average video quality.

2.4.4 VBV buffer rate control

Rate control is not a part of the MPEG standard, but the standard group has issued non-normative guidance to aid in implementation. A rate control algorithm dynamically adjusts encoder parameters to achieve a target bit rate *R*. It allocates a budget of bit to each group of pictures, individual picture and/or sub-picture unit in a video sequence and depending on the predicted next picture complexity, it calculates the quantization parameter *QP* to employ. The quantization parameter *QP* determines the encoding quality and the real amount of bits needed to code a residual picture. *QP* is typically limited in changes to no more than ±25% units between pictures, in order to guarantee stability and to minimize perceptible variations in quality.

If the sequence's complexity varies quickly, as it is common at a scene change, *QP* cannot follow this variation at the same velocity, thus some pictures will be encoded at higher quality than the one allowed by the selected bit rate, which will result in a bit allocation overhead.

This data overhead will cause a VBV buffer occupancy increment which will be compensated later by the rate control algorithm itself. However, in a low delay application the VBV buffer size may be very small and this mechanism can generate a buffer overflow, with consequent errors and frame skipping at decoder side. So the lower delay we try to achieve, the higher the risk of overflow using a standard rate control method. A way to overcome this problem is to take into account a bandwidth overhead. If *R* is the channel bit rate, we can set the encoding rate to R/α where $\alpha \geq 1$. The residual bandwidth $R(1 - 1/\ \alpha)$ will be used by stuffing packets during most of the time, but in the case of data overhead it will avoid the buffer overflow. In other words, we read the data from the buffer at a higher rate than expected, so the encoder buffer level is kept low for most of the time and if the picture complexity varies quickly we achieve a more efficient control.

Using this buffer control strategy, under normal working conditions, the encoder buffer is kept quite empty while the decoder buffer quite full, thus the VBV buffer delay is completely moved to decoder side. Therefore, the decoder implementation becomes a very important point in low delay applications.

2.5 Conclusions

The analysis of MPEG low delay mode puts in evidence that only a sub-set of coding features can be employed to achieve low latency, with a consequent loss of compression efficiency. The buffer size and its control strategy become a critic point in the latency optimization. Therefore, an audio low delay coding technique is required in order to support the minimum latency which has been achieved on video encoding side. So we are going to use IPPPP…up to ∞ picture, intra slice refresh and rate control based on each slice and rate control is based on PD (proportional + derivative) control.

Chapter 3

3 Implementation of multi stream approach

3.1 Work contribution for the project

The HD (High Definition) project wants to investigate a low delay coding architecture to target a HD MPEG-4 platform. This part of preliminary study will be achieved on a visual C++ MPEG-4 simulator at simple profile, which can process a YUV input video sequence as showed in figures 3-1 and 3-2.

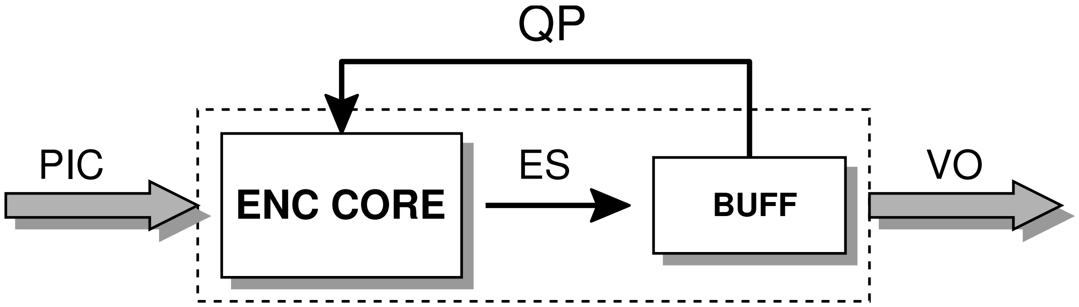


Figure 3-1 simple encoder

Each input PICture which compose the video sequence, is encoded by the MPEG-4 SP core, which generates an ES bitstream (Elementary Stream). A classic buffer feedback is used in order to smooth the ES VBR (Variable Bit-Rate) into a CBR (Constant Bit-Rate), which is required for transmission over a bandwidth limited channel. The final result is a VO (Video Object), which is written in a binary file.

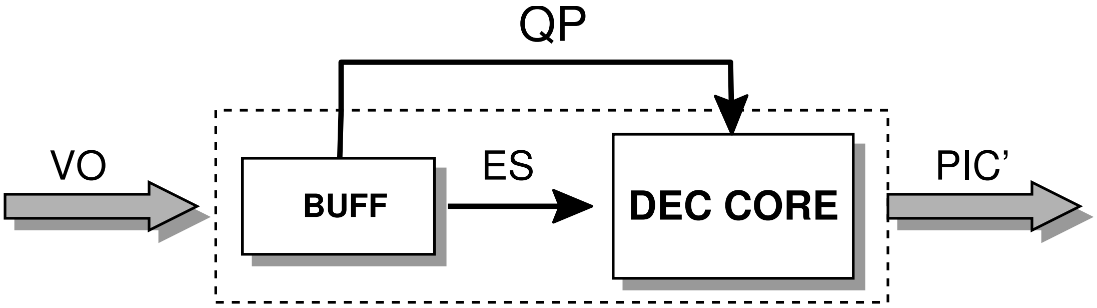


Figure 3-2 simple decoder

The decoding software is an independent task which starts reading the VO.bit file and generates an approximation of the input video sequence. Therefore PIC' is not the same than PIC because of the quantization process performed in the encoder, which is intrinsically not reversible.

The general architecture showed in figures 3-1 and 3-2 makes use of the basic technique to reduce the coding/decoding latency. Further improvements are made using a multi-stream approach (Figure 3-4). The idea is to divide the each input PICs into four non-overlapping regions, which can be processed by four independent encoders. The global TS (Transport Stream) is then generated multiplexing the four TSs related to each sub-picture. Figure 3-3 shows the strip scan (a); strip scan (b) and square scan methods. In the three methods the most intuitive way to divide the input PIC into 4 sub-PICs are strip scan (b) and square scan. In strip scan (a) when encoded there will give many artifacts because the motion cannot be compensated properly because of the cut. We had chosen the other methods for splitting.

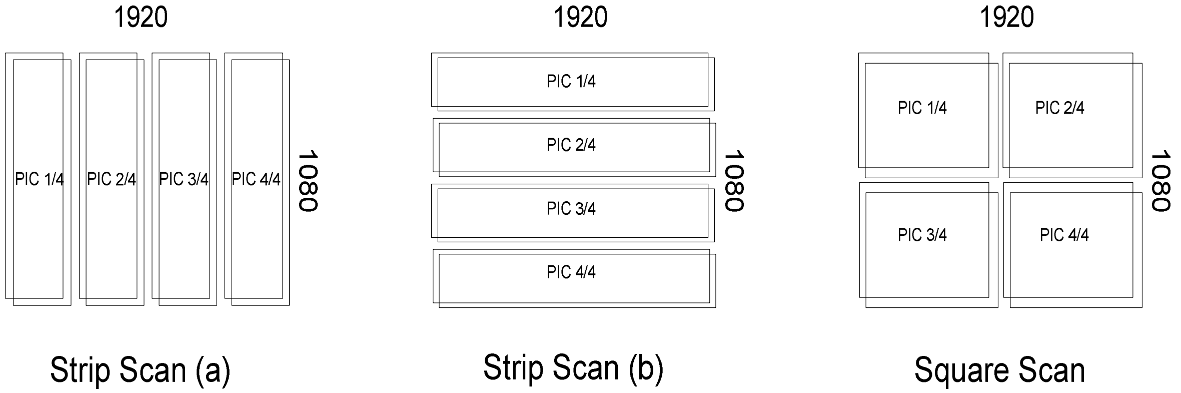


Figure 3-3 Possible Splitting of HD Sequence

Some modifications to the general software are required to implement the encoding scheme showed in Figure 3-4, as similar modification will be required to the decoding process.

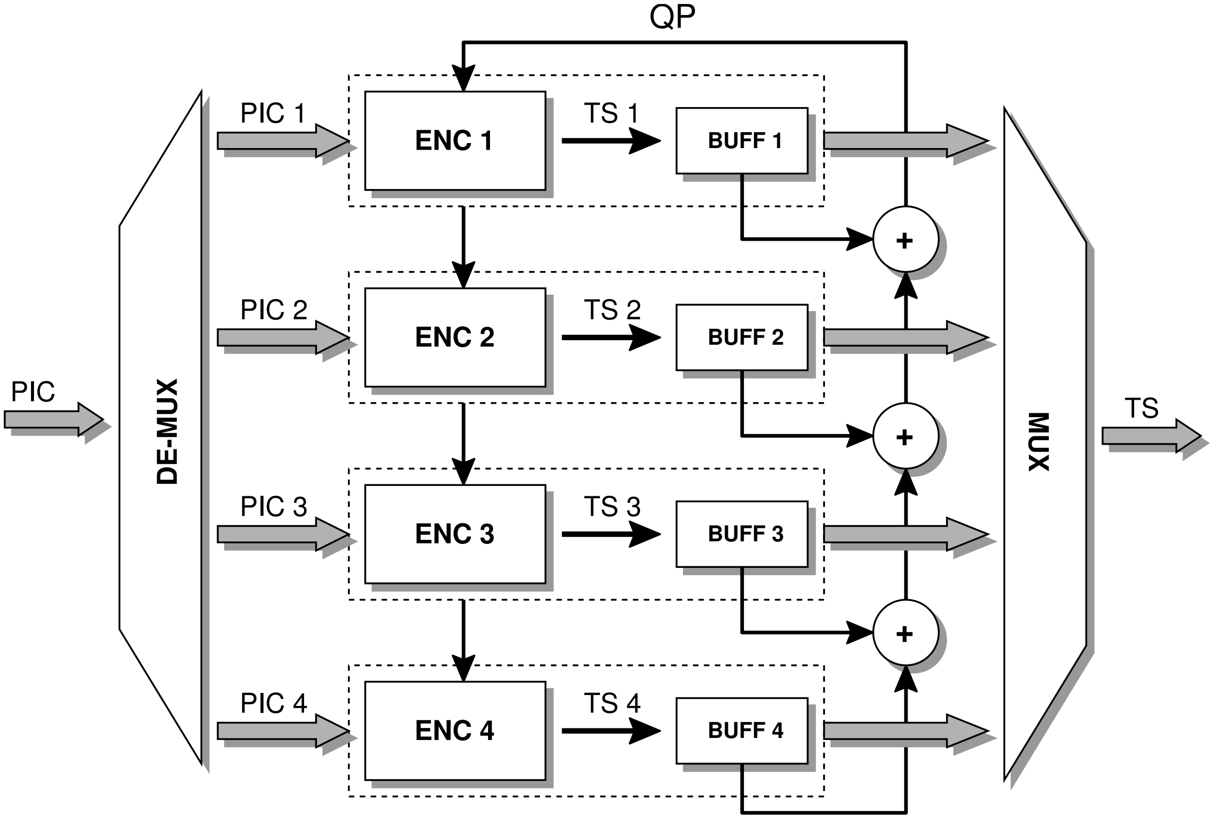


Figure 3-4 Design of multi- stream approach for Encoder

3.2 Implementation of intra slice refresh

The main aim to implementation of Intra slice refresh is to maintain our GOP without B frame and without intermediate I frame/fields. If we use without B frame/ field we do not get high amount of data in different levels. In our process we are applying IPPPPPPPPPP….∞ so if we are trying for this approach we need to fresh Intra slice which can finish Intra picture after some certain frame/ field.

The main advantage of intra slice refresh is our buffer maintain constant level fullness because we are introducing our intra frame/ field as slice vice, so while applying intra slice refresh we include some slice of intra by force fully with this it will maintain constant buffer occupancy and we will introduce intra frame/field as slice vice depending upon how many slice we will introduce per frame/field.

We introduce different level of intra slice refresh depending upon how many slices per field we want introduce. Depending upon this we classified two types for flexibility.

- Half slice intra refresh
- Full slice intra refresh(here we can apply 1,2,3.. … slice refresh)

3.2.1 Half slice intra refresh

In this section we implemented half slice intra refresh per field. The implementation of intra slice refresh flow chart is shown in below figure 3-5, which will give step vice implementation technical details. The time to make complete refresh is 68 fields (1.36sec) made for full version of HD sequence (1920 ×1088). So within 4 seconds 3 times complete intra slice refresh will be done.

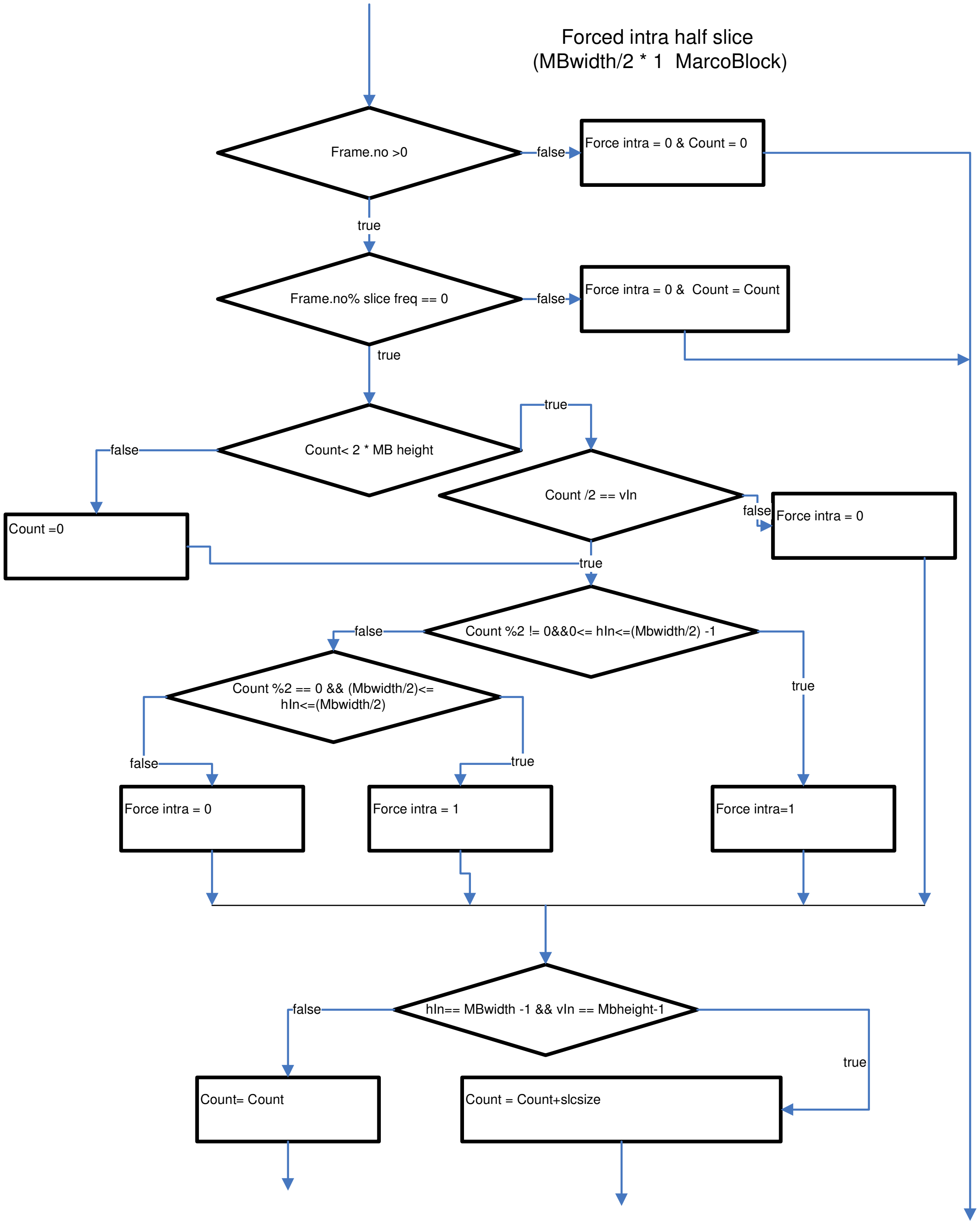

Forced intra half slice
(MBwidth/2 * 1 MarcoBlock)
Frame.no >0
false
Force intra = 0 & Count = 0
true
Frame.no% slice freq == 0
false
Force intra = 0 & Count = Count
true
Count< 2 * MB height
true
false
Count /2 == vIn
false
Force intra = 0
Count =0
true
Count %2 != 0&&0<= hIn<=(Mbwidth/2) -1
false
true
Count %2 == 0 && (Mbwidth/2)<= hIn<=(Mbwidth/2)
false
true
Force intra = 0
Force intra = 1
Force intra=1
hIn== MBwidth -1 && vIn == Mbheight-1
false
true
Count= Count
Count = Count+slcsize

Figure 3-5 shows Implementation of half slice intra refresh

In this the main implementation steps are

- Checking the frame number depending upon the number first frame (I-frame) we do not want to apply because it is already intra frame. Every time even times we want to increment the count number because we will finish one slice in two fields so we increment every even number of frame.

- Check the count number if the count number is more than 2 * MB height (2*34) than make it as zero (0). And check if count number/2 is vertical position than go for next step otherwise don’t force for slice intra for that slice. Next step if the count number is factor of 2 and checks the horizontal position alternative for two parts (left and right side slices).

- If force is 1 than apply force intra for those macro blocks rest of the blocks is to apply normal way and increase the count number when one complete frame is finished.

3.2.2 Full slice intra refresh

In this section we implemented full slice intra refresh per field. The implementation of intra slice refresh flow chart is shown in below figure 3-6, which will give step vice implementation technical details. The time to make complete refresh is 34 fields (0.68 sec) made for full version of HD sequence (1920 ×1088). So within 3 seconds more than 4 times complete intra slice refresh will be done. In this method we can apply how much intra slice per frame we want just like 1, 2, 3, 4 etc. depending upon how many slice we are introduce we indicate the value of slc_freq (slice frequency). If the value of the slc_freq is 1 then one slice will be introduced for one frame.

In this the main steps implementation is

- Checking the frame number depending upon the number first frame we don’t want to apply because it is already intra frame. Next we have to check for slice frequency depending on that we will decide which frame has to done intra refresh.

- Check the count number if the count number is more than MB height (34) or equal than make it as zero (0), and check if count number is equal to vertical position than go for slice intra for that slice.

- If force is 1 than apply force intra for those macro blocks rest of the blocks is to apply normal way and increase the count number when one complete frame is finished.

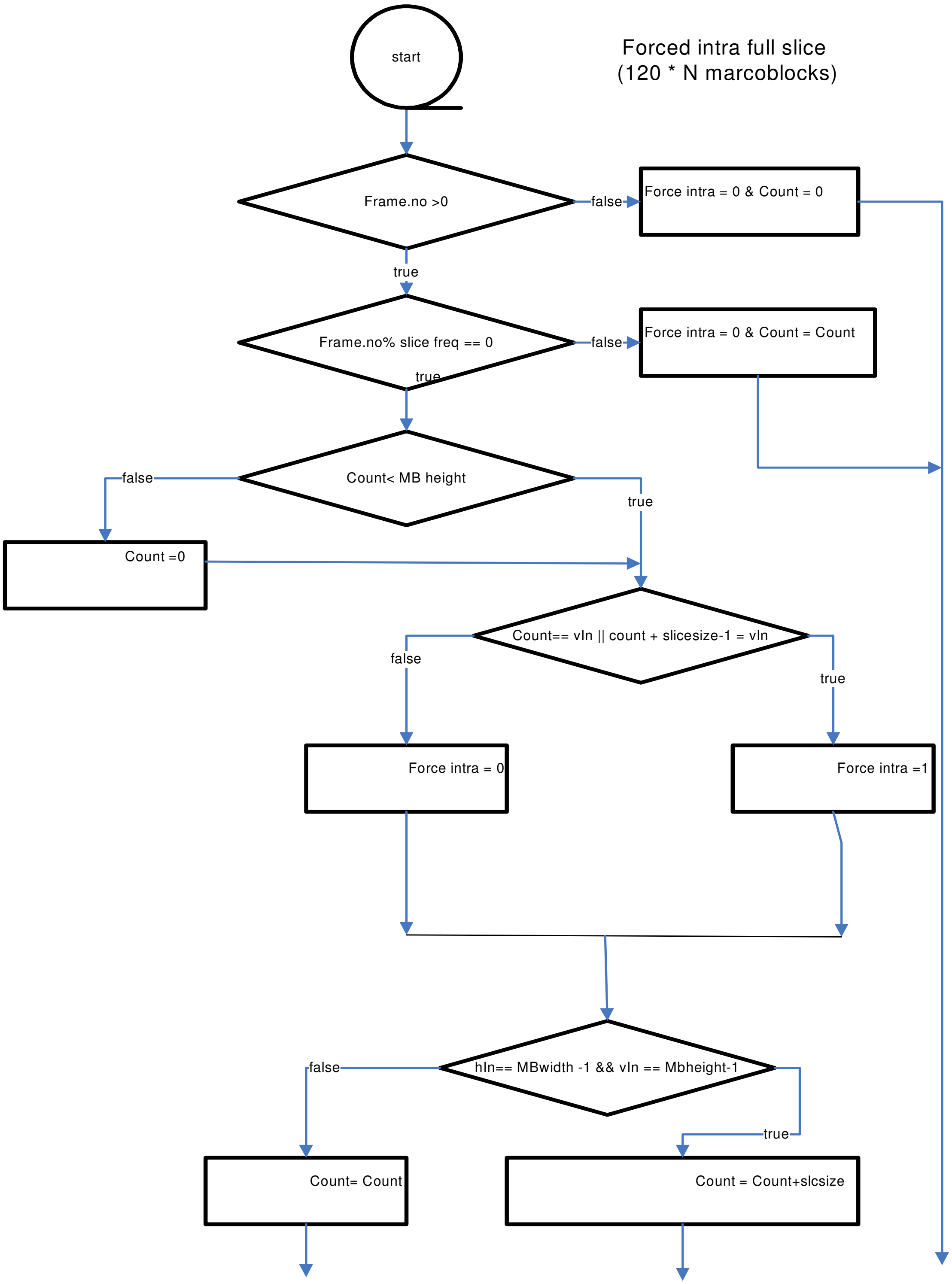


Figure 3-6 shows Implementation of full slice intra refresh

3.3 Observations done with simulations after intra refresh

After implementation we made some simulation work to check results and to finalize which type of intra refresh is more effective for our application for different bitrates. The results of simulation work are shown in below figure 3-7. In this complete test we have taken two different types of HD sequences of 1920 ×1088(we added 8 lines to 1920 × 1080 because 1080 is not a suitable with mpeg standards for division with 16) of intra and it is converted into 1920 × 544 of progressive.

- Parkrun: - parkrun is standard HD sequence in which the sequence starts with a person running in park and he runs for 6 to 7 seconds than he will stop for 2 to 3 seconds. In this starting encoding easy because of less information but later it is very difficult.
- Mobcal: - mobcal is also same as parkrun sequence but in this camera will shoot a calendar than moving train. In this the sequence is difficult in beginning any easy when motion is increased because of less information in the sequence.

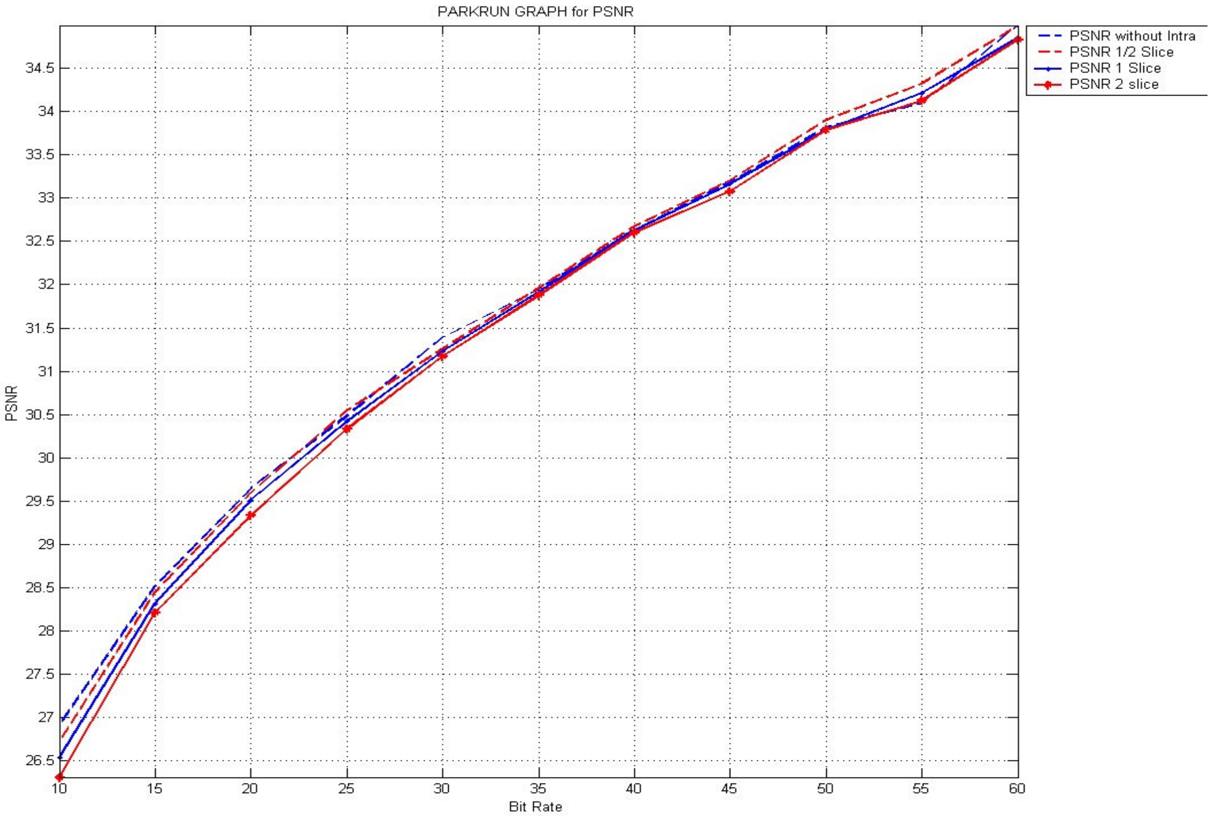


Figure 3-7 Graph between different bitrates and PSNR for different levels of intra slice refresh

We made different simulation on parkrun sequence for different levels of just like without intra refresh, half slice intra refresh, one slice refresh and two slice intra refresh. From this simulation work we came for a conclusion to make refresh in half slice per field or if require one slice per field, because those are really good with compared to two slice refresh per field.

In this application we are looking to adapted 20 to 40 M bits per second. If we check at region with half slice intra refresh is equal to without intra refresh and if we want to refresh faster for difficult sequence we can use one slice refresh, which is perfect to refresh within one second of time so that human eye cannot find any artifact in frames.

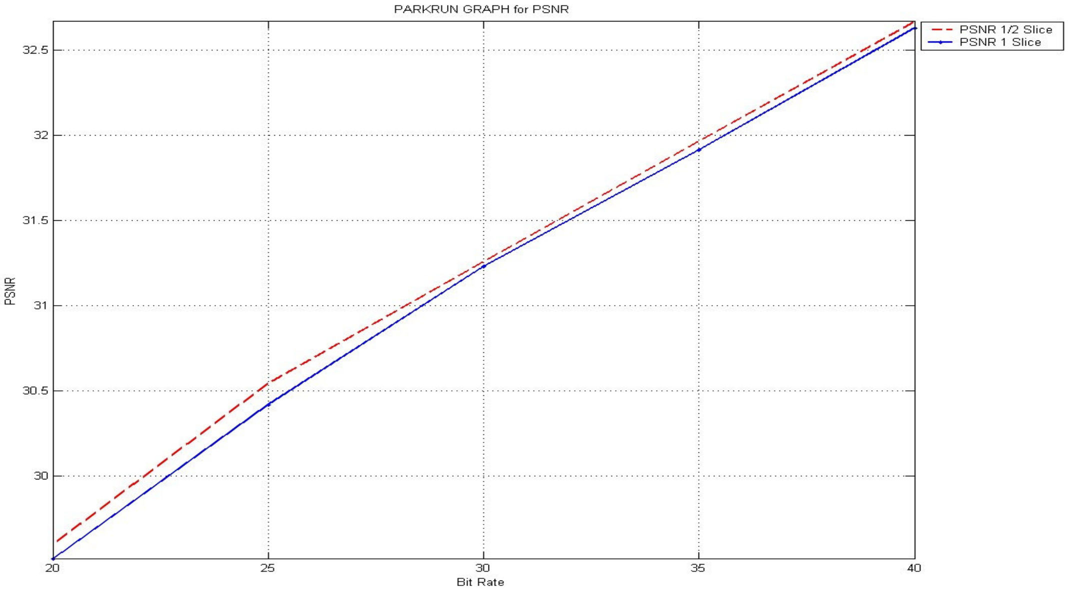


Figure 3-8 Graph show main interesting bitrates for 1/2 and full slice refresh for parkrun

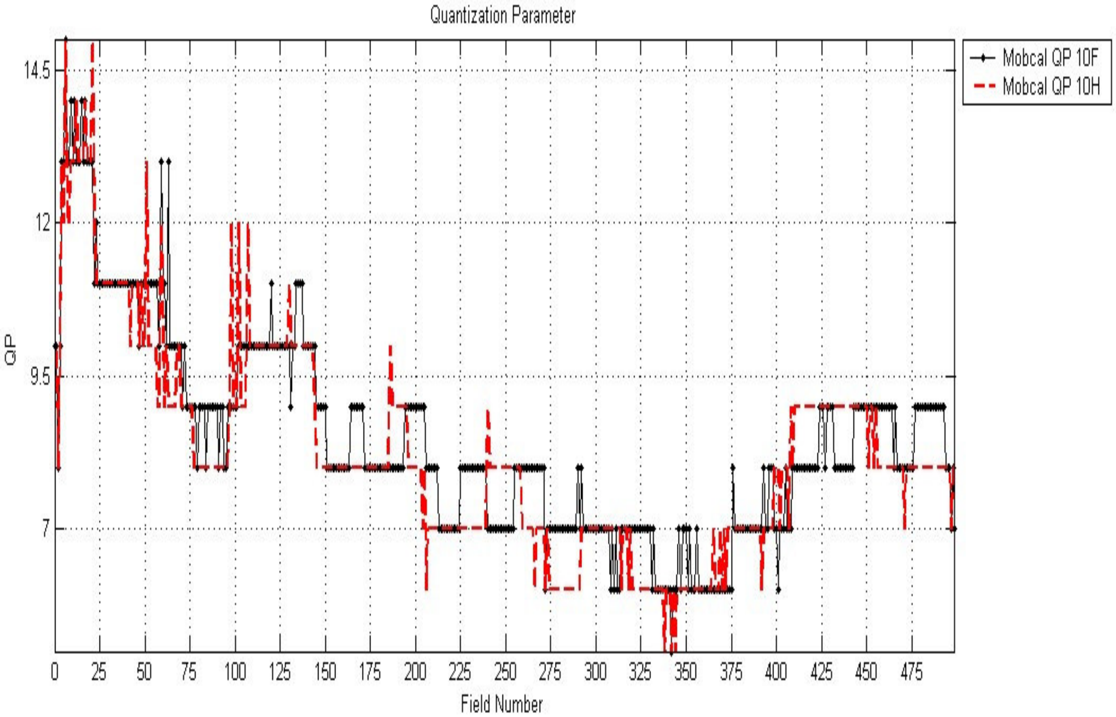


Figure 3-9 Graph is between field number Vs. QP for half slice and full slice intra refresh for mobcal

In the below figures 3-10,3-11,3-12 and 3-13 we can see the QP value for different bit rates at different levels of intra slice refresh.

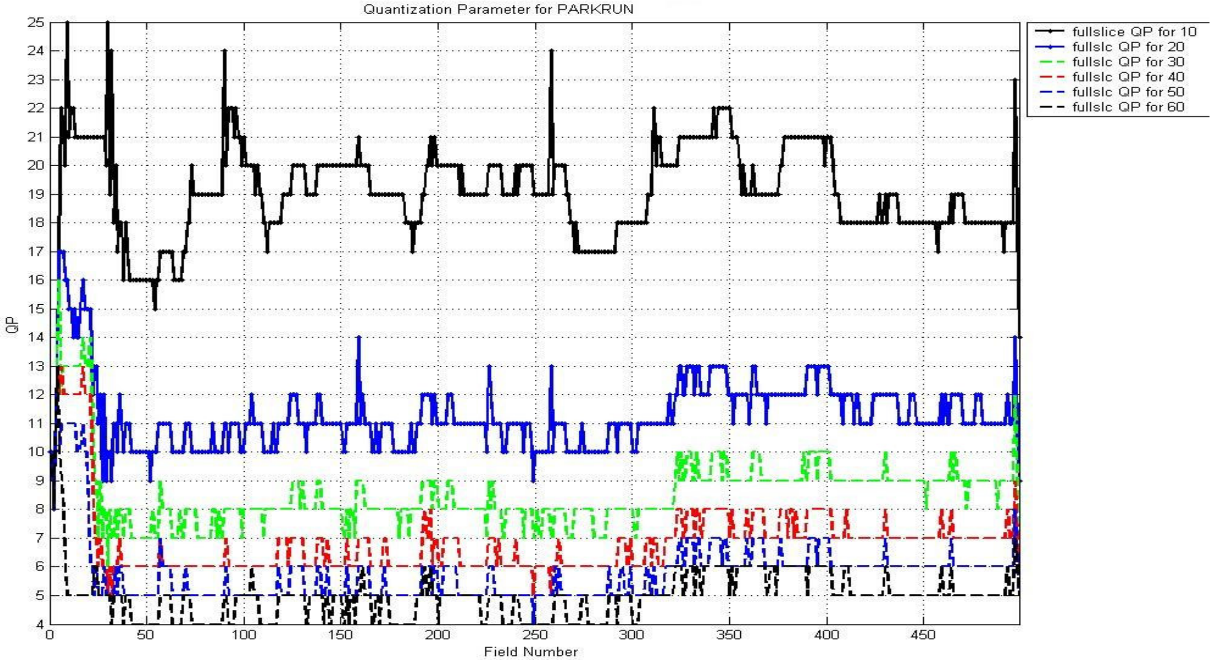


Figure 3-10 Graph between field number vs. QP for full slice intra refresh for parkrun (10 to 60 MBPS)

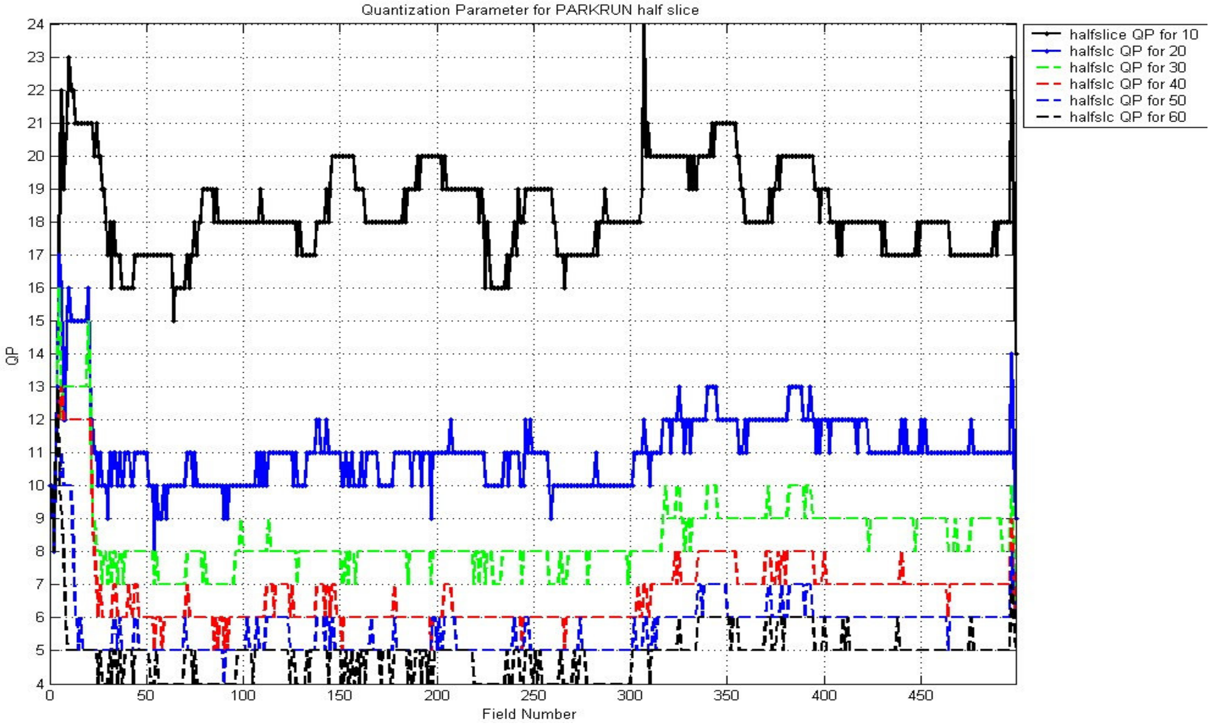


Figure 3-11Graph between frame number vs. QP for full slice intra refresh for parkrun (10 to 60 MBPS)

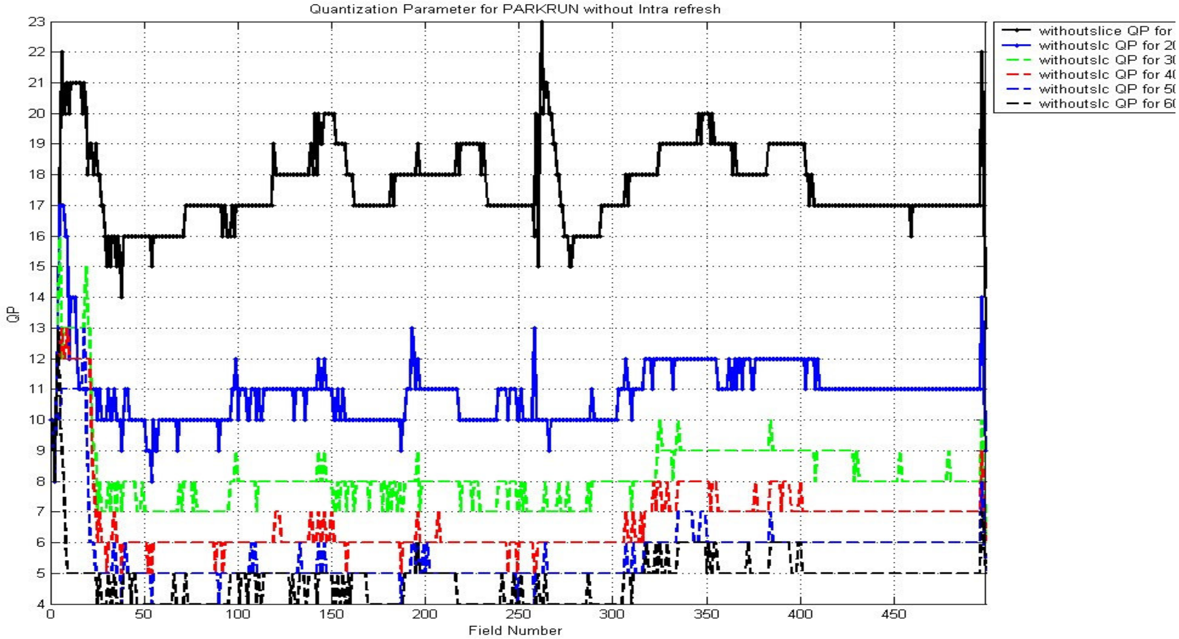


Figure 3-12 Graph between field number vs. QP for without slice intra refresh for parkrun (10 to 60 MBPS)

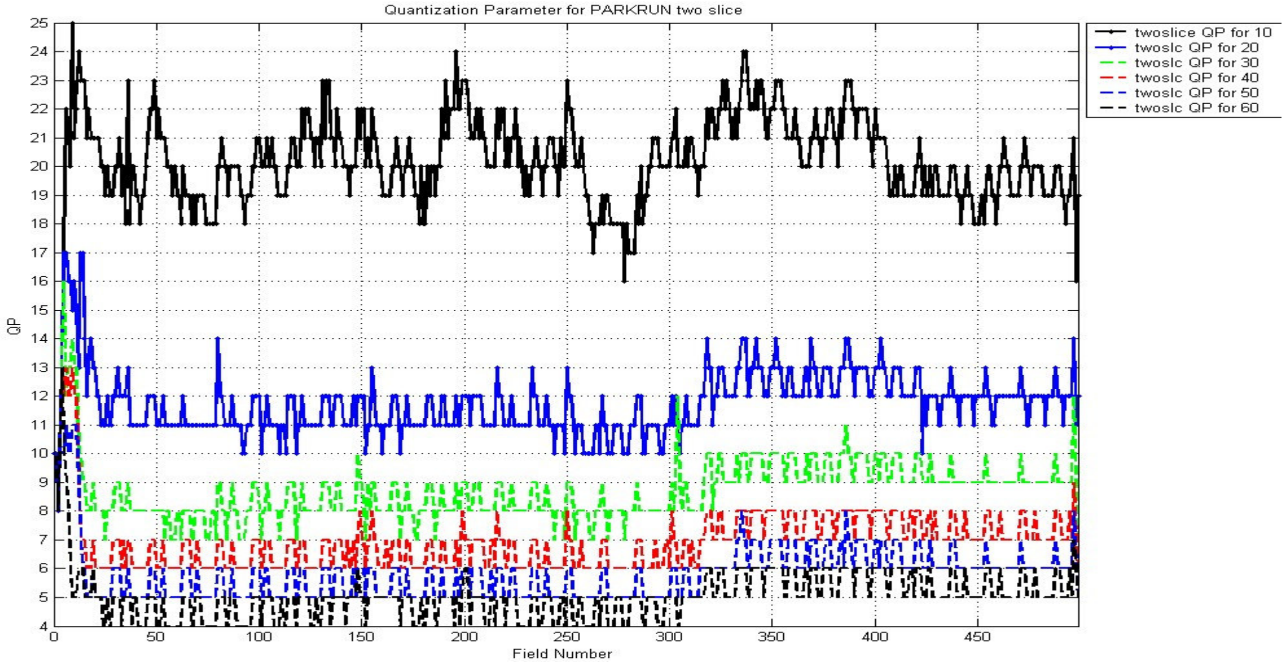


Figure 3-13 Graph between field number vs. QP for two slice intra refresh for parkrun (10 to 60 MBPS)

The above graphs shows how the QP value is applied at different levels of intra refresh for same range of bitrates. The more the data we introduce as intra slice more compression has to do for same bitrate so we apply higher QP than less number of slice frequency. If we want do refresh more frequently than we can use higher refresh frequency otherwise single or half slice is best options for intra refresh for normal conditions.

3.3.1 Square, Band split evaluation

The idea of encoding in multi-stream approach is quite a promising one despite of its several drawbacks. For example in the motion compensation and estimation of the encoder, in which it cannot compensate the motion properly and estimate because if the object in a particular frame moves very rapidly there will be a high residue. The way to reduce the delay of overall system with this approach is good to cut a complete full HD into either Square or Band. For this the single raw data should be cut into pieces before giving it to several encoders to encode.

The cutting of the raw data file is carried out into 4 pieces. As decided it should be either Square or Band split. So the splitting is done before the input is given to an encoder by reading each frame into memory first and then read the required macro block. After encoding and decoding there might occur some artifacts or loss in data due to encoding and decoding each piece separately. After you join all the pieces you have to evaluate the PSNR and draw which is perfectly suitable for this multitasking. It also has gained its importance in this multi-streaming approach.

3.4 Multithreading, Pipes and Forks

3.4.1 Introduction:

The world runs in parallel, but our usual model of software does not. Programming languages are sequential. This mismatch makes it hard to write systems software that provides the interface between a computer (or user) and the world. Imagine that there is a large field to be plowed. It takes a certain amount of time to plow the field with one tractor. If two equal tractors are available, along with equally capable personnel to man them, then the field can be plowed in about half the time. The field can be divided in half initially and each tractor given half the field to plow. One tractor does not get into another's way if they are plowing disjoint halves of the field. Thus they don't need to communicate. Note however that there is some initial overhead that was not present with the one-tractor model, namely the need to divide the field. This takes some measurements and might not be that trivial. In fact, if the field is relatively small, the time to do the divisions might be more than the time saved by the second tractor. Such overhead is one source of dilution of the effect of parallelism.

Rather than dividing the field only two ways, if N tractors are available, it can be divided N ways, to achieve close to an N-fold gain in speed, if overhead is ignored. However, the larger we make N, the more significant the overhead becomes (and the more tractors we have to buy). So we can understand that there will be quite an amount of gain occurred during the multitasking of the same single process. This can be done by either multithreading or by forks and pipes.

Basic Terminology:

- **Process** -A *process* is defined as a particular course of action intended to achieve a result.

- **Multithreading** -*Multithreading* is defined as a processing technique that allows two or more of the same type of transaction to be carried out simultaneously.

- **Fork** - *Fork* is a function in 'C' which is used to turn a single process into 2 identical processes, known as the ***parent*** and the ***child***.

- **Pipe** - *Pipe* is a function in 'C' which is used in a process where the input of one process is made the input of another.

3.4.2 Multithreading

Multi-threading is a term used to describe the facility provided by the operating system that enables an application to create threads of execution within a process. The applications architecture also takes the advantage of the multi-threading provided by the operating system. A thread is basically a path of execution through a program. It is also the smallest unit of execution that Win32 schedules. A thread consists of a stack, the state of the CPU registers, and an entry in the execution list of the system scheduler. Each thread shares all of the process's resources.

A process consists of one or more threads and the code, data, and other resources of a program in memory. Typical program resources are open files, semaphores, and dynamically allocated memory. A program executes when the system scheduler gives one of its threads execution control. The scheduler determines which threads should run and when they should run. Threads of lower priority may have to wait while higher priority threads complete their tasks. On multiprocessor machines, the scheduler can move individual threads to different processors to "balance" the CPU load.

Each thread in a process operates independently. Unless you make them visible to each other, the Threads execute individually and are unaware of the other threads in a process. Threads sharing common resources, however, must coordinate their work by using semaphores or another method of interprocess communication. When an application spawns a new thread, that thread becomes an independent entity inside of the application's process space. Each thread has its own execution stack and is scheduled for runtime separately by the kernel. A thread can communicate with other threads, other processes, perform I/O operations, and do anything else you might need it to do. Because they are inside the same process space, though, all threads in a single application share the same virtual memory space and have the same access rights as the process itself.

Threads let your program perform multiple tasks in parallel. For example, you can use threads to perform several, lengthy calculations while your user interface continues to respond to user commands. You can also use threads to divide a large job into several smaller jobs.

Of course, the performance advantages of multithreading come at the cost of increased complexity in the design and maintenance of your code. Because threads of an application share the same memory space, so you should synchronize access to shared data structures to prevent the application from entering into an invalid state or crashing. Overprotection of your data structures can cause problems too, however. To protect you data you use locks. An overuse of locks may protect your data but could lead to performance that is worse than that of a single threaded application. Striking a balance between performance and protection requires careful consideration of your data structures and the intended usage pattern for your extra threads to accomplish the task.

3.4.3 Appropriate situations for using threads

Before you start using threads in your program, you should look at the problem you are trying to solve and decide if threads are the best solution. In some cases, there may be alternatives to threading that are simpler. The following list shows some of the situations where threads might benefit your program:

- You can avoid blocking your user interface and have work that can be done in the background.
- You can handle multiple I/O sources in parallel.
- You can employ multiple CPU cores to increase the amount of work done by your application in a given amount of time.

Even if one of these situations seems appropriate to you, you should still consider alternatives to threading. Other solutions may exist that provide the same benefits without requiring the use of threads. The following list provides some situations in which threading may not be the best solution:

- An asynchronous API exists to perform the same behavior.
- The duration of work to perform is short (less than a few milliseconds).
- The work you have to do is serialized or does not lend itself easily to parallelization. Note that you might still create one background thread (rather than multiple threads) to process data serially without blocking your user interface.
- The underlying subsystems you are using are not thread safe.

Some reasons to avoid Multithreading in our programming is

- The simplest and easiest way to avoid thread-related resource conflicts is to give each thread in your program its own copy of whatever data it needs. Here in our project we don’t want to modify much of the program structure and data flow by introducing new copies of data which is required by each process unnecessarily.

- Parallel code works best when you minimize the communication and resource contention among your threads. Here we want to use parallel code but does not want to rewrite everything from scratch due to time constrain. So creating a multithreaded application is hard and time consuming.
- Even if you are very careful and lock shared data structures at all the right junctures in your code, your code may still be semantically unsafe. This may also cause blockage of your program and sudden crash during execution. So we want a safe program exit with out failure during execution.

So we have to choose to avoid this kind of programming for achieving our result of multitasking.

3.4.4 Forks and Pipes

The programming for multi-tasking an application has been made simple in UNIX which has later been adopted in windows. In windows you need to introduce some library functions and the same pipes and forks which are quite popular for interprocess communication and creating a multiple process through one single process. The header files which are used from the ‘C’ library are #include <sys/types.h> and
#include <unistd.h>. So the added advantage to use both in UNIX and windows helped us to select and implement forks and pipes made easy to select this method of achieving our task.
So the following figure explain how a single parent process is spawn into a single child process.

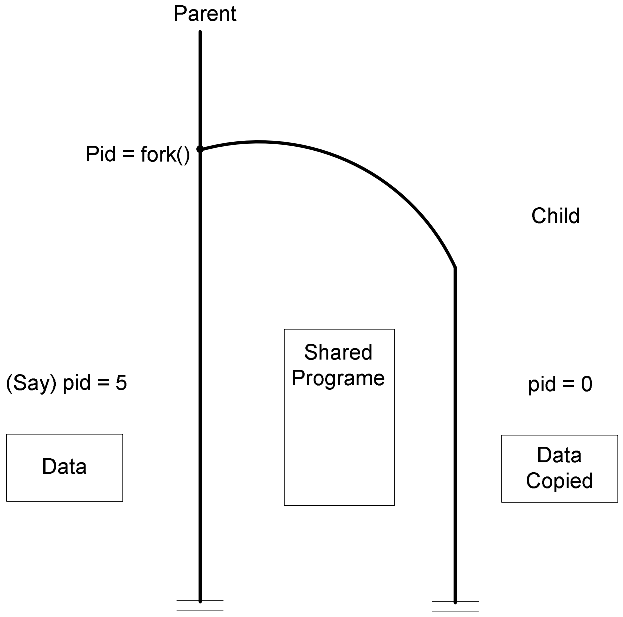


Figure 3-14 forks functionality

What does fork () do?

Forks is quite dominant approach for achieve our task than multithreading. If we fork a process we create a parent and a child. So if we fork several times we can create as many as process which are identical and can also inherit and behave same like the parent process.

Syntax of fork function is: *pid_t fork (void);*

NOTE: The child process will have its own unique PID. PID is nothing but process ID which helps it to tag a particular process. On success, fork() returns 0 to the child process and returns the process ID of the child process to the parent process. On failure, fork() returns -1 to the parent process, sets errno to indicate the error, and no child process is created.

The execution of fork() creates a new *process* (program in execution) that is executing the same code as the program that created it. The new process is called a *child*, with the process executing fork being called the *parent*. The child gets a complete, but independent, copy of all the data accessible by the parent process.

When a child is created using fork, it comes to life as if it had just completed the call to fork itself. The only way the child can distinguish itself from its parent is by the return value of fork. The child process gets a return value of 0, while the parent gets a non-zero value. Thus a process can tell whether it is the parent or the child by examining the return value of fork. As a consequence, the program can be written so as to have the parent and child do entirely different things within the same program that they share.

The value returned by fork to the parent is known as the *process id* (PID) of the child. This can be used by the parent to control the child in various ways. One of the uses of the PID, for example, is to identify that the child has terminated. The system call wait, when given the PID of the child, will wait for the child to terminate and then return. This provides a mechanism for the parent to make sure that something has been done before continuing. A more liberal mechanism involves giving wait an argument of 0. In this case, the parent waits for termination of any one of its children and returns the PID of the first that terminates.

When a programmer is unaware he may disturb the program execution due to mismatching the things. In ‘C’ when using this fork function he has to know just what is and is not inherited by the child. This list may vary depending on UNIX implementation, so take it with a grain of salt. Note that the child gets *copies* of these things, not the real thing.

Features that are inherited by the child from the parent:

- process credentials (real/effective/saved UIDs and GIDs)
- environment
- stack
- memory
- open file descriptors (note that the underlying file positions are shared between the parent and child, which can be confusing)
- close-on-exec flags
- signal handling settings
- nice value
- scheduler class
- process group ID
- session ID
- current working directory
- root directory

- file mode creation mask (umask)
- resource limits
- controlling terminal

Some features which are unique to the child are:

- process ID
- different parent process ID
- Own copy of file descriptors and directory streams.
- Process, text, data and other memory locks are NOT inherited.
- process times, in the tms struct
- resource utilizations are set to 0
- pending signals initialized to the empty set
- timers created by timer_create not inherited
- asynchronous input or output operations not inherited

Communication between Parent and Child

A parent and child can communicate through any of the normal inter-process communication schemes (pipes, sockets, message queues, shared memory), but also have some special ways to communicate that take advantage of their relationship as a parent and child. One of the most obvious is that the parent can get the exit status of the child. To get the communication pipes concept has been introduced here.

Conceptually a pipe can be thought of much like a hose-pipe, in that it is a conduit where we pour data in at one end and it flows out at the other. A pipe looks a lot like a file in that it is treated as a sequential data stream. Unlike a file, a pipe has two ends, so when we create a pipe we get two end points back in return. We can write to one end point and read from the other. Also unlike a file, there is no physical storage of data when we close a pipe, anything that was written in one end but not read out from the other end will be lost.

We can illustrate the use of pipes as conduits between processes in the following diagram:

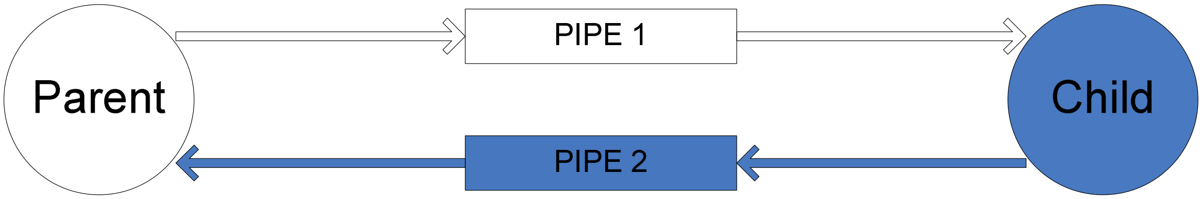


Figure 3-15 communication between parent and child

Here we see two processes which I've called Parent and Child, for reasons that will be because one process is spawning into another with the use of fork command. So such process which is used here is called as Parent and the spawned process is called as child as explained above. The parent can write to Pipe 1 and read from Pipe 2. The Child can read from Pipe 1 and write to Pipe 2.

Note that each pipe can be used to send a request or return data depending upon which process is initiating the *transaction*. This gives us an interesting challenge in naming the pipes which we will discuss a little later.

What does pipe () do?

Another innovation of UNIX was the development of pipes, which gave programmers the ability to string together a number of processes for a specific output. Piping is done if there is more process which has to communicate individually to a single parent. Here we also have two (or more) forked processes which need to communicate between them with out overlapping any data. So we have to first open a *pipe*. 'C' programming in UNIX allows two ways of opening a pipe.

popen() -- Formatted Piping

FILE *popen(char *command, char *type) -- opens a pipe for I/O where the command is the process that will be connected to the calling process thus creating the *pipe*. The type is either ``r'' - for reading, or ``w'' for writing. popen() returns is a stream pointer or NULL for any errors. A pipe opened by popen() should always be closed by pclose(FILE *stream).

We use fprintf() and fscanf() to communicate with the pipe's stream.

pipe() -- Low level Piping

int pipe(int fd[2]) -- creates a pipe and returns two file descriptors, fd[0], fd[1]. fd[0] is opened for reading, fd[1] for writing. pipe() returns 0 on success, -1 on failure and sets errno accordingly. The standard programming model is that after the pipe has been set up, two (or more) cooperative processes will be created by a fork and data will be passed using read() and write().

Pipes opened with pipe() should be closed with close(int fd).

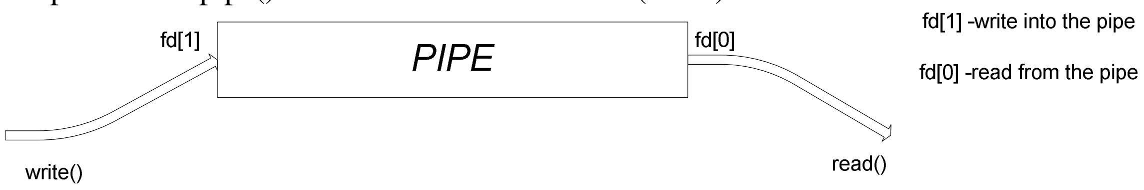


Figure 3-16 Pipes functional Architecture

Here we made a selection and used the Low level piping, but how can we implement two way communications between one server and several clients?

It is possible that more than one client is communicating with your server at once. As long as each command they send to the server is smaller than PIPE_BUF (If each piece of data written to the pipe is less than PIPE_BUF in size, and then they will not be interleaved.

However, the boundaries of writes are not preserved; when you read from the pipe, the read call will return as much data as possible, even if it originated from multiple writes.), they can all use the same named pipe to send data to the server.

All clients can easily know the name of the server's incoming FIFO. However, the server can not use a single pipe to communicate with the clients. If more than one client is reading the same pipe, there is no way to ensure that the appropriate client receives a given response.

A solution is to have the client create its own incoming pipe before sending data to the server, or to have the server create its outgoing pipes after receiving data from the client. Using the client's process ID in the pipe's name is a common way to identify them. Using FIFOs named in this manner, each time the client sends a command to the server, it can include its PID as part of the command. Any returned data can be sent through the appropriately named pipe with which we can solve the problem.

Implementation of Forks and Pipes for Multitasking using on Single Encoder

We have a single Encoder Process which encodes the complete video sequence. Now the figure 3-17 gives you a clear description of how we need to use the pipes and forks for multi-tasking.

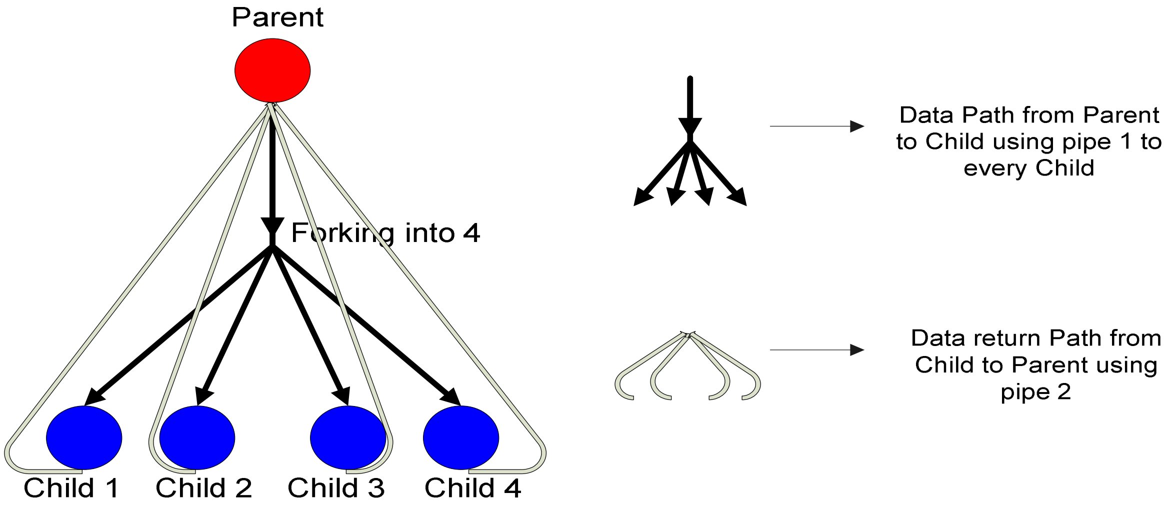

Parent
Forking into 4
Child 1
Child 2
Child 3
Child 4
Data Path from Parent to Child using pipe 1 to every Child
Data return Path from Child to Parent using pipe 2

Figure 3-17 Parent and Child connections using pipes

A parent process (Encoder here) is forked to create several processes which are identical to the parent. The parent work is to get the data from the input and supply data safely to the child and get the output from all the children safely without overlapping any memory locations and writing back to a file. So we need to design a channel using pipes for safe communication. Keeping this in mind we had started using the pipes and forks. For data safety as explained used two pipes for each child. Of which one pipe is used to receive the input and the other is used to send in the output to parent. Thus the safety for the data has also found the solution for the given problem with ease to use multiple encoders working simultaneously.

3.5 Rate control

The DCT coefficient quantization, run-level and VLC coding produce a Variable Bit Rate (VBR) which depends on the complexity of the picture information and the amount and type of motion in the sequence. To achieve a Constant Bit Rate (CBR), which is required for transmission over a fixed bandwidth system, a buffer is employed to smooth out the bit rate variations. In order to achieve the requested CBR avoiding critical conditions of buffer overflow and underflow, a control feedback is applied to the coding process. We have different model of rate control, first we will see about Q2 model of rate control.

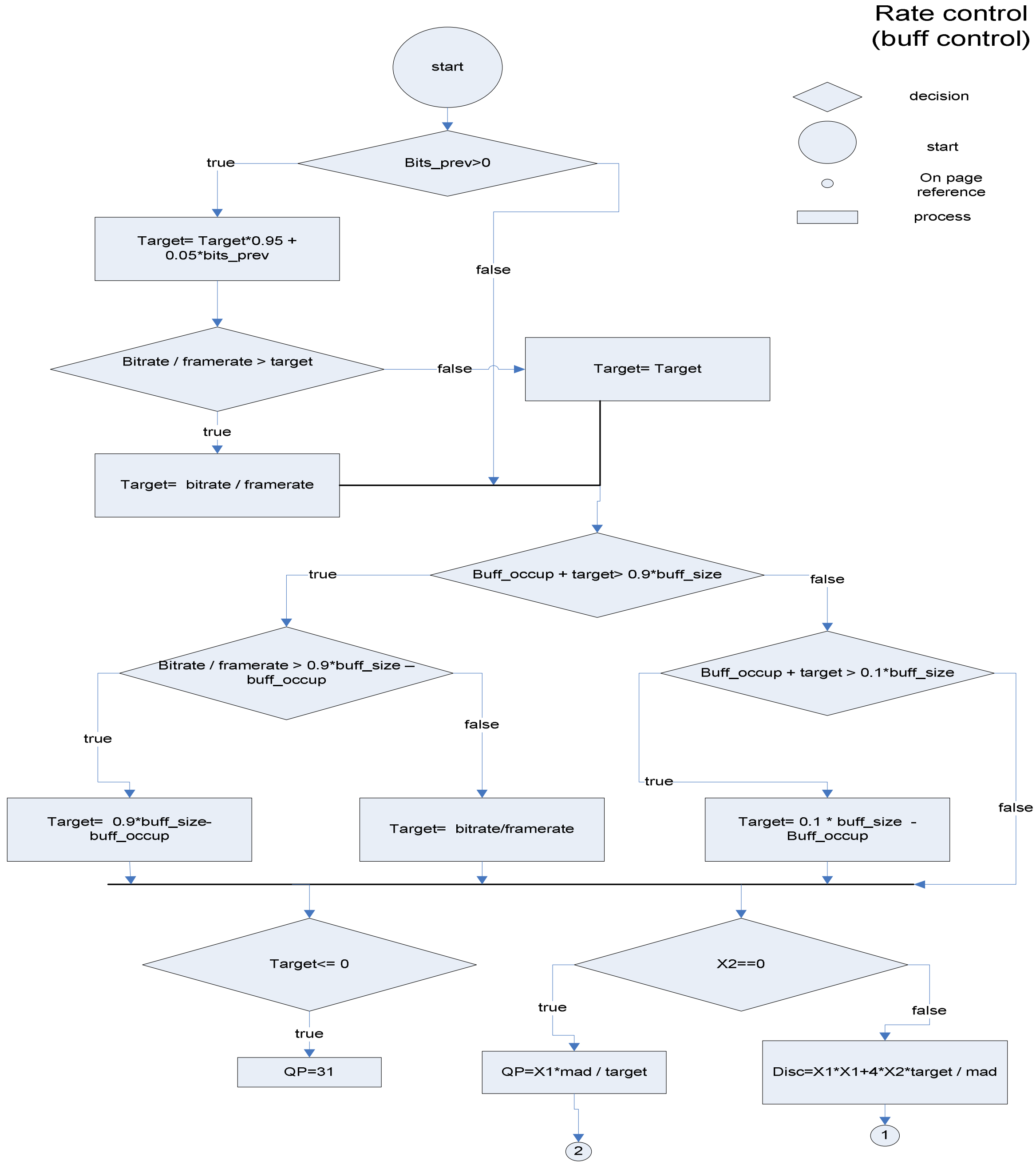

Rate control
(buff control)
decision
start
On page reference
process
start
Bits_prev>0
true
false
Target= Target*0.95 + 0.05*bits_prev
Bitrate / framerate > target
false
Target= Target
true
Target= bitrate / framerate
Buff_occup + target> 0.9*buff_size
true
false
Bitrate / framerate > 0.9*buff_size – buff_occup
Buff_occup + target > 0.1*buff_size
true
false
true
false
Target= 0.9*buff_size-buff_occup
Target= bitrate/framerate
Target= 0.1 * buff_size - Buff_occup
Target<= 0
X2==0
true
true
false
QP=31
QP=X1*mad / target
Disc=X1*X1+4*X2*target / mad
2
1

Figure 3-18 Q2 model for rate control (1)

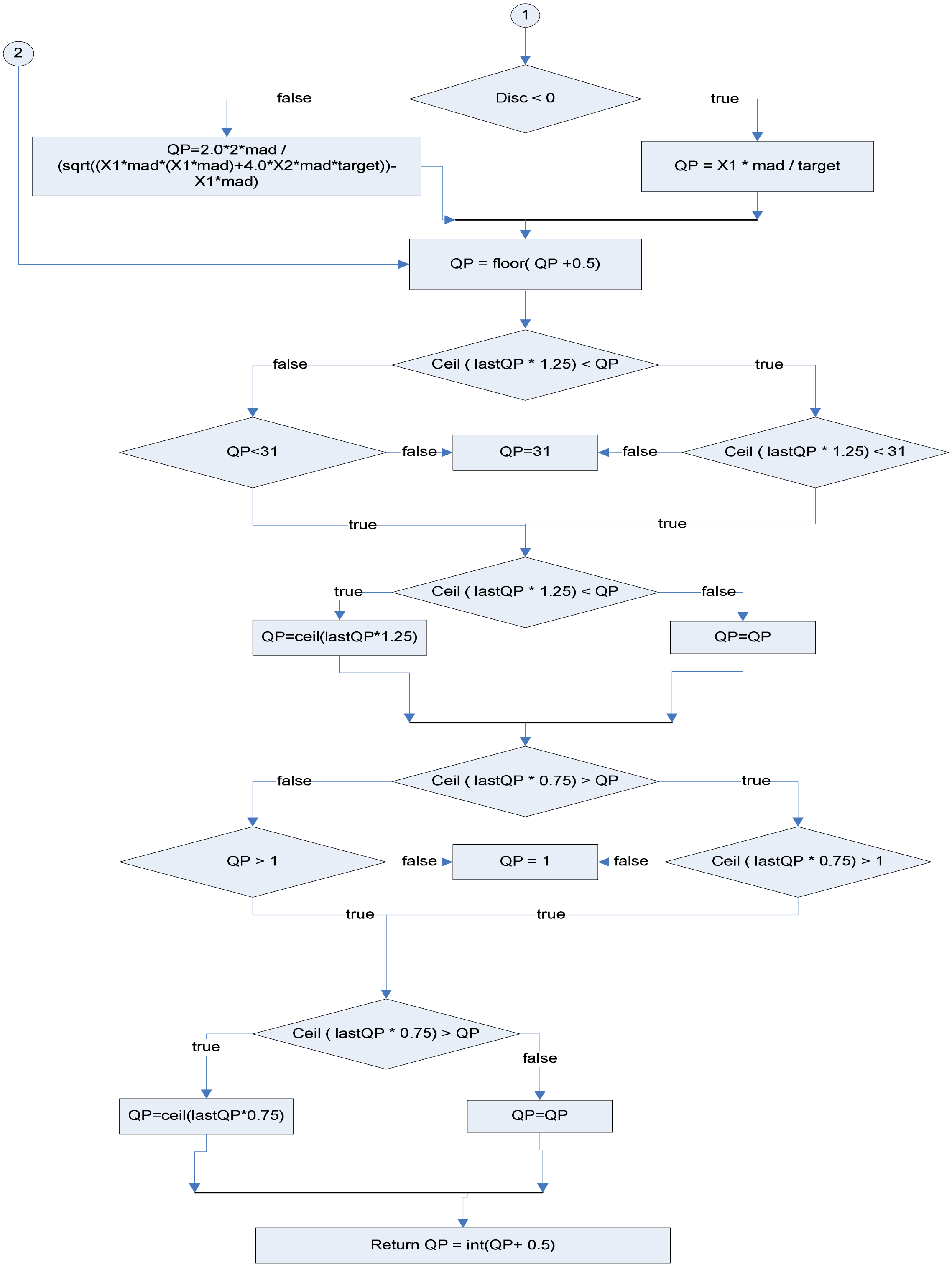


Figure 3-19 Q2 model for rate control (2)

The main aim of the rate control is to calculate the QP which is major factor for compression. This is calculated depending target bit rate, buffer occupancy last QP and mean absolute difference. The QP range is 1 to 31.

This model is used to calculate QP in normal way where we will how the buffer occupancy and what is bit rate we are apply. The main to handle quantization parameter is to maintain constant quality as much as possible and try to avoid buffer underflow and overflow condition.

The main problem with buffer overflow is we lose information which will be worst thing in encoding a sequence. Once we lost the information in overflow we cannot recover easily so we need refresh one complete cycle. So in this condition the human eye can catch artifacts easily. In ours design of rate control first major point is do not overflow at any situation, next is to try to maintain constant quality while encoding.

The main problem with underflow is we lose our bandwidth. So we will try to avoid this problem for effective usage of bandwidth, but it is not a major problem to consider as overflow. If possible we can try to avoid underflow condition.

The Q2 having some standards just like shown in flow chart where we calculate target bit per field and QP value.

- In calculation of target bits per field are having some calculations. 95% of present number and 5% of old bits.
- Target bit rate is bit rate / frame rate and compared old target rate and select maximum value.
- Than check with buffer occupancy, if buffer is going to overflow than select previous target otherwise 90% buffer size – buffer occupancy.
- Once we calculate the target than next step is to calculate QP value. If the target is less than zero than QP should be maximum (31) without any calculations.

Otherwise calculate some factor values(X1, X2, MAD) than using this QP calculation is done. Where X1 and X2 are constants, Mad is mean absolute difference.

Once QP is calculated first we will check with last QP because to maintain approximate constant quality, so for this we need to check change of QP not more than 25% QP compared to last QP and at the same time we need to check for boundaries of QP(1 to 31).

Our main aim is to design low delay design of MPEG4 so we need to minimize the buffer size as much as we can that is 40ms according to MPEG standards, but the basic model of MPEG4 and Q2 model is working on 500ms delay. So we need to change the control methodology. The buffer occupancy of Q2 model for different buffer sizes are shown in below figure 3-20 and 3-21. In those figure 3-20 is for 40ms buffer occupancy and 3-21 is for 500ms, which is made on field number vs. buffer occupancy.

In those figure we can see clearly about buffer underflow and overflow conditions for 40ms. Where we can see buffer overflow and underflow which are the two things we need to from the buffer control methodology. So we changed the methodology.

The buffer size of 40ms is 1.2Mbit and 500ms is 15Mbit, with those sizes we can see clearly underflow and overflow in 40ms delay buffer.

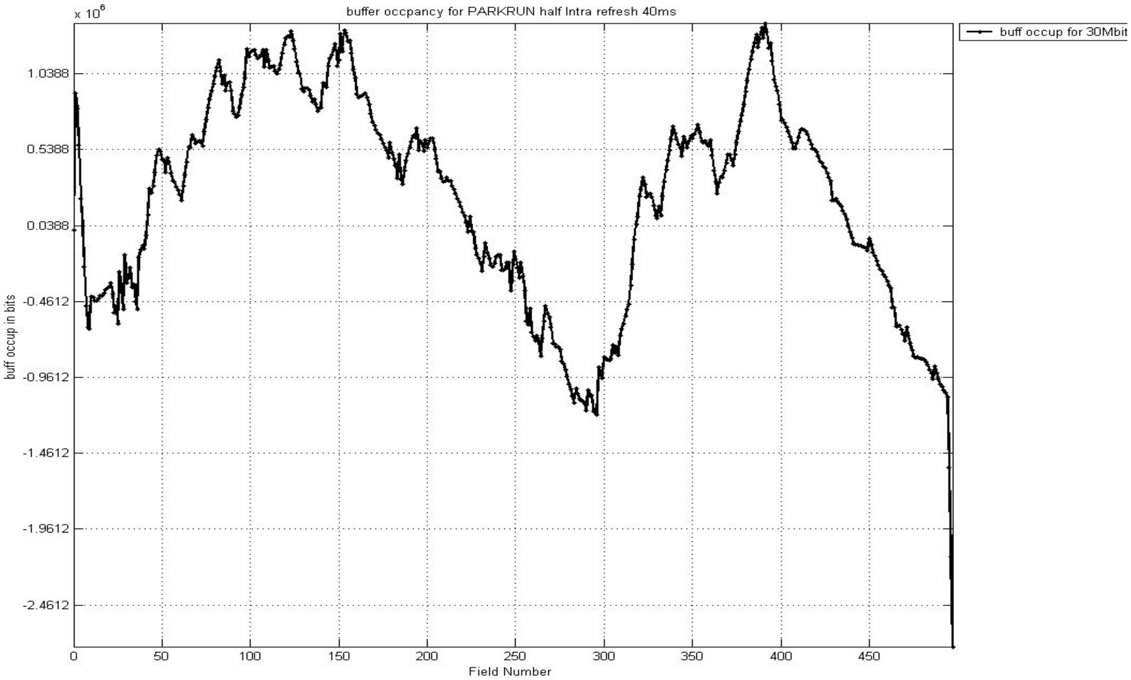


Figure 3-20 Graph between field number vs. buffer occupancy at 30 M bit per second at 40ms delay for parkrun sequence

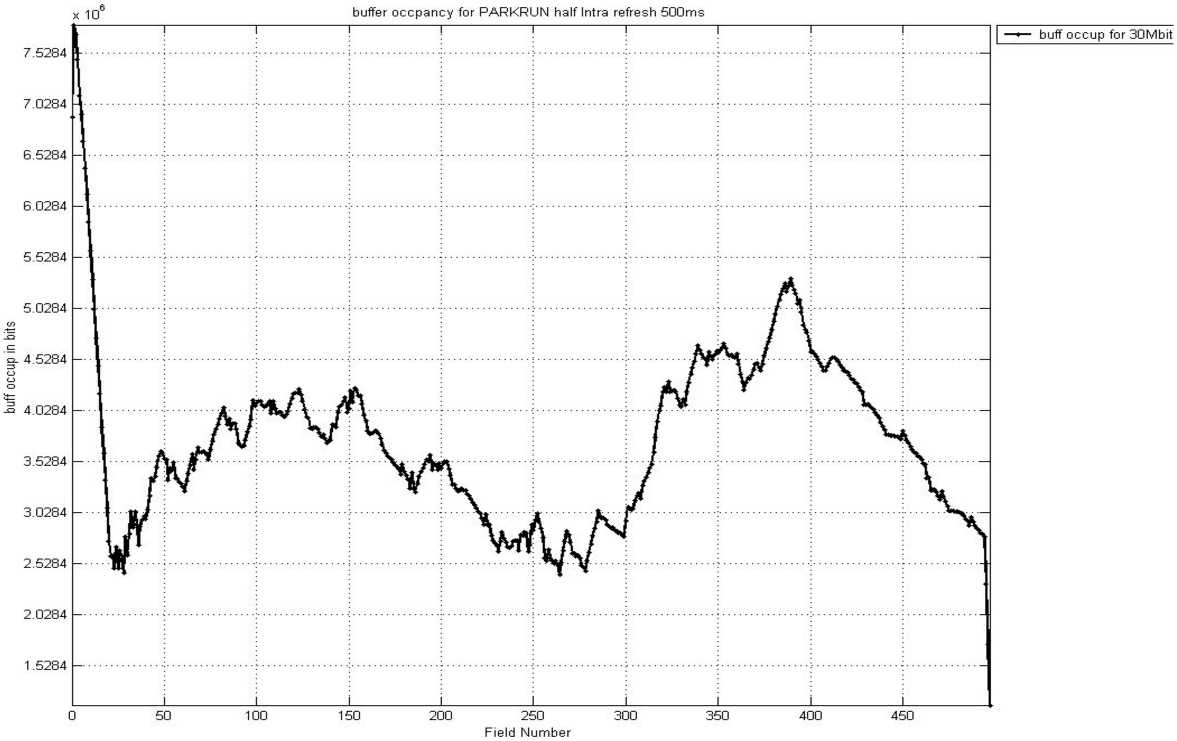


Figure 3-21 Graph between field number vs. buffer occupancy at 30M bit per second at 500ms delay for parkrun sequence

With some more modifications in stepwise QP calculations with reference to buffer occupancy made some emergency conditions like if buffer is going to overflow than increase the QP up to twice the old QP and if it is underflow than 50% of old QP.

The results of the simulation on Q2 model for the parkrun sequence are shown below.

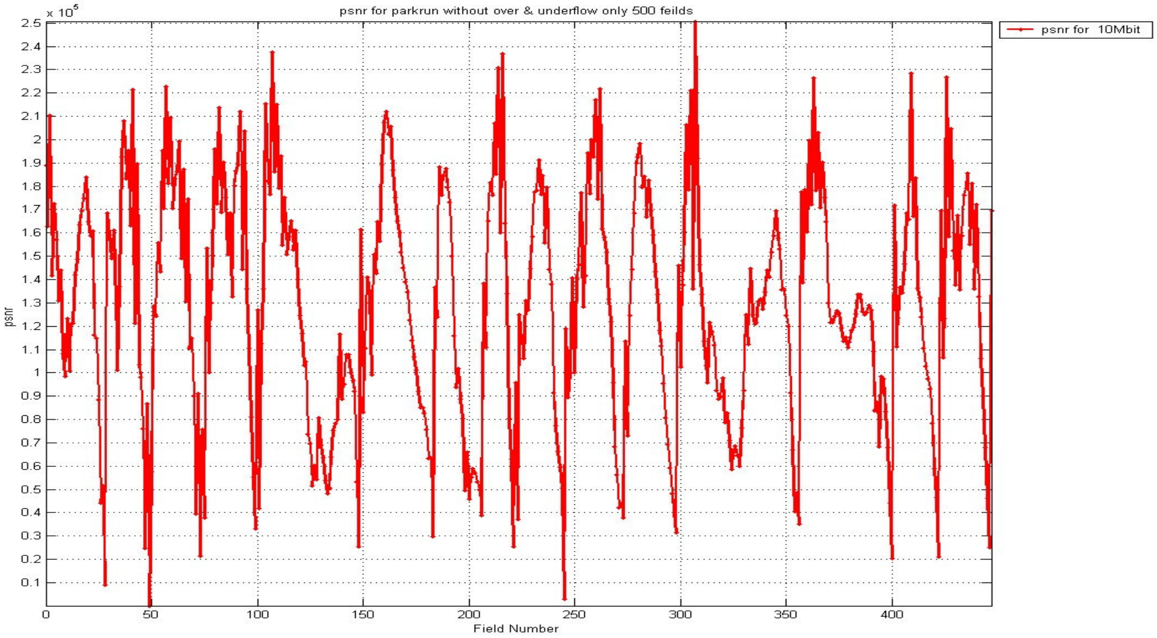


Figure 3-22 Graph between field number vs. buffer occupancy after step change in Q2 model

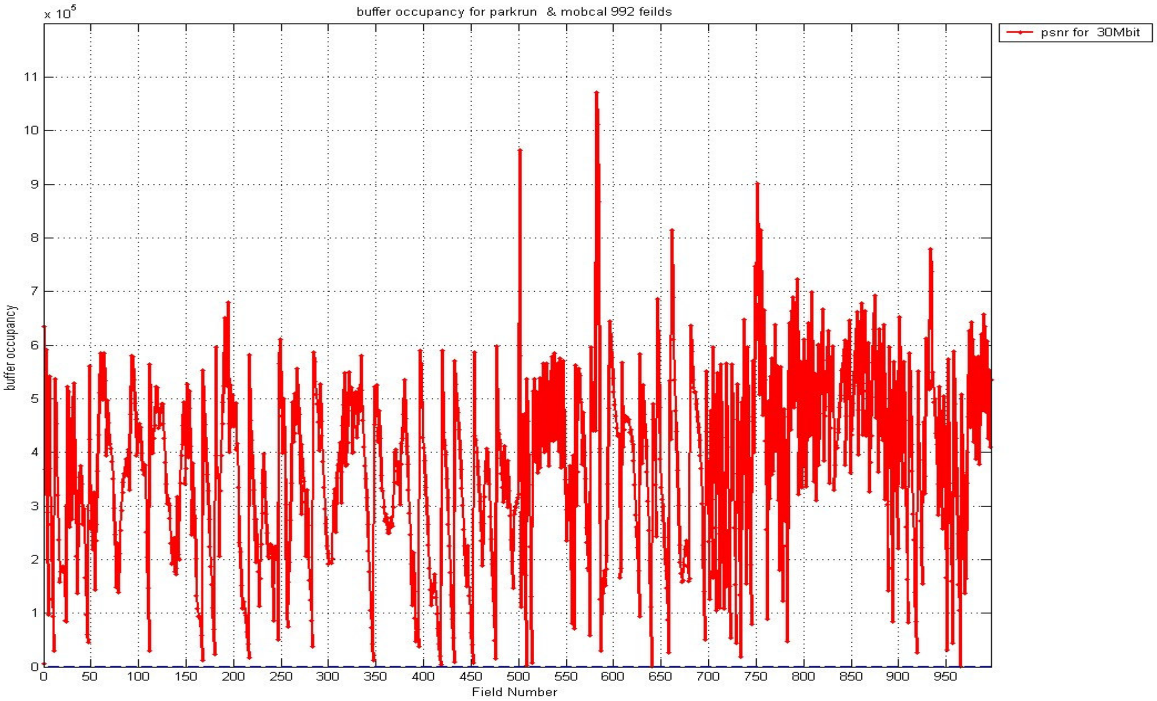


Figure 3-23 Graph between field number vs. buffer occupancy at 30Mbit for mixed (mobcal and parkrun) sequence

The change in step sequence of QP compared with last QP is only worth full for certain sequences. Just like single sequence without any change in sequence or if there is any change if it is not immediate change than also it works successfully. If the sequence is changing continuously and we are not able to control with this methodology for this case we design slice encoding. The slice encoding will identify the sequence and buffer level at each slice, in order to act on the QP as soon as possible.

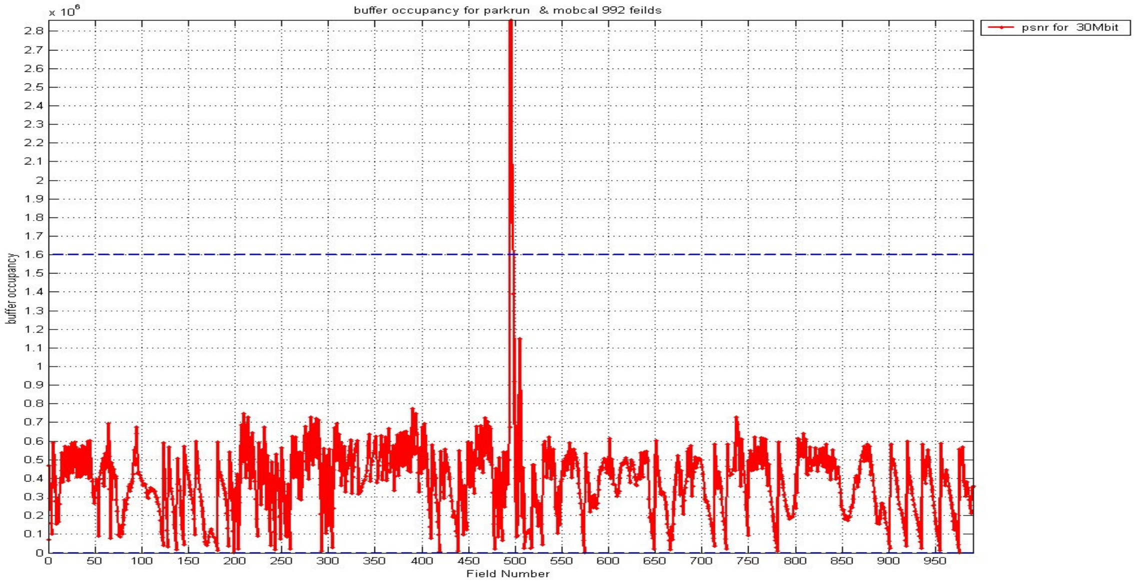


Figure 3-24 Graph between field number vs. buffer occupancy of mixed sequence with sudden change (without any gray fields in between) for 30 Mbit

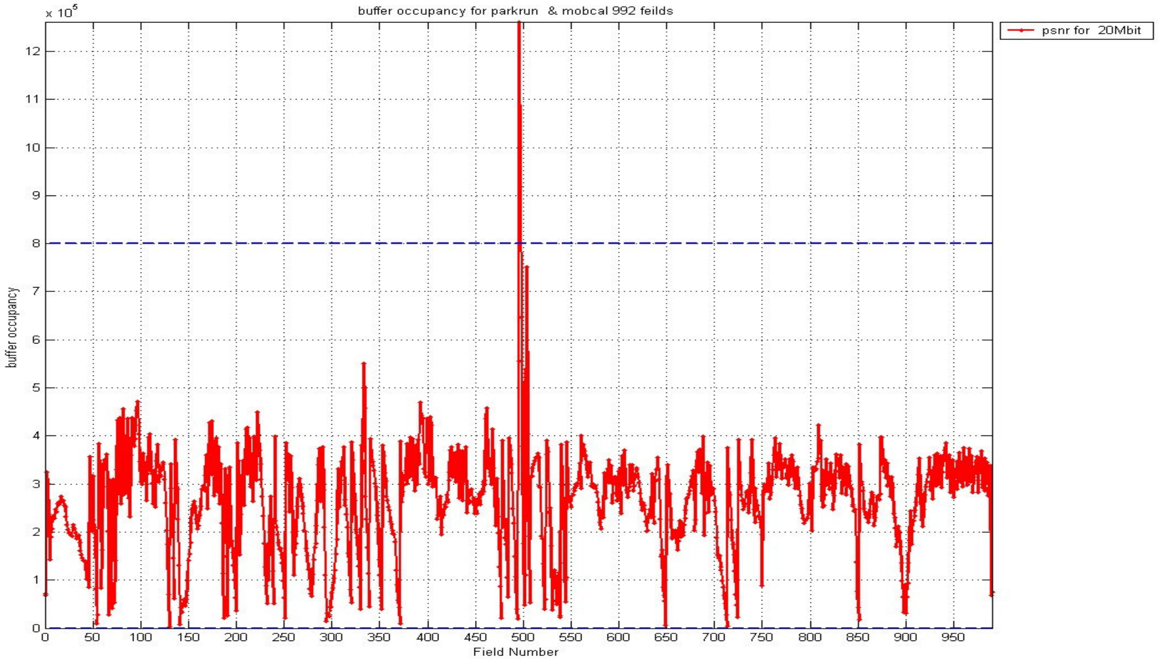


Figure 3-25 Graph between field number vs. buffer occupancy of mixed sequence with sudden change for 20Mbit.

The figure 3-24 and 3-25 graphs show how the sudden increase buffer fullness when sudden sequence changes. The mobcal sequence is having less information than the parkrun sequence when sudden change from mobcal sequence to parkrun the main problem is we do not have common sequence so we get more residual so our rate control methodology change cannot identify because it acts only on field. That is why we want to apply rate control in slicewise so that we can get information as soon as possible at each end of slice with this information we can control at buffer overflow condition.

3.5.1 Slice encoding

Slice encoding is more effecting strategy compared to previous effects, so for a worst condition like mixed sequence (mobcal and parkrun). Here we change the QP per slice with concerning to buffer fullness, so that we can get clear information after completion each slice. With this information we can easily identify the sequence change and if the buffer is going towards overflow than we can act on emergency condition where the QP will very strong to act on sequence to control overflow.

The main disadvantages of this type of implementation is the QP is changing per slice depending upon the sequence toughness, so in single frame we may find different compression so human can identify those artifacts some times. We implemented special rate control for the application. We used P and D^3 (one proportional and three derivates). With this change we can find the difference between QP applied at different levels of sequence. The figure 3-26 and 3-27 graphs shows the different effects in QP and buffer control in two different methodologies for rate control.

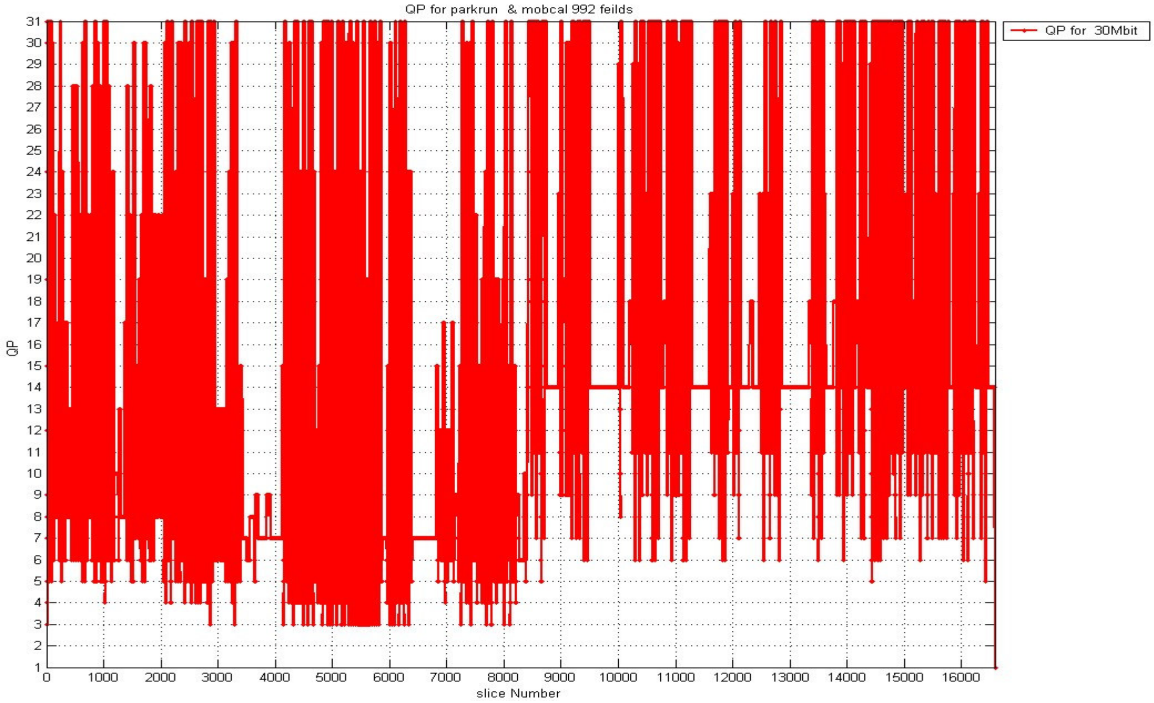


Figure 3-26 Graph between slice number vs. QP after slice update of QP value.

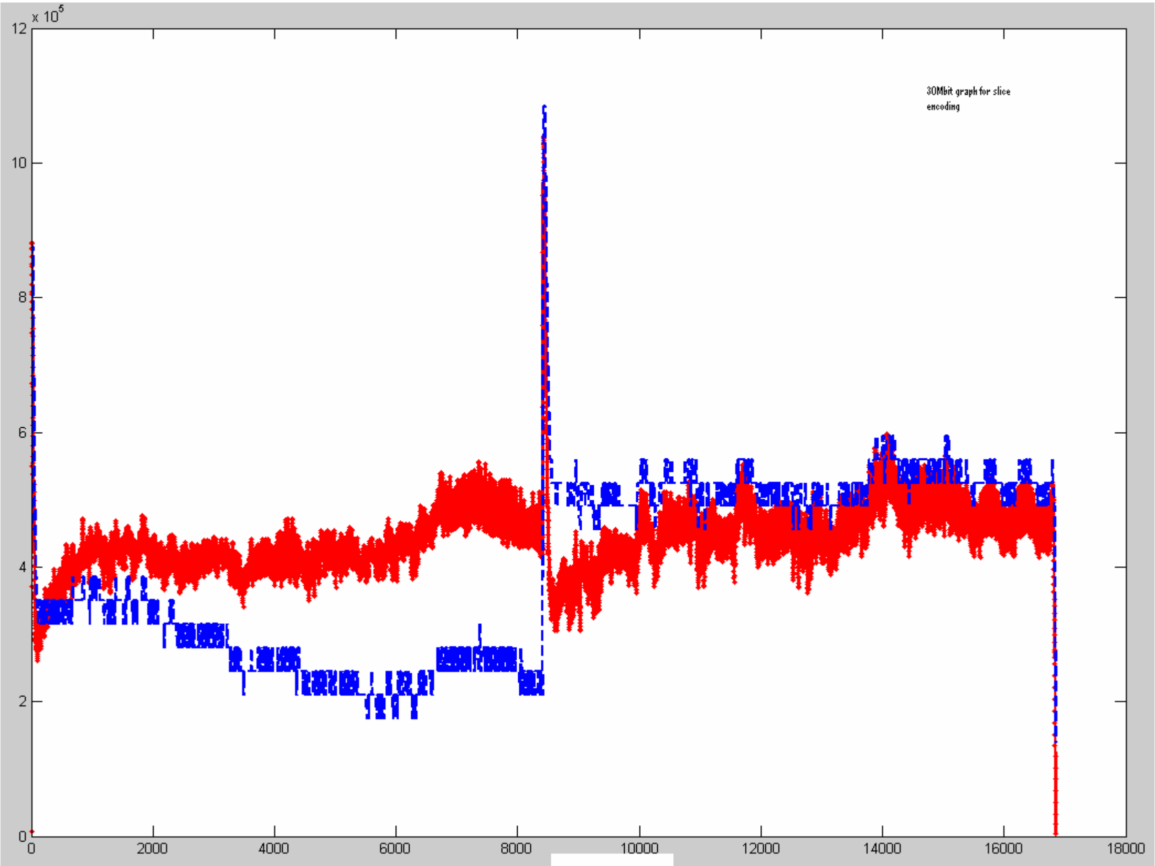


Figure 3-27 Graph between slice number vs. QP and slice number vs. buffer occupancy with different scales after encoding slicewise with proportional and derivate approach.

Figure 3-26 shows the QP change depending upon residue and buffer occupancy.

Figure 3-27 shows the QP and buffer occupancy for slice encoding without much variation in sequence after (P+ D^3).

Here we can see clear factor that the variation in QP for different slice which can effect output sequence quality which is the major factor to be considered in video compression.

Chapter 4

4 Hardware Implementation on FPGA

4.1 Introduction

Fourth generation terminals and mobile networks will be available within the next few years. Due to the heavy demand in market there is an increasing attention which has been drawn especially to the processing of digital video sequences over the last few years. As VLSI technologies have advanced, the processing power of general-purpose processors has increased dramatically.

Real-time video processing applications tend to be implemented nowadays by software design with the help of a powerful processor. However, this is not a complete solution to fully fulfilling the real-time requirement. However, there is a trade-off between hardware and software implementation. Various factors, such as processing speed, flexibility, power consumption, and development cost, should be taken into account. In general, hardware implementation takes lot of resources like time and money which are the important aspects for a company. To save this time and cost, a model is first prepared in the software. Using the performance and optimizations done in software a new architecture is developed for hardware. Generally software has the capability of adding a pre-processing block for handling the input noise components easily than in hardware.

This chapter presents the design of a multi-stream approach with low delay for an HDTV application, the main objective of this thesis work. The application is an algorithmic-software implementation written in ‘C’. The architecture designed should be complaint with the decoder which can decode the encoder output. The implementation of the hardware model for the specific application makes use of the Xilinx MPEG-4 Simple profile in software and use the background knowledge introduced in the preceding chapters. In order to take full advantage of both hardware and software implementations, Xilinx has designed its own video codec. This Video Codec has several functional modules of hardware which is implemented in software in the form of simple ‘C’ functions which will handle the input and output through arguments in a function. The salient feature of this kind of design of software video codec for hardware modules when used in Xilinx IP tool is converted into equivalent VHDL. This VHDL code when used in any professional hardware design tools produce the appropriate hardware modules.

The MPEG-4 software design of Xilinx is suitable for reducing the cost and time for effective implementation of hardware. So the obtained VHDL code is transferred to the FPGA by synthesizing and then place and route. The proposed software design for this project is implemented and tested in real-time using the Hardware. Figure 4-1 shows the main board which is of a detachable part used for live cameras.

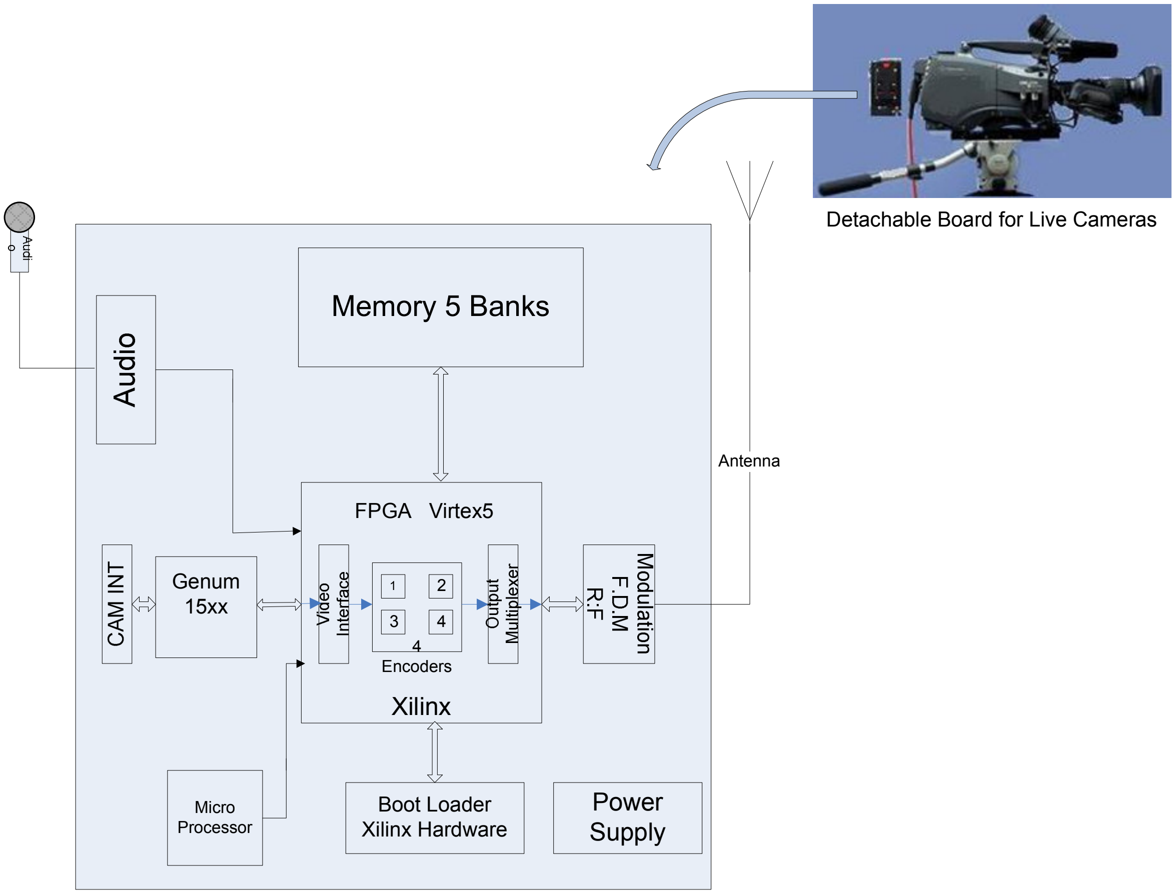


Figure 4-1 Main board design for HDTV application

Easy to Use Design : The end product which is made can be very easy to use. It is just a detachable piece for any live camera. Coming to the inner view of the piece has several main components of which you have an FPGA, power supply, FDM modulator, Micro Processor, Boot loader by Xilinx and the audio and video interfaces along with 5 blocks of memory. The design of four-encoders with multiplexers for switching with input and output interfaces are finished first in the FPGA. Then keeping in mind the power usage of the device the power supply is fixed. Then the board is supplied with a micro processor and a boot loader which are used to reset and load the required things when power is switched on. The most important things on board are handling the easy plug in user interface used for the input and output through several ports. There are 5 blocks of memory designed to significantly handle and process the data without any trouble.

4.2 FPGA implementation

In the complete hardware implementation our concern is mainly about the design of 4 encoders in the FPGA for efficient use of resources and optimal implementation of the design. In FPGA we are handling the input interface of live video sequence and the four encoders and the output which is multiplexed not to overlap during the streaming of data.

Input interface: In the input interface we have several modules of which the most important are sync detector, pattern generator, a chroma resampler, a memory controller and a dispatcher. Here the input is a 4:2:2 which are converted to 4:2:0 for our codec MPEG-4 SP requirements. The sync detect is used to detect the input first before sending it to the encoders. It detects the start of a new frame or a start of a new macro block. The pattern generator is used to reset the hardware and clear the buffers by generating some test patterns for checking the hardware after the power is switched on. As the data pass the two important steps, the luma is passed directly to the FIFOs for handling it with the memory controller later.

The chroma which is left is then passed into the chroma resampler which is also a Xilinx IP core where the chroma samples are made into 8 bits from 10 bits. The memory controller checks the input data which is handled to encode with the help of the FIFOs. There is a dispatcher which checks the FIFOs whether there is one macro block of data and sends it to the appropriate encoder for encoding.

After the data is encoded it is collected into some small FIFOs where a multiplexer works very fast to clear the FIFOs and create the elementary stream. This elementary stream is then combined with a header and made into a transport stream and then passed on to the channel for distribution to the clients. In this design as we have a transmission work, which is carried out with the RF modulator.

4.3 Power constrains

The Power constrains of the Targeted application is less than 7 watts. We are going to use Virtex5 of Xilinx which is developed in 65nm scale and has better performance than any other FPGA of this generation when compared.

4.4 Area requirements

The Virtex5 has lot of resources for our design of four encoders. As of now we can clearly see that it has advanced features than any other FPGA of handling the design. Coming to the board area there is no specific requirement. The board is equally spacious and has lot of room to accommodate many other components for various other features.

Chapter 5

5 SIMULATION RESULTS

5.1 Quality calibration

To achieve the required output CBR the encoding quality is tuned by the buffer control algorithm at each slice beginning. Therefore each slice is encoded at constant quality but different qualities can be used in different slices. To make this mechanism effective, it is fundamental that an upper quality level is associated to a greater amount of bits (lower compression factor) and vice versa. Thus we have to define the quality states according to a monotone relationship with the compression ratio.

A quality state is characterized by Quantization Precision (QP). QP is inversely proportional to quality. If we increase the QP than we will apply more compression ratio on data, we are sure that lose of more information compared to less QP we get higher compression ratios at the cost of lower objective coding qualities (PSNR).

In order to specify, evaluate and compare video communication systems it is necessary to determine the quality of the video images displayed to the viewer. A difficult and often imprecise art because there are so many factors that can affect the results. Visual quality is inherently *subjective* and is influenced by many factors that make it difficult to obtain a completely accurate measure of quality. For example, a viewer's opinion of visual quality can depend very much on the task at hand, such as passively watching a DVD movie, actively participating in a videoconference, communicating using sign language or trying to identify a person in a surveillance video scene. Measuring visual quality using *objective* criteria gives accurate, repeatable results but as yet there are no objective measurement systems that completely reproduce the subjective experience of a human observer watching a video display.

Subjective quality measurement

Factors Influencing Subjective Quality

Our perception of a visual scene is formed by a complex interaction between the components of the Human Visual System (HVS), the eye and the brain. The perception of visual quality is influenced by spatial fidelity (how clearly parts of the scene can be seen, whether there is any obvious distortion) and temporal fidelity (whether motion appears natural and 'smooth'). However, a viewer's opinion of 'quality' is also affected by other factors such as the viewing environment, the observer's state of mind and the extent to which the observer interacts with the visual scene. A user carrying out a specific task that requires concentration on part of a visual scene will have a quite different requirement for 'good' quality than a user who is passively watching a movie. For example, it has been shown that a viewer's opinion of visual quality is measurably higher if the viewing environment is comfortable and non-distracting (regardless of the 'quality' of the visual image itself).

Other important influences on perceived quality include visual attention (an observer perceives a scene by fixating on a sequence of points in the image rather than by taking in everything simultaneously) and the so-called 'recency effect' (our opinion of a visual sequence is more heavily influenced by recently-viewed material than older video material) [2, 3]. All of these factors make it very difficult to measure visual quality accurately and quantitatively.

Objective quality measurement

The complexity and cost of subjective quality measurement make it attractive to be able to measure quality automatically using an algorithm. Developers of video compression and video processing systems rely heavily on so-called objective (algorithmic) quality measures. The most widely used measure is Peak Signal to Noise Ratio (PSNR) but the limitations of this metric have led to many efforts to develop more sophisticated measures that approximate the response of 'real' human observers.

PSNR

Peak Signal to Noise Ratio (PSNR) (Equation) is measured on a logarithmic scale and depends on the mean squared error (MSE) of between an original and an impaired image or video frame, relative to $(2n - 1)^2$ (the square of the highest-possible signal value in the image, where *n* is the number of bits per image sample).

$$PSNR_{dB} = (10 \log10 (2n - 1)^{2)} /(MSE) \quad \textbf{Equation 5.1}$$

PSNR can be calculated easily and quickly and is therefore a very popular quality measure, widely used to compare the 'quality' of compressed and decompressed video images.

The PSNR measure suffers from a number of limitations. PSNR requires an unimpaired original image for comparison but this may not be available in every case and it may not be easy to verify that an 'original' image has perfect fidelity. PSNR does not correlate well with subjective video quality measures such as those defined in ITU-R 500. For a given image or image sequence, high PSNR usually indicates high quality and low PSNR usually indicates low quality.

Some conclusions and observations

We made some simulations and find some interesting stuff for different levels of delay and different types of encoding.

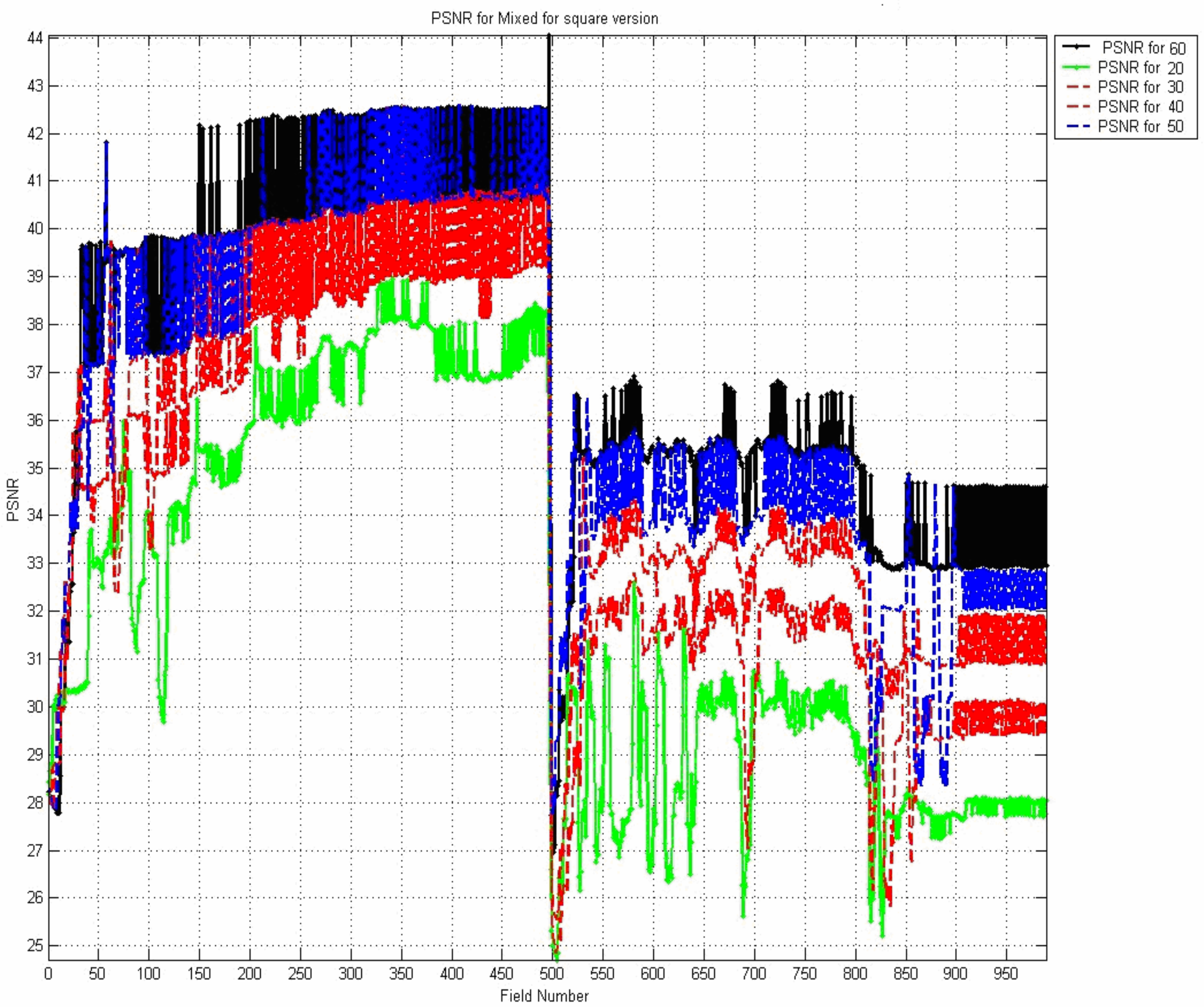


Figure 5-1 Graph between Field number vs. PSNR for multi streaming at 40ms delay (mobcal and parkrun mixed sequence

The figure 5-1 graph is drawn for different levels of bitrate from 20Mbit to 60 Mbit for 4 encoder multistreaming approach at 40ms delay. In the mixed sequence we can see sudden change in seen so when mobcal is finished and next field is parkrun there is a lot of residue. We applied higher QP that gives less quality there we can find sudden jump in PSNR in graph.

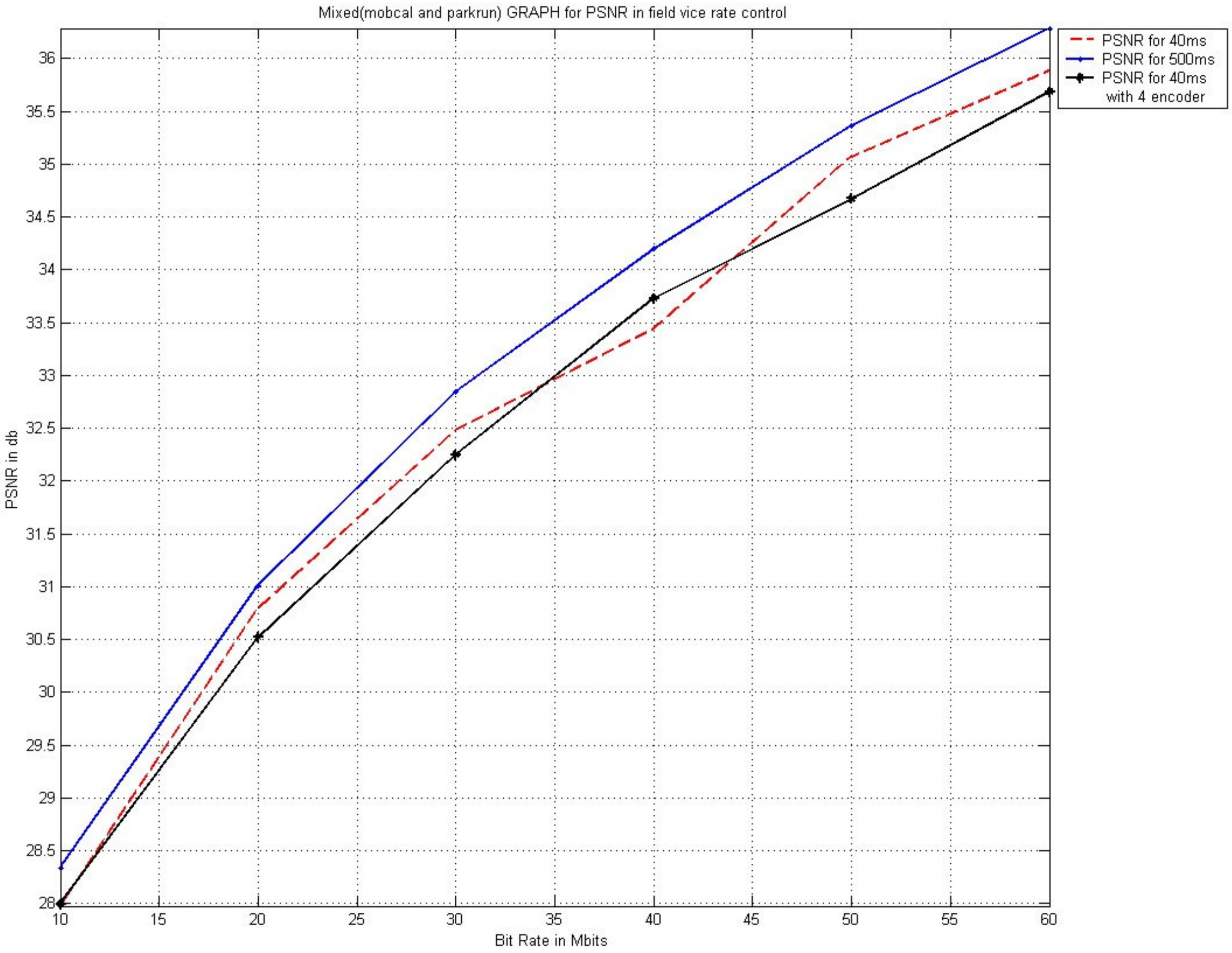


Figure 5-2. Graph between bitrate vs. PSNR for different delay 40ms and 500ms with single encoder and third one is with 4 encoders multi stream approach with 40ms delay.

In figure 5-2 we can see the video compression of mixed (mobcal and parkrun sequences) for field vice rate control. The 500ms delay system is more effective in quality compared with the other two ways of approach but still we are designing for low delay solution so we will see the comparison between 40ms with single encoder and multi streaming level, where we can see there is not much difference between those two approaches and with multi approach we can reduce the encoding time.

The only problem with multi streaming (instead of single encoder we use four encoder in problem) is we need lot hardware compared to single encoder, but we cannot reduce the delay. So we try to gain the maximum quality with four encoders. With using of single or four we can get almost same quality so we can use this multi streaming approach.

In the below figures we can see different PSNR graphs for different delays and with 4 encoder and single one.

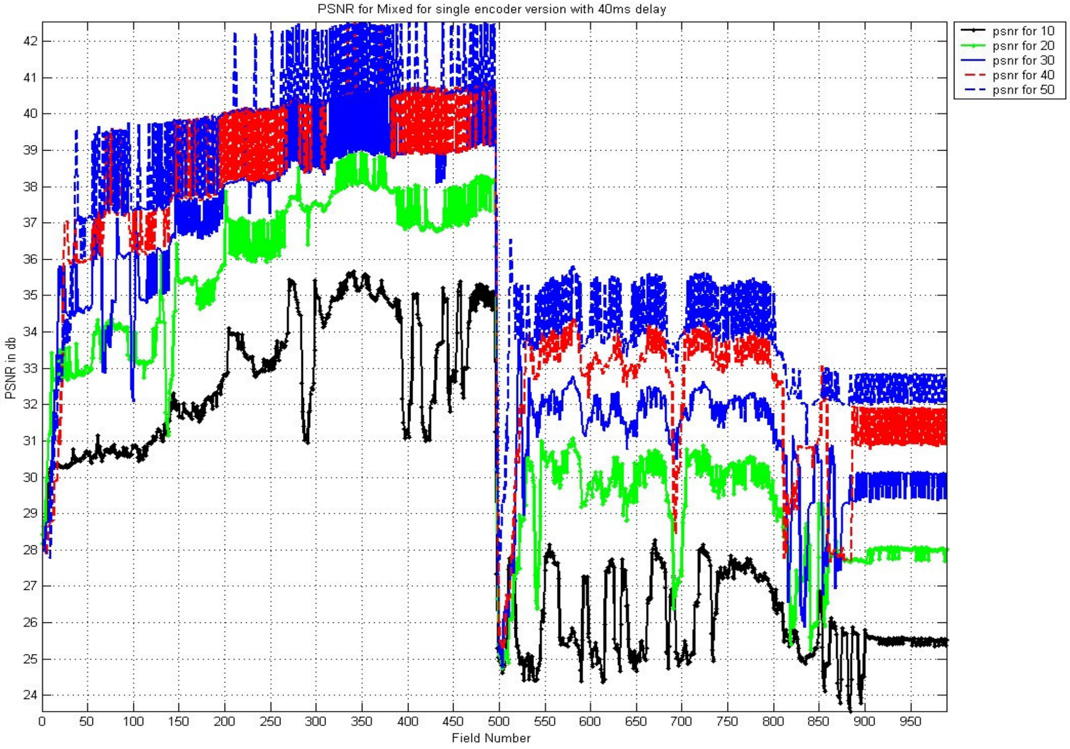


Figure 5-3 PSNR for mixed sequence with single encoder (40ms delay) from 10 to 50MBPS

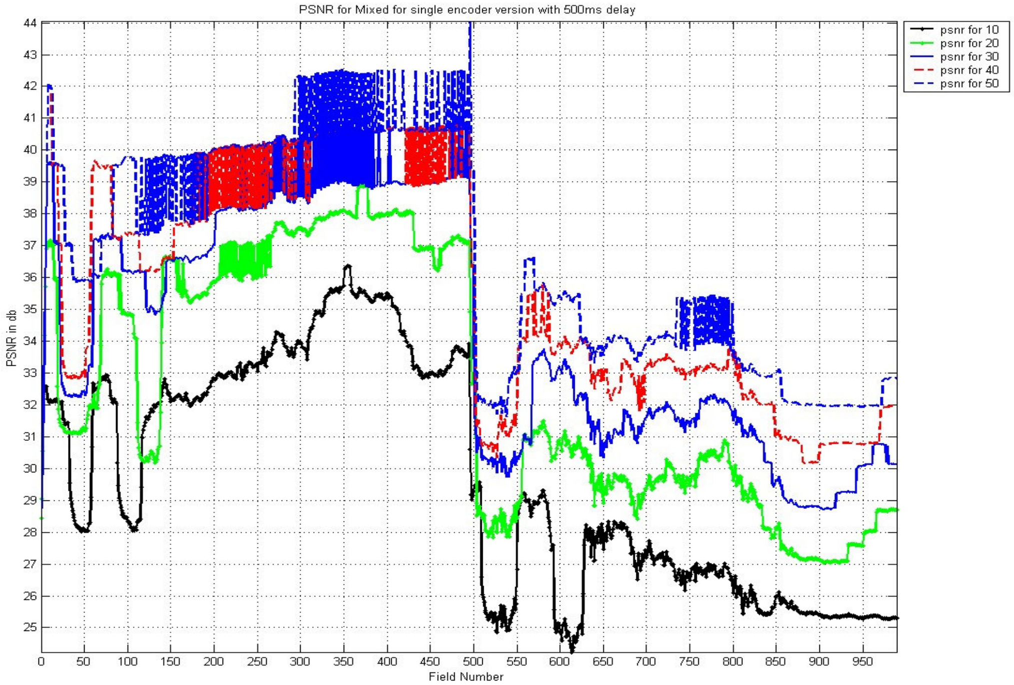


Figure 5-4 PSNR for mixed sequence with single encoder (500 ms delay) from 10 to 50MBPS

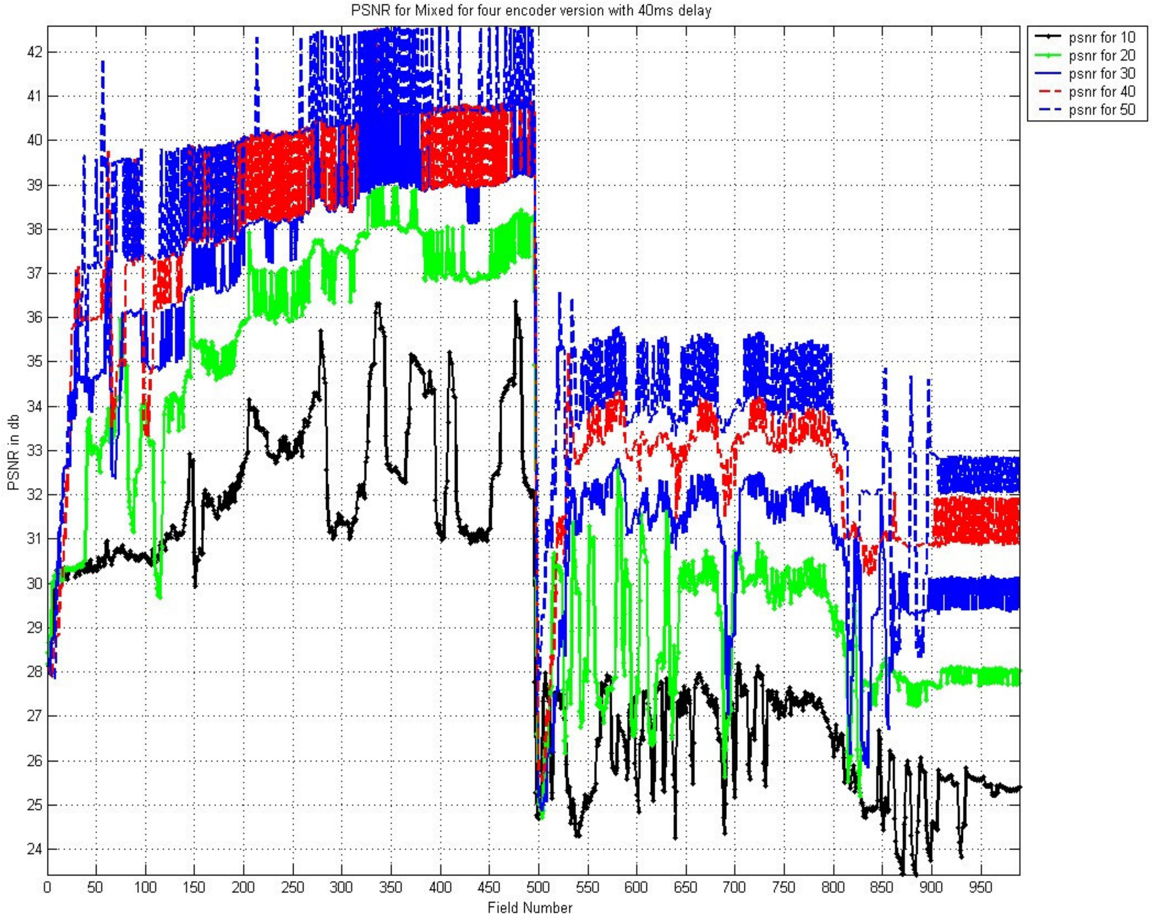


Figure 5-5 PSNR for mixed sequence with four encoder square split (40ms delay) from10 to 50MBPS

The figure 5-3 is with low delay (40ms) and single streaming, figure 5-4 is with standard delay (500ms) and single streaming and finally figure 5-5 is with low delay (40ms) and multi streaming (4 encoder parallel). Among those three the 500ms delay is showing good performance but rest of the two cases also good and the main change in quality is less 1db which good compared with delay level and multi-streaming approach.

5.2 Buffer control

The implemented rate control strategy is the one described which is based on both a *hard* and *soft* reaction depending on the *buffer fullness BF*. This method is based on the quality states and has been fully tested on the active range of encoding bit rates 10 to 60 Mbps.

The buffer control one of major constraint of quality. The buffer is maintained at constant level which would be the ideal stage of buffer control. The main rate control is check and applied depending on buffer fullness.

5.2.1 Buffer virtual update

In this part we update buffer depending on bitrate, which is made virtually but it is exactly what the hardware do. The stepwise implementation is

- **First check frame number and frame type** – if frame number is 0 than it is necessary to have I picture and frame type represents type of the frame means I or B or P the corresponding number as 0,2,1.
- **Check the underflow condition** – if the buffer is going to underflow than we send one flag which will give us as '0' which represents buffer is underflow so don't send any compressed bits, which is real time hardware related. So depending up on field base or slice base encoding we will make buffer occupancy.
- **Updating buffer occupancy** – the equation 5.2 show how the buffer occupancy is calculated according bits per field or slice, bitrate and frame rate.

buff_fullness += ((vopbits – (bit_rate/ (Frame_rate)) × flag × x) **Equation 5.2**

This equation 5.2 representation style of buffer update
Where
Buff_fullness is buffer occupancy, vopbits is compressed bits, bitrate is at what bitrate we are encoding, frame rate is how many frames per second, flag represents underflow condition and x is represents slicewise updating the buffer or field base. If x =1 its field base and x=34 than it is slice base for HD (1920 × 544) sequence.

5.2.2 QP calculation

The QP (quantization parameter) is one of the major factors for video compression. This QP is inversely proportional to quality and this QP is calculated according buffer size and buffer fullness.

The main steps for calculation of QP are

- **Check the overflow emergency conditions –** first main important condition is overflow. If buffer fullness is more than buffer size than this is overflow condition which is the worst condition for any video compression method. We need to check worst condition for overflow if the buffer is going towards overflow so Increase the QP twice than previous and not more 31(because QP boundary is 1 to 31).
- **Check the underflow emergency condition –** second part is to check underflow condition because if the buffer is continuously underflow than our bandwidth will fall. So to utilize it properly we will check buffer underflow condition and make QP 75% with compared to previous QP, but QP should not decrease less than 1.
- **Normal condition –** when the buffer is normal or coming from overflow to normal condition than this case is applied. Here we will do the different approach and make some calculations depending on buffer size and occupancy.

$$QP = QP + a\times y\times y\times y + b\times x \qquad \text{Equation 5.3}$$

Equation 5.3 shows QP calculation methodology where a and b are constants

$$x = (\text{buffer fullness} - \text{oldbufferfullness}) \div \text{buffer size} \qquad \text{Equation 5.4}$$

$$y = (\text{buffer fullness} \div \text{buffer size}) - 0.40 \qquad \text{Equation 5.5}$$

x and y is calculated by using equation 5.4 and 5.5 for a and b value we made different simulation and find some good values.

i. a = 1.0, b = 0.5: buffer well bounded under normal condition; some "jitter" in QP, cannot withstand a scene change: overflow.

ii. a = 0.5, b = 1.0: buffer well bounded under normal condition; some "jitter" in QP can withstand a scene change, but reaction is still slow (if...else at 75% prevents overflow).

iii. a = 4.0 (y^3), b = 1.0: buffer well bounded (a little more relaxed than previous cases) under normal condition.

iv. a = 4.0 (y^3), b = 2.0: buffer well bounded (a little more relaxed than previous cases) under normal condition and more effective after overflow.

The final results after 75% emergency and a = 4.0 and b = 2.0 is shown in figure 5-6, 5-7 and 5-8 at different bitrates 20 ,30 and 40 MBPS.

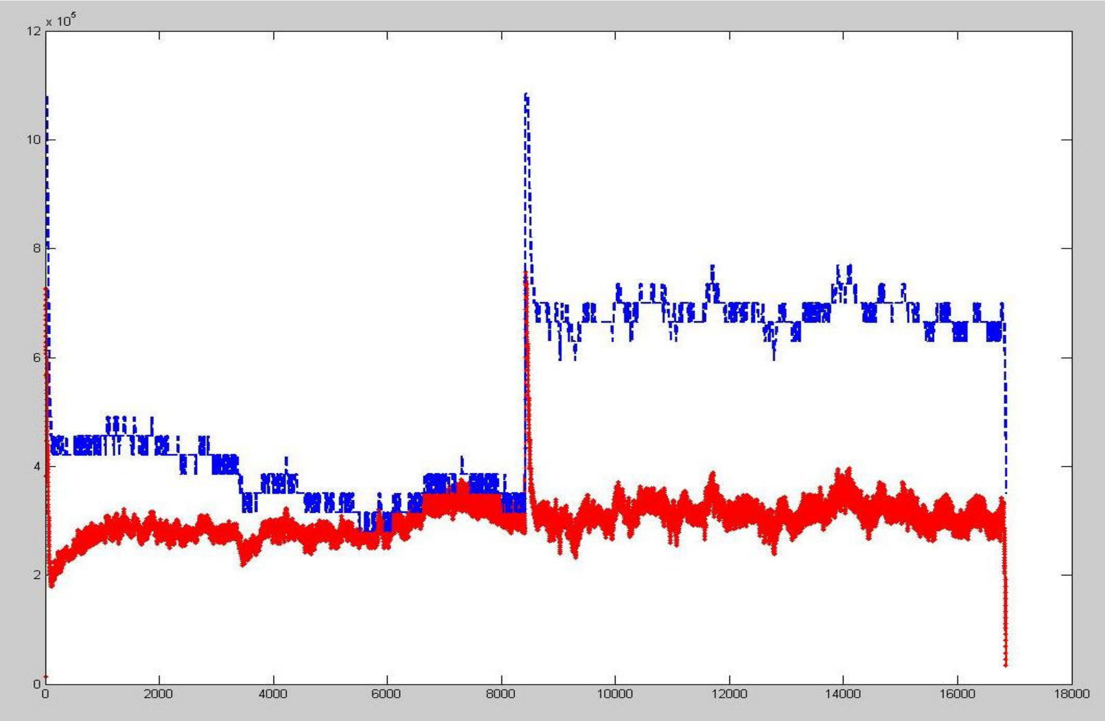


Figure 5-6 Graph between slicenumber vs. QP and slice number vs. bufferfullness at 20 MBPS

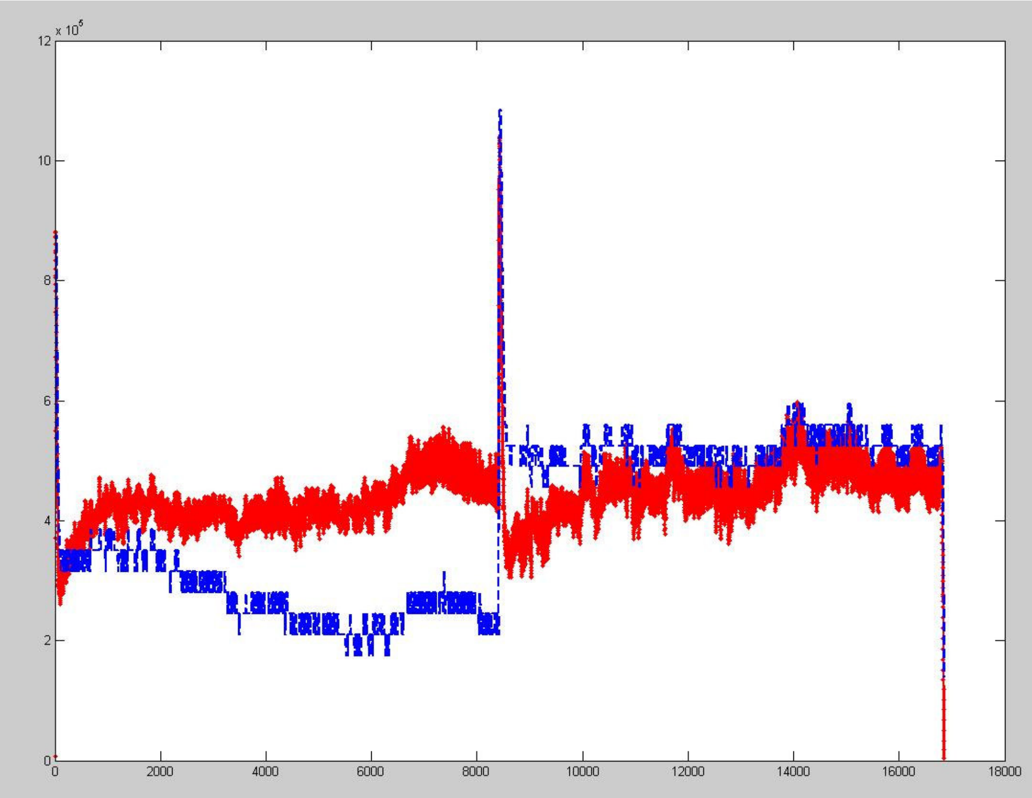


Figure 5-7 Graph between slice number vs QP and slice number vs. bufferfullness at 30MBPS

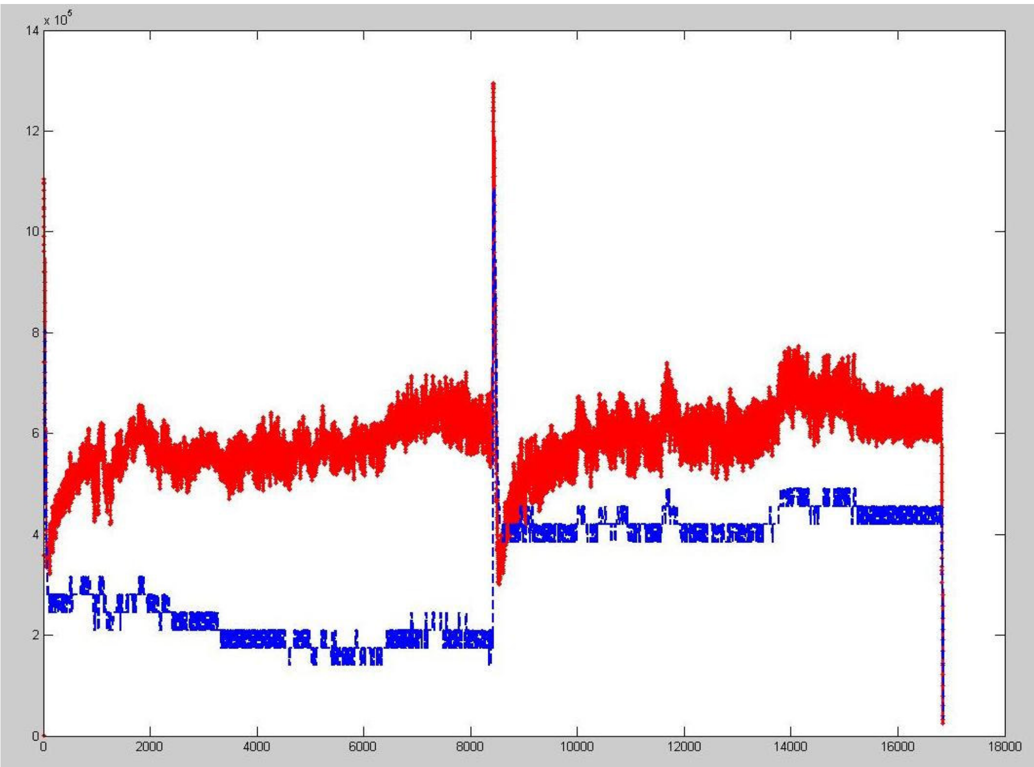


Figure 5-8 Graph between slice number vs. QP and slice number vs. bufferfullness at 40 MBPS.

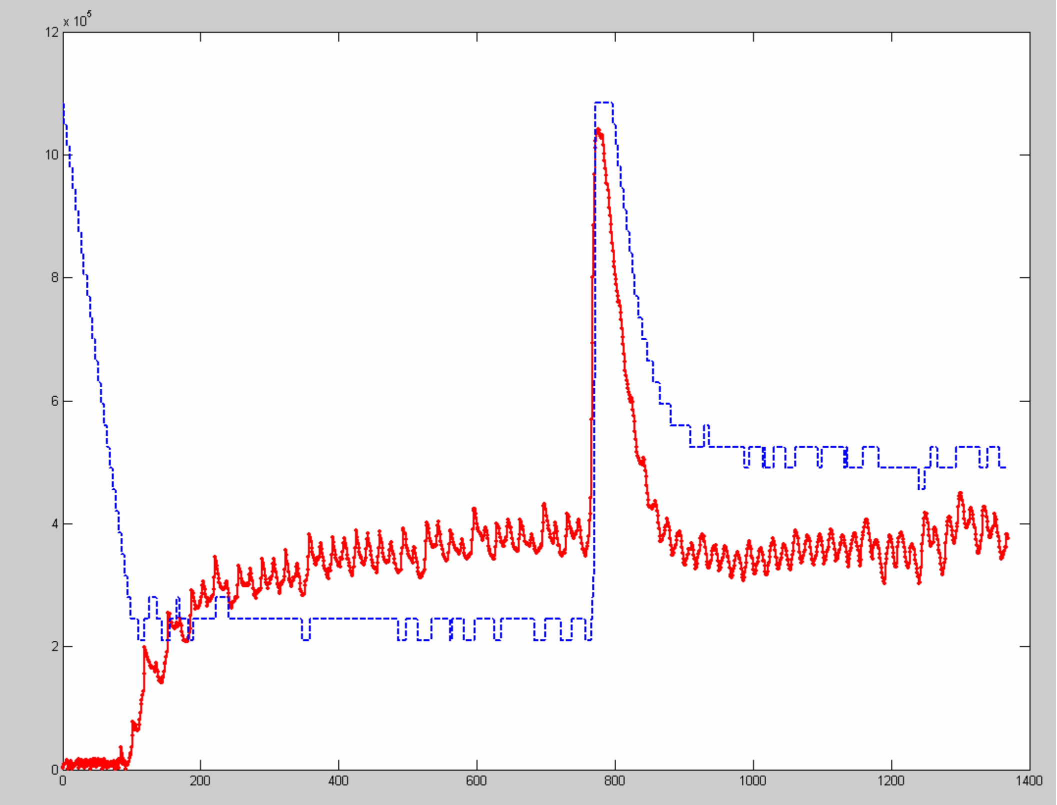


Figure 5-9 Graph between slice number vs. QP and slice number vs. buffer fullness at 40MBPS in important seen change for mixed sequence at 40MBPS

The figure 5-9 shows how quick the response for bufferfullness with the a = 4.0 and b = 2.0, the reaction is very quick but after full QP also the bufferfullness is increasing for 10 slices so for this case we use low pass filter to reduce input residue.

5.2.3 Conclusion

The buffer and rate control is made for low delay design with multi-streaming approach, where the slice encoding is more effective for critical situations like sudden seen change any high residue. The rate control is made in slice base for effective control and multi streaming approach can reduce some delay in encoding process. For multi streaming also we need to apply global QP only to make good quality compression.

Chapter 6

6 Conclusion and Future Work

6.1 Summary of the achievements

The demand for digital wireless links is more and more increasing in professional video domain. However some problems of compatibility are still open concerning in many applications which require installing new hardware in the place of existing ones. The industry has to choose the best solution with respect to the input cost and output performance ratio.

This kind of solution comes from using advanced technologies and advanced method of achieving what you want from the existing hardware with the available bandwidth. This attitude also comes from the analysis of the most advanced MPEG standards. MPEG-4 and AVC/H.264 has more efficient compression and unparalleled flexibility to meet the needs of practical multimedia communication applications. The diverse set of coding tools described in these standards address the wider range of applications ever supported. However, the algorithm complexity has reached high levels and compact implementations require big efforts from developers and manufacturers in this era. To find a solution you need to investigate more and improve the coding styles with respect to various output constrains.

In order to implement a low latency system based on MPEG standard, a subset of simple coding tools can be identified first. The study of an existing MPEG-4 makes it possible to understand the implementation issue coming from the VBV buffer control strategy, which puts in evidence a strong trade-off between latency and bandwidth overhead.

We were asked to implement a decoding compliant architecture we are provided with a Xilinx MPEG-4 Encoder and Decoder which is a software codec compliant with hardware model. We had to first make some simulation work initially to know the existing Mpeg profile. After we had implemented intra refresh slicewise for removing the drift occurred due to compression. After this we had again made some simulations to get the optimal slice refresh value. Then we had decided to go for a multi-stream approach to reduce the delay of the system, while the design is still under development and optimization for the buffer control strategy.

We had achieved the multi-stream approach using pipes and forks for a clean exit with out modifying the structure of the encoder. Then we had designed a global control strategy for all the encoders which are used during the multi-stream approach to maintain a good quality with respect to the bandwidth. There is a considerable improvement due to all the achievements in the encoder.

6.2 Further work

The optimization of the coding/decoding latency in MPEG is both a theoretical and implementation problem. The set of MPEG features which allows a low delay coding can be isolated and analyzed, but the real system performance depends on few key factors such as: field-encoding, intra-slice mode, infinite GOP and others, which are not actually widely supported by hardware developers. Furthermore, the set of MPEG low delay processing tools does not include bidirectional coding and impose strong constraints on the GOP, which limit the MPEG compression efficiency. To achieve a low delay coding in MPEG is possible at the price of higher encoding rates and a sub-utilization of the available coding tools supported by standard platforms for a required bandwidth.

On the other hand, the theoretical problem of the VBV buffer optimization for low delay purpose is still open. MPEG does not give any indications about the buffer control strategy, a task which has been completely delegated to the developer.

Here we have two solutions for the future work of this project

- An implementation of a low-pass filter before sending the data to the encoder might be more useful to eliminate the problem of overflow of the buffer after encoding.
- A slice implementation of the quantization value for a global buffer control strategy is a good optimal solution to compress the data when there is an emergency occurred due to overflow. It is also a better solution to control the quality of the encoders in the multi-stream approach to achieve the required bitrate.

7 User Manual

7.1 Terminology

Some of the terminology adopted in the MPEG-4 standard i.e. Simple profile in specific is as follows:

Codec

A *video codec* is a software module that enables video compression or decompression of digital video.

MPEG

The **Moving Picture Experts Group** (MPEG) is a working group of ISO/IEC in charge of the development of international standards for compression, decompression, processing, and coded representation of moving pictures, audio and their combination.

Frames and Fields

A video signal may be sampled as a series of complete frames (progressive sampling) or as a sequence of interlaced fields (interlaced sampling). In an interlaced video sequence, half of the data in a frame (one field) is sampled at each temporal sampling interval. A field consists of either the odd-numbered or even-numbered lines within a complete video frame and an interlaced video sequence. It contains a series of fields, each representing half of the information in a complete video frame. The advantage of this sampling method is that it is possible to send twice as many fields per second as the number of frames in an equivalent progressive sequence with the same data rate, giving the appearance of smoother motion.

Slice

A frame is divided into a slice which is a sequence of macro blocks. A coded frame consists of macro blocks, each containing 16 × 16 luma samples and associated chroma samples (8 × 8 Cb and 8 × 8 Cr samples in the current standard).

Macro Block

A macro block is a set of samples of data which is usually in mpeg-4 represented as 16x16 pixels.

Bit Rate

Bit rate describes how much information there is per second in a stream of data. The important thing to understand is that the higher the bit rate, the more information, and thus the more effort it takes to decode that information, and the more space the file requires. So bitrate is a highly influential factor now-a-days.

PSNR

Peak Signal to Noise Ratio (PSNR) is measured on a logarithmic scale and depends on the mean squared error (MSE) of between an original and an impaired image or video frame, relative to $(2^n - 1)^2$ (the square of the highest-possible signal value in the image, where n is the number of bits per image sample).

$$PSNR_{db} = 10 \log_{10} (2^n - 1)^2 / MSE$$

PSNR can be calculated easily and quickly and is therefore a very popular quality measure, widely used to compare the quality of compressed and decompressed video images.

7.2 List of figures

7.3 Glossary

4:2:0 (sampling)	Sampling method: chrominance components have half the horizontal and vertical resolution of luminance component
4:2:2 (sampling)	Sampling method: chrominance components have half the horizontal resolution of luminance component
4:4:4 (sampling)	Sampling method: chrominance components have same resolution as luminance component arithmetic coding Coding method to reduce redundancy artefact Visual distortion in an image
ADC	Analog to Digital Converter
Arithmetic coding	Coding method to reduce redundancy
Artefact	Visual distortion in an image
ASIC	Application-Specific Integrated Circuit
ASO	Arbitrary Slice Order
B-picture (slice)	Coded picture (slice) predicted using bidirectional motion compensation
BAB Binary	Alpha Block, Indicates the boundary of a region (MPEG-4 Visual)
Block	Region of macroblock (8 x 8 or 4 x 4 samples)
Block matching	Motion estimation carried out on rectangular picture areas
Blocking	Square or rectangular distortion areas in an image
CABAC	Context-based Adaptive Binary Arithmetic Coding
CAE	Context-based Arithmetic Encoding
CAVLC	Context Adaptive Variable Length Coding
CBR	constant bitrate
CD	Compact Disk
Chrominance	Color difference component
CIF	Common Intermediate Format
CODEC	COder / DECoder pair
COFDM	Coded-OFDM
Color	space Method of representing color images
DAC	Digital to Analog Converter
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
Direct prediction	A coding mode in which no motion vector is transmitted
DPCM	Differential Pulse Code Modulation
DSCQS	Double Stimulus Continuous Quality Scale
DSP	Digital Signal Processor
DV	Digital Video. International standard for a consumer digital video format introduced by Sony Corp. in 1995
DVB	Digital Video Broadcasting
DVB-C	DVB-Cable

DVB-S	DVB-Satellite
DVB-T	DVB-Terrestrial
DWT	Discrete Wavelet Transform
Entropy coding	Coding method to reduce redundancy
ES	elementary stream
FFT	Fast Fourier Transform
Field	Odd or even numbered lines from an interlaced video sequence
FIFO	First-In, First-Out
FIR	Finite Impulse Response
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
GOP	Group Of Pictures, a set of coded video images
H.261	A video coding standard
H.263	A video coding standard
H.264	A video coding standard
HDTV	High Definition TeleVision
Huffman coding	Coding method to deduce redundancy
HVS Human	Visual System, the system by which the humans perceive and interpret the images
I-picture (slice)	Coded picture (slice) without reference to any other frame
IEC	International Standard Commission
IEEE	Institute of Electrical and Electronics Engineers
IFFT	Inverse Fast Fourier Transform
IIR	Infinite Impulse Response
Inter (coding)	Coding of video frames using temporal prediction or compensation
Interlaced (video)	Video data represented as a series of fields
Intra (coding)	Coding of video frames without temporal prediction
ISO	International Standards Organization
ITU	International Telecommunication Union
JPEG	Joint Photographic Expert Group (also an image coding standard)
JPEG2000	An image coding standard
Level A	set of conformance parameters (applied to a Profile)
LSB	less Significant Bit
Macroblock (MB)	A region of frame (usually 16 x 16 pixels) formed of blocks
MAD	Mean Absolute Difference
Motion compensation	Video frame prediction method, based on a motion model
Motion estimation	Method to estimate relative motion between two or more video frames

Motion vector (MV)	Vector indicating a displacement block or region to be used in motion compensation
MPEG	Moving Picture Experts Group
MPEG-1	A multimedia coding standard
MPEG-2	A multimedia coding standard
MPEG-3	A multimedia coding standard
MSB	Most Significant Bit
NAB	National Association of Broadcasters
NAL	Network Abstraction Layer
NTSC	National TV Standards Committee
OFDM	Orthogonal Frequency Division Multiplexing
Picture (coded)	Coded (compressed) video frame
P-picture (slice)	Coded picture (slice) using motion-compensated prediction from one reference frame
PAL	Phase-Alternating Line
PDA	Personal Digital Assistant
Profile A	set of functional capabilities (related to a video CODEC)
Progressive (video)	Video data represented a series of consecutive completes frames
PSNR	Peak Signal to Noise Ratio, an objective quality measure
QAM	Quadrature Amplitude Modulation
QCIF	Quarter Common Intermediate Format
QPSK	Quaternary Phase Shift Keying
Quantize	Reduce the precision of a scalar or vector quantity
Rate control	Control of bit rate of encoded video signal
Rate-distortion	Measure of CODEC performance (distortion at a range of coded bit rates)
RGB	Red Green Blue color space
RLC	Run level coding
RTL	Register Transfer Logic
SAD	Sum of Absolute Differences
Scalable coding	coding a signal into a number of layers
SDRAM	Synchronous Dynamic Random Access Memories
SDTV	Standard Definition TeleVision
SECAM	System Sequential Colour A Memory
SNR	Signal to Noise Ratio
Studio quality	Lossless or near-lossless video quality
TS	Transport Stream
VALIDATE	Verification And Launch of Integrated Digital Advanced Television in Europe

VBV	Video Buffering Verifier
VCEG	Video Coding Expert Group (committee of ITU)
VCL	Video Coding Layer
VHDL	Very High level Design Language
VLC	Variable Length Code
VLSI	Very Large Scale Integration
VO	Video Object
VOP	Video Object Plane
VQEG	Video Quality Expert Group
YCbCr	Luminance, Blue chrominance, Red chrominance colour space
YUV	A colour space (see YCbCr)

8 References

[1] Graziano VAROTTO, LOW DELAY VIDEO CODING, Ph.D. Dissertation, EPFL, Lausanne, March 2006.

[2] Iain E.G. Richardson, H.264 and MPEG-4 Video Compression, Video Coding for Next-generation Multimedia, The Robert Gordon University, Aberdeen, John Willey & Sons Ltd (2003).

[3] Jie Chen, Ut-Va Koc, K.J. Ray Liu, Design of digital video coding systems, Signal Processing and Communications Series, Marcel Dekker, AG (2002).

[4] Pan Feng Nanyang, Digital Television Terrestrial Broadcasting Primer, Technological University, Singapore, TechOnLine Publications (October 2001).

[5] www.livetools.tv

[6] www.epfl.ch

[7] www.xilinx.com

[8] ITU-R Recommendation BT.601-5, Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios

[9] ITU-R Recommendation BT.500-11, Methodology for the subjective assessment of the quality of television pictures, Question ITU-R 211/11, (1974 - 1978 - 1982 - 1986 - 1990 - 1992 – 1994 - 1995 - 1998 - 1998 - 2000 - 2002).

[10] Woo Young Choi, Rae-Hong Park, Motion vector coding with conditional transmission Signal Processing, Vol. 18, No. 3, November 1989.

[11] D. A. Huffman, A method for the construction of minimum redundancy codes, Proc. of the IRE, 40, p 1098-1101, (1952).

[12] I. Witten, R. Neal and J. Cleary, Arithmetic coding for data compression, communications of the ACM, 30 (6), June 1987.

[13] P. A. A. Assuno and M. Ghanbari, Buffer Analysis and Control in CBR Video transcoding, IEEE Transaction and systems for video technology, Vol. 10, No. 1, February 2000.

[14] ISO/IEC 14496-2, Information technology, generic coding of audio-visual objects-Part 2: Visual, Geneva, Switzerland, 1998, MPEG4.

[15]ISO/IEC 14496-10 and ITU-T Rec. H.264, Advanced Video Coding, 2003.

[16]MPEG 2 Test Model 5, Rev. 2, Section 10 Rate Control and Quantization Optimization, ISO/IEC/JTC1SC29WG11, April 1993.

[17] Xilinx Company MPEG-4 Encoder V1.7, Product Specification, December 15, 2006.

[18] http://www.erlenstar.demon.co.uk/unix/faq_2.html

[19] http://www.unixguide.net/unix/programming/