REAL-TIME VIDEO ENCODER ON TMSC6000 PLATFORM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BARAN ERDOĞAN

IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

NOVEMBER 2004

Approval of the Graduate School of Natural and Applied Sciences

_____
Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____
Prof. Dr. İsmet Erkmen
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____
Assoc. Prof. Dr. Gözde Bozdağı Akar
Supervisor

Examining Committee Members

Prof. Dr. Hasan Güran	(METU,EE)	_____
Assoc. Prof. Dr. Gözde Bozdağı Akar	(METU,EE)	_____
Assoc. Prof. Dr. Aydın Alatan	(METU,EE)	_____
Asst. Prof. Dr. Cüneyt Bazlamaççı	(METU,EE)	_____
Alpaslan Lorasdağı	(ASELSAN)	_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Baran Erdoğan

Signature :

ABSTRACT

REAL-TIME VIDEO ENCODER ON TMSC6000 PLATFORM

Erdoğan, Baran

M.Sc., Department of Electrical and Electronics Engineering

Supervisor: Assoc. Prof. Dr. Gözde Bozdağı Akar

November 2004, 77 pages

Technology is integrated into daily life more than before as it evolves through communication area. In the past, it started with audio devices that help us to communicate while far between two ends of communication line. Nowadays visual communication comes in front considering the communication technology. This became possible with the improvement in the compression techniques of visual data and increasing speed, optimized architecture of the new family processors. These type processors are named as Digital Signal Processors (DSP's). Texas Instruments TMS320C6000 Digital Signal Processor family offers one of the fastest DSP core in the market. TMS320C64x sub-family processors are newly developed under the TMS320C6000 family to overcome disadvantages of its predecessor family TMS320C62x. TMS320C64x family has optimized architecture for packed data processing, improved data paths and functional units,

improved memory architecture and increased speed. These capabilities make this family of processors good candidate for real-time video processing applications. Advantages of this core are used for implementing newly established H.264 Recommendation. Highly optimizing C Compiler of TMS320C64x enabled fast running implementation of encoder blocks that bring heavy computational load to encoder. Such as fast implementation of Motion Estimation, Transform, Entropy Coding became possible. Simplified Densely Centered Uniform-P Search algorithm is used for fast estimation of motion vectors. Time taking parts enhanced to improve the performance of the encoder.

Keywords: H.26L, H.264, Video Compression, Real-time, Digital Signal Processor

ÖZ

GERÇEK ZAMANLI VIDEO KODLAYICISININ TMSC6000 PLATFORMUNDA GERÇEKLENMESİ

Erdoğan, Baran

Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Gözde Bozdağı Akar

Kasım 2004, 77 sayfa

Günümüzde teknoloji, iletişim teknolojileri alanında daha fazla ilerledikçe, günlük hayata daha da fazla girmektedir. Geçmişte bu durum uzak noktalar arasında sesli iletişimi sağlayan iletişim araçlarıyla başladı. Bugünlerde ise, iletişim teknolojisi alanında görüntülü iletişim ön plana çıkmaktadır. Bu, sıkıştırma yöntemlerindeki gelişmeyle ve hızı artan, mimarisi bu tür işler için tasarlanan işlemci ailelerinin gelişmesi ile mümkün olmuştur. Bu tip işlemciler Sayısal Sinyal İşlemciler (SSİ) olarak adlandırılmaktadır. Texas Instruments TMS320C6000 Sayısal Sinyal İşleyici ailesi şu anda geliştirilen en hızlı SSİ çekirdeklerinden birine sahip bulunmaktadır. Bu işlemci ailesi içinde bulunan TMS320C64x işlemci alt ailesi, kendinden once geliştirilen, TMS320C62x işlemci alt ailesinin olumsuz yönlerini gidermek üzere geliştirilmiştir. TMS320C64x ailesi işlemcilerin paketlenmiş veri

işlenmesi için geliştirilmiş mimarisi, genişletilmiş veriyolları ve işlevsel üniteleri, geliştirilmiş hafıza mimarisi bulunmaktadır ve daha hızlıdırlar. Bu yetenekleri, TMS320C64x işlemci ailesini, gerçek zamanlı görüntü işleme uygulamaları için iyi bir seçenek yapmaktadır. Bu mimarinin artıları yeni geliştirilen H.264 standardını gerçeklemek için kullanılmıştır. TMS320C64x in C de yazılmış kodları işlemci mimarisine en uygun şekilde derleyen derleyicisiyle, fazla sayıda hesaplama gerektiren kodlayıcı bloklarının hızlı çalışacak bir şekilde gerçeklenmesi mümkün olmuştur. Hareket kestiriminde kullanılmak üzere basitleştirilmiş "Yoğun Kümelenmiş Düzgün-P Arama" algoritması kullanılmıştır. Zaman alan kısımlar daha hızlı çalışacak şekilde yeniden düzenlenmiştir.

Anahtar Kelimeler: H.26L, H.264, Görüntü Sıkıştırma, Gerçek Zamanlı, Sayısal Sinyal İşlemci.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Assoc. Prof. Dr. Gözde Bozdağı Akar for her valuable guidance, advice and encouragements.

I would like to thank ASELSAN for technical support.

I am grateful to my family for everything.

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURE

LIST OF ABBREVIATIONS

CABAC	:	Context Adaptive Binary Arithmetic Coding
CAVLC	:	Context Adaptive Variable Length Coding
CBP	:	Coded Block Pattern
CCIR	:	International Radio Consultative Committee
CCITT	:	International telephone and Telegraph Consultative Committee
CCS	:	Code Composer Studio
CIF	:	Common Interface Format
CPU	:	Central Processing Unit
DCT	:	Discrete Cosine Transform
DCUPS	:	Densely Centered Uniform-p Search
DMA	:	Direct memory Access
DSP	:	Digital Signal Processor
DVD	:	Digital Video Disc
EMIF	:	External Memory Interface
FIR	:	Finite Impulse Response
GEL	:	Graphical Extension Language
HDTV	:	High Definition TV
HPI	:	Host Port Interface
I/O	:	Input/Output
IDE	:	Integrated Development Environment
IDR	:	Instantaneous Decoding Refresh
ISDN	:	Integrated Services Digital Network
ISO	:	International Organization for Standardization
ITU	:	International Telecommunication Union
JVT	:	Joint Video Team
MAC	:	Multiply And Accumulate

MB	:	Macro Block
MCBSP	:	Multi Channel Buffered Serial Port
MIPS	:	Million Instructions Per Second
MPEG	:	Moving Picture Experts Group
NAL	:	Network Adaptation Layer
PC	:	Personal Computer
PCI	:	Peripheral Component Interconnect
PSTN	:	Public Switching Telephone Networks
QCIF	:	Quarter Common Interface Format
RAM	:	Random Access Memory
SAD	:	Sum of Absolute Difference
SDRAM	:	Synchronous Dynamic Random Access Memory
SQCIF	:	Sub-Quarter Common Interface Format
VLIW	:	Very Long Instruction Word
VCL	:	Video Coding Layer
VLC	:	Variable Length Coding

CHAPTER 1

INTRODUCTION

1.1 Problem Definition and Motivation

Communication has become an important part of our life. Innovations in technology have given a way to commercial use of real-time audiovisual communication applications, which require high bandwidth and have real-time characteristics. Without video compression, it is not possible to meet requirements of widely used visual services such as, Video Conferencing, Videophone, Video E-mail, and Video Streaming over Internet, High Definition TV (HDTV), and Digital Video Disc (DVD) due to large bandwidth needs [1]. For this reason, there has been great effort to develop new video compression methods.

Up to this time, several video compression standards were developed. ITU-T standards H.261 [2], H.263 [3] and ISO/IEC standards MPEG-1 [4], MPEG-2 [5] are examples to these standards. Brief descriptions of these standards are given in the following chapters. Joint Video Team (JVT) [6], which is a committee formed by both ITU-T and MPEG members, have released H.264 Recommendation [7], which uses latest innovations in video compression technology, supporting wide range of frame sizes, aiming all bit-rate applications.

Real-time implementation of video compression standards mentioned above requires high processing power due to complexity of the video processing

operations. It may not be possible to meet real-time requirements of these standards on a desktop PC with commercial operating system. Video processing dedicated kernel and hardware may be required to fulfill real-time timing constraints. If portability is needed, Digital Signal Processors (DSP's) are considered as an alternative to commercial processors due to their low power consumption and signal processing optimized hardware. Comparing with its predecessors, in order to meet increased computational complexity requirements and real-time timing constraints of H.264 Recommendation, Texas Instruments TMS320C64x [8] Family DSP's, which is a processor family under TMSC6000 Platform, have been selected among different DSP alternatives. Texas Instruments TMS320C64x DSP Family is designed to meet real-time performance requirements of imaging and video applications for both fixed and portable devices. TMS320C64x Family DSP' s have come in front with their high processing capability, large internal memory, and large external memory support and low power consumption [6]. As a result, TMS320C64x Family DSP's are chosen for real-time implementation of H.264 Encoder.

1.2 Scope of the Thesis

In this thesis, we have implemented H.264 Recommendation on TM320C64x Digital Signal Processor for QCIF frame size fitting real-time timing constraints. In the implementation, we try to enhance performance with TMS320C64x specific optimization methods and regular software development techniques. In addition to these methods, we try to improve encoder performance and reduce computational power by using Densely Centered Uniform-P Search for motion estimation and Sum of Absolute Differences for matching criterion.

1.3 Outline of Dissertation

Chapter 2 Predecessor video compression standards are explained briefly.

Chapter 3 Brief functional description of the H.264 standard, which mainly forms the core of the standard is stated. Some critical information about intra and

inter coding stages is explained. Enhancements achieved in H.264 Recommendation are pointed out.

Chapter 4 Hardware architecture and capabilities of TMS320C64x processor family is explained.

Chapter 5 Software architecture is explained, encoder flowcharts are given, Used algorithms that are not in the scope of the recommendation are explained. Code optimization methods are explained.

Chapter 6 Performance evaluation is made, results are presented

Chapter 7 Conclusions are stated.

CHAPTER 2

OVERVIEW OF VIDEO CODING STANDARDS

2.1 ITU-T Standards

International Telecommunication Union (ITU) was established in 1934. Two different committees inside ITU, which were CCITT (International Telephone and Telegraph Consultative Committee) and CCIR (International Radio Consultative Committee) joined in 1992 under the roof of CCITT. They formed ITU-T and ITU-R. ITU-T released H.26x Recommendations for video compression, which is mentioned briefly in the following chapters.

2.1.1 H.261

H.261 [2] is an international standard for Integrated Services Digital Network(ISDN) picture phones and video conferencing systems. ITU-T released H.261 in 1990. Typical image formats supported are QCIF (176x144) and CIF (352x288) at a frame rate between 7.5-30 frames per second. Aimed bit rate is multiple of 64 kbps, which is a bit rate of single ISDN channel.

H.261 standard uses 16x16 pixel macroblocks. For sampling of luminance and chrominance components, 4:2:0 sampling format is used.

For inter prediction, Integer pixel accuracy is accepted in recommendation. For single macroblock, single displacement vector exists. Adaptive loop filter is used for reducing blocking artifacts between macroblocks after reconstruction. Motion vectors are differentially encoded for inter frames.

For coding of the residuals, 8x8 DCT is used as a transform method. Quantization is performed in two steps for intra macroblock residuals. DC coefficients are quantized with uniform quantizer. AC residual coefficients of intra macroblocks and all residual coefficients of inter macroblocks is quantized using uniform threshold quantizer. Quantized coefficients are zig-zag scanned. Run-level coding is used for the entropy coding stage.

2.1.2 H.263

H.263 [3] is published by ITU-T in 1995 for picture phones over analog subscriber lines. Typical image formats supported are sub-QCIF, QCIF and CIF. Frame rate is below 10 frames per second. Targeted bit rate is about 20 kbps for Public Switching Telephone Networks (PSTN). This standard offers picture quality same as H.261 with half bit rate. H.263 is used for network video streaming. This standard is also core for H.263+ and H.264 standards.

In H.263, motion compensation accuracy is improved from integer pixel to half pixel accuracy. For DCT coefficients, 3-D Variable Length Coding (VLC) is used as an entropy coding method. Streaming overhead is reduced when compared with H.261. Number of picture formats supported is increased in H.263. Unrestricted motion vectors are used for motion vector compensation. Advanced prediction mode is added. Overlapped block motion compensation is supported.

2.1.3 H.263+

H.263+ [9] is the extended version of H.263. This standard can be thought to be between H.263 and H.264. Most of the improvements in H.264 are based on H.263+ enhancements.

2.2 ISO/IEC Standards

ISO (International Organization for Standardization) was established in1947. The mission of the organization is “to facilitate the international coordination and

unification of industrial standards". IEC (International Electro-technical Commission) was founded in 1956 for establishing international standards for all electrical technologies. Joint ISO/IEC Technical Committee has released MPEG-1 and MPEG-2 standards for video content storage.

2.2.1 MPEG-1

ISO/IEC Joint Technical Committee released MPEG-1 [4] standard in 1991 for video storage on CD-ROM media. Targeted bit rate was about 1.5 Mbps. Standards Typical image format was CIF with a frame rate of 30 fps. Standard does not support compression for interlaced frames.

2.2.2 MPEG-2

MPEG-2 [5] standard is released ISO/IEC in 1994. This standard is an extension of MPEG-1 for interlaced frames and optimized for resolution of 704x480 pixels, which is the resolution of NTSC TV standard. MPEG-2 is used for High Definition TV. Comparing with the MPEG-1, this standard supports efficient compression of interlaced digital video data at broadcast quality, which is not present in MPEG-1. New prediction modes are added for I frames. Additional scan pattern for DCT coefficients is used for scanning. For inter frames, new motion compensation scheme with 16x8 size blocks is introduced. Quantization scheme is also different from MPEG-1's, which increases coding efficiency. VLC tables have been improved and different scalability modes have been added.

CHAPTER 3

THE H.264 RECOMMENDATION

3.1 Overview

ITU-T and ISO/IEC are the two independent standardization organizations working independently. ITU-T video coding standards are named with H.26x (H.261, H.262, H.263 etc.) ISO/IEC standards are named with MPEG-x (MPEG-1, MPEG-2 etc. These two committees were working independently except the development of H.262/MPEG-2, which was released in 1990. Recently, these two committees have joined together and form Joint Video Team for the development of new video standard called H.264 [7] (H.26L, MPEG4-Part 10 Advanced Video Coding). H.264 Recommendation introduces latest innovations in video compression technology and represents great improvement when compared with the other standards in terms of performance and quality. H.264 standard aims all range of video applications from low bit rate applications to HDTV and high resolution DVD content. Development flow chart of the standards that are introduced by these two committees are shown in Figure 1 [6].

3.2 Comparison of H.264 with Other Standards

New compression techniques bring great compression efficiency to H.264 standard. H.264 standard offers up to 2x compression compared with MPEG-4 simple profile H.263, H.263+ and improvement in perception for almost all bit rates.

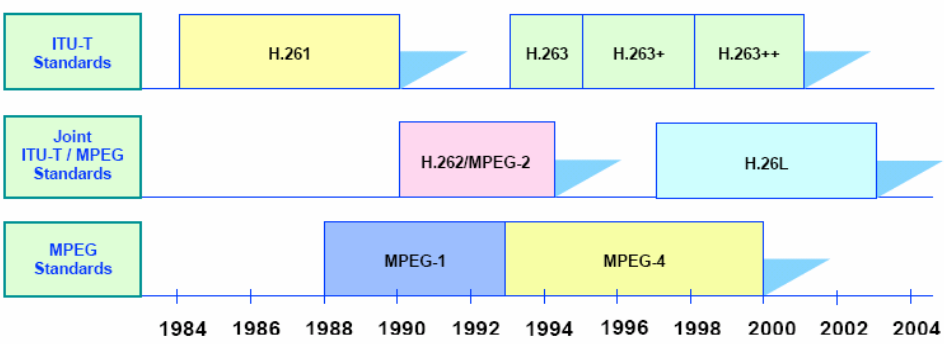

Figure 1 - Progression of ITU-T recommendations and MPEG standards [6].

H.264 gives great error resilience due to introduced VCL and NAL layer enhancements [10],[12],[13].

H.264 supports 28 different video input formats supporting interlaced scanned pictures in field mode [7].

H.264 has adapted for network transmission by adding one layer called Network Adaptation Layer (NAL) over Video Coding Layer (VCL) [10],[6],[11].

VCL enables efficient transmission of video data on network by representing video content in integer number of bytes units.

H.264 is highly adaptive for different applications, which require different type of delay characteristics. It may operate with low delay constraints for real-time applications or higher delay constraints for applications requiring more processing power, such as video content storage [10],[12].

Introduction of SP and SI frames enables fast random access for video decoders. These frames are switching frames. For example, these types of frames may be used for switching from low bit rate to high bit rate in the same video stream by monitoring the available bandwidth [12].

Considering the functional building blocks of the encoder, H.264 is similar as previous standards. It has, transform, quantization, motion estimation and motion

compensation, entropy coding blocks. In order to reduce blocking artifacts, loopfilter [14].block is introduced in H.264.

Multiple reference frame selection feature allows motion to be estimated from five different past reference frames. This option brings the maximum compression efficiency for periodic motion. Advantage of multiple reference frame selection is shown in Figure 2 [15].

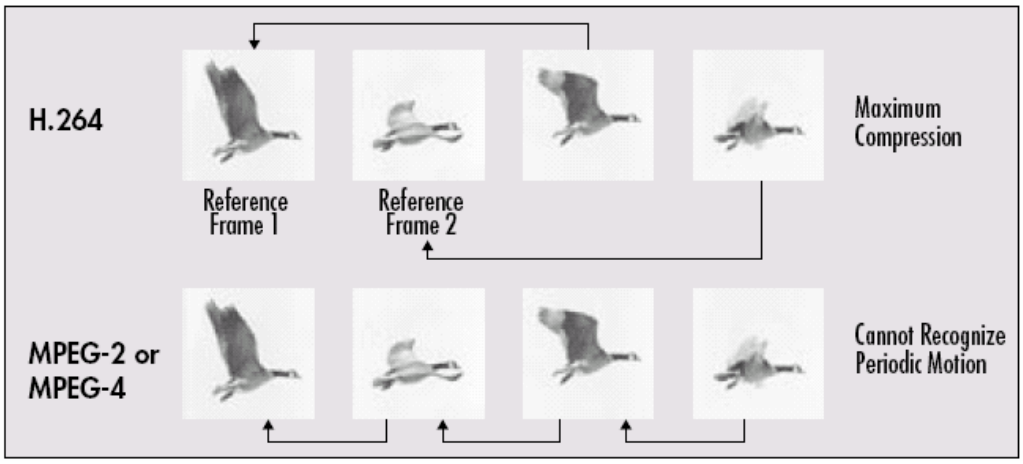

Figure 2 - Advantage of H.264 compression [15].

Reduced size of motion blocks supplies strong motion isolation from neighboring blocks compared with other standards [15].

The major enhancements are inside functional blocks. Considering intra prediction, H.264 uses 16x16 block sizes or 4x4 block sizes optionally. New prediction modes for intra prediction have been added to improve performance. For the inter prediction, motion estimation is done with motion blocks of different shapes, that are not used in predecessor standards. Fractional sample resolution is ¼ pixel for luminance, 1/8 pixel for chrominance vector component. This increases precision of motion vectors. Considering transform coding block, newly developed integer based transform is used and this increases real-time performance. For the entropy coding block, Context Adaptive Variable Length

Coding (CAVLC) [16] and Context Based Adaptive Arithmetic Coding (CABAC) [17] is developed.

3.3 Description of H.264 Recommendation

H.264 Encoder diagram is given in Figure 3 [11]. Following chapters takes encoder diagram shown in the Figure 3 [11]. as a reference.

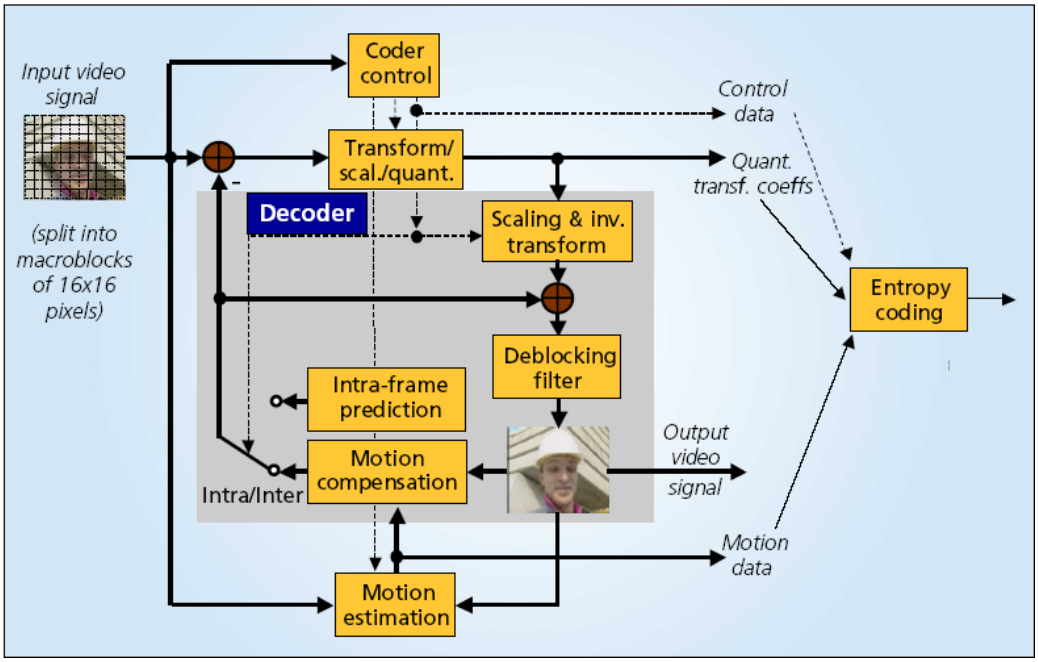

Figure 3 - H.264 Encoder block diagram [11]

3.3.1 Video Input and Video Frame Structure

Example QCIF picture consists of two slices is given in Figure 4. Each slice is represented by two different colors. Slice is the largest building block of H.264 frame. All the parameters of decoder are refreshed at slice boundaries. Picture may consist of one or more slices. Incrementing the number of slices in single picture improves error resilience but reduces coding efficiency [10].

Each slice consists of regions called macroblocks shown with squares. For luminance and chrominance size of the macroblocks are different. Each

macroblock consists of 16x16 luminance pixels, two 8x8 chrominance pixels. Sampling format is chosen as 4:2:0. Chrominance samples are down-sampled because chrominance component of pixel does not effect perception as much as luminance component. Human eye is more sensitive to luminance component.

Sample QCIF picture of H.264 is shown in Figure 4.

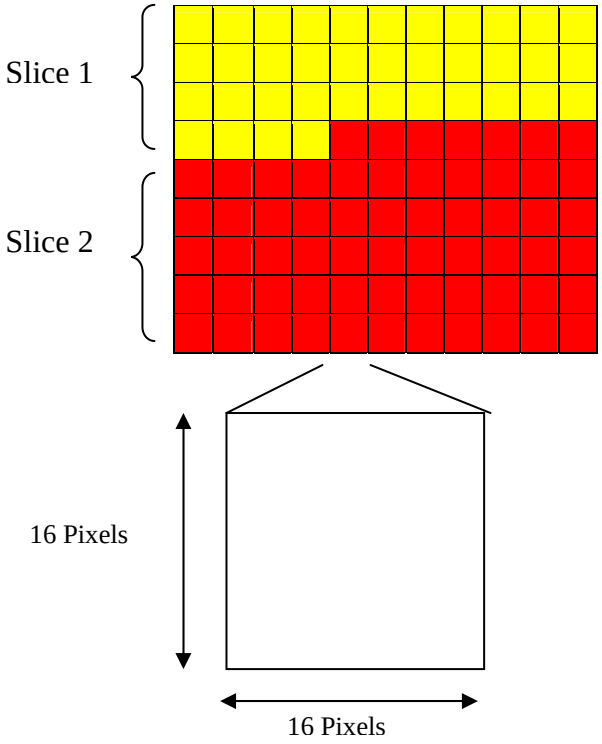

Figure 4 - Sample QCIF frame with two slices showing single luminance macroblock

This example of QCIF picture consists of two slices as shown in figure by two different colors. Slice is the largest building block of H.264 frame. All the parameters of decoder are refreshed at slice boundaries. Picture may consist of one or more slices. Incrementing the number of slices in single picture improves error resilience but reduces coding efficiency.

Each slice consists of regions called macroblocks shown with squares in Figure 4. For luminance and chrominance data, size of the macroblocks is different due to sub-sampling of chrominance components.

Each macroblock consists of 16x16 luminance pixels, two 8x8 chrominance pixels for U and V component. Sampling format is chosen as 4:2:0. Chrominance samples are down-sampled because chrominance component of pixel does not effect perception as much as luminance component. Human eye is more sensitive to luminance component than chrominance component.

If the current picture is in field mode (interlaced scan), slices contain integer number of macroblock pairs that, each macroblock pair contains two macroblocks.

3.3.2 Network Adaptation Layer

Predecessor standards have some disadvantages due to representation of compressed video stream while transmitting or storing video content. H.264 introduced new approach for representation of video data, which is called as Network Adaptation Layer (NAL). This concept separates Video Coding Layer, where compressed video data is efficiently represented, from the transport layer or storage media. NAL enables easier packetization of VCL data. Every unit in NAL contains integer number of bytes. This reduces overhead of processing for communication and content storage or data retrieval. NAL unit standardizes an interface for packet oriented and bit-stream systems.

3.3.3 Video Coding Layer

Video Coding Layer (VCL) of H.264 bit-stream represents compressed visual data without any packetization overheads introduced with the NAL. Description of the following parts belongs to VCL data processing.

3.3.3.1 Intra (Spatial) Prediction

Intra prediction exploits spatial redundancies in a frame. Considering a single macroblock in a frame, neighboring macroblocks usually have correlated pixel values. If neighboring blocks in the same frame are used for predicting current block pixel values, which are being constructed at the decoder side, this type of prediction is called intra prediction. Intra predicted frames have greater PSNR than inter encoded frames, but they contain more information than inter encoded frames, which reduces compression ratio. This type of pictures is sent less frequently, compared with inter encoded pictures, due to their bandwidth cost. These pictures are used for refreshing the screen and diminishing propagation errors that came from adjacent decoding of inter encoded frames. Error propagation occurs due to error on the channel in which video stream is being transmitted or due to encoding inefficiency resulting from compression. Intra encoded frames diminish propagation errors from inter encoded pictures due to their spatial compression nature. For this reason, intra encoded pictures are called as Instantaneous Decoding Refresh (IDR) pictures in the recommendation.

Error propagation in the inter frames was also problem for the preceding standards. H.264 also offers new macroblock based solution for erroneous transmission of inter frames. If the distortion of the video transmission channel can be monitored in real-time, using intra prediction, intra encoded macroblocks can be sent inside inter frames between inter encoded macroblocks. In order to optimize the number of intra encoded macroblocks in an inter frame, rate-distortion optimization method can be used [13].

3.3.3.1.1 Intra Prediction Modes

3.3.3.1.1.1 Luma Prediction modes

Intra prediction modes represent linear combination of neighboring macroblocks or partitions' pixel values with some predefined scaling factors. According to the

prediction direction, weights of the scaling coefficients change.

3.3.3.1.1.1.1 Prediction Modes for 16x16 Blocks

There are four modes, meaning four prediction directions of spatial prediction for 16x16 block size. Up and left neighboring 16x16 macroblocks' samples are used for predicting the current macroblock's pixel values.

Samples of current 16x16 block is predicted from linear combinations of neighboring 16x16 blocks' samples changing according to type of the prediction mode. Details can be found in [7].

3.3.3.1.1.1.2 Prediction Modes for 4x4 Blocks

There are nine intra prediction modes, meaning nine prediction directions, for the prediction of current 4x4 block pixel values from the neighboring blocks

Samples of 4x4 block is predicted from linear combinations of neighboring 4x4 blocks' samples changing according to type of the prediction mode. Details can be found in [7].

3.3.3.1.1.1.3 Chroma Prediction Modes

Prediction of chrominance samples is the same for both 16x16 and 4x4 luminance predicted macroblocks. There are four modes for intra chrominance prediction

Chrominance samples are predicted by using neighboring blocks' coefficients with one of these four prediction modes. Predicted pixel values are calculated by using linear combinations of neighboring pixels with linear combination coefficients changing according to the prediction mode (direction). Details can be found in [7].

3.3.3.2 Inter (Temporal) Prediction

Inter prediction exploits temporal redundancies in video stream. Temporal means related with time. This type of prediction uses similarity of one or more video frames in the time order. Motion vectors belonging some specific block of a frame are used for representing the single motion of a motion block in the picture. Main elements of this type of prediction are motion vector data and residual data after motion estimation.

3.3.3.2.1 Motion Vector Data

H.264 introduces different motion block sizes for motion estimation with resolution of ¼ for luminance motion vectors, 1/8 for chrominance motion vectors. Smallest size of blocks for motion block is 4x4. For a single macroblock, up to sixteen, motion vector data may have to be sent. If this data is sent directly, it may cost large amount of bits for transmission of motion vector data [18]. Motion vectors are highly correlated to their neighboring motion vectors meaning, prediction residuals are usually small. So that motion vector data is differentially coded, which means, it is predicted and prediction residuals are sent to decoder.

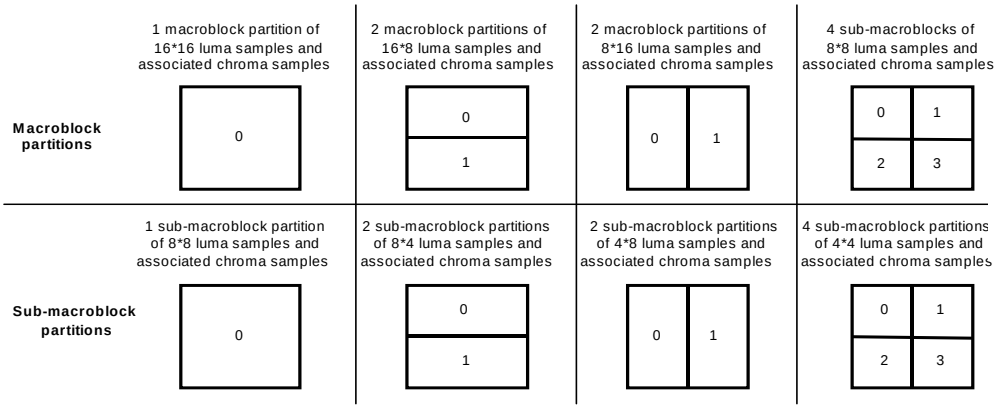

Figure 5 – Available macroblock partitions for H.264 [7].

Macroblock partitions for estimation of the motion vector components are given in Figure 5 [7]. Optimization of macroblock partition sizes become challenging issue for encoder. For the regions where there is a small motion, larger macroblock partitions have to be chosen.

For detailed surfaces, smaller macroblock partitions must be chosen for prediction. This affects encoder efficiency and performance. Sample of optimized selection of motion vector blocks is shown in Figure 6 [19].

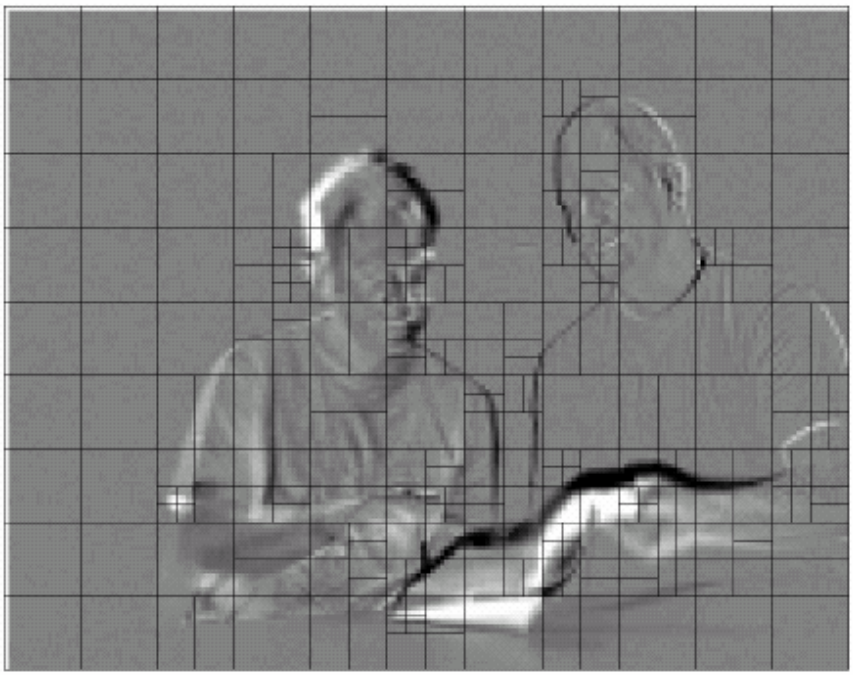

Figure 6 – Appropriate selection of partitions [19].

Motion vector data composed of, motion vector component in the horizontal direction, motion vector component on the vertical direction and the reference index for the source frame. This data is used for reconstruction of current picture from the previously reconstructed frames at the decoder side.

3.3.3.2.2 Prediction of Luma Motion Vector Components

Luminance motion vectors are predicted to send motion vector data differentially thus reduce bandwidth used for sending motion vector data. Two types of

prediction methods are used for motion vector prediction these are Directional Segmentation Prediction and Median Prediction.

3.3.3.2.2.1 Directional Segmentation Prediction

Directional segmentation prediction is only used for motion block types of 8x16 and 16x8. Prediction rule is given below.

If macroblock partition width is 16 and macroblock partition height is 8 and if this is the partition on top, predicted motion vector this macroblock partition is the same as motion vector above. If this is the partition below, Predicted motion vector is the same as motion vector on the left of this partition.

If macroblock partition width is 8 and macroblock partition height is 8 and this is the macroblock partition on the left, predicted motion vector data for this partition is the same as motion vector on the left of this partition. If this is the macroblock partition on the right, predicted motion vector is the same as the motion vector, which belongs to the block on the corner of this macroblock partition.

Further details can be found in [7].

3.3.3.2.2.2 Median Prediction

For this type of prediction, motion vector components are preprocessed according to their availability status and median of neighboring motion vector components are used for prediction current motion vector block's motion vector components. Details can be found in [7].

3.3.3.2.2.3 Chrominance Vector Prediction

Chrominance motion vectors are derived from the luminance motion vectors. These vectors have higher resolution than luminance vectors. Chrominance vectors are exactly the half of the luminance vectors. As a result, maximum

resolution of chrominance motion vectors is increased to 1/8 pixel.

3.3.3.2.3 Reference Frame Selection

For each macroblock, there can be at most four different reference frames for inter prediction for H.264 recommendation. Each reference frame belongs to each 8x8 sub-partition of a luminance macroblock, thus 4x4 sub partition of chrominance macroblock.

3.3.3.2.4 Fractional Sample Interpolation

In order to construct macroblock partition from reference picture data and fractional motion vector data, some processing is needed. To find the pixel values with the resolution of ½ pixels, Six-tap Finite Impulse Response Filtering (FIR) of pixel values on integer locations are needed.

Pixels values at ¼ pixel resolution is found with bilinear interpolation of pixel values at half or integer resolution positions.

To find chrominance samples at fractional pixel positions, simple linear interpolation of pixel values at integer locations are used.

Details of fractional sample interpolation can be found in [7].

3.3.3.3 Coded Block Pattern (CBP)

Coded bock pattern is information sent to decoder increase decoding speed. This information is not sent for Intra 16x16 coded macroblocks. Coded Block Pattern shows, which of the 8x8 blocks out of 4 for luminance and which of the 4x4 blocks out of 4 for chrominance contain non-zero transform coefficients.

3.3.3.4 Transform and Quantization

Prediction residuals from intra or inter estimation are transformed from spatial

domain to frequency domain. This is achieved in transform block of the encoder. Transformed residuals are called transform coefficients. Number of nonzero coefficients are reduced and gathered together in the scan order, which is given in following chapters. After quantization, number of non-zero coefficients reduces. Quantization of transformed residuals reduces the number of bits to represent transformed nonzero values, which results in compression. De-quantization stage recovers quantized data with some inverse quantization mismatch; this error is called as quantization error. This mismatch is kept at optimum by changing the quantization parameter on each macroblock differentially.

3.3.3.4.1 Integer Transform

H.264 standard introduces new transform method called Integer Transform. This transform is also called Simplified Discrete Cosine Transform (Simplified DCT)[20]. Transform block size is reduced from 8x8 of DCT to 4x4 of Integer Transform. Reduced size of transform block reduces blocking and ringing artifacts caused by inverse transform and quantization.

Integer Transform is derived from DCT by approximating transform coefficients on transform matrix of DCT to fractional numbers, which changes the form of DCT. Resulting new transform requires no multiplications, only additions and shifts. Integer transform requires at most 16 bits to complete all arithmetic procedures with so little PSNR performance decrease from DCT. Implementation of Integer Transform with 16-bit arithmetic with shifts and additions makes it widely usable for conventional processor types with higher performance than DCT. Inverse transform mismatch problem of DCT diminishes on Integer transform because there are no rounding errors resulting from division and multiplication of DCT coefficients. As a result, transform method without inverse mismatch problem gives ride to fast implementations on large variety of processor families [20].

Different matrices are used for transformation and inverse transformation. For

4x4 luminance and chrominance coefficient transformation and inverse transformation, matrices in Figure 7 [20] are used.

$$Y = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix}$$

$$X = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} y_{00} & y_{01} & y_{02} & y_{03} \\ y_{10} & y_{11} & y_{12} & y_{13} \\ y_{20} & y_{21} & y_{22} & y_{23} \\ y_{30} & y_{31} & y_{32} & y_{33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix}$$

Figure 7 - Integer transform and inverse transform matrices [20]

H.264 performs additional transform for DC coefficients of luminance blocks and chrominance blocks. 4x4 Luminance DC coefficient matrix is formed from DC coefficient of each 4x4 transformation block. These DC coefficients are further transformed with matrix given in Figure 8 [20].

$$Y_D = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} x_{D00} & x_{D01} & x_{D02} & x_{D03} \\ x_{D10} & x_{D11} & x_{D12} & x_{D13} \\ x_{D20} & x_{D21} & x_{D22} & x_{D23} \\ x_{D30} & x_{D31} & x_{D32} & x_{D33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) // 2$$

$$X_{QD} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} y_{QD00} & y_{QD01} & y_{QD02} & y_{QD03} \\ y_{QD10} & y_{QD11} & y_{QD12} & y_{QD13} \\ y_{QD20} & y_{QD21} & y_{QD22} & y_{QD23} \\ y_{QD30} & y_{QD31} & y_{QD32} & y_{QD33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

Figure 8 - Luminance DC transform and DC inverse transform matrices [20]

The same process as luminance DC coefficients is applied to chrominance coefficients too. 2x2 Chrominance coefficient matrix is formed with DC

coefficients of each 4x4 chrominance transform blocks. This 2x2 matrix is transformed with the matrices given in Figure 9 [20].

$$Y_D = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_{D00} & x_{D01} \\ x_{D10} & x_{D11} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$X_{QD} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} Y_{QD00} & Y_{QD01} \\ Y_{QD10} & Y_{QD11} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Figure 9 - Chrominance DC transform and DC inverse transform matrices [20]

3.3.3.4.2 Quantization

Transformed coefficients are quantized with newly developed H.264 quantization scheme ([20],[21]). This method eliminates redundant precision on high frequency coefficients, supplying the same perceptual quality as original frame by eliminating high frequency components. For a single macroblock, total of 52 quantization parameters can be chosen adaptively and may be sent differentially for each macroblock. These step sizes increase about 12.5% rate, instead of increasing constantly. This introduces better quantization steps because higher transform coefficient levels are quantized with higher step sizes.

3.3.3.5 Scanning

Two scan patterns are used for H.264 transform coefficients. One is zig-zag scan, and the other is field scan. Zig-zag scan is used for frame macroblocks, field scan is used for field macroblocks

Scanning is used for ordering transformed coefficients in the frequency order. Coefficients are scanned in the order shown in Figure 10 [7].

Scanning stage prepares transformed and quantized residuals for entropy coding stage.

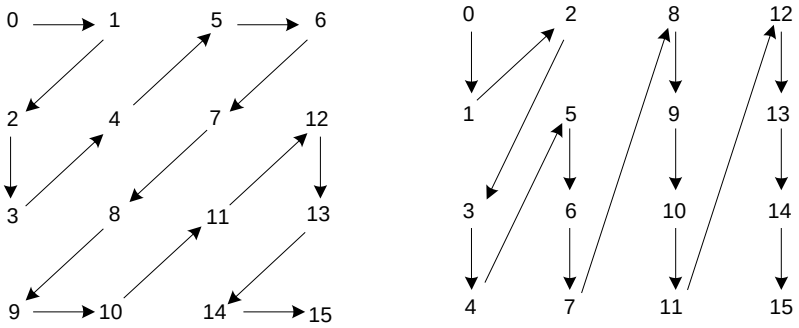

Figure 10 - Coefficient scan patterns for Zig-zag Scan and Field Scan [7]

3.3.3.6 Deblocking Filter

Video coding using block-based methods introduces artifacts caused by the method. These artifacts are visible at the edges of compression blocks due to different compression characteristics of coded blocks from the neighboring blocks in terms of transform, quantization etc. Deblocking filtering reduces blocking artifacts at the edges of compression blocks and increases visual quality. H.264 uses in-loop deblocking filtering method to overcome blocking effects [14]. Deblocking filter is used before motion compensated prediction stage. This increases compression efficiency as well. Deblocking filter is used adaptively by using some compression parameters of corresponding blocks [7].

3.3.3.7 Entropy Coding

Entropy coding stage is used for representing generated compressed data stream with lesser bits that as it is. Entropy coding stage compresses stream by representing higher frequency symbols with less bits than lower frequency symbols. Entropy coding compression is perfectly reconstructable; there is no inverse mismatch problem. Compressed stream is get exactly as it was before compression. H.264 introduces two new entropy coding compression methods for entropy coding stage of decoder. These methods are Context Adaptive Binary

Arithmetic Coding (CABAC) and Context Adaptive Variable Length Coding (CAVLC) [7].

3.3.3.7.1 Context Adaptive Binary Arithmetic Coding (CABAC)

One of the most important enhancements that H.264 introduces is new entropy coding method named Context Adaptive Binary Arithmetic Coding (CABAC)[22].

CABAC ’ s extended name specifies characteristics of this method. It makes use of Arithmetic Coding [23] with finite binary alphabet. Arithmetic coding implementations can be made with larger alphabet sizes [24], [25], but this increases complexity of the implementation. Due to reduction of alphabet size to binary, arithmetic coding implementation in microprocessors is simplified. As it can be understood from the binary alphabet, all the symbols are first binarized according to their type then they are arithmetic coded. Depending on the symbol type, probability variables used by arithmetic coding are initialized with some initial values, and then they are updated according to distribution of symbol values, to increase efficiency of arithmetic coding.

CABAC is actually composed of three main stages;

Binarization is applied to symbols that have non-binary values. All non-binary symbols are binarized using four different binarization methods, which exploit coding efficiency by using symbols’ binary bit distribution. Details of these can be found in [7]. Binary valued symbols are directly coded with bypass mode of arithmetic coding engine.

Context model selection is applied some specific bins of symbol. “Bin” means bit position in symbol. For all bins of specific symbol type, there are different probability models used for initializing binary arithmetic coding engine parameters, called context models. Context model parameters show probability of each bin being “1” or “0”.

After context model decision, Adaptive Binary Arithmetic Coding is used for binarized symbol. Probability parameters needed for arithmetic coding is taken from context model. After encoding a symbol, context model, which keeps the probability statistics, is updated.

3.3.3.7.2 Variable Length Coding (VLC)

If entropy coding mode for stream is not CABAC, Variable Length Coding is used as an alternative entropy coding method. VLC simply uses the principle of assigning shorter code words to frequently observed symbols, longer code words to rarely observed symbols to reduce overall entropy of the stream. Details of this well known method can be found in [1]. H.264 uses Context Adaptive Variable Length Coding (CAVLC) for coding of residual data. Other syntax elements are Exp-Golomb coded. CAVLC is a new concept brought by H.264

CHAPTER 4

TEXAS INSTRUMENTS TMS320C64X DIGITAL SIGNAL PROCESSOR

Innovation in technology resulted in need for higher processing capabilities than before. One of these innovation areas is a signal processing area. Commercial Signal processing applications are used widely today. As the throughput needed for signal processing applications increased, commercial processors were unable to handle real-time processing power requirements. Digital Signal Processors (DSP' s) are designed to meet the digital signal processing requirements. They were designed to be faster than commercial while computing signal processing operations by taking advantage of their architectures. Main advantage of a DSP is, its ability to perform one or more Multiply And Accumulate (MAC) instructions in a single machine cycle, which is not supported by commercial processors. DSP architectures have begun to evolve for their efficient use in signal processing area.

Texas Instruments TM320C64x DSP family is specifically designed for signal processing applications that require high processing power, such as video processing, speech processing and communication applications. Reduced power consumption of this processor family is one of the most important feature considering embedded designs. Texas Instruments 32-bit TMS320C64x [8] core is designed by using second-generation VelociTI™ Very Long Instruction Word (VLIW) architecture by Texas Instruments. This architecture allows fetching of

multiple instructions in a single machine cycle. 600 MHz TMS320C64x DSP offers 4800 MIPS (Million Instructions Per Second), with the increasing clock rate, 1.1 GHz or over, TMS320C64x can process 8800 MIPS or more. C64x processors have three external bus interfaces for peripheral accesses, bus clock speed of one of these buses may be increased to 133 MHz for use of new fast memories, A-D converters etc. Another bus provides interface for slow peripherals. Third type of bus is used for communicating with industrial standard buses like PCI. Three Multi Channel Buffered Serial Ports (MCBSP) supplies connectivity to serial peripherals with up to 100 Mbits/sec[8] transfer rate. C64x's instruction set and pipeline allows multiple instructions to be scheduled in parallel. Due to these reasons, this processor family is chosen for the implementation of the H.264 encoder.

4.1 Architecture

TMS320C64x architecture is an extension of VelociTI™ architecture developed by Texas Instruments in 1997. This architecture first applied to C62x and C67x family DSP families. TMS320C64x architecture formed of multiple execution blocks (multiplier, added etc.) working in parallel. C64x architecture is extended version of the C62x architecture. So that all C64x family processors are code compatible with C62x family processors. Increased orthogonality and parallelism of instruction set, extended data load and store paths, increased number of register files, packed data processing, increased clock speed are main enhancements from the C62x family of processors.

TMS320C64x CPU core is shown in Figure 11 [8],core is composed of two register files, eight functional units, instruction fetch, instruction dispatch units with advanced instruction packing capability, instruction decode unit, control register and Interrupt control logic.

TMS320C64x CPU Core consists of two register files, Register File A and Register File B. Each of these register files have thirty-two 32-bit general-purpose registers, total of sixty-four 32-bit registers. These registers may be used

for arithmetic or conditional operations. Registers A0, A1, A2, B0, B1, B2 may be used for conditional operations. Registers A4-A7 and B4-B7 can be used for circular addressing registers. These register files support data sizes 8-bits, 16-bits, 32-bits, 40-bits, 64-bits data sizes for processing. Packed data types can store four 8-bit or two 16-bit data in a single 32-bit register or four 16-bit data in two 32-bit registers.

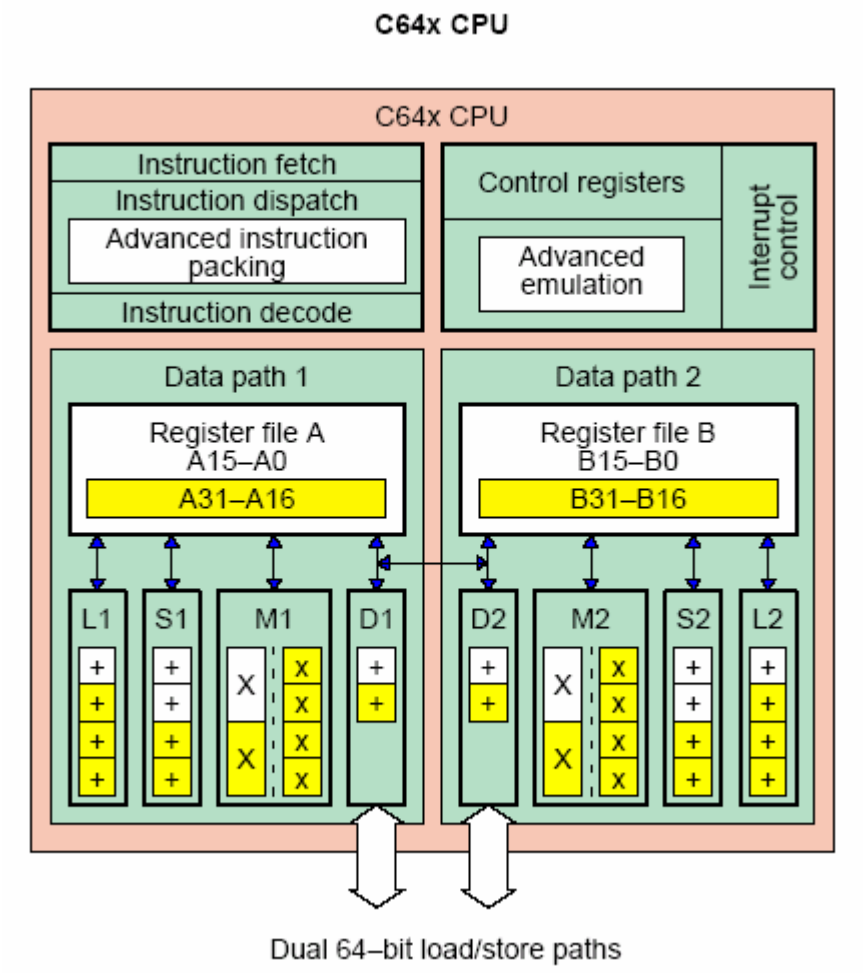

Figure 11- TMS320C64x CPU Core [8]

Eight functional units can be divided into two groups. Each group has the same functionality. There are four different types of functional groups. Names of these functional units are L, S, M, D, which are shown in Figure 11 [8].

Operations that can be performed in these functional units are given in Appendix A.

There are two register cross paths between two groups of functional registers, allowing that, functional from a data path 1 or 2 can access 32 bit operand from the opposite path as shown in Figure 13. This increases orthogonality, thus compiler efficiency.

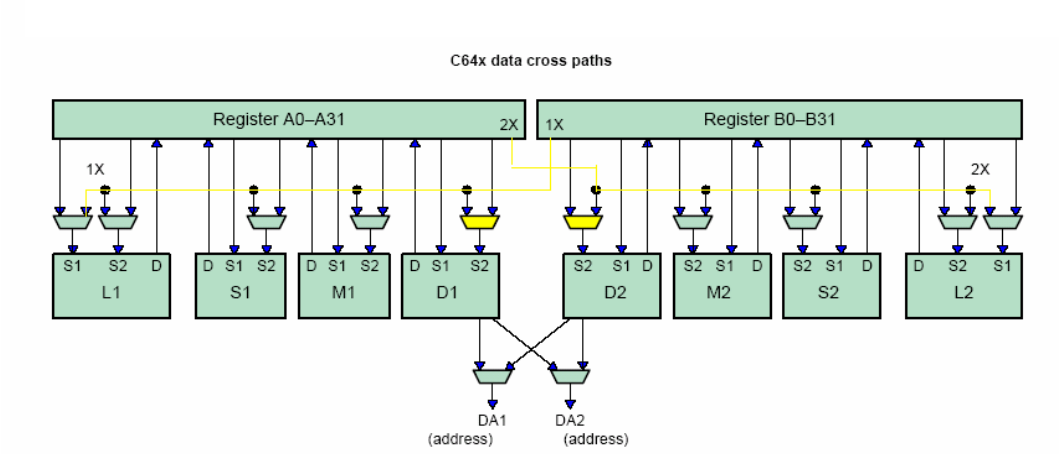

Figure 12 – C64x data cross paths [8]

C64x allows 32 bit load and store operations. Here are total of four, two load, two store paths to registers. These load and store paths are shown in Figure 12-13 [8]. C64x has one major enhancement compared with C62x architecture, which is, it can access any word or double word at byte boundaries without alignment. It can use non-aligned load or store operations.

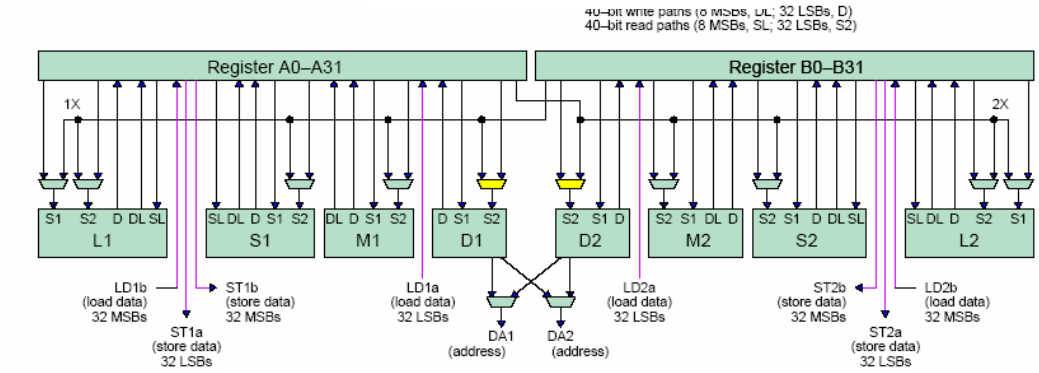

Figure 13 - C64x memory load and store paths [8]

C64x can operate on packed data types. As an example, four 8-bit operands may be subtracted with other four operands without requirement of being on different 32-bit words. This is called data packing. C64x instructions support 8-bit, 16-bit operations. Data packing reduces number of machine cycle and memory needed to perform instructions with 8-bit or 16-bit operands.

C64x family processors have programmable two level cache. 16 Kbytes of Level 1 program and data cache running at full CPU speed is present in the core. There is 1024 Kbytes of on chip RAM on C64x processors. Level 2 Cache may be mapped over this memory by configuring the related registers. Size of the cache Level 2 cache can extend from 32 Kbytes up to maximum of 256 Kbytes. Structure of these memories are shown in Figure 14 [8].

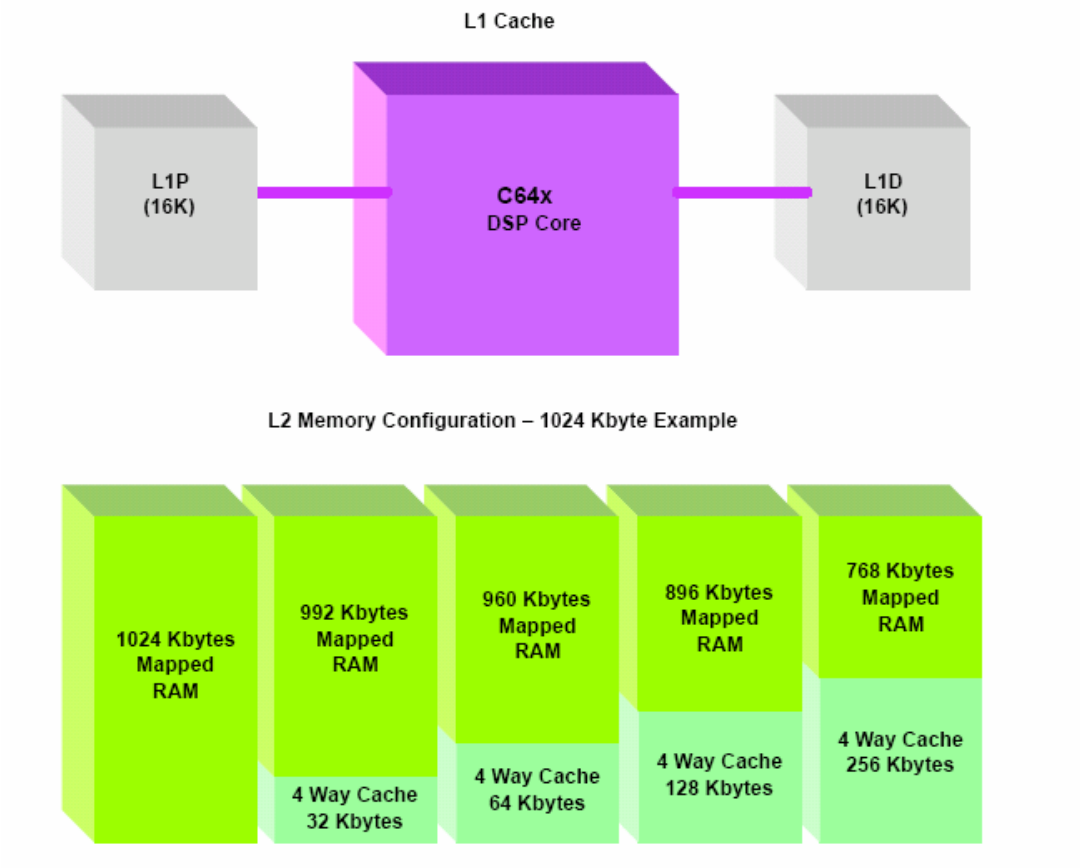

Figure 14 – C64x internal memory architecture [8]

There is Enhanced DMA Controller on C4x family and it can support up to 64 independent channel transfers

C64x processors have three external buses for communication with parallel peripherals. Two External Memory Interfaces (EMIFA, EMIFB) and one Host Ports Interface (HPI) exist. EMIFA is 64-bit transfer bus and may be used connection to higher speed memories such as fast SRAM s. EMIFB is 16-bits wide and may be used for any peripheral supporting this bus width. 32-bit HPI supports communication interface between other processors of industrial type. In some models of C64x, HPI is replaced by PCI interface. PCI bus supplies interface for PCI devices.

4.2. Code Generation on TI TMS320C64x Platform

Texas Instruments TMS320C64x family Digital Signal Processors use Code Composer Studio Integrated Development Environment (CCS IDE) for all stages of code generation. CCS is an integrated tool that, code generation, linking and compiling is performed on the same platform. It uses built-in compiler, linker and assembler for the code generation without the need for manual manipulation such as command line entries. Project environment details are given in Appendix B.

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 Virtual System Architecture

Texas Instruments TMS320C64x Cycle Accurate Simulator is used for implementing the proposed system and measuring the system performance. Virtual system architecture is designed and according to this virtual architecture, system memory map is formed. Video source supplies QCIF frames to external fast SDRAM. The same memory is used for storage of rarely accessed space consuming data (Frame store data, final compressed stream etc.).Virtual system architecture is shown in Figure 15.

Project based implementation details are explained in Appendix B.

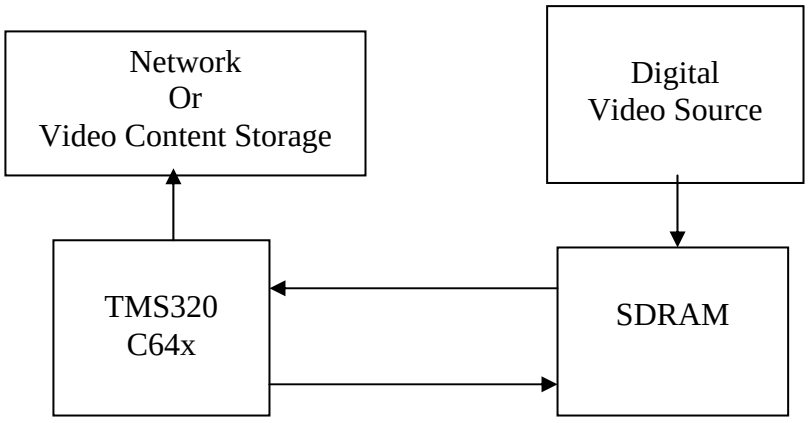

Figure 15 – Virtual system architecture

5.2 Encoder Architecture

H.264 encoder block diagram is given in. Figure 16.

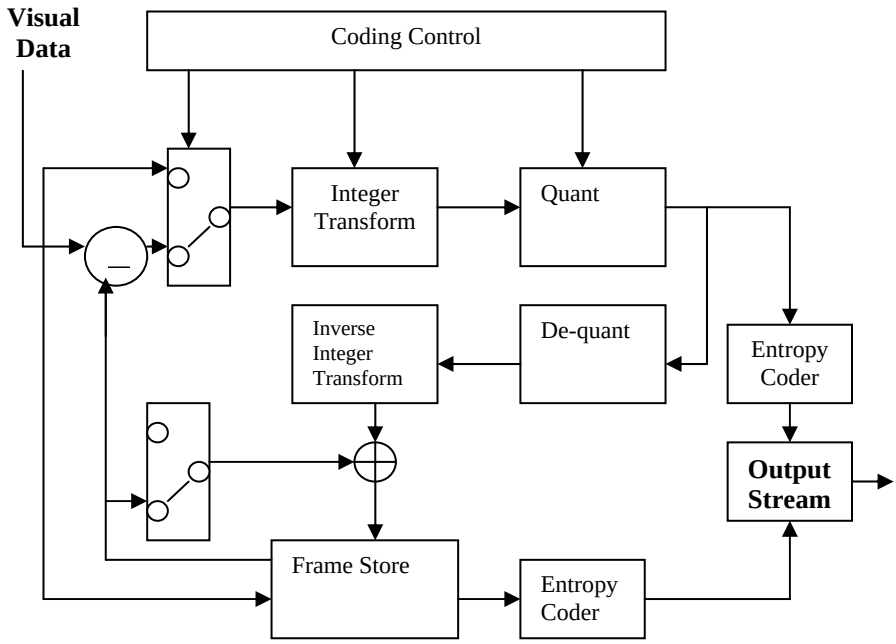

Figure 16 - H.264 Encoder block diagram

As it can be seen, building block operations run sequentially, so that, performance of an encoder can be calculated as sum of individual performances of building blocks of an encoder.

Encoder blocks are implemented considering sequential structure of the encoder architecture. Each building block is built up individually and integrated together.

5.2 Encoder Specific Algorithms

H.264 Recommendation only specifies the decoder of the standard. H.264 encoder also consists of decoder block for feedback part of an encoder, which includes motion compensation block. Some parts of an encoder is not specified in the recommendation. As an example, best matching criterion for motion estimation and intra prediction is not specified in the recommendation. Implementation of some parts of an encoder left to the one who implements the

encoder. Decoder only specifies how to reconstruct coded video sequence with the format given in the recommendation.

In order to find best matching prediction, some comparison between predicted and original block should be made. For this implementation Sum of Absolute Differences (SAD) method is used for best matching criterion.

For motion estimation, Densely Centered Uniform-P Search algorithm is used.

5.2.1 Sum of Absolute Differences

While predicting pixel values of a macroblock or a sub block, there should be criterion in order to find best matching prediction. Sum of Absolute Differences is used for this aim in this implementation.

While calculating SAD, absolute difference of predicted and original target block if found for each entry. Then sum of these absolute differences is calculated. Result is Sum of Absolute Differences for corresponding macroblock or sub block.

Smaller SAD value represents better prediction for corresponding block.

5.2.2 Densely Centered Uniform-P Search

Densely Centered Uniform-P Search algorithm [26] is a fast search algorithm using the strategy of sampling the search area with some predefined motion vectors and making comparison among that motion vector positions.

Observations on block matching algorithms in [26] showed that motion vectors tend to congregate at the origin of the search area. It is found that [26] in cheerleaders video sequence, %50 of the motion vectors have the length zero or one. So that, scan starts at positions (0,0), (0,1), (0,-1), (1,0), (-1,0). Weighting function (SAD) is used for that target points and best weighting motion vector is kept as best vector. Other levels of DCUPS are applied for some encoder specific

conditions that are explained in following chapters.

5.4 Software Architecture

In this section, functionality of the implemented encoder sections and flowcharts that are specific to the implementation are given and explained.

5.4.1 Encoder Flowcharts

Flow diagrams of the encoder implementation and encoder specific implementation details are given in the following chapters.

5.4.1.1 Main Loop

H.264 Entry point is the starting point of the software, which performs encoding and prepares bit-stream. Before that point, some environmental initializations, necessary for TMS320C64x platform is completed by the compiler initialization routines. These platform initialization routines are embedded in the “rts6400.lib“ software library created by Texas Instruments and automatically used by the C Compiler. After platform initialization, main loop runs. Main loop of the encoder is given in Figure 17.

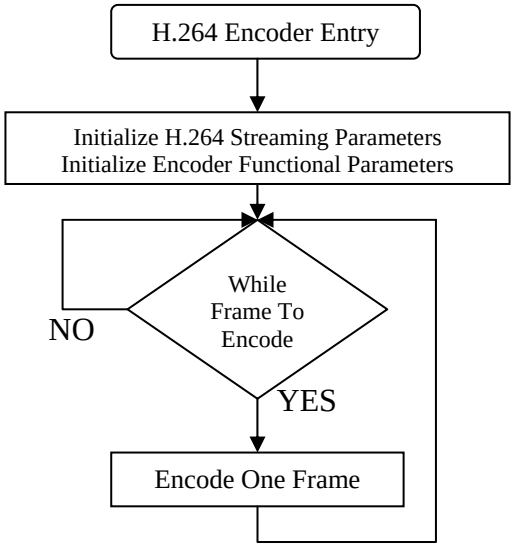

Figure 17 - Main loop flowchart

Necessary NAL units and global parameter sets, such as sequence parameter sets and picture parameter sets are inserted into stream at this stage.

Frame decoding that are common for inter and intra blocks are given in Figure 18.

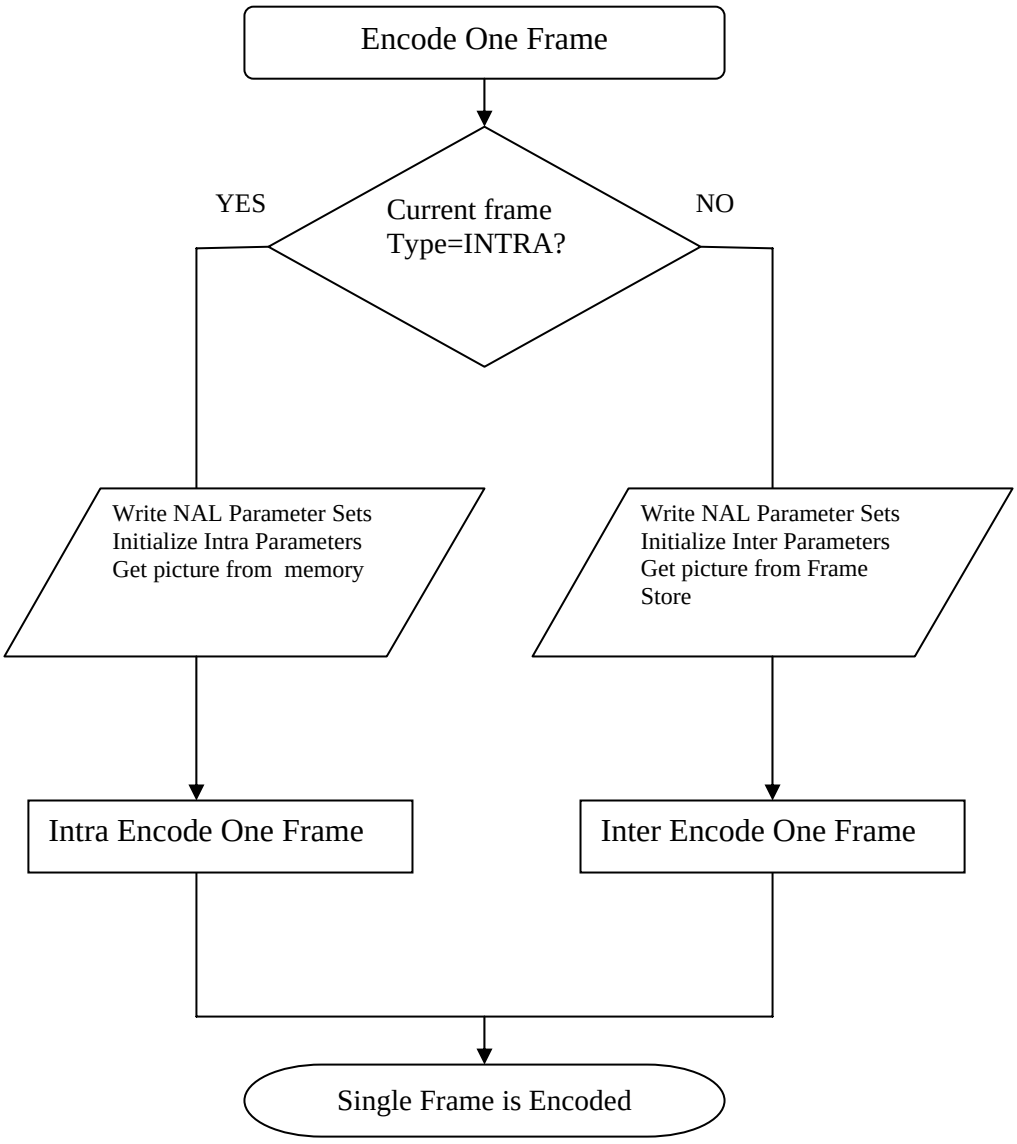

Figure 18 - Encode One Frame flow chart

Basic implementation steps of encoding single frame are the same for intra and inter encoding stages. For this reason their single frame encoding flowcharts are the same and shown in Figure 19.

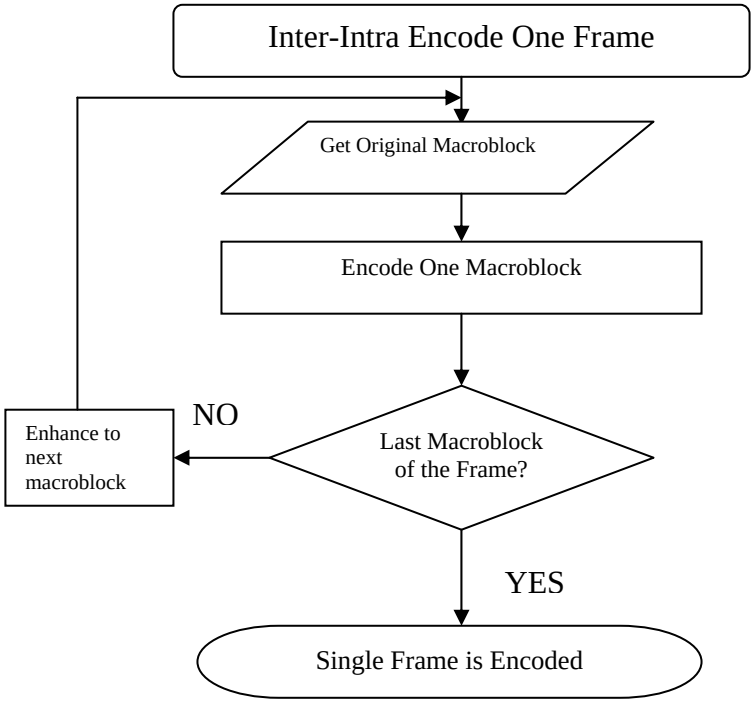

Figure 19 - Inter-Intra Encode One Frame flow chart

5.4.1.2 Intra Encoder

For decision of Intra prediction modes, variance of the processed macroblock is calculated. If variance is larger than predefined threshold value, 4x4 intra mode is used for intra prediction. For smoother surfaces, 16x16 intra modes are used. In order to find best 16x16 or 4x4 intra prediction mode, all prediction modes for target macroblock or sub blocks are applied and the mode, which gives lowest SAD value, is chosen as best intra prediction mode.

For both Intra4x4 prediction and Intra 16x16 prediction same algorithm is used for prediction. Before prediction of the samples, availability of the neighboring pixels and pixel values are decided. After this process, prediction is performed for each mode. Best result is stored in global storage.

Intra encoding flowchart for a single intra macroblock is given in Figure 20.

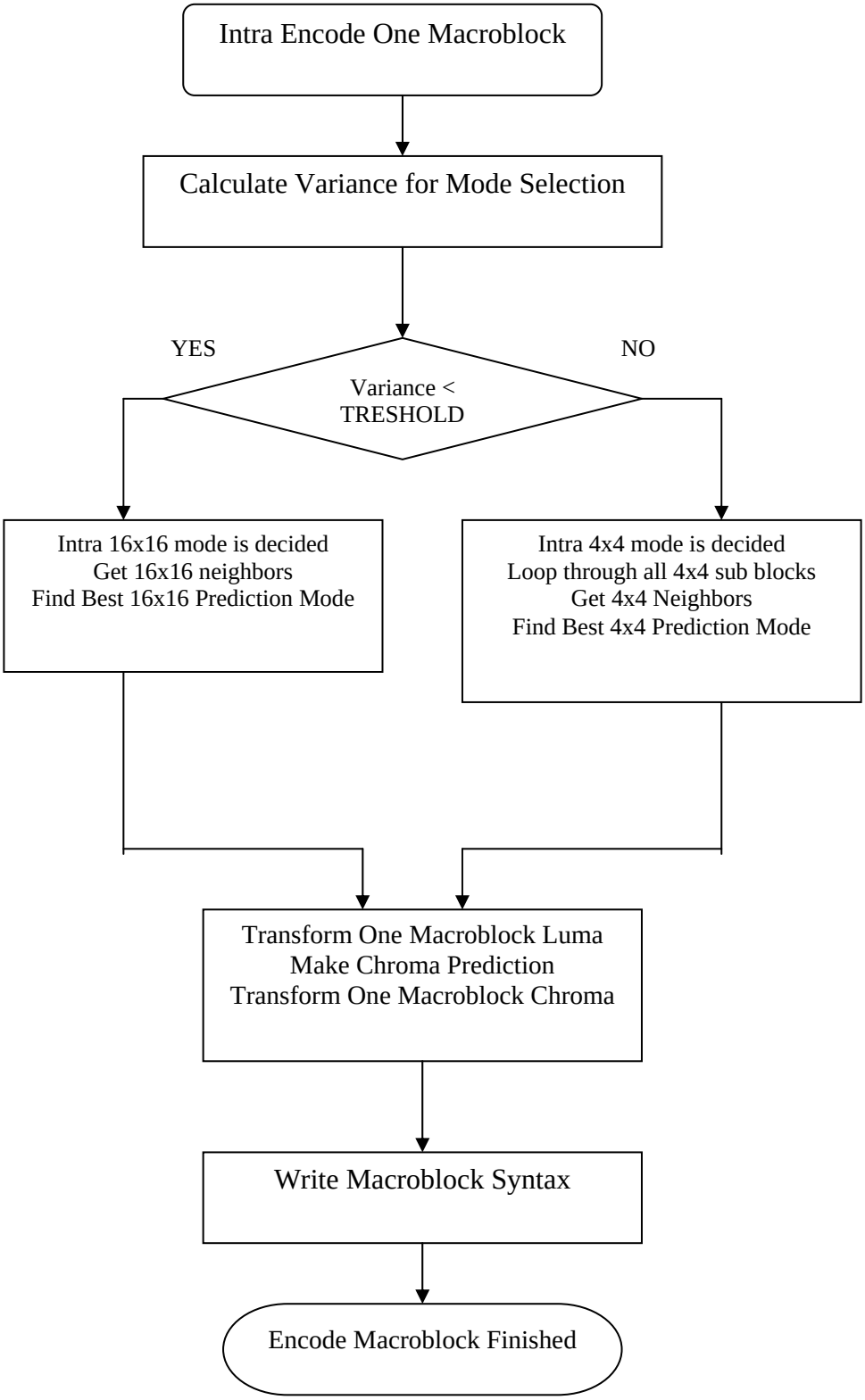

Figure 20 - Intra Encode One Macroblock flow cart

5.4.1.3 Inter Encoder

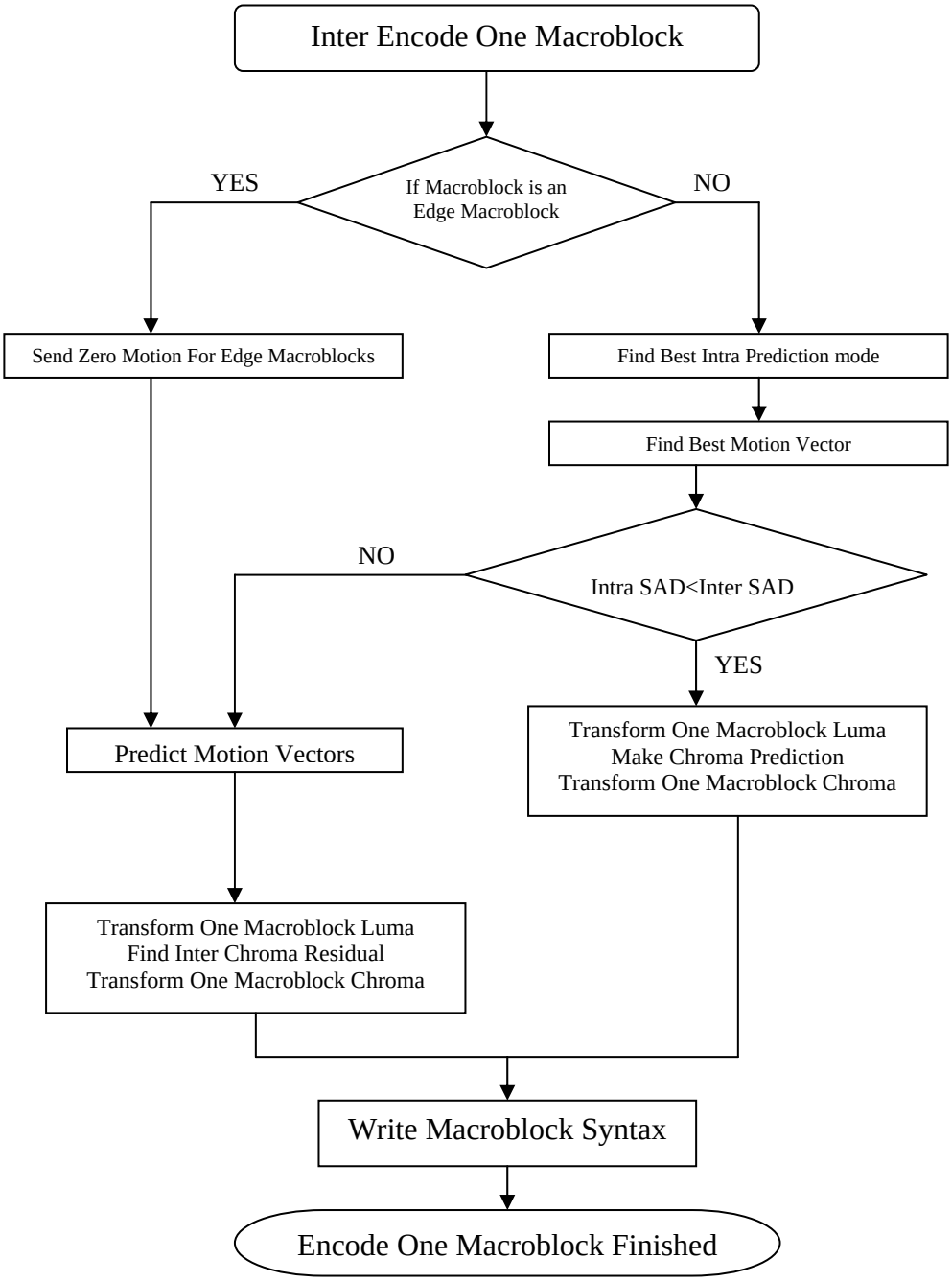

Figure 21 - Inter Encode One Macroblock flow chart

Although number of intra coded macroblocks in an inter coded picture is decided by channel bit error rate for real world system[13], for this implementation, compared SAD values between intra and inter decisions are accepted as criterion

for sending intra macroblock instead of inter encoded macroblock inside an inter frame. Decision scheme is given in Figure 21.

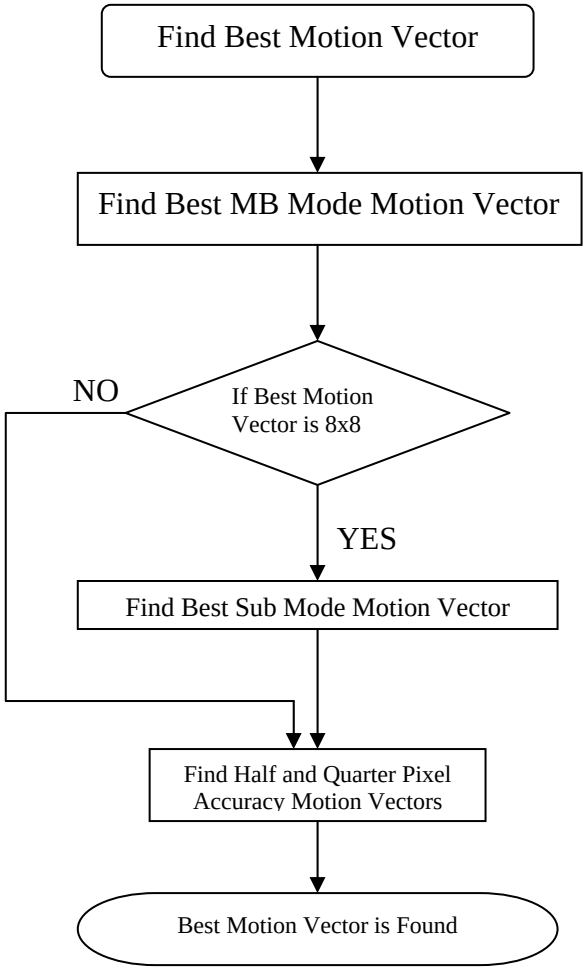

Figure 22 - Find Best Motion Vector flow chart

Best motion vector decision scheme is given in Figure 22

Macroblocks at picture boundaries are directly sent with zero motion assumption. Their reference motion blocks are assumed as 16x16 blocks, reference motion block is taken from the previous picture in time order. For each macroblock not being at picture boundaries, up to five available reference frames are scanned from the frame store to find the best motion vector.

Motion estimation process starts with search window construction from the current reference frame. After search window is constructed, integer pixel

accuracy motion vector search algorithm starts. Integer pixel accuracy motion vector that gives best SAD is found. Then half and quarter pixel motion estimation starts and result is placed in global storage.

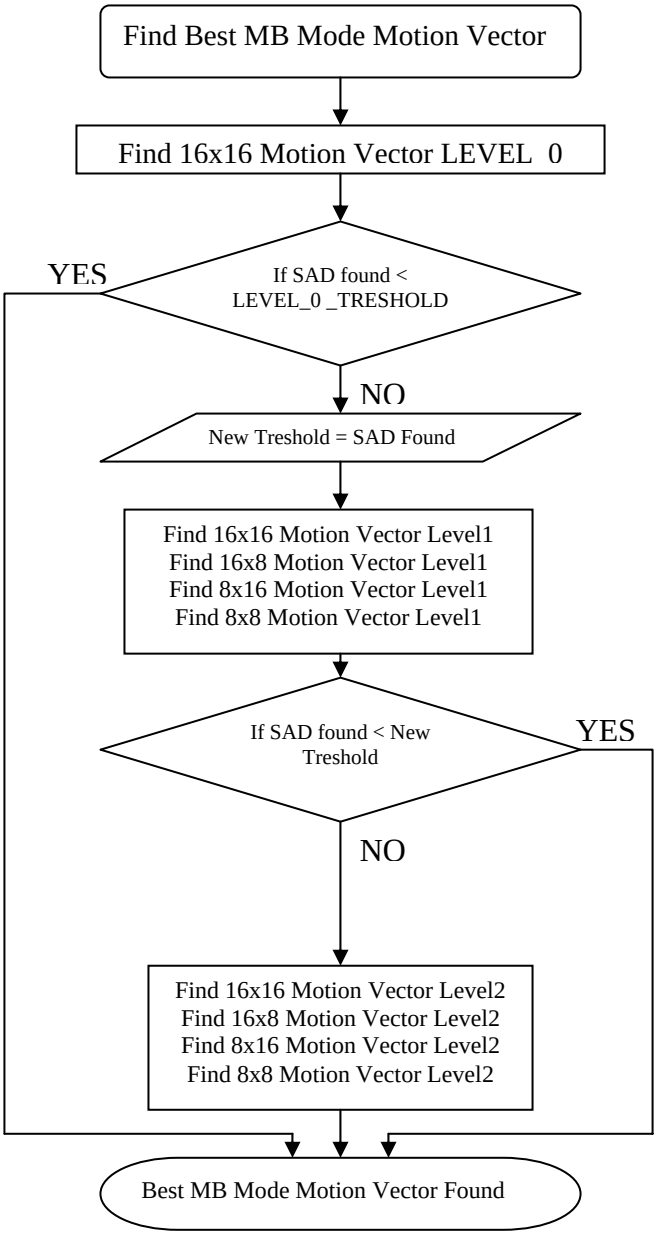

Figure 23 - Find Best MB Mode Motion Vector flow chart

Best motion vector is decided with two steps decision scheme shown in Figure 23. In the first step, for each reference frame in frame store, motion vector modes

of 16x16, 16x8, 8x16 and 8x8 for different search positions specified with DCUPS search algorithm. These positions are gathered into groups by considering distance from the origin. These groups are named under search LEVEL. There are three search levels. Each level is distinguished by its search positions. Best match search positions used for each level is given below with (x,y) locations in a plane.

- LEVEL0, (0,0) location
- LEVEL1, (1,0),(0,1),(-1,0),(0,-1) locations
- LEVEL2, (3,2),(-3,2,),(3,-2),(-3,-2),(6,5),(-6,5),(6,-5),(-6,-5) locations

Search modes that use block sizes of 16x16, 16x8, 8x16 and 8x8 modes are called MB modes in the encoder software. If the best MB mode motion vector has 8x8 block size, sub block modes of 8x8 block, which are 8x4, 4x8, 4x4 block modes are used to find best integer pixel motion vector. After finding best integer pixel motion vector, half and quarter pixel motion vectors are decided by using filtering and interpolation [7].

5.4.1.4 Transform and Quantization Block

This block is one of the common blocks for inter and intra prediction in the encoder. Because of this reason, function of this block is explained only in intra encoder sub-topic. Prediction residuals from the difference of predicted and original values of each macroblock are processed in this functional block. Besides transform and quantization, inverse transform and inverse quantization are also performed in this stage. Transformed and quantized residuals are put into global storage for entropy coding module. Inverse-quantized and inverse transformed coefficients are put into frame store.

5.4.1.5 Motion Compensation Block

Motion compensation block is not a separate functional block from the other functional blocks. Motion compensation for H.264 is done by using motion vectors found and reference frames in frame store to find the predicted frame at the decoder side. After predicted frame is found, prediction residuals can be found.

5.4.1.5.1 Luma Motion Compensation

Considering the motion estimation stage, predicted pixel values and prediction residuals must be calculated to find best motion vector. Meaning that, during motion estimation stage, all motion compensation values are already calculated. These values are used for motion compensation.

5.4.1.5.2 Chroma Motion Compensation

Motion estimation uses luminance pixel values, while detecting the motion, because only luminance component of the frame is a reference for the motion estimation. Chrominance components decide the colors. Chrominance blocks should also move along the luminance motion path. Motion vectors for chrominance components are calculated from luminance motion vectors for H.264 Recommendation.

Unlike luminance component, motion compensation for chrominance components of picture should be performed separately due to the reason explained above.

5.4.2 CABAC

Entropy coding stage is the output stage of the encoder. All bits generated from the encoder must pass through from this stage. After this stage, parsing stage for the encoder starts. Parsing is specified in detail in H.264 recommendation.

Context Adaptive Binary Arithmetic Coding is chosen as entropy coding method. Binary encoding, context initialization and CABAC routines in the JVT encoder are ported. Some simplifications are made for compatibility.

5.4.3 Data Input/Output

This implementation assumes virtual visual source for raw video stream. Due to this reason, some specific file format for Texas Instruments IDE must be prepared [30] from the raw byte stream. After loading that special file to the IDE, visual data can be processed from the memory.

5.5. Optimizations

Real-time systems have to complete some their tasks in some specific timing constraints required for the application.

Desktop platform implementation of real-time systems is very commonly used implementation method for real-time systems. When mobility is required, embedded systems are considered for implementation. Embedded systems introduce a drawback for implementation of real-time systems. They have limited processing resources, such as memory and processing power, compared with desktop systems.

This issue leads to software and hardware architecture differences between desktop and embedded systems. Someone who deals with an embedded system has to be more careful while using available memory and processing power.

In this work software architecture is designed in a way that, memory and processing power consumption does not prevent further development steps for future enhancements of the H.264 encoder.

Both Intra and Inter prediction have some special functional stages for compression. Some of these functional blocks are common for two of the

prediction methods. Excluding these common functional blocks, Intra and Inter encoder stages can be implemented completely independent.

Using advantage of independent implementation, first intra encoder block is implemented. Each functional block of intra encoder is sequentially connected to each other. Functional blocks are implemented with module based architecture. Modularity increased debugging speed for the encoder implementation. Common blocks with inter encoder have also been implemented wile developing intra encoder. Afterwards, inter encoder functional blocks are implemented. Modularity of inter encoder blocks are also taken into consideration while designing the software. In the last phase, entropy coding stage is implemented. By this way, all of the modules are implemented and tested independently.

5.5.1 Optimizations Regarding Software Architecture

Modularity of the software architecture made works easier to optimize the H.264 encoder source code. By this way, each module can be optimized independently without affecting function of the other modules. Functional blocks of the code are implemented in more than one way. Then profiling tool of Code Composer Studio is used for profiling code size and execution time. Fastest implementation of the code is chosen for the final version.

Most of the code is written in C language. This way is chosen due to efficiently optimizing compiler used by Code Composer Studio IDE and to reduce implementation time. Some basic and frequently used codes are written with intrinsic codes. In some parts of the code, assembler optimized signal processing libraries of Texas Instruments are used. Optimization at this level is expected to fulfill performance requirements of the encoder.

Regarding the capability of the TMS320C64x family Digital Signal Processors, code development flow chart is given as shown in Figure 24 [27].

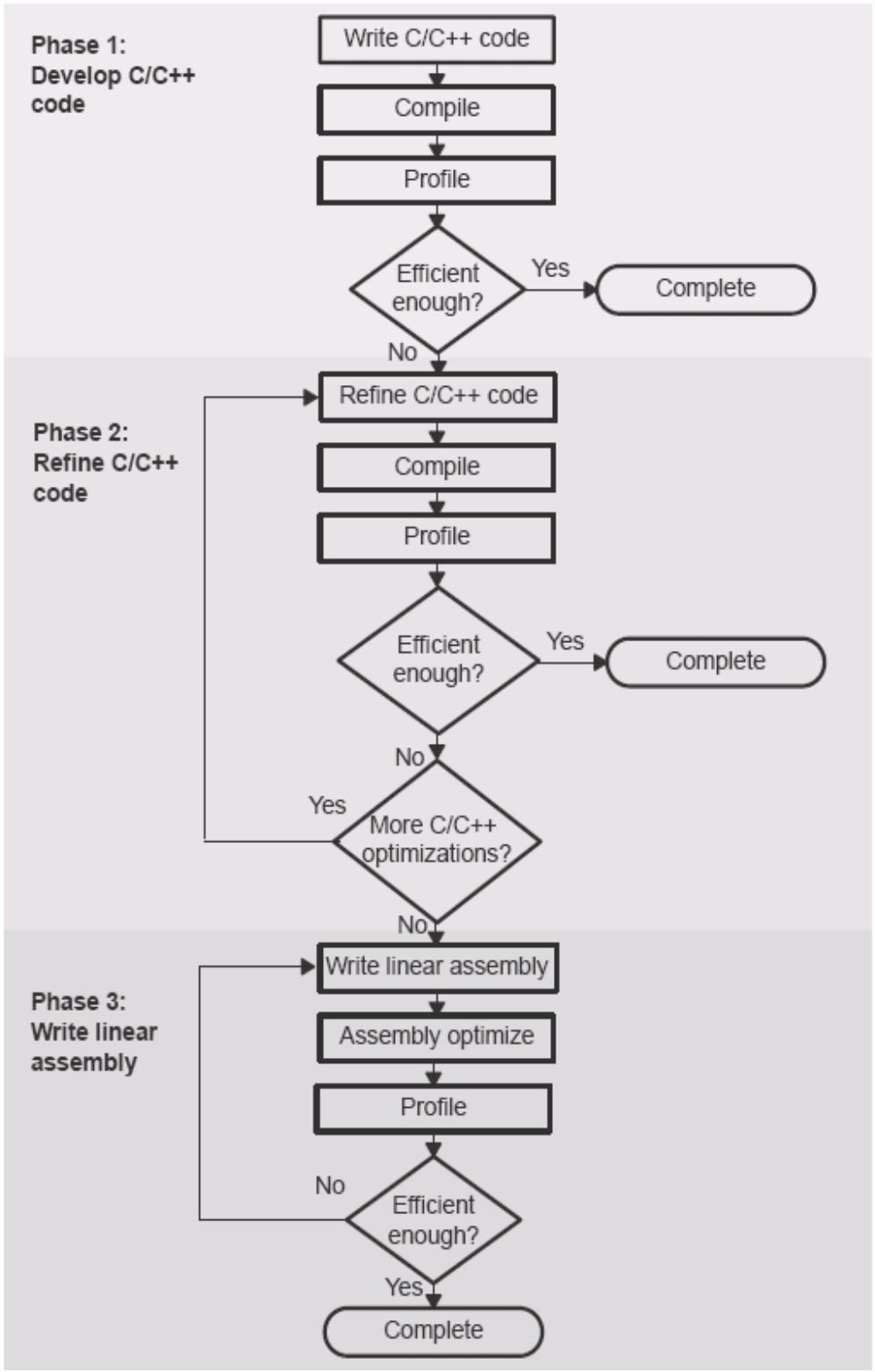

Figure 24 – Code development cycle for TMS320C64x [27]

5.5.2 Optimizations Regarding Compiler

Software implementations using C language require two kinds of ability to enhance C code. The first one is to know programming skills for all programming languages. This allows you to write better code for any programming language, not only for C. For example using global variables for data exchange between functions allows you to get rid of unnecessary stack allocations, when this parameter has to be used by the functions called inside functions in which parameter is passed. At the same time stack overflow probability is reduced. These types of optimizations are compiler and platform independent optimizations. The second is to think like a compiler to reduce load on the compiler in order to generate better output when writing source codes. Texas Instruments C6000 compiler enables software designer to improve code efficiency by putting compiler into more precise state of generating code by providing feedback to the compiler.

Time-consuming parts of software are inside loops. Therefore, loop optimization becomes important issue. To optimize the loop, number of instruction cycles spent in loop must be reduced. Method that can be used is “Loop Unrolling [27]” for the loop optimization. If the inside loop can be unrolled efficiently, compiler performs optimization on unrolled inner part. For this implementation, loop unrolling is used where it is possible.

Performance of C code, which assumes “long” and “int” types as the same bit-length reduces. Because “long” type is 40-bit length for TMS320C6000 compiler and it uses additional data unit for loading and saving values of these variables. This reduces performance, generates extra code for handling of “long” operations.

Using 32-bit “int” variables for loop counters generates more efficient code, when compared with the code using “short” variables for loop counters.

Another important issue for a compiler is to handle memory independencies present. As number of parallel instructions increases, code performance increases at the same time. It is important for a compiler to schedule instructions in parallel not to waste processing power. Sometimes feedback is needed for the compiler to understand independent instructions. Dependency is a case where one instruction must occur before another one. This type of dependent instructions cannot be scheduled in parallel. In order to give information about independencies to compiler, there are some methods, which are also used in this implementation. For example, one of them is using “restrict” keyword for current scope of the variable. This keyword shows that, pointer object that is declared with “restrict” keyword is used for only one object inside the current scope. By this way, compiler can assume that this pointer location is not changing in the current scope and increase parallelism.

There are more issues about compiler optimization. All the details can be found in [27].

5.5.3 Used Optimization Methods

In this chapter used optimization types will be explained. Performance enhancements will be shown.

5.5.3.1 Loop Unrolling

Loop unrolling is actually optimization technique to make job of the compiler easier. This reduces code size and also number of cycles required for completion of the same job.

In the first example there are two forms of the same code. In the first code, two loops is used for completing the job completed in second code with only single loop and unrolled inner loop.

```
for (j=0;j<4;j++)
{
 for(i=0;i<2;i++)
 {
   m[2*i]=y[i][j]+y[i+2][j];
   m[2*i+1]=y[i][j]+y[i+2][j];
 }
}
```

(a)

```
for (j=0;j<4;j++)
{
  m[0]=y[0][j]+y[2][j];
  m[1]=y[0][j]+y[2][j];
  m[2]=y[1][j]+y[3][j];
  m[3]=y[1][j]+y[3][j];
}
```

(b)

Figure 25 – Loop Unrolling optimization
a) Straightforward code b) Loop unrolled code

Statistics for the code in Figure 25 a) and b) are given below

	Code Size	**Number of Cycles**
Original Code	108	240
Loop unrolled code	76	136

Table 1 - Loop unrolling statistics

As it can be seen from the Table 1, code size of the original code is larger than loop unrolled code size. This means, loop unrolling also reduces generated code size for loops. Considering the performance, original code is executed at 240 CPU cycles. Whereas loop unrolled code takes only 136 cycles to perform the same job. This shows that loop unrolling enhances code performance. Loop unrolling method is used in the H.264 encoder implementation where it is possible, in order to take advantages of reduced code space and increased execution performance.

5.5.3.2 Use of Intrinsic Operators

After C optimization step in the code development flow, use of the intrinsic operators comes for further optimization. Code lines below belongs the part of

unutilized code.

```
for(i=0;i<16;i++)
 for(j=0;j<16;j++)
 {
  sum_of_squares+=mb[i][j]*mb[i][j];
  sum+=mb[i][j];
 }
```

Figure 26 – Straightforward code

In the Figure 26, there is a program code, which finds sum of squares and sum of each entry separately. Considering this part of the code, variable “mb[i][j]” is an 8 bit entry and “mb” is 16x16 two dimensional array. Each entry is treated as 8-bit entities for the calculation.

In the optimized code of Figure 27, allocation of the “mb” array in the memory is taken into consideration. Variable named “var” is a pointer for unsigned integer, which is 32 bits for TMS320C64x processor. For each line of “mb” array, beginning address of the corresponding line is assigned to “var”. Packed instruction “_dotpu4” is used for calculation of both sum of squares and sum. This instruction takes two 32-bit arguments, separates each argument into four bytes(which are actual entry values of “mb” array) and takes dot product of each byte from two of the arguments. Finally, instruction returns sum of dot products.

```
for(i=0;i<16;i++)//sweep all lines of the mb array
{
        var=(unsigned int *)&mb[i][0];//assign beginning address of line to var
        sum_of_squares+=_dotpu4(var[0],var[0]);//sum of squares for first 4 bytes
        sum_of_squares+=_dotpu4(var[1],var[1]);// sum of squares for second 4 bytes
        sum_of_squares+=_dotpu4(var[2],var[2]);// sum of squares for third 4 bytes
        sum_of_squares+=_dotpu4(var[3],var[3]);// sum of squares for fourth 4 bytes
        sum+=_dotpu4(0x01010101,var[0]);//sum for first 4 bytes
        sum+=_dotpu4(0x01010101,var[1]); //sum for second 4 bytes
        sum+=_dotpu4(0x01010101,var[2]); //sum for third 4 bytes
        sum+=_dotpu4(0x01010101,var[3]); //sum for fourth 4 bytes
}
```

Figure 27 – Intrinsic optimized code

Comparing the statistics in the Table 2, even though code size increases, execution time reduces drastically. Intrinsic operators enable programmer to use TMS320C64x specific assembler instructions under C programming language. This brings great flexibility while optimizing code, without rewriting assembly.

	Code Size	**Number of Cycles**
Original Code	248	13706
Intrinsic Optimized Code	452	2634

Table 2 –Intrinsic optimization statistics

5.5.3.3 Use of Assembly Optimized TMS320C64x Libraries

Texas Instruments offer libraries for programmers [28] dealing with image or video processing. These libraries contain assembly optimized routines. For this implementation, functions from these libraries are used where it is possible. For example, 16x16 SAD is calculated by using this library function. Before using the library function, lines in Figure 28 were being used for SAD calculation, where "_abs" intrinsic operator was used for absolute value calculation.

```
for(i=0;i<16;i++)
 for(j=0;j<16;j++)
  sad+=_abs(mb[i][j]-mb2[i][j]);
```

Figure 28 - Straightforward SAD code

Statistics for the library function and the straightforward calculation of SAD with C routine in Figure 27 are given in Table 3.

	Code Size	**Number of Cycles**
Original Code	32	2180
Library Function	168	67

Table 3 – Assembly optimization statistics

As it can be seen from the Table 3, execution time decreases drastically with the usage of the library function. Even if its code length is larger than original code, its performance is a lot better than original codes.

5.6 Encoder Conformance

H.264 Recommendation specifies some predefined profile levels. In order to achieve interoperability, there are three profile levels for H.264 decoder. These profiles are named as Basic Profile, Main Profile and Extended Profile. If the decoder supports one of these profile levels, it can decode streams generated with up to same profile level. For example, Main profile can decode streams prepared with Basic or Main Profile levels. Extended Profile Decoder can decode Basic, Main and Extended profile streams.

This implementation of H.264 encoder can generate H.264 Main Profile Level output streams. Basic Profile is a base for this work. However, CABAC entropy coding scheme implemented for entropy coding stage, which is supported starting from the Main Profile decoders.

CHAPTER 6

PERFORMANCE EVALUATION

The proposed real-time H.264 Video Encoder processes colored frames with 144 lines x 160 pixels dimension, which is known as QCIF frame resolution.

While evaluating the performance of the encoder, execution time and PSNR results are taken from CCS IDE. Execution time results are measured with Profiler Tool of CCS IDE. PSNR results are calculated on simulator.

We have used three YUV video sequences that are used for compression evaluation are chosen as test sequences. In addition, four different quantization parameters are used in the test. Results are collected and compared with the results presented in the previous works.

Development and evaluation environment for the proposed system is given below.

Code Generation Environment : TI Code Composer Studio for C6000 Family.

Execution Time Analysis Tool : TI CCStudio Profiler.

Platform : TMS320C64x Cycle Accurate Simulator.

CPU Clock : 800MHz.

Gathered profiling results are represented in the following tables.

Frame Type	QP=20	QP=28	QP=36	QP=44
Intra	49353703	40741236	35155552	32330613
Inter	72965755	74674142	78144253	79170510

Table 4 – Profile results for Foreman sequence (Number of cycles)

Frame Type	QP=20	QP=28	QP=36	QP=44
Intra	47983922	39738933	34119128	31672727
Inter	45801810	48172325	68910747	71121933

Table 5 – Profile results for Container sequence (Number of cycles)

Frame Type	QP=20	QP=28	QP=36	QP=44
Intra	46184506	38715393	33837658	31300781
Inter	69391931	67032258	74662078	75737128

Table 6 – Profile results for Carphone sequence (Number of cycles)

Profile results for the evaluated sequences are given above. Number of cycles required for encoding single intra or inter frame is written in the corresponding entries. Number of cycles gives execution time of the specific encoding task, when the clock frequency of the processor is specified.

Evaluation results are given in the tables from 7 to 10.

Sequence Name	Frame Type	Frame Size	Frames/Sec	Bit-rate (Kbits/sec)	Average PSNR(dB)
Foreman	color	QCIF	10	500.992	42.7
Container	color	QCIF	16	390.256	43.3
Carphone	color	QCIF	11	480.720	42.9

Table 7 – Evaluation results with QP=20

Sequence Name	Frame Type	Frame Size	Frames/Sec	Bit-rate (Kbits/sec)	Average PSNR(dB)
Foreman	color	QCIF	10	292.440	36.5
Container	color	QCIF	16	252	36.7
Carphone	color	QCIF	11	259.768	37.2

Table 8 – Evaluation results with QP=28

Sequence Name	Frame Type	Frame Size	Frames/Sec	Bit-rate (Kbits/sec)	Average PSNR(dB)
Foreman	color	QCIF	10	129.304	30,6
Container	color	QCIF	11	169.336	30.45
Carphone	color	QCIF	10	132.056	31.5

Table 9 – Evaluation results with QP=36

Sequence Name	Frame Type	Frame Size	Frames/Sec	Bit-rate (Kbits/sec)	Average PSNR(dB)
Foreman	color	QCIF	10	100.032	26.5
Container	color	QCIF	10	119.208	25.4
Carphone	color	QCIF	10	102.816	27.1

Table 10 – Evaluation results with QP=44

Quantization parameters used for evaluation purpose are chosen intentionally to compare the results with the ones in [28] and [29].

PSNR performance of the encoded regarding quantization parameter is similar to the results [28] with the same quantization parameters.

For this implementation, by using different search strategies, better PSNR results can be obtained. This may increase execution time and reduce the performance of the encoder. However, there is another tradeoff at this point. Although execution time increases, picture quality of the reconstructed frame on the frame store will increase with increasing efficiency of the search algorithm. This may result in degradation in output bit stream length due to smaller residuals. There is an optimum point for this decision, which is not in the context of this work.

In this work, only I and P frames are implemented. B frames are not present in the stream, which significantly improve encoder performance. Using B frames with better search strategies will lead to results close to given in [29].

It is observed from the tables that, increasing quantization parameter reduces frames/second rate of the encoder. This is due to decreased quality of picture in the frame store for motion estimation. Time needed for motion estimation increases with reduced picture quality, which is dependent on quantization parameter.

Sending zero motion for edge macroblocks increased the system execution time performance. For the proposed system, nearly %30 of the macroblocks are sent with zero motion assumption. Considering larger picture sizes, system performance will decrease under linear performance degradation curve.

Prediction mode decision for inter macroblocks is made with SAD comparison between inter and intra coding results in the proposed system. In the real life, this will not be the case. For the proposed system, considering single inter macroblock, both intra and inter prediction is performed excluding edge macroblocks. This situation results in performance degradation for inter encoded macroblocks that are not at the edge of the picture. Changing this strategy will introduce significant performance increase to the proposed system.

CHAPTER 7

CONCLUSIONS

In this thesis work, we aimed to implement real-time H.264 Encoder on TI TMS32C64x Digital Signal Processor. Performance evaluation results are presented in the previous chapter. Gathered results showed that, this implementation of H.264 Encoder meets real-time requirements with satisfying picture quality.

While implementing the encoder, modularity was an important issue. Flexible encoder architecture is created. This will enable easy further manipulations of the code and encoder structure.

It is observed that, encoder performance depends on several critical global parameters, such as quantization parameter, mode decision thresholds. These parameters are not decided after large number of trials. In addition, these critical parameters are fixed in this implementation All performance evaluation results may change by changing these critical parameters. Obviously optimum value for these parameters will result in better encoder performance. Further improvement can be achieved by changing these parameters according to channel parameters. As an alternative, optimum values for these parameters can be decided after number of trials.

Motion search algorithm significantly affects the encoder performance. DCUPS algorithm has given moderate results in this work. Different search algorithms

such as three step search or exhaustive search may be used as an alternative motion search algorithms.

In this work, every motion block size can used for motion estimation. Finding best motion block size by considering all block sizes takes great time of the encoder. By limiting the motion vector block sizes, better performance results can be obtained.

Another issue we address in this thesis is the real-time implementation of CABAC entropy coding scheme. Due to CABAC ‘s computational complexity, in order to achieve real-time performance, it may be implemented with hardware. It is shown that, for the proposed system, software CABAC implementation works with real-time timing constraints on TMS320C64x DSP.

Proposed H.264 encoder does not implement the full recommendation. Main building blocks are implemented in this work for only QCIF frame size. Conforming stream syntax requirements is another issue that is to be accomplished with full attention. In order to generate output stream with correct syntax, one hundred percent compatible decoder should be present. Even JVT decoder has some bugs and it is being updated. For full compatibility, decoder may be explicitly implemented.

Unutilized portions of the code may be revised for better performance. Giving support for different frame sizes may be implemented in the future work.

Almost hundred percent of the code is written in the C programming language. Considering the code development flowchart in the Figure 24, all development stages may be applied to the code. This will introduce significant improvement to performance of the encoder.

Rate-distortion optimization method can be further added to the implementation and performance of the encoder can be monitored afterwards.

Additional features, such as support for field pictures, generation of B, SP, SI frames, Rate-Distortion Optimization proposed in [13] can be implemented in the future work.

REFERENCES

[1] B. Bhatt, D. Birks and D. Hermreck, “About Digital Video Compression”,IEEE Spectrum Magazine, vol. 36, no.8, pp20-28, October 1997.

[2] ITU-T Recommendation H.261, “Video Codec for Audivisual Services at px64 kbit/s”, 1993.

[3] ITU-T Recommendation H.263, “Video Coding for very Low Bit rate Communication”, 1996.

[4] ISO/IEC International Standard 11172; "Coding of moving pictures and associated audio for digital storage media up to about 1,5 Mbits/s", November 1993.

[5] ISO/IEC International Standard 13818; "Generic coding of moving pictures and associated audio information", November 1994.

[6] UB Video Inc., “Emerging H.26L Standard: Overview and TMS320C64x Digital Media Platform Implementation”, White Paper, 2002.

[7] International Telecommunication Union, “Advanced video coding for generic audiovisual services” 2003.

[8] Texas Instruments, “TMS320C64x Technical Overview, Literature Number: SPRU395B”, January 2001.

[9] ITU-T Recommendation H.263 Version 2, “Video Coding for very Low Bit rate Communication”, September1997.

[10] T. Wiegand, G. Sullivian, G. Bjontegaard, A. Luthra, “Overview of the H.264/ AVC Video Coding Standard”, IEEE Transactions On Circuits And Systems For Video Technology, July 2003.

[11] R. Schafer, T. Wiegand, H. Schwarz, “The Emerging H.264/AVC Standard”, EBU Technical Review, January 2003.

[12] T. Stockhammer, T. Wiegand, “H.264/AVC for Wireless Applications”, 2003.

[13] T. Stockhammer, D. Kontopodis, T. Wiegand, “Rate- Distortion Optimization For JVT/ H.26L Video Coding In Packet Loss Environment”.

[14] Iain E.G. Richardson ,”H.264/ MPEG-4 Part 10: Intra Prediction White Paper, “Reconstruction Filter”, April 2003

[15] LSI Logic Cooperation, “H.264 /MPEG 4 AVC Video Compression Tutorial”.

[16] Iain E.G. Richardson , H.264/ MPEG-4 Part 10: Variable Length Coding White Paper, “Variable-Length Coding”.April 2003

[17] D. Marpe, H. Schwarz, T. Wiegand, “Context Based Adaptive Binary Arithmetic Coding in the H.264/ AVC Video Compression Standard” IEEE Transactions On Circuits And Systems For Video Technology, Vol.13, No.7, July 2003

[18] T. Wiegand. ITU- Telecommunications Standardization Sector Study Group 16, “H.26L Test Model Long Term Number 8 (TML-8) draft0”, May 2001.

[19] Iain E.G. Richardson , H.264/ MPEG-4 Part 10: Inter Prediction White Paper, “Prediction of Inter Macroblocks in P-Slices”, April 2003.

[20] Antti Hallapuro, Marta Karczewicz,” Low Complexity Transform and Quantization – Part I: Basic Implementation”, jvtb038.doc, January 2002

[21] Iain E.G. Richardson , H.264/ MPEG-4 Part 10: Transform and Quantization White Paper, “Transform and Quantization”., April 2003

[22] D. Marpe, G. Blattermann, G. Heising, T. Wiegand, “Video Compression Using Context- Based Adaptive Arithmetic Coding ” ICIP, Thessaloniki, Greece, 2001.

[23] A. Moffat, R. M. Neal, and I. H.Witten, “Arithmetic coding revisited,” in*Proc. IEEE Data Compression Conf.*, Snowbird, UT, 1996, pp. 202–211.

[24] P. G. Howard, J. S. Vitter, “Practical Implementations of Arithmetic Coding”, Brown University Depertment of Computer Science, Techinical Report No.92-18, Revised Version April 1992.

[25] A. Said, “Comparative Analysisof Arithmetic CodingComputational Complexity”, HP Laboratories Palo Alto HPL-2004-25, April 2004.

[26] Raymond Westwater, Borko Furht, Joshua Greenburg, “Motion Estimation Algorithms for Video Compression”, Kluwer Academic Publishers, 1997

[27] Texas Instruments, “TMS320C6000Optimizing Compiler User’s Guide , Literature Number: SPRU187K”, October 2002.

[28] Till Halbach, Mathias Wien, “ Concepts and Performance of Next Generation Video Compression Standardization”

[29] “MPEG-4 Video and Image Coding Tools”, MPEG-4 Industry Forum Presentation.

[30] Texas Instruments, “TMS320C62x Image/Video Processing Library Programmer’s Reference, Literature Number: SPRU400”, March 2000.

[31] Texas Instruments, “TMS320C64x DSP Library Programmer’s Reference , Literature Number: SPRU565B”, October 2003.

[32] H. Sabikhi, “Example of GEL Usage With File I/O for Code Composer Studio v2.1”, Code Composer Studio, Applications Engineering Application Report, SPRA381, April 2002.

APPENDICES

APPENDIX A

TMS320C64x Functional Unit Operations

Functional Unit	Fixed-Point Operations
.S unit (.S1, .S2)	32–bit arithmetic operations
	32/40–bit shifts and 32–bit bit–field operations
	32–bit logical operations
	Branches
	Constant generation
	Register transfers to/from control register file (.S2 only)
	Byte shifts
	Data packing/unpacking
	Dual 16–bit compare operations
	Quad 8–bit compare operations
	Dual 16–bit shift operations
	Dual 16–bit saturated arithmetic operations
	Quad 8–bit saturated arithmetic operations
.D unit (.D1, .D2)	32–bit add, subtract, linear and circular address calculation
	Loads and stores with 5–bit constant offset
	Loads and stores with 15–bit constant offset (.D2 only)
	Load and store double words with 5–bit constant offset
	Load and store non–aligned words and double words
	5–bit constant offset generation
	32–bit logical operations
	Dual 16-bit arithmetic operations

Figure 29 – TMS320C64x functional unit operations [8]

Functional Unit	Fixed-Point Operations
.M unit (.M1, .M2)	16 x 16 multiply operations
	16 x 32 multiply operations
	Quad 8 x 8 multiply operations
	Dual 16 x 16 multiply operations
	Dual 16 x 16 multiply with add/subtract operations
	Quad 8 x 8 multiply with add operations
	Bit expansion
	Bit interleaving/de–interleaving
	Galois Field Multiply
	Rotation
	Variable shift operations
.L unit (.L1, .L2)	32/40–bit arithmetic and compare operations
	32–bit logical operations
	Leftmost 1 or 0 counting for 32 bits
	Normalization count for 32 and 40 bits
	Byte shifts
	Data packing/unpacking
	5–bit constant generation
	Dual 16–bit arithmetic operations
	Quad 8–bit arithmetic operations
	Dual 16–bit min/max operations

Figure 30 – TMS320C64x functional unit operations (ctd) [8]

APPENDIX B

Code Composer Studio Project Description

B.1 Project View

Project folder appearance is given in Figure 31.

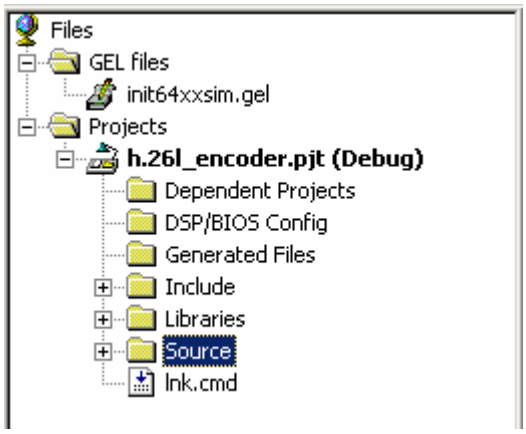

Figure 31 - CCStudio project window

GEL Folder consists Code Composer Studio Graphical Extension Language functions. GEL file is used to initialize Code Composer Studio IDE, and makes it compatible with the target board configuration. For this purpose, default gel file “init64xxsim.gel” that comes with the CCS IDE is used. Gel file mainly initializes some internal registers of the processor.

Projects folder contains some subfolders, which may be used for project generation. Only “Include”, “Libraries” and “Source” folders contain items in themselves. File named ”lnk.cmd” contains linker commands that are given as an order to linker executable file which resolves linking issues and allocates program and data into specified memory regions. Linker Command file is given in Figure 32

Linker command file starts with two assignments for system heap length and system stack length used by the functions. Heap is used for dynamically allocated objects. Regarding this implementation, dynamically allocated objects are not used. For this reason heap size is chosen as 256 bytes, which is represented by “0x100” hexadecimal notation.

```
-heap  0x100
-stack 0x4000
MEMORY
{
PAGE 0:
L2_PROG_RAM(RW):        origin =0x0          , length=0x60000
L2_DATA_RAM(RW):        origin =0x60000      , length=0xB0000
FRAME_STORE:            origin =0x80000000   , length=0x60000
TABLES  :               origin =0x800D0000   , length=0x40000
OUTSTREAM:              origin =0x80110000   , length=0x10000/* output stream*/
}

SECTIONS
{
.text       >     L2_PROG_RAM
.stack      >     L2_DATA_RAM
.bss        >     L2_DATA_RAM
.cinit      >     L2_DATA_RAM
.cio        >     L2_DATA_RAM
.const      >     L2_DATA_RAM
.data       >     L2_DATA_RAM
.switch     >     L2_DATA_RAM
.sysmem     >     L2_DATA_RAM
.far        >     L2_DATA_RAM
.framestore :           {framestore.obj(.bss)}          >FRAME_STORE
.tables:    {tables.obj(.bss)}                          >TABLES
.outstream:{outstream.obj(.bss)}                        >OUTSTREAM
}
```

Figure 32 -. CCStudio Linker Command File

Someone, who uses heap, must allocate more heap area than specified here. System stack length is specified as “0x4000” bytes. Depending on the implementation, stack size may be specified larger than here. Usage of global variables reduced the need for the stack. This was another issue for this implementation. Using less stack area allows encoder use more data memory for H.264 functional objects for future enhancements.

Linker command file is mainly composed of two sections, “MEMORY” and “SECTIONS” part.

Section called “MEMORY” defines the hardware configuration of the platform for linker.

“MEMORY” section has subsections, which are different regions of platform memory used for linking. These memory regions are defined by their starting point “origin” and their “length” in byte. On the TMS320C64x platform, these regions cannot be overlapped. Regions and their explanations are given below

- “L2_PROG_RAM”, this memory is used as program memory for storing generated program code. This is an internal memory of TMS320C64x processor.
- “L2_DATA_RAM”, this memory is used as data memory for storing used data structures, variables etc. This is an internal memory of TMS320C64x processor.
- “FRAME_STORE”, this memory is used for frame storage of H.264 Encoder. This memory is an external memory.
- “TABLES”, this memory is used for storing initialization tables. This memory is an external memory.
- “OUTSTREAM”, this memory is used for storing output stream generated by the encoder.

While arranging memory regions, access speed requirements of memory regions come into work. Frequently accessed structures, variables etc. are put into fast internal memory. Less accessed memory regions, tables used for initialization are put in slower external memory region.

“SECTIONS” section of linker command file explains linker to put which program segment to which physical memory region defined by “MEMORY” section.

- “.text” section is the program code generated by the compiler and must be put into the physical memory region allocated for program which is L2_PROG_RAM.

- ”stack”, ”.bss”, “.cinit”, “cio”, “.const”, “.data”, “.switch”, “.sysmem”, “.far” sections are data sections generated by compiler and must be put in data memory, which is L2_DATA_RAM. Detailed description of sections generated by compiler can be found in [27].

- “.framestore” section is created for putting video data that belongs to frame store into physical memory section called FRAME_STORE.

- “.tables” section is created for putting initialization tables and frame residuals into physical memory region called TABLES.

- “.outstream” section is created for put output stream into physical memory region called OUTSTREAM.

“Include” folder of project folder contains C header files used by the source files.

“Libraries” folder contains used run time support libraries as shown in Figure 33.

- “dsp64x.lib” contains optimized digital signal processing routines that are used in the encoder. Detailed functional description of this library can be found in [31].

- “img64x.lib” contains optimized image processing functions that are used in the encoder. Detailed functional description of this library can be found in [30].

- “rts6400.lib” contains run-time support functions that are used for standard ANSI-C syntax.

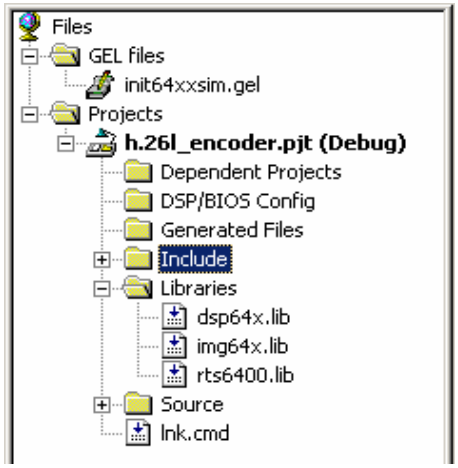

Figure 33 - CCS Libraries view

“Source” folder contains source files created for H.264 encoder as seen in Figure 34.

Description of contents for these source files are given below,

- “basic.c”, this file contains basic encoder routines.
- “binariencode.c”, this file contains functions that manipulate arithmetic encoding environment.
- “buffer_pointers.c”, this file contains declaration pointers needed to manipulate buffers and assignments of them to related structures.
- “cabac.c”, this file contains Context Adaptive Binary Arithmetic Coding routines used for entropy coding section.
- “context_init.c”, this file contains initialization functions for context models used for entropy coding.
- “encode.c”, this file contains basic initialization functions for H.264 streaming headers.

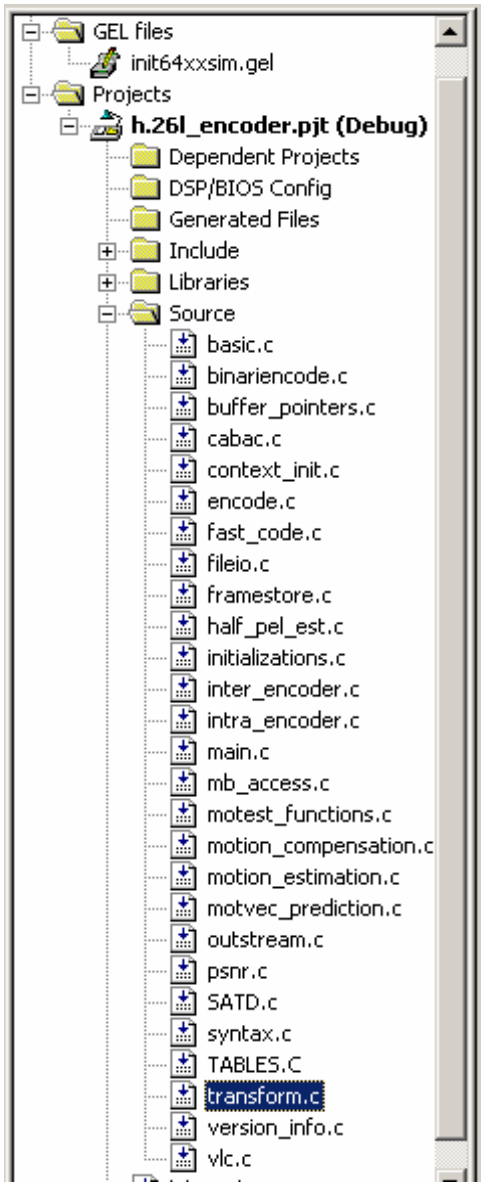

Figure 34 - Source folder view

- "fast_code.c", this file contains six-tap filtering function used for motion estimation. Further optimized routines may be put under this file.

- "fileio.c", this file contains file-IO routines used for taking video stream data from hard disk of computer to internal memory of the TMS320C64x platform.

- "framestore.c", this file contains frame store manipulating functions.

- “half_el_est.c”, this file contains half and quarter pixel motion estimation functions used for inter prediction.
- “initializations.c”, this file contains initialization functions for H.264 encoder data structures.
- “inter_encoder.c”, this file contains necessary function calls for inter encoding and syntax preparation.
- “intra_encoder.c”, this file contains necessary functions for intra coding of a frame.
- “main.c”, this file contains entry point of the encoder, necessary function calls for initialization of streaming and H.264 data types and stream manipulating functions.
- “mb_access.c”, this file contains macroblock access functions used in various parts of the code.
- “motest_functions.c”, this file contains functions used for motion estimation.
- “motion_compensation.c”, this file contains motion compensation routines.
- “motion_estimation.c”, this file contains manipulation routines for motion estimation functions.
- “motvec_prediction.c”, this file contains routines for motion vector prediction.
- “outstream.c”, this file only contains the declaration for output stream buffer.
- “psnr.c”, this file contains Peak Signal over Noise Ratio function.

- “satd.c”, this file contains Sum of Absolute Difference routines with Haddamard transform.
- “syntax.c”, this file contains syntax preparation functions.
- “tables.c”, this file contains initialization tables for CABAC and residual buffer declarations.
- “transform.c”, this file contains transform coding routines.
- “version_info.c”, this file contains version and update information for the encoder. Nothing executable is put here.
- “vlc.c”, this file contains VLC coding functions that are needed for some syntax elements, even if CABAC is used as an entropy coding scheme.

B.2 Encoder Global Data Fields

In this chapter, description of data fields that are used globally is given. These fields are explained by considering the source file they are declared.

B.2.1 “cabac.c” Global Data Fields

- “motinfo_ctx”, this field keeps the current state of the motion info contexts for CABAC.
- “textinfo_ctx”, this field keeps the current state of the texture info contexts for CABAC.

B.2.2 “encode.c” Global Data Fields

- “pred_frame”, this field keeps predicted macroblock data.

B.2.3 “framestore.c” Global Data Fields

- “frame_store_buffer”, this field is used for keeping reconstructed past five frame data used for inter prediction.

B.2.4 “inter_encoder.c” Global Data Fields

- “s_win”, this field is used for keeping luminance data search window for motion vector search

- “win_chroma_u”, this field is used for chroma motion compensation for U data.

- “win_chroma_v”, this field is used for chroma motion compensation for V data.

B.2.5 “intra_encoder.c” Global Data Fields

- “img_UV”, this field keeps chroma pixel values for U and V data for encoding of intra macroblocks.

- “chipred_mode”, this field keeps chroma intra prediction modes for each macroblock.

- “img_cbp”, this field keeps the coded block pattern data for current encoded image.

- “Intra4x4_pred_mode”, this field keeps intra 4x4 prediction mode for intra 4x4 predicted macroblocks of the current frame.

B.2.6 “main.c” Global Data Fields

- “current_mb”, this field keeps encoding parameters, residuals, predicted values for only current macroblock.

B.2.7 “motest_functions.c” Global Data Fields

- “motvec_comp_buf”, this field keeps necessary motion vector data for motion vector comparison. For current macroblock
- “sad_comp_buf”, this field keeps evaluated SAD data for current macroblock.
- “reframe_array”, this field keeps reference frame indices for inter encoded macroblocks of the frame.
- “mvd_array”, this field keeps motion vector data for inter encoded macroblocks of the current frame
- “mvd_res_array”, this field keeps motion vector residuals remaining after motion vector prediction.

B.2.8 “motion_estimation.c” Global Data Fields

- “motion_vector_buffer”, this field keeps found motion vectors after motion estimation stage.

B.2.9 “outpstream.c ”Global Data Fields

- “output_stream”, this field is used for keeping generated final output stream for a single frame.

B.2.10 “syntax.c” Global Data Fields

- “curr_strm”, This field keeps the current bit pointer of the generated stream used for writing output after CABAC stage of the encoder.

B.2.11 “tables.c” Global Data Fields

- “frame_residual_u”, this field keeps residual chrominance u data of the

current frame after prediction

- “frame_residual_v”, this field keeps residual chrominance v data of the current frame after prediction

All of the remaining global arrays declared are used for initialization of context models used for CABAC Encoding.

B.2.12 “transform.c” Global Data Fields

- “quantMat”, this field is used for storing table needed for quantization.
- “dequantMat”, this field is used for storing table needed for de-quantization.
- “QPC”, this field is used for keeping values needed for chrominance quantization parameter determination.