IMPLEMENTATION OF H.264 DECODER ON COMBINE PLATFORM OF ARM AND DSP

BY

SHOBHEN GOHEL

07MCE005

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

MAY 2009

Implementation of H.264 Decoder On Combine Platform of ARM And DSP

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Computer Science and Engineering

By

Shobhen Gohel

07MCE005

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

May 2009

Certificate

This is to certify that the Major Project entitled "Implementation of H.264 Decoder on Combine Platform of ARM And DSP" submitted by Shobhen Gohel (07MCE005), towards the partial fulfillment of the requirements for the degree of Master of Technology in Computer Science and Engineering of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Dr. S.N. Pradhan
Guide and Professor,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Prof. D. J. Patel
Professor and Head,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Dr K Kotecha
Director,
Institute of Technology,
Nirma University, Ahmedabad

Abstract

H.264 is an emerging video coding standard, which aims at compressing high-quality video contents at low bit-rates. While the new coding and decoding processes are similar to many previous standards, the new standard includes a number of new features and thus requires much more computation than most existing standards do. The complexity of H.264 standard poses a large amount of challenges to implementing the coding scheme in real-time via software on personal computer.

H.264 Decoder require high computation because of which it take very high execution time. Decoder process is introduced as a sequential process which works on a single processor but to take advantage of heterogenous environment or use of multi processor the requirement of parallelizing the decoder process can be helpful using .

The main goal for Implementing H.264 decoder on combine platform is to take advantage of both processors execution power. OMAP provides combine platfoem of ARM and DSP processor with Hardware accelerator which can use execution power of two processors and can makes the performance fast.

Acknowledgements

I would like to thanks to Dr. S.N. Pradhan , Professor, Department of Computer Engineering, Institute of Technology, Nirma University, Ahmedabad for his valuable guidance and continual encouragement throughout the Major project. I heartily thankful to him for his time to time suggestion and the clarity of the concepts of the topic that helped me a lot during this study.

I like to give my special thanks to Prof. D.J.Patel, Head, Department of Computer Engineering, Institute of Technology, Nirma University, Ahmedabad for his continual kind words of encouragement and motivation throughout the Major Project. I am also thankful to Dr. K Kotecha, Director, Institute of Technology for his kind support in all respect during my study.

I am thankful to all faculty members of Department of Computer Engineering, Nirma University, Ahmedabad for their special attention and suggestions towards the project work. The blessings of God and my family members makes the way for completion of major project. I am very much grateful to them. The friends, who always bear and motivate me throughout this course, I am thankful to them.

- Gohel Shobhen

07MCE005

Contents

Certificate iii

Abstract iv

Acknowledgements v

List of Tables ix

List of Figures x

Abbreviations xi

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Next Generation Media Application . . . 3
- 1.3 Scope Of Work . . . 3
- 1.4 Thesis Organization . . . 4

2 Literature Survey 5
- 2.1 General . . . 5
- 2.2 Network Abstraction Layer . . . 6
  - 2.2.1 NAL Units . . . 7
  - 2.2.2 NAL unit type . . . 7
  - 2.2.3 parameter set . . . 8
- 2.3 VIDEO CODING LAYER . . . 8
  - 2.3.1 Pictures, frames, and fields . . . 9
  - 2.3.2 YCbCr color space and 4:2:0 sampling . . . 9
  - 2.3.3 Division of the picture into macroblocks . . . 10
  - 2.3.4 Slices and slice groups . . . 10
- 2.4 Block diagram of H.264 decoder . . . 12
  - 2.4.1 Entropy Decoding . . . 13
  - 2.4.2 Inverse transformation . . . 15
  - 2.4.3 Deblocking filter . . . 15
  - 2.4.4 Intra prediction . . . 16

2.4.5 Motion Compensation (Inter-frame Prediction) 16

3 Decoder Parallelization process **18**
3.1 Problem Definition 18
3.2 Existing Methodologies 18
3.2.1 Task-Level Decomposition 18
3.2.2 Data-Level Decomposition 19
3.3 Parallel Decoding Process Implementation 22
3.3.1 Decoding Process 22
3.3.2 H.264 Decoder Load Analysis and partitioning 23
3.3.3 Motion Compensation 24

4 Motion Compensation **26**
4.1 Inter-Frame Prediction in P Slices: 27
4.2 Inter-Frame Prediction in B Slices 30
4.2.1 Reference Picture Lists 31
4.2.2 Motion Vector Calculation 32
4.2.3 Sub-Sample Interpolation 33

5 Implementation and Simulation **34**
5.1 Implementation Environment 34
5.1.1 Scratchbox 34
5.1.2 SimIt-ARM 35
5.2 OMAP 36
5.3 Implementation 38
5.3.1 Derivation process for motion vector components and reference indices 39
5.3.2 Derivation process for luma motion vector 40
5.3.3 Derivation process for chroma motion vectors 40
5.3.4 Decoding process for Inter prediction samples 41
5.3.5 Fractional sample interpolation process 42
5.3.6 Weighted sample prediction process 43

6 Experimental Result and Analysis **45**
6.1 Result 45
6.1.1 PSNR ratio 46

7 Conclusion and Future Scope **48**
7.1 Conclusion 48
7.2 Future Scope 49

A Open Multimedia Application Platform **50**
A.0.1 How the Architecture Works 52

B List of websites **54**

References **55**

List of Tables

I Example for Exp-Golomb codes 14
II Example for unsigned-to-signed-mapping 14

I Output of SimitARM 46
II Comparison of PSNR ratio 47

List of Figures

2.1 Structure of H.264/AVC video encoder 6
2.2 NAL Unit 7
2.3 Data Structure of H.264 9
2.4 Frame Field Types 10
2.5 Slice Group 11
2.6 Block diagram of H.264 decoder 13

3.1 slices 19
3.2 Macroblock Parllelism 21
3.3 Computation profile result for H.264 Decoder 24

4.1 Sample Position 28
4.2 Inter Compensation 31

5.1 OMAP 35x Processor 37

A.1 OMAP Architecture 51

Abbreviations

CABAC	Context-based Adaptive Binary Arithmetic Coding
CAVLC	Context-based Adaptive Variable Length Coding
FMO	Flexible Macroblock Ordering
GOP	Group-of-Pictures
HD	High Definition
IDR	Instantaneous Decoding Refresh
MC	Motion Compensation
ME	Motion Estimation
MV	Motion Vector
NAL	Network Abstraction Layer
OMAP	Open Multimedia Application Platform
PSNR	Peak Signal-to-Noise Ratio
SD	Standard Definition
SPS	Sqeuence Parameter Set
VCL	Video Coding Layer

Chapter 1

Introduction

H.264/AVC is the newest international video coding standard. it is the latest block-oriented motion-compensation-based codec standard developed by the ITU-T Video Coding Experts Group (VCEG) together with the ISO/IEC Moving Picture Experts Group (MPEG), and it was the product of a partnership effort known as the Joint Video Team (JVT).

The MPEG-2 video coding standard, which was developed about 10 years ago primarily as an extension of prior MPEG-1 video capability with support of interlaced video coding, was an enabling technology for digital television systems worldwide. It is widely used for the transmission of standard definition (SD) and High Definition (HD) TV signals over satellite, cable, and terrestrial emission and the storage of high-quality SD video signals onto DVDs. However, an increasing number of services and growing popularity of high definition TV are creating greater needs for higher coding efficiency. Therefore H.264 Coding Standard was introduced.

Advanced Coding technology H.264 requires high computation and complex data dependencies which require high execution time for encode and decode the video. The H.264 Decoder Process can be Parallelized such that the data dependences between parallel process can be minimized and reduce the execution time.

1.1 Motivation

One key attribute of a video compression application is the bit rate of the compressed video stream. Codecs that target specific applications are designed to stay within the bit rate constraints of these applications, while offering acceptable video quality.

H.264 is an emerging video coding standard, which aims at compressing high-quality video contents at low-bit rates. While the new encoding and decoding processes are similar to many previous standards, the new standard includes a number of new features and thus requires much more computation than most existing standards do. The complexity of H.264 standard poses a large amount of challenges to implementing the encoder/decoder in real-time.

To distribute the computation the Decoding process is to be parallelized such that the dependences between process can minimize and the execution time can be optimized. To execute the parallel decoding process a combination platform of ARM and General Purpose DSP can be used in tendence to improve the performance of decoding.

Use of OMAP architecture can make it possible to combine platform of ARM and DSP. OMAP architecture also provide communication between processor so it is possible to make execution more efficient.

1.2 Next Generation Media Application

a. Broadcast over cable, satellite, Cable Modem, DSL, wireless, etc.

b. Interactive or serial storage on optical and magnetic devices, DVD, etc.

c. Conversational services over ISDN, Ethernet, LAN, DSL, wireless and mobile networks, modems, etc.

d. Video-on-demand or multimedia streaming services over ISDN,Cable Modem, LAN, wireless, etc.

e. Multimedia Messaging Services (MMS) over Ethernet,LAN, wireless etc.

1.3 Scope Of Work

Main Functionality of the H.264 decoder is to decode the bit stream generated by H.264 encoder in to video output sequence. Decoding process is distribute in processes such as Entropy Decoding, Inverse Quantization , Inverse Transform, Intra Prediction, Motion Compensation and De-blocking filter.

From Decoding process the Motion Compensation requires half of the computation power of total computation. So to make the decoder more efficient the Motion Compensation process of decoder will be parallelize using the concept of Macro block level Parallelism and dynamic scheduling are useful to distribute the high computation requirement.

This is achieved through the use of a DSP core and an ARM Cortex A-8 CPU. Both processors utilize an instruction cache to reduce the average access time to instruction memory .For that OMAP Architecture will be used which will provide heterogeneous environment.

1.4 Thesis Organization

The rest of the thesis is organized as follows.

Chapter 2, *Literature Survey* , Literature Survey describes the H.264 decoding process and also showing internal details about the flow of the process and the requirements of parallelize the process. It also covers the ARM and DSP processor background require for the project . And In last Simulation tool overview will be shown which shows how to use the simulation tool.

Chapter 3, *Decoder Parallelization Process* , This chapter introduce the different Parallelization technique ,dependencies in the parallelization , Problem regarding Parallelization and solution to overcome the problem . And it introduce proposed the Decoder process parallelization in such way that dependencies between process can minimize and the execution time can be optimize.

Chapter 4, *Motion Compensation*, This Chapter introduce the Motion compensation of the Decoder.It shows internal detail of the motion compensation and shows how the processes are implemented on the conventional decoder.

Chapter 5, *Implementation and Simulation*, This Chapter introduce the implementation of the Parallelize Decoder designed in the previous chapter . And then the process shows how it is simulate in the simulation tool to finally analyze the behavior of the Parallelize Decoder process's Output.

Chapter 6, *Experimental Result and Analysis*,

This chapter shows the experimental results obtained using the simulation and then finally analysis of the Result regarding improvement in the execution time.

Chapter 7, *Conclusion and future scope*, This chapter shows the concluding remarks regarding the optimization of decoing process of H.264 Decoder and scope of future work.

Chapter 2

Literature Survey

2.1 General

H.264 is a standard for video compression, and is equivalent to MPEG-4 Part 10, or MPEG-4 AVC (for Advanced Video Coding).It is the latest codec standard developed by the ITU-T Video Coding Experts Group (VCEG) together with the ISO/IEC Moving Picture Experts Group (MPEG), and it was the product of a partnership effort known as the Joint Video Team (JVT). The ITU-T H.264 standard and the ISO/IEC MPEG-4 Part 10 standard.

For Flexibility and customization, the H.264/AVC design covers a Video Coding Layer (VCL), which is designed to efficiently represent the video content, and a Network Abstraction Layer (NAL), which formats the VCL representation of the video and provides header information in a manner appropriate for conveyance by a variety of transport layers or storage media.

From VCL Video is formate in to Macroblocks which is given to the DataPartition which further devide these Macroblocks in to Slices depends on the configuration of the codec.Then these slices are given to the NAL unit which format the Slices in

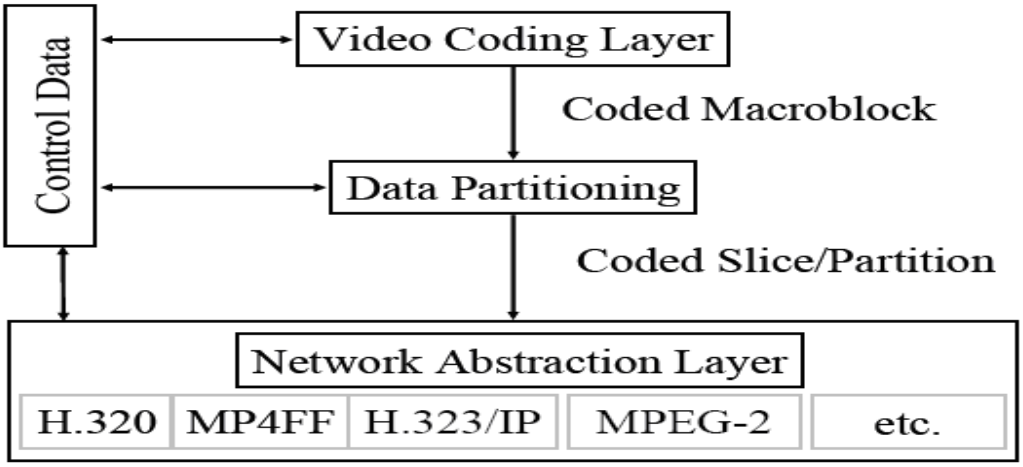

Figure 2.1: Structure of H.264/AVC video encoder

to packets with header information which is useful to forward or transform to the transmission medium.

Following two section contain the detail information of Network Abstraction Layer and Video Coding layer

2.2 Network Abstraction Layer

Network Abstraction Layer (NAL) is designed to provide network friendliness for simple and effective customization of the use of the VCL for a broad variety of systems.

The NAL facilitates the ability to map H.264 VCL data to transport layers such as

- RTP/IP for any kind of real-time wire-line and wireless Internet services
- File formats, e.g. ISO MP4 for storage and MMS
- H.32X for wireline and wireless conversational services

- MPEG-2 systems for broadcasting services, etc.

2.2.1 NAL Units

The coded video data is organized into NAL units, each of which is effectively a packet that contains an integer number of bytes. The first byte of each NAL unit is a header byte that contains an indication of the type of data in the NAL unit, and the remaining bytes contain payload data of the type indicated by the header. The

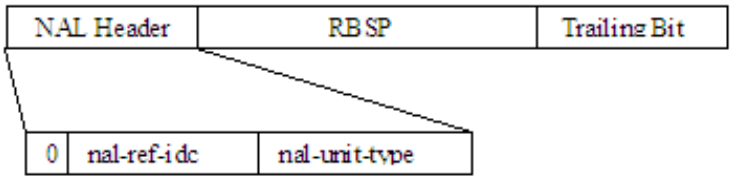

Figure 2.2: NAL Unit

payload data in the NAL unit is interleaved as necessary with emulation prevention bytes, which are bytes inserted with a specific value to prevent a particular pattern of data called a start code prefix from being accidentally generated inside the payload.

2.2.2 NAL unit type

Currently 1-12 NAL unit type nal_unit_type are defined. From these 12 types Type 1-5 and 12 are coded video data called VCL NAL units. The rest of the nal_unit_type are called non-VCL NAL units and contain information such as parameter sets and supplemental enhancement information such as IDR pictures, SPS and PPS .

An instantaneous decoding refresh(IDR) picture is a picture placed at the beginning of a coded video sequence. When the decoder receives an IDR picture, all information is refreshed , which indicates a new coded video sequence.

A Sqeuence parameter set (SPS) contains important header information that applies to all NAL units in the coded video sequence. A Picture Parameter Set contain header information that applies to the decoding of one or more pictures within the coded video sequence.

2.2.3 parameter set

A parameter set is supposed to contain information that is expected to rarely change and offers the decoding of a large number of VCL NAL units. There are two types of parameter sets:

- sequence parameter sets, which apply to a series of consecutive coded video pictures called a coded video sequence

- picture parameter sets, which apply to the decoding of one or more individual pictures within a coded video sequence.

2.3 VIDEO CODING LAYER

The VCL design follows the block-based hybrid video coding approach , in which each coded picture is represented in blockshaped units of associated luma and chroma samples called macroblocks. The basic source-coding algorithm is a hybrid of inter-picture prediction to exploit temporal statistical dependencies and transform coding of the prediction residual to exploit spatial statistical dependencies.

A video is divided in sequence of pictures. Each Sequence of picture is divide in group of pictures. Each GOP is divide in to the picture. These picture is further divide in to Slices and finally the slice is divide in to the Macroblock.

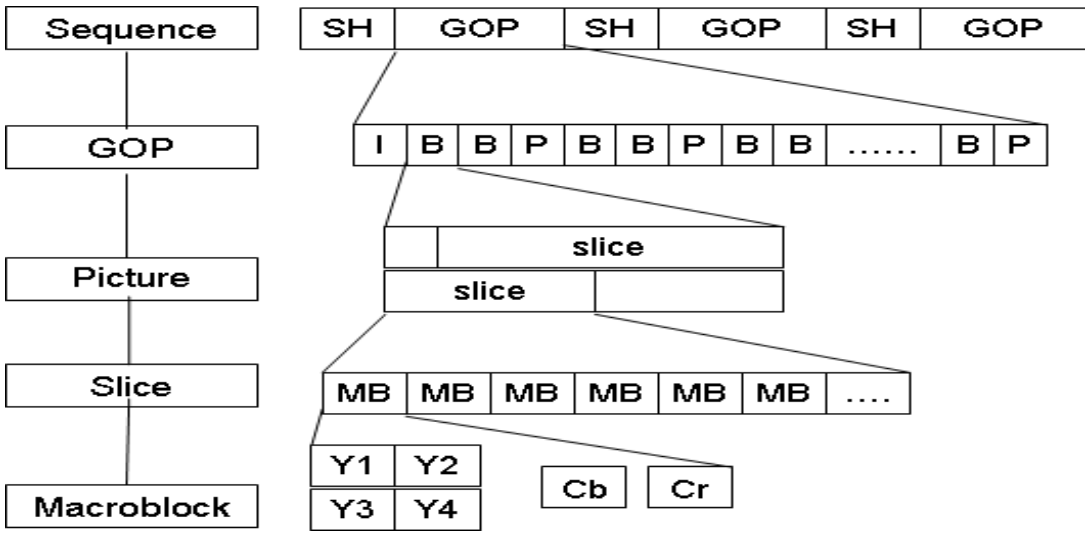

Figure 2.3: Data Structure of H.264

2.3.1 Pictures, frames, and fields

A coded video sequence in H.264/AVC consists of a sequence of coded pictures. A coded picture can represent either an entire frame or a single field, as was also the case for MPEG-2 video.

Generally, a frame of video can be considered to contain two interleaved fields, a top and a bottom field. The top field contains even-numbered rows 0, 2,..., H/2-1 with H being the number of rows of the frame. The bottom field contains the odd-numbered rows (starting with the second line of the frame).

2.3.2 YCbCr color space and 4:2:0 sampling

The video color space used by H.264/AVC separates a color representation into three components called Y, Cb, and Cr. Component Y is called luma, and represents brightness. The two chroma components Cb and Cr represent the extent to which the color deviates from gray toward blue and red, respectively.

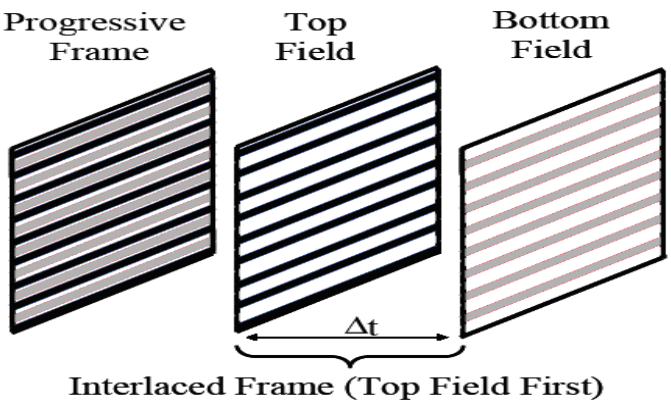

Figure 2.4: Frame Field Types

Because the human visual system is more sensitive to luma than chroma, H.264/AVC uses a sampling structure in which the chroma component has one fourth of the number of samples than the luma component (half the number of samples in both the horizontal and vertical dimensions). This is called 4:2:0 sampling with 8 bits of precision per sample.

2.3.3 Division of the picture into macroblocks

A picture is partitioned into fixed-size macroblocks that each cover a rectangular picture area of 16x16 samples of the luma component and 8x8 samples of each of the two chroma components. This partitioning into macroblocks has been adopted into all previous ITU-T and ISO/IEC JTC1 video coding standards since H.261. Macroblocks are the basic building blocks of the standard for which the decoding process is specified.

2.3.4 Slices and slice groups

Slices are a sequence of macroblocks which are processed in the order of a raster scan when not using flexible macroblock ordering (FMO). A picture maybe split

into one or several slices. A picture is therefore a collection of one or more slices in H.264/AVC. Slices are self-contained in the sense that given the active sequence and picture parameter sets, their syntax elements can be parsed from the bitstream and the values of the samples in the area of the picture that the slice represents can be correctly decoded without use of data from other slices provided that utilized reference pictures are identical at encoder and decoder. Some information from other slices maybe needed to apply the deblocking filter across slice boundaries.

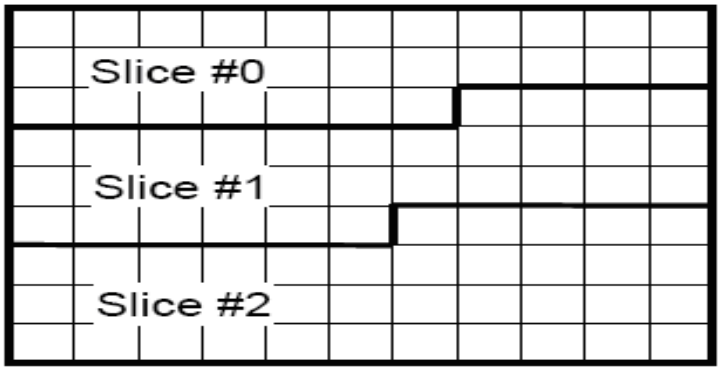

Figure 2.5: Slice Group

FMO modifies the way how pictures are partitioned into slices and macroblocks by utilizing the concept of slice groups. Each slice group is a set of macroblocks defined by a macroblock to slice group map, which is specified by the content of the picture parameter set and some information from slice headers. Using FMO, a picture can be split into many macroblock scanning patterns such as interleaved slices, a dispersed macroblock allocation, one or more "foreground" slice groups and a "leftover" slice group, or a checker-board type of mapping.

Regardless of whether FMO is in use or not, each slice can be coded using different coding types as follows:

- I slice: A slice in which all macroblocks of the slice are coded using intra pre-

diction.

- P slice: In addition to the coding types of the I slice, some macroblocks of the P slice can also be coded using inter prediction with at most one motioncompensated prediction signal per prediction block.

- B slice: In addition to the coding types available in a P slice, some macroblocks of the B slice can also be coded using inter prediction with two motion compensated prediction signals per prediction block.

The above three coding types are very similar to those in previous standards with the exception of the use of reference pictures as described below.

The following two coding types for slices are new:

- SP slice: A so-called switching P slice that is coded such that efficient switching between different precoded pictures becomes possible.

- SI slice: A so-called switching I slice that allows an exact match of a macroblock in an SP slice for random access and error recovery purposes.

2.4 Block diagram of H.264 decoder

H.264/AVC video coding standard has been introduced with significant enhancements in both video coding efficiency and flexibility over a variety of network domains. In video coding layer (VCL), some of important enhancements are the use of a small block-size (4x4) exact-match transform, adaptive in-loop deblocking filter and motion-prediction capability. As shown in Fig , after receiving the data from network abstraction layer (NAL), entropy decoder decodes and the inverse quantized and inverse transformed data is created and added to the predicted data from the previous frames depending upon the header information. Then, the original block can be reconstructed after deblocking filter

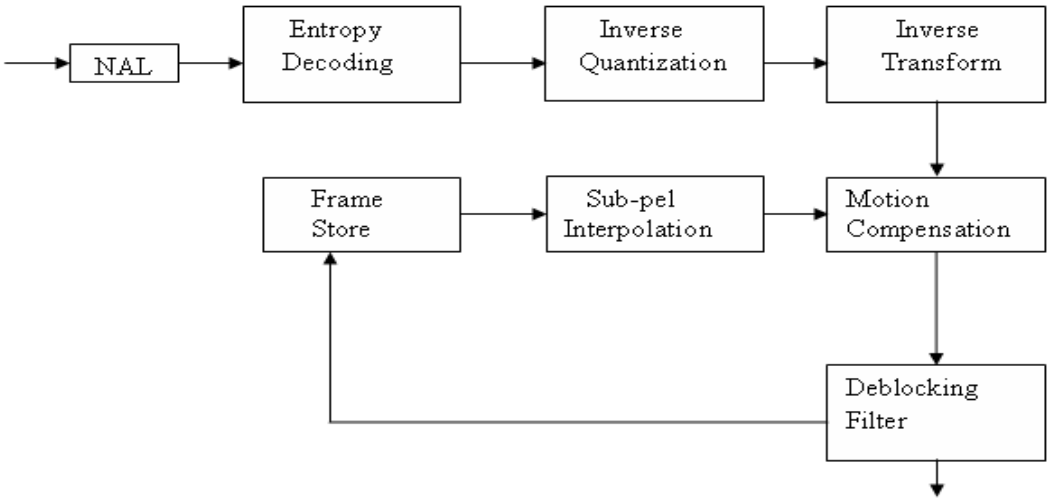

Figure 2.6: Block diagram of H.264 decoder

2.4.1 Entropy Decoding

There are two types of entropy coding used in H.264, and they are the Exp-Golomb codes and CAVLC. The Exp-Golomb codes encode integers using a fixed codeword table, so it is fairly simple to decode. The Exp-Golomb codes are used for all entropy coded parts of the coded video stream except for the residual data. CAVLC stands for Context-based Adaptive Variable Length Coding, and it is a kind of entropy coding where the codeword table continually changes based on the previous data seen. Accordingly, the decoding is much more complicated [1]. CAVLC is used to encode the residual data as described in the previous section.

Exp-Golomb Code Syntax

The Exp-Golomb code is used to encode many different syntax elements in H.264, and not all of them are unsigned integers. Therefore, for many of the syntax elements, an additional table is specified for the mapping between the codeNum and the values they represent. One such example is the mapping to signed integers. For these mapping tables, the more common values of the syntax elements are mapped to

Value	**code**
0	1
1	010
2	011
3	00100
4	00101
.	.
.	.
.	.
42	00000101011
43	00000101100

Table I: Example for Exp-Golomb codes

Unsigned Exp-Golomb code	0	1	2	3	.	.	42	43
signed Exp-Golomb code	0	1	-1	2	.	.	-21	22

Table II: Example for unsigned-to-signed-mapping

smaller codeNums, so they can be coded more efficiently.

Exp-Golomb codes consist of three parts: First, there is a number of n 0 bits, followed by one 1 bit and again n bits with an offset value v. The final code value x is derived by x = 2^n-1+v

Signed Exp-Golomb codes are parsed by retrieving an unsigned Exp-Golomb code c from the bitstream and mapping it from unsigned to signed using the following rule: If c is even, -c/2 is returned; if c is odd, (c+1)/2 is returned; zero remains unchanged.

CAVLC Decoding Process

The typical output of the CAVLC decoding process is an array of 16 integers, which are arranged in such a way that the numbers at the start of the array are more likely to have large absolute values. The numbers toward the end of the array are likely to have small values like 0, 1, or -1.

A typical example of such an array would be 8, 5, -1, -2, 0, 0, 1, 0, -1, 0, 0, 0, 0, 0, 0, 0. To take advantage of these known properties, the array of integers is encoded into the following components, each of which has unique code tables. The number of non-zero QTCs and that of trailing ones are first decoded. Then, non-zero QTCs with value equal to 1 are reconstructed with their corresponding sign bits. After that, non-zero QTCs and zeros before each non-zero QTC are decoded. Finally, non-zero QTCs are stored into a 1-D array in the zig-zag scan order.

CABAC Decoding Process

In context-based adaptive binary arithmetic coding (CABAC) a context table including contexts which are calculated accrding to reference data. The context table includes contexts, Most Probable Symbols (MPS) and probability index (pState). The pState is between 0 and 63, and the larger number means that there is a higher probability of the occurrence of 0/1.

2.4.2 Inverse transformation

Transformation is used to convert samples from spatial to frequency domain. The AVC uses 4x4 integer transformation, similar to the 4x4 DCT. The decoder reverses this process by performing inverse transformation. This conversion is separable and can be performed in two steps - vertically and horizontally. Fast algorithms require eight additions and two binary shifts.

2.4.3 Deblocking filter

H.264 employs an adaptive deblocking filter, which is applied to each decoded macroblock to reduce blocking artifacts. Its aim is to smooth the blocking edges around the boundary of each macroblock without affecting the sharpness of the picture, thus

improving the subjective video quality of the compressed video. The filtered data are used for motion compensated prediction of further video frames. The filter affects the perceptual quality of decoded frames and the efficiency of compression. If the filtered image is a close "match" to the original, the motion compensated residual is likely to be reduced, leading to higher compression efficiency .

2.4.4 Intra prediction

Intra prediction can be invoked for I and SI macroblock types for both luminance and chrominance. The samples' values are predicted from context which is previously reconstructed left, above and above-right spatially neighboring samples in Intra4x4 or Intra16x16 modes. Prediction type selection is conditioned by the content of processed block.

For the 4x4 luminance blocks, nine directions has been defined, including vertical, horizontal and DC prediction. In horizontal and vertical modes samples are predicted directly from context values, whereas each pixel from a block is estimated by weighted mean of up to three neighboring context samples when more sophisticated procedure is selected.

2.4.5 Motion Compensation (Inter-frame Prediction)

The operation of motion compensation can be regarded as copying the predicted macroblock from reference frame. The predicted macroblock is added to the residual macroblock (generated by inverse transforms and quantization) to reconstruct the macroblock in the current frame.

Motion compensator is the most demanding component of the decoder, consuming more than half of its computation time . Intending to increase the coding efficiency,

the H.264/AVC standard adopted a number of relatively new technical developments. Most of these new developments rely on the motion prediction process, like: variable block-size, multiple reference frames, motion vector over picture boundaries, motion vector prediction, quarter-sample accuracy, bi-predictive slices and weighted prediction.

Motion compensation module is composed by three main sub-modules :

Motion vector predictor (MVP) :

which infers the motion vectors from the syntactic elements available in the bitstream and from neighbor MBs information

Sample Processor :

which generates the quarter-pixel interpolated samples

Frame Memory Access :

which brings the reference frames data to be processed by the sample processing module.

This work presents the design of a memory hierarchy solution for the frame memory access unit. The H.264 standard extends this in several ways providing support for variable block sizes (up to 4x4), fine (fractional) sub-pixel accuracy for motion vectors (pixel in the luma component), and multi-frame references for accurate prediction.

The motion compensation is treated in two types of slice (P and B) respectively. In the case of motion compensation in P slices, the luminance component of each macroblock (16x16 samples) may be split up into four ways to produce a macroblock partition. When the 8x8 mode is chosen, each of the four 8x8 macroblock partitions within the macroblock may be further split into 4 ways.

Chapter 3

Decoder Parallelization process

3.1 Problem Definition

The coding efficiency gains of advanced video codecs like H.264, come at the price of increased computational requirements. The demands for computing power increases also with the shift towards high definition resolutions. As a result, current high performance uniprocessor architectures are not capable of providing the required performance for real-time processing . Thus, it is necessary to exploit parallelism.

3.2 Existing Methodologies

The H.264 codec can be parallelized either by a task-level or data-level decomposition.

3.2.1 Task-Level Decomposition

In a task-level decomposition the functional partitions of the algorithm are assigned to different processors. For example, Inverse Quantization IQ) and the IDCT can be done in parallel with the Motion Compensation (MC) stage. The main drawbacks of task-level decomposition are load balancing and scalability. Balancing the load is difficult because the time to execute each task is not known a priori and depends on

the data being processed. Scalability is a problem because it is limited to the number of tasks, which typically is small .

3.2.2 Data-Level Decomposition

In a data-level decomposition the work (data) is divided into smaller parts and each assigned to a different processor. Each processor runs the same program but on different (multiple) data elements. In H.264 data decomposition can be applied at different levels of the data structure: Group-of-Pictures (GOP) level, frame-level, slice-level, macroblock-level, and block-level. The most important of these will be explained next.

Frame-Level Parallelism

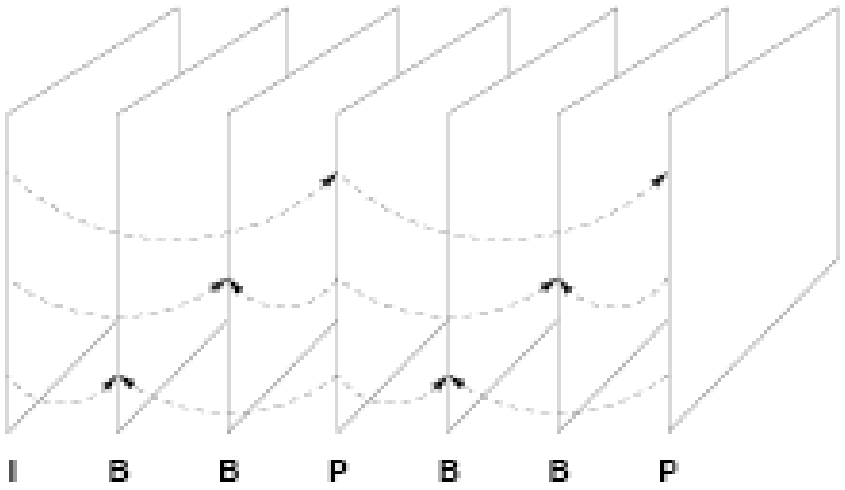

Figure 3.1: slices

As shown in Figure 3.1 in a sequence of I-B-B-P frames some frames are used as reference for other frames (like I and P frames) but some frames (B frames) are not used as references. That means that B frames can be processed in parallel [2]. In this

case, a control processor can assign independent frames to different processors. Frame-level parallelism has good load balancing but scalability problems. This is due to the fact that usually there are no more than two or three B frames between P frames. This limits the amount of TLP to a few threads. However, the main disadvantage of frame-level parallelism is that, unlike previous video standards, in H.264 B frames can be used as reference. In such a case, there is no or little parallelism available.

Slice-Level Parallelism

In H.264 and in most current hybrid video coding standards each picture is partitioned into one or more slices. As slices are completely independent from each other they can be processed in parallel without dependency or ordering constraints. [3, 4].However, as the number of slices is determined by the encoder, there is a scalability and load balancing problem if there is no control of what the encoder does. More important, increasing the number of slices per frame increases the bitrate for the same quality level. Going from 1 to 64 slices per frame increases the bitrate up to 34%.

Macroblock-Level Parallelism

To exploit parallelism between MBs it is necessary to take the dependencies between them into account. In H.264, motion vector prediction, intra prediction, and the de-blocking filter use data from neighboring MBs defining a complex set of dependencies. Processing MBs in a diagonal wavefront manner satisfies all the dependencies and at the same time allows to exploit parallelism between MBs.

At time slot T7 three independent MBs can be processed: MB (4,1), MB (2,2) and MB (0,3) [5]. The number of independent MBs varies over time. At the start of decoding a frame it increases with one MB every two time slots, then stabilizes at its maximum, and finally decreases at the same rate it increased. For a low resolution like QCIF there are at most 6 independent MBs during 4 time slots. For Full High Definition (1920x1088) there are at most 60 independent MBs during 9 time slots.

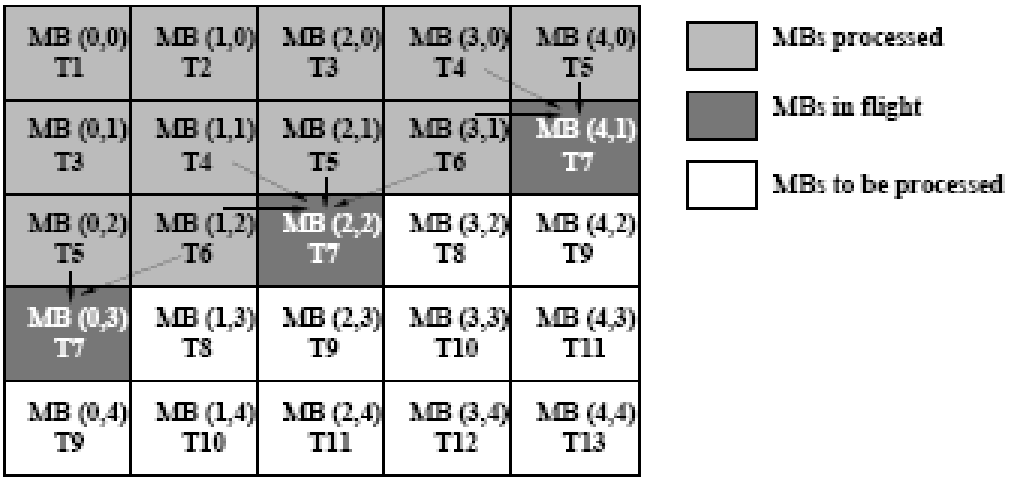

Figure 3.2: Macroblock Parllelism

MB-level parallelism has many advantages over other schemes for parallelization of H.264. First, this scheme can have a good scalability. Second, it is possible to achieve a good load balancing if a dynamic scheduling system is used. Additionally, because in MB-level parallelization all the processors/threads run the same program the same set of software optimizations can be applied to all processing elements.

However, MB-level parallelism has some disadvantages. The first one is that the entropy decoding cannot be parallelized using data decomposition, due to the fact that the lowest level of data that can be parsed from the bitstream are slices. Only after entropy decoding has been performed the parallel processing of MBs can start. This disadvantage can be overcome by using special purpose instructions or hardware accelerators for entropy decoding. The second disadvantage is that the number of independent MBs is low at the start and at the end of decoding a frame. Therefore, it is not possible to sustain a certain processing rate during the decoding of a frame.

3.3 Parallel Decoding Process Implementation

To parallelize H.264 Decoder effectively, data partitioning method is widely used [6]. However, due to the dependencies among data, it cannot be applied easily. Actually, the following data dependencies exist between neighboring blocks of the same frame in H.264/AVC.

- Predicted Motion Vector
- Intra Prediction and Mode decision
- Quarter-pel interpolation and deblocking-filter

3.3.1 Decoding Process

An overview of the decoding process is given as follows.

- Decoding of NAL units
- Decoding processes using syntax elements in the slice layer and above.
  - Variables and functions relating to picture order count (only needed to be invoked for one slice of a picture)
  - Variables and functions relating to the macroblock to slice group map (only needed to be invoked for one slice of a picture)
  - The method of combining the various partitions when slice data partitioning is used
  - Prior to decoding each slice, the derivation of reference picture lists is necessary for inter prediction.
  - When the current picture is a reference picture and after all slices of the current picture have been decoded, the decoded reference picture marking

process specifies how the current picture is used in the decoding process of inter prediction in later decoded pictures.

- The processes specify decoding processes using syntax elements in the macroblock layer and above.
  - The intra prediction process for I and SI macroblocks except for I_PCM macroblocks provides the intra prediction samples being the output. For I_PCM macroblocks directly specifies a picture construction process. The output are the constructed samples prior to the deblocking filter process.
  - The inter prediction process for P and B macroblocks is with inter prediction samples being the output.
  - The decoding process transform coefficient and picture construction prior to deblocking filter process . The transform coefficient decoding process derives the residual samples for I and B macroblocks as well as for P macroblocks in P slices. The output are the constructed samples prior to the deblocking filter process.
  - The decoding process for transform coefficients and picture construction prior to deblocking for P macroblocks in SP slices or SI macroblocks . The output are the constructed samples prior to the deblocking filter process.
  - The constructed samples prior to the deblocking filter process that are next to the edges of blocks and macroblocks are processed by a deblocking filter with the output being the decoded samples.

3.3.2 H.264 Decoder Load Analysis and partitioning

The normal H.264/AVC decoder [7] for full HD resolution H.264 stream to estimate the computation portion of the modules in Figure 3.3 The profiling was performed with an H.264 HD (1920x1080) resolution progressive sequence, encoded in 8Mbps

bit-rate and IBBP prediction mode, using CAVLC entropy coding.

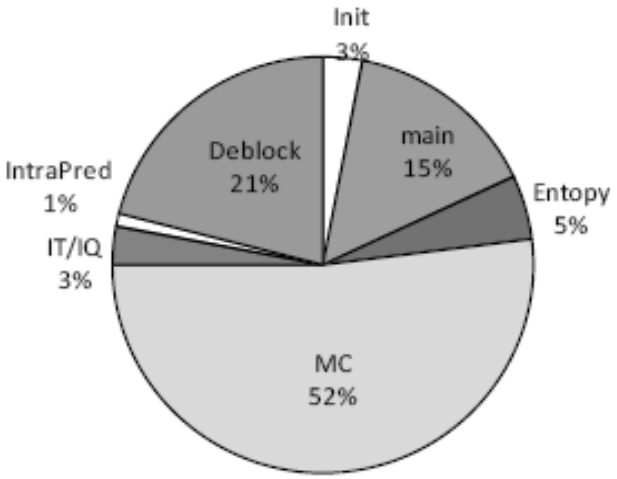

Figure 3.3: Computation profile result for H.264 Decoder

Among the modules, motion compensation and deblocking filter occupy most of execution time and are considered for offloading in the first step.Therefore to make execution faster first of all motion compensation process is to be parallelize.Block motion compensation divides up the current frame into non-overlapping blocks, and the motion compensation vector tells where those blocks come from a common misconception is that the previous frame is divided up into non-overlapping blocks, and the motion compensation vectors tell where those blocks move to. The source blocks typically overlap in the source frame.

3.3.3 Motion Compensation

H.264 provides various block sizes for inter prediction to improve the coding gain. One 16x16 MB can be partitioned into one 16x16 block, two 16x8 or 8x16 blocks or four 8x8 blocks. Each 8x8 blocks can be further partitioned into two 8x4 or 4x8 blocks or four 4x4 blocks. Moreover, each block whose size is larger than 8x8 can be predicted using

different reference frames. In addition, H.264 provides fractional samples of motion estimation (ME). The Motion Vector(MV) can be of half- or quarter-pel resolution.

The half-pel value is obtained by applying a one-dimension 6-tap FIR interpolation filter horizontally (the x-direction) or vertically (the y-direction). The quarter-pel value is obtained by the average of two nearest half-pel values. Furthermore, H.264 provides the so-called P_Skip mode to code an MB in a large area with slow motion. When one MB is coded as the P_Skip mode, neither MV nor residual error data are transmitted. The MV of the P_Skip mode MB is decided by those MVs of its neighboring MBs. Further details regarding the Motion Compensation is given in next chpater.

Chapter 4

Motion Compensation

H.264 provides various block sizes for inter prediction to improve the coding gain. One 16x16 MB can be partitioned into one 16x16 block, two 16x8 or 8x16 blocks or four 8x8 blocks. Each 8x8 blocks can be further partitioned into two 8x4 or 4x8 blocks or four 4x4 blocks. Moreover, each block whose size is larger than 8x8 can be predicted using different reference frames. In addition, H.264 provides fractional samples of motion estimation (ME). The Motion Vector(MV) can be of half- or quarter-pel resolution.

The half-pel value is obtained by applying a one-dimension 6-tap FIR interpolation filter horizontally (the x-direction) or vertically (the y-direction). The quarter-pel value is obtained by the average of two nearest half-pel values. Furthermore, H.264 provides the so-called P_Skip mode to code an MB in a large area with slow motion. When one MB is coded as the P_Skip mode, neither MV nor residual error data are transmitted. The MV of the P_Skip mode MB is decided by those MVs of its neighboring MBs.

Recently, an MCP decoding complexity model for H.264/AVC was proposed in [8] to overcome the difficulties described above. The Mostdecoding complexity is modeled as a function of the number of cache misses N_c , the number of y-direction interpolation filters N_y, the number of x-direction interpolation filters N_x and the number of MVs

per MB N_v . Mathematically, it can be written as of

$$\alpha \cdot N_c + \beta \cdot N_y + \gamma \cdot N_x + \mu \cdot N_v \tag{4.1}$$

These new developments rely on the motion prediction process, like: variable block-size, multiple reference frames, motion vector over picture boundaries, motion vector prediction, quarter-sample accuracy, bi-predictive slices and weighted prediction [9]

4.1 Inter-Frame Prediction in P Slices:

In addition to the intra macroblock coding types, various predictive or motion- compensated coding types are specified as P macroblock types. Each P macroblock type corresponds to a specific partition of the macroblock into the block shapes used for motion- compensated prediction.

The prediction signal for each predictive-coded M X N luma block is obtained by displacing an area of the corresponding reference picture, which is specified by a translational motion vector and a picture reference index. Thus, if the macroblock is coded using four 8 X 8 partitions and each 8 X 8 partition is further split into four 4 X 4 partitions, a maximum of 16 motion vectors may be transmitted for a single P macroblock.

The accuracy of motion compensation is in units of one quarter of the distance between luma samples. In case the motion vector points to an integer-sample position, the prediction signal consists of the corresponding samples of the reference picture; otherwise the corresponding sample is obtained using interpolation to generate noninteger positions. The prediction values at half-sample positions are obtained by applying a one-dimensional 6-tap FIR filter horizontally and vertically. Prediction values at quarter-sample positions are generated by averaging samples at integer- and half-sample positions. Figure 4.1 illustrates the fractional sample interpolation

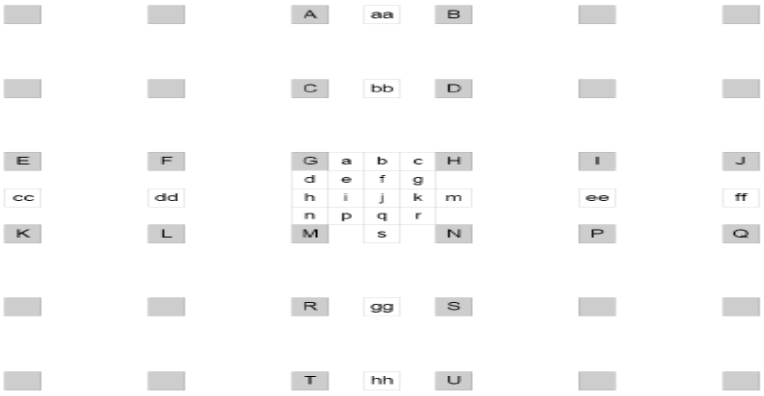

Figure 4.1: Sample Position

for samples ak and nr. The samples at half sample positions labeled and are derived by first calculating intermediate values and , respectively by applying the 6-tap filter as follows:

$$b_1 = (E - 5F + 20G + 20H - 5I + J) \tag{4.2}$$

$$h_1 = (A - 5C + 20G + 20M - 5R + T) \tag{4.3}$$

The final prediction values for locations and are obtained as follows and clipped to the range of 0255:

$$b = (b_1 + 16) >> 5 \tag{4.4}$$

$$h = (h_1 + 16) >> 5 \tag{4.5}$$

The samples at half sample positions labeled as are obtained by

$$j_1 = cc - 5dd + 20h_1 + 20m_1 - 5ee + ff \tag{4.6}$$

where intermediate values denoted as cc ,dd ,ee ,m_1 and ff are obtained in a manner

similar to h_1. The final prediction value is then computed as

$$j = (j_1 + 512) >> 10 \tag{4.7}$$

and is clipped to the range of 0 to 255. The two alternative methods of obtaining the value of illustrate that the filtering operation is truly separable for the generation of the half-sample positions.

The samples at quarter sample positions labeled as a, c, d, n, f, i, k, and q are derived by averaging with upward rounding of the two nearest samples at integer and half sample positions as, for example, by

$$a = (G + b + 1) >> 1 \tag{4.8}$$

The samples at quarter sample positions labeled as e, g, p, and r are derived by averaging with upward rounding of the two nearest samples at half sample positions in the diagonal direction as, for example, by

$$e = (b + h + 1) >> 1 \tag{4.9}$$

The syntax allows so-called motion vectors over picture boundaries, i.e., motion vectors that point outside the image area. In this case, the reference frame is extrapolated beyond the image boundaries by repeating the edge samples before interpolation.

Multiframe motion-compensated prediction requires both encoder and decoder to store the reference pictures used for inter prediction in a multipicture buffer. The decoder replicates the multipicture buffer of the encoder according to memory management control operations specified in the bitstream. Unless the size of the multipicture buffer is set to one picture, the index at which the reference picture is located inside the multipicture buffer has to be signalled.

Motion compensation for smaller regions than 8 X 8 use the same reference index for prediction of all blocks within the 8 X 8 region. In addition to the motion-

compensated macroblock modes described above, a P macroblock can also be coded in the so-called P_Skip type. For this coding type, neither a quantized prediction error signal, nor a motion vector or reference index parameter is transmitted.

The reconstructed signal is obtained similar to the prediction signal of a P_16 16 macroblock type that references the picture which is located at index 0 in the multipicture buffer. The motion vector used for reconstructing the P_Skip macroblock is similar to the motion vector predictor for the 16 16 block. The useful effect of this definition of the P_Skip coding type is that large areas with no change or constant motion like slow panning can be represented with very few bits.

4.2 Inter-Frame Prediction in B Slices

Thus, the substantial difference between B and P slices is that B slices are coded in a manner in which some macroblocks or blocks may use a weighted average of two distinct motion-compensated prediction values for building the prediction signal. B slices utilize two distinct lists of reference pictures, which are referred to as the first (list 0) and second (list 1) reference picture lists, respectively. Which pictures are actually located in each reference picture list is an issue of the multipicture buffer control and can be enabled if desired by the encoder.

In B slices, four different types of inter-picture prediction are supported: list 0, list 1, bi-predictive, and direct prediction. For the bi-predictive mode, the prediction signal is formed by a weighted average of motion-compensated list 0 and list 1 prediction signals. The direct prediction mode is inferred from previously transmitted syntax elements and can be either list 0 or list 1 prediction or bi-predictive. B slices utilize a similar macroblock partitioning as P slices.

Beside the P_16 X 16, P_16 X 8, P_8 X 16, P_8 X 8, and the intra coding types, bi-predictive prediction and another type of prediction called direct prediction, are provided. For each 16 X 16, 16 X 8, 8 X 16, and 8 X 8 partition, the prediction method (list 0, list 1, bi-predictive) can be chosen separately.

An 8 X 8 partition of a B macroblock can also be coded in direct mode. If no prediction error signal is transmitted for a direct macroblock mode, it is also referred to as B_Skip mode and can be coded very efficiently similar to the P_Skip mode in P slices. The motion vector coding is similar to that of P slices with the appropriate modifications because neighboring blocks may be coded using different prediction modes.

Each macroblock, in B-slices, can be derived from one or two reference frames using one of several ways. The alternatives are direct prediction, prediction from List 0, prediction from List 1 or biprediction which is a linear combination of List 0 and List 1 motion compensated references.

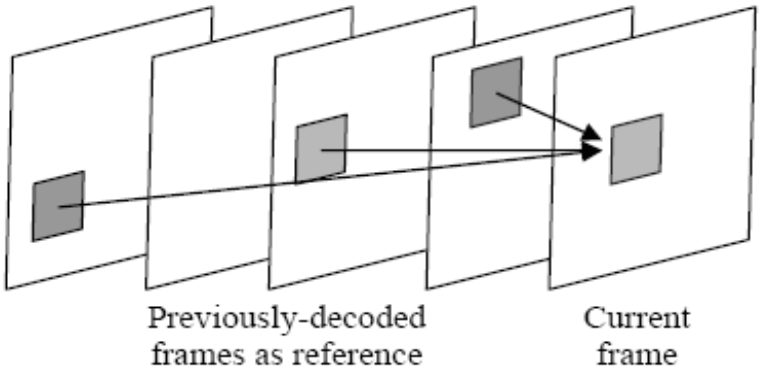

Figure 4.2: Inter Compensation

Inter-prediction consists of three main components, the reference picture list construction, motion vector calculation, and sub-sample interpolation. Each of these components will be described in a subsection below.

4.2.1 Reference Picture Lists

After the decoding of each frame, the decoded frame can be placed into either the short-term pic list or the long-term pic list so that it can be referenced for the decoding of future inter-predicted frames. Usually, the new frame is entered into the short-term

pic list, replacing the oldest frame in the list if necessary. The encoded stream can contain instructions on how the frames should be placed inside the pic lists.

For each slice to be decoded using inter-prediction, a reference pic list is constructed using the frames in the short-term pic list and the long-term pic list. Again, the encoded stream can contain instructions on how to reorder these frames. The more frequently used reference frames are placed in the beginning of the list (smaller index) to take advantage of the entropy coding feature of H.264.

Default order of short term reference pictures in reference lists:

List0 (B slice) : Decreasing order of PicOrderCount (for pictures with POC earlier than current picture) and increasing order of PicOrderCount (for pictures with POC later than current picture).

List1 (B slice): increasing order of PicOrderCount (later than current picture) then decreasing order of PicOrderCount (earlier than current picture)

4.2.2 Motion Vector Calculation

For each block (which can be 16x16, 16x8, 8x16, 8x8, 8x4, 4x8, 4x4 pixels in size) to be inter-predicted, the encoded stream specifies which frame in the reference pic list should be used. In addition, a motion vector is needed, which gives the location of the block used for motion compensation in the reference frame, relative to the current location in the frame being decoded. For example, if the motion vector is (5,2), then the predicted value for pixel location (1,0) in the current frame comes from pixel location (6,2) in the reference frame.

The motion vector for the current block is predicted using the motion vectors used for neighboring blocks that have already been decoded, taking into account whether each of the neighboring blocks uses the same reference picture or not. The difference between the predicted motion vector and final motion vector that should be used is given by the encoded stream.

4.2.3 Sub-Sample Interpolation

The motion vectors used in H.264 do not necessarily have integer components. For each of the vertical and horizontal components, the motion vector can have quarter precision. For example, For example, if the motion vector is (5.75,2.5), then the predicted value for pixel location (1,0) in the current frame comes from pixel location (6.75,2.5) in the reference frame. The values of these fractional sample locations are generated by interpolating the values of the nearby integer samples.

Luma Interpolation To illustrate the interpolation process for the luma samples, figure 4.1 shows some integer samples (grey squares) and fractional samples (white squares), each labeled with upper case or lower case letters. For example, pixel location (6.75,2.5) corresponds to k in figure 4.1. The half samples b, h, and j, are calculated using a 6-tap filter. To calculate the value of sample b, the 6-tap filter is applied to the samples E, F, G, H, I, and J.

Similarly, the same 6-tap filter is applied, except vertically, to get sample h. To get the sample j, the 6-tap filter is first applied horizontally to obtain samples aa, bb, b, s, gg, and hh, then the filter is applied vertically to get the value of j. Alternatively, j can be calculated by filtering vertically then horizontally, as the result would be identical. The quarter samples a, c, d, e, f, g, i, k, n, p, q, and r, are calculated by averaging the two closest integer samples or half samples.

Chroma Interpolation Since there is only one chroma sample for each 2x2 block of luma samples, quarter luma positions would actually correspond to eighth chroma positions. However, the chroma interpolation does not use the 6-tap filter. Instead, a fractional sample value is calculated by using a weighted average of the 4 surrounding integer samples

Chapter 5

Implementation and Simulation

5.1 Implementation Environment

5.1.1 Scratchbox

Scratchbox is a cross-compilation toolkit designed to make embedded Linux application development easier. It also provides a full set of tools to integrate and cross-compile an entire Linux distribution.

A brief summary of features:

- Scratchbox is used to cross-compile which generate binaries e.g.ARM
- Supports ARM and x86 targets.
- Provides glibc and uClibc as C-library choices.

5.1.2 SimIt-ARM

The SimIt-ARM package contains an instruction-set simulator and a cycle-accurate simulator for the StrongARM architecture. Both simulators read ELF32 little-endian ARM-linux binaries and can simulate.

SimIt-ARM 3.0 is an instruction-set simulator that runs both system-level and user-level ARM programs. It supports two popular simulation styles: interpretation and dynamic-compiled simulation. It is developed for the IA32 Linux platform but may work for other platforms as well. SimIt-ARM is free, open-source software under the GNU General Public License.

SimIt-ARM features:

Full system simulation SimIt-ARM supports the ARM v5 architecture, including the memory management unit and some fundamental I/O devices. High simulation speed On a Pentium D 2.8GHz desktop, the interpreter runs at above 30MIPS.

Built-in debugger The '-d' flag enables a light-weight debugging interface, allowing one to step through a program and to observe register/memory values.

Modular design SimIt-ARM is developed in C++. All simulation states are packaged in C++ classes. Therefore it is easy to create multiple simulator objects for modeling multiprocessor targets.

Dynamic compiled simulation

The dynamic-compiled simulator combines the functionality of the interpreter and the static-compiled simulator. It interprets the target program at the beginning. Meanwhile it profiles the program by keeping a counter for each block of code. The

counter records the cumulative number of simulated instructions in the page. If the number is beyond a predefined threshold, then this page is deemed hot and is translated. Translation is performed in two steps.

First the page is translated to a C++ function. Next the C++ function is compiled into a shared library and linked to the simulator itself. Compilation of the C++ function can be done by the simulator process itself, or can be distributed to a separate thread, or a remote translation server. Obviously, using of more CPUs or translation servers means faster simulation speed.

5.2 OMAP

The OMAP (Open Multimedia Application Platform) architecture is based on a combination of TI's state-of-the-art TMS320C55x DSP core and high performance ARM Cortex A-8 CPU.

The OMAP35xx family of high-performance, applications processors are based on the enhanced OMAP 3 architecture and are integrated on TI's advanced 65-nm process technology.

The architecture is designed to provide best-in-class video, image, and graphics processing sufficient to support the following:

- Streaming video
- 2D/3D mobile gaming
- Video conferencing
- High-resolution still image
- Video capture in 2.5G wireless terminals, 3G wireless terminals, and rich multimedia-featured handsets.

For instance, a single DSP can process, in realtime, a full videoconferencing application (audio and video at 15 images/sec.), using only 40 percent of the available computational capability. Therefore, 60 percent of the capacity can be employed to run other applications concurrently. At the same time, in the OMAP dual-core architecture, the ARM processor stands ready to handle any other application requirements or can be suspended, thus saving battery life. As a result, the mobile user can enjoy access to popular OS applications (Word, Excel, etc.) while also engaging a videoconferencing application.

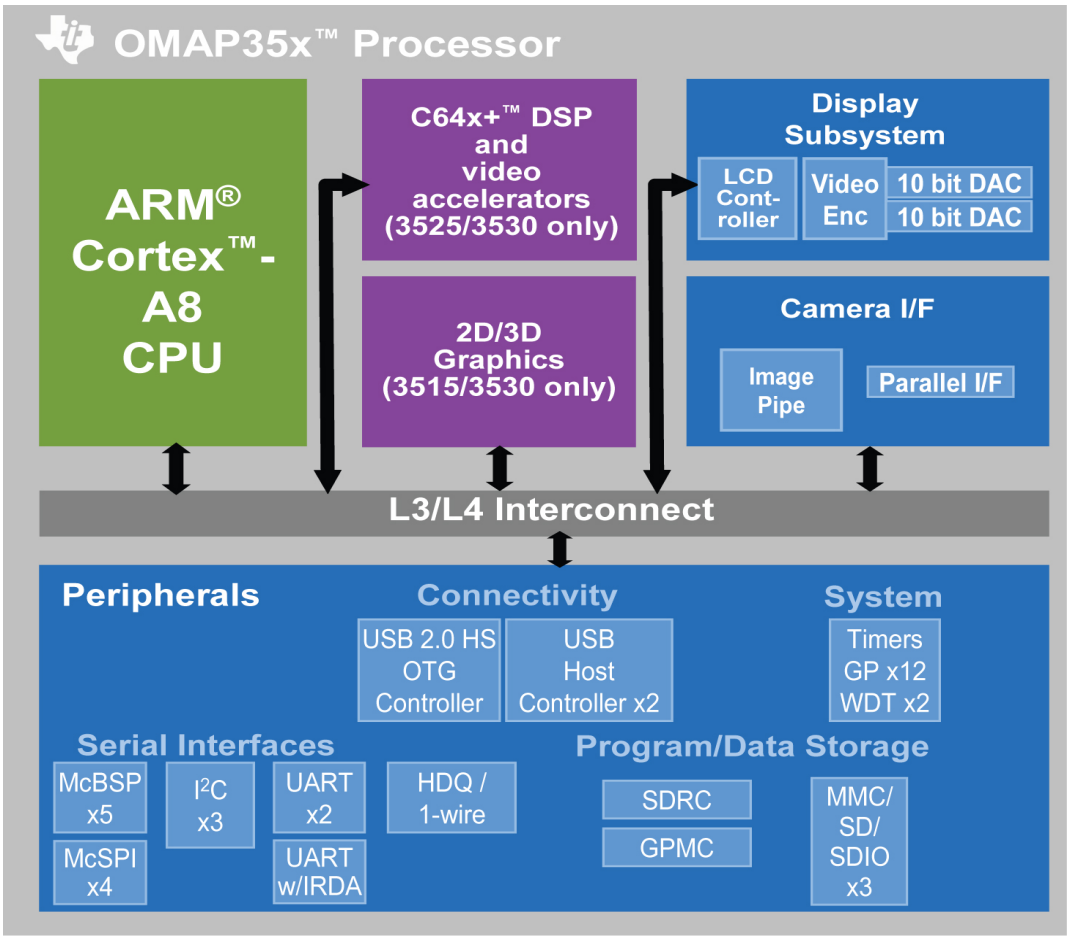

Figure 5.1: OMAP 35x Processor

The following subsystems are part of the device:

- Microprocessor unit (MPU) subsystem based on the ARM Cortex-A8 microprocessor
- IVA2.2 subsystem with a C64x+ digital signal processor (DSP) core
- SGX530 subsystem for 2D and 3D graphics acceleration to support display and gaming effects
- Camera image signal processor (ISP) that supports multiple formats and interfacing options connected to a wide variety of image sensors
- Display subsystem with a wide variety of features for multiple concurrent image manipulation, and a programmable interface supporting a wide variety of displays. The display subsystem also supports NTSC/PAL video out.
- Level 3 (L3) and level 4 (L4) interconnects that provide high-bandwidth data transfers for multiple initiators to the internal and external memory controllers and to on-chip peripherals

5.3 Implementation

This process is invoked when decoding P and B macroblock types. Outputs of this process are Inter prediction samples for the current macroblock that are a 16x16 array predL of luma samples and two 8x8 arrays predCr and predCb of chroma samples, one for each of the chroma components Cb and Cr.

In motion compensation module of decoder first of all the required memory for the prediction is allocated to the Motion compensation prediction module.It will allocate 2D memory array for the temp block 0 and 1.which are useful to store the required blocks.

The partitioning of a macroblock is specified by mb_type. Each macroblock partition is referred to by macroblock Partition Index. When the macroblock partitioning consists of partitions that are equal to sub-macroblocks, each sub-macroblock can

be further partitioned into sub-macroblock partitions as specified by sub macroblock type.

The Inter prediction process for a macroblock partition macro block Partition Index and a sub-macroblock partition sub Macroblock Partition Index consists of the following ordered steps

5.3.1 Derivation process for motion vector components and reference indices

Inputs to this process are a macroblock partition mbPartIdx and a sub-macroblock partition subMbPartIdx. Outputs of this process are- luma motion vectors mvL0 and mvL1 and the chroma motion vectors mvCL0 and mvCL1 ,reference indices refIdxL0 and refIdxL1 ,prediction list utilization flags predFlagL0 and predFlagL1

For the derivation of the variables motion vector list 0 and list 1 as well as reference indoces 0 and reference index 1, the following applies.

- If macroblock type is equal to P_Skip, the derivation process for luma motion vectors for skipped macroblocks in P and SP slices is invoked with the output being the luma motion vectors 0 and reference indices 0, and prediction Flag 0 is set equal to 1. motion vector 1 and reference indices 1 are marked as not available and prediction Flag 1 is set equal to 0.
- Otherwise, if macroblock type is equal to B_Skip, or B_Direct_16x16 or sub macroblock type is equal to B_Direct_8x8, the derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8 in B slices is invoked with macroblock Part indices and sub Macroblock Part indices as the input and the output being the luma motion vectors 0 and 1, the reference indices reference indices 0 and 1, and the prediction utilization flags prediction flag 0 and 1.

5.3.2 Derivation process for luma motion vector

The derivation process for the neighbouring blocks for motion data is invoked with macroblock Part indices, sub macroblock part indices, and list suffix as the input and with macroblock address , macroblock part indices and sub macroblock part indices , reference indices and the motion vectors with N being replaced by A, B, or C as the output.

If the macro block is 16 X 8 and macroblock part indices is equal to 0 then motion vector prediction is set to B.If macro block is 16 X 8 and macroblock part indices is 1 then motion vector prediction is set to A. If the macro block is 8 X 16 and macroblock part indices is equal to 0 then motion vector prediction is set to A.If macro block is 8 X 16 and macroblock part indices is 1 then motion vector prediction is set to C.

To get the luma block the Interpolation of 1/4 subpixel is used to read the luma block from memory.Same as in chroma block if the reference picture is not available then luma block will be filled with 128 values.Then depending on the Independent picture or dependent picture the current reference frame will be decided.Then depending on the vertical interpolation,horizontal interpolation, diagonal interpolation, vertical and horizontal interpolation the block will be taken from memory and set in luma block.

5.3.3 Derivation process for chroma motion vectors

A chroma motion vector is derived from the corresponding luma motion vector. Since the accuracy of luma motion vectors is one-quarter sample and chroma has half resolution compared to luma, the accuracy of chroma motion vectors is one-eighth sample, i.e., a value of 1 for the chroma motion vector refers to a one-eighth sample displacement.

For example when the luma vector applies to 8x16 luma samples, the corresponding chroma vector applies to 4x8 chroma samples and when the luma vector applies

to 4x4 luma samples, the corresponding chroma vector applies to 2x2 chroma samples.For the derivation of the motion vector for chroma samples, the following applies.

-If the current macroblock is a frame macroblock, the horizontal and vertical components of the chroma motion vector are derived by multiplying the corresponding components of luma motion vector by 2, through mapping one-quarter sample motion vector units to one-eighth sample motion vector units.

- Otherwise (the current macroblock is a field macroblock), only the horizontal component of the chroma motion vector is derived . The vertical component of the chroma motion vector is dependent on the parity of the current field or the current macroblock and the reference picture, which is referred by the reference index.

Now, in get chroma block chroma function will be get chroma block required for decoding. It will use full pel position to get chroma block.First of all block values will be filled with the 128 values if no reference picture is available.If reference picture is available on the position of horizontal and vertical values the accordingly from the memory the values will be get and stored in chroma block.

5.3.4 Decoding process for Inter prediction samples

Outputs of this process are the Inter prediction samples , which are a array of prediction luma samples, and two arrays of prediction chroma samples, one for each of the chroma components Cb and Cr.

In block bi-prediction B pictures use both future and past pictures as references. The prediction signals are obtained by linearly combining the prediction values separately from forward reference picture and backward reference picture based on motion compensation. B pictures can achieve higher compression ratio by more effectively

exploiting the correlation between reference pictures and current B picture.

At the decoder, B pictures will be reconstructed from transmitted pictures I/P pictures. Such a technique is referred to as frame interpolation. Therefore, there are no residual and motion vectors transmitted for interpolated frames. The motion vectors of every block in B pictures are derived from the transmitted pictures. However, frame interpolation is only suitable for video sequences with simple translational motion.

Reference picture list Reference Picture List is a list of variables Picture number for short-term reference pictures and Long Term Picture Number for long-term reference pictures of previously decoded reference frames, complementary reference field pairs, or non-paired reference fields that have been marked as used for reference.

Depending on field picture flag, the meaning of Picture number and Long Term Picture number is specified as follows.
- If field picture flag is equal to 1, all entries of the Reference Picture List are variables Picture Number and Long Term Picture Number of decoded reference fields or fields of decoded reference frames.
- Otherwise (field picture flag is equal to 0), all entries of Reference Picture List are variables Picture Number and Long Term Picture Number of decoded reference frames or complementary reference field pairs.

5.3.5 Fractional sample interpolation process

The other function check motion vector range is useful after the Motion Estimation is done and the motion vectors are ready then the motion vector range is check whether it is in expectable limit or not on the horizontal limit as well as on vertical limit. If it exceed the range limit then it will show error that the horizontal or vertical motion vector is out of allowed range.

5.3.6 Weighted sample prediction process

For macroblocks or partitions with prediction Flag 0 equal to 1 in P and SP slices, the following applies.

- If weighted prediction flag is equal to 0, the default weighted sample prediction process is invoked with the same inputs and outputs
- Otherwise (weighted prediction flag is equal to 1), the explicit weighted prediction process is invoked with the same inputs and outputs

For macroblocks or partitions with prediction Flag 0 or prediction Flag 1 equal to 1 in B slices, the following applies.

- If weighted bi-prediction idc is equal to 0, the default weighted sample prediction process is invoked with the same inputs and outputs as the process.
- Otherwise, if weighted bi-prediction idc is equal to 1, the explicit weighted sample prediction process for macroblocks or partitions with prediction flag 0 or prediction flag 1 equal to 1 with the same inputs and outputs as the process

Weighted bi-prediction is a method for decoding video signal data for an image block, receiving a substantially uncompressed image block; assigning a weighting factor for the image block corresponding to a particular reference picture; weighting the reference picture by the weighting factor; computing motion vectors corresponding to the difference between the image block and the weighted reference picture; and motion compensating the weighted reference picture in correspondence with the motion vectors.

When weighted biprediction idc is equal to 2, the following applies.
- If prediction Flag 0 is equal to 1 and prediction Flag 1 is equal to 1, the implicit weighted sample prediction is invoked with the same inputs and outputs as the process.
- Otherwise (predFlagL0 or predFlagL1 are equal to 1 but not both), the default weighted sample prediction process is invoked with the same inputs and outputs.

For computing motion vectors testing within a search region for every displacement within a pre-determined range of offsets relative to the image block; calculating at least one of the sum of the absolute difference and the mean squared error of each pixel in the image block with a first motion compensated reference picture corresponding to the first predictor.

Selecting an offset with the lowest sum of the absolute difference and mean squared error as the motion vector for the first predictor; calculating at least one of the sum of the absolute difference and the mean squared error of each pixel in the image block with a second motion compensated reference picture corresponding to the second predictor; and selecting an offset with the lowest sum of the absolute difference and mean squared error as the motion vector for the second predictor.

And free prediction memory will free the memory allocated to the luma and chroma blocks and frees memory for next block.

Chapter 6

Experimental Result and Analysis

6.1 Result

H.264 Encoder will use the yuv file as a input that generate a Bit-stream file which is encoded bit-stream and a test_dec.yuv which is a reference file to remove any error during decoding. Here 'sample.yuv' with 880 frames size of 352 X 288 and given to the H.264 Encoder to generate the bit-stream for the decoder.

When the generated bit-stream by encoder is givenas a input to Decoder it will generate following output.

———————— JM 15.1 (FREXt)————————

Decoder config file : (decoder.cfg)

Input H.264 bitstream : test.264

Output decoded YUV : test_dec.yuv

Output status file : log.dec

Input reference file : test_rec.yuv

Finally it show the SNR ratio for y,u and v component.

——————— Average SNR all frames ———————————

SNR Y(dB) : 37.77

SNR U(dB) : 37.55

SNR V(dB) : 36.99

Total decoding time : 16.419 sec (53.613 fps)

———————————————————————————————

Exit JM 15 (FRExt) decoder, ver 15.1

The Bit-stream generated by the encoder is given to the Decoder on the SimitARM for execution of the decoder on to the ARM processor which will give following error.

Total user time	192.308 sec.
Total system time	1.436 sec.
Simulation user time	166.638 sec.
Simulation system time	0.492 sec.
Caching user time	25.670 sec.
Caching system time	0.944 sec.
Simulation speed	2.960e+08 inst/sec.
Total instructions	57352355495 (57G), including 98.88% compiled
Requested blocks	52.
Cache-hit count	31.

Table I: Output of SimitARM

6.1.1 PSNR ratio

The phrase peak signal-to-noise ratio, often abbreviated PSNR, is an engineering term for the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. The PSNR is most commonly used as a measure of quality of reconstruction of lossy compression codecs. The signal in this case is the original data, and the noise is the error introduced by

PSNR	**Before Macroblock Partition**	**After Macroblock Partition**
PSNR Y	37.77	38.47
PSNR U	37.35	38.23
PSNR V	36.99	37.16

Table II: Comparison of PSNR ratio

compression. When comparing compression codecs it is used as an approximation to human perception of reconstruction quality, therefore in some cases one reconstruction may appear to be closer to the original than another, even though it has a lower PSNR (a higher PSNR would normally indicate that the reconstruction is of higher quality).

$$PSNR = 10 * log_{10}(\frac{MAX_I^2}{MSE}) = 20 * log_{10}(\frac{MAX_I}{\sqrt{MSE}}) \tag{6.1}$$

Using Macro block level paratition in the Motion Compensation It can be seen from Table II, the PSNR ratio of Y,U and V component improves by using Macro Block partition

Chapter 7

Conclusion and Future Scope

7.1 Conclusion

It is envisaged that analysis of the computation requirement for H.264 decoder is carried out which shows that the motion compensation is required more then half of the total Computation to decode the bit-stream. So Modification of motion compensation part is done using the Macro-Block level partition.

For simulation environment Scratchbox is used for cross compile the decoder in to the ARM binary files and the generated binary file will be executed on SimitARM which will simulate the decoder on the ARM processor.

Present result shows that using the concept of Macro-block level partition can improved the execution time and PSNR ratio.But as the number of instruction is increased in the decoder the complexity of the Decoder is also increased. So to overcome the complexity the OMAP architecture is useful to make execution more faster.

7.2 Future Scope

The result show that the number of instruction is increased due to the MacroBlock partition the complexity is improved.As OMAP provides the Heterogenous Environment for ARM and DSP platform and which provides communication between ARM and DSP,in future the H.264 decoder using macroblock partition can be implement on the OMAP architecture.OMAP also provide dedicated 2D/3D graphics hardware accelerator which will helpful to implement general processes of the decoder. Thus Using of execution power of two processor ARM and DSP with Hardware Accelerator the execution can be faster.

Appendix A

Open Multimedia Application Platform

The OMAP architecture is based on a combination of TI's state-of-the-art TMS320C55x DSP core and high performance ARM925T CPU. A RISC architecture, like ARM925T, is well suited for control type code (Operating System (OS), User Interface, OS applications). A DSP is best suited for signal processing applications, such as MPEG4 video, speech recognition, and audio playback. The OMAP architecture combines two processors to gain maximum benefits from each. TI conducted a comparative benchmarking study, based on published data, which shows that a typical signal processing task executed on the latest RISC machine (StrongARM, ARM9E) requires three times as many cycles as the same task requires on a C55x DSP.

For instance, a single C55x DSP can process, in realtime, a full videoconferencing application (audio and video at 15 images/sec.), using only 40 percent of the available computational capability. Therefore, 60 percent of the capacity can be employed to run other applications concurrently. At the same time, in the OMAP dual-core architecture, the ARM processor stands ready to handle any other application requirements or can be suspended, thus saving battery life. As a result, the mobile user can enjoy access to popular OS applications (Word, Excel, etc.) while also engaging

a videoconferencing application.

The OMAP3430 processor is the first processor in the industry to integrate the superscalar ARM Cortex-A8 core, the newest generation of ARM RISC processors. With features such as deeper pipelines, a dedicated level-2 cache and execution of up to twice as many instructions per clock cycle, the new high-performance ARM processor provides the processing power to support high-quality productivity applications and faster user interfaces. Mobile workers will be able to access databases, work on spreadsheets and presentations, use email, send instant messages, browse and download from the Web, attend videoconferences and play audio-video clips faster. Outstanding gaming capabilities will also be possible, thanks to ARMs integrated vector floating-point acceleration working with the OMAP3430s dedicated 2D/3D graphics hardware accelerator.

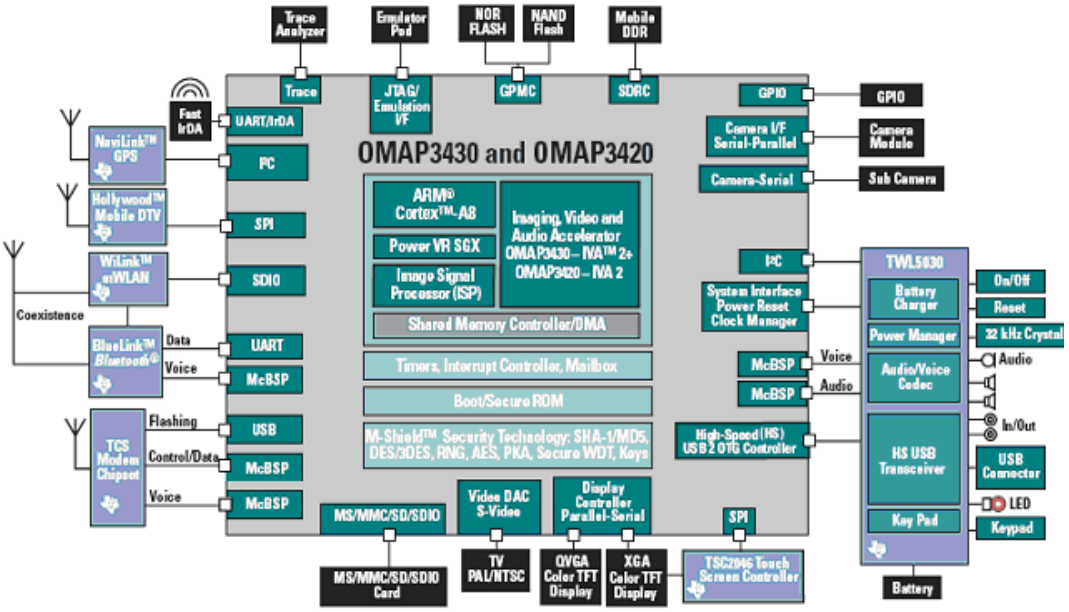

Figure A.1: OMAP Architecture

In addition, the OMAP3430 processor integrates the advanced IVA 2+ acceleration with new support for critical-coded functions. A second-generation, powerop-

timized version of the imaging, video and audio accelerator used in TIs DaVinci technology, IVA 2+ improves multimedia processing up to 4X from previous OMAP processors. The increased capabilities of the IVA2+ enable multi-standard (MPEG4, H.264, Windows Media Video, RealVideo, H.263, etc.) encode and decode at DVD resolution. With the advanced multimedia capabilities of the OMAP3430, a multistandard DVD-quality camcorder can be added to a phone for the first time. In addition, the IVA2+ advances video teleconferencing by providing H.264-based video at greater than CIF resolutions.

A.0.1 How the Architecture Works

The OMAP hardware architecture, shown in Figure 1, is designed to maximize the overall system performance of a 3G terminal while minimizing power consumption. This is achieved through the use of a C55x DSP core and an ARM925T CPU. Both processors utilize an instruction cache to reduce the average access time to instruction memory and eliminate power-hungry external accesses. In addition, both cores have a memory management unit (MMU) for virtual-to-physical memory translation and task-to-task memory protection.

The OMAP core contains two external memory interfaces and one internal memory port. The external memory interfaces support direct connection to synchronous DRAMs at up to 100 MHz and to standard asynchronous memories, such as SRAM, FLASH, or burst FLASH devices. The latter interface is typically used for program storage. It can be configured as 16 or 32 bits wide. The internal memory port allows direct connection to on-chip memory, such as SRAM, and can be used for frequently-accessed data, such as critical OS routines or the liquid crystal display (LCD) frame buffer.

This reduces the access time and eliminates costly external accesses. All three interfaces are completely independent and allow concurrent access from either processor

or direct memory access (DMA) unit.

The OMAP core also contains numerous interfaces to connect to peripherals or external devices from either the DSP or GPP. To improve system efficiency, these interfaces also support DMA from each respective processor's DMA unit. The local bus interface is a highspeed, bidirectional, multimaster bus that can be used to connect to external peripherals or additional OMAPbased devices in a multicore product. Additionally, a high-speed access bus is available to allow an external device to share the main OMAP system memory (SDRAM, FLASH, internal memory). This interface provides an efficient mechanism for data communication and also allows the designer to reduce system cost by reducing the number of external memories required in the system.

the OMAP architecture includes several peripheralstimers, general purpose input/output interfaces (I/Os), a UART, and watchdog timers. These are the minimum peripherals required in the system; other peripherals can be added on the TI peripheral bus (TIPB) interfaces. A color LCD controller is also included to support a direct connection to the LCD panel. The ARM DMA engine contains a dedicated channel that is used to transfer data from the frame buffer to the LCD controller, where the frame buffer can be allocated in the SDRAM or internal SRAM.

Appendix B

List of websites

- H.264 Decoder JM Reference Software
  http://iphome.hhi.de/suehring/tml/
- SimIt-ARM simulator
  http://simit-arm.sourceforge.net/
- Scratch-box Cross compiler
  http://www.scratchbox.org/
- Open Multimedia Application Platform(OMAP)
  http://focus.ti.com/omap/docs/omaphomepage.tsp
- H.264 / Advanced Video Coding (AVC) Standard
  http://www.vcodex.com/h264.html

References

[1] S.-W. Lee and C.-C. Kuo, "Complexity modeling of h.264/avccavlc/uvlc entropy decoders," *Ming Hsieh Department of Electrical Engineering and Signal and Image Processing*, MAY 2006.

[2] L. E. Z. X. G. S. Chen, Y., "Implementation of h. 264 encoder and decoder on personal computers.," *Journal of Visual Communications and Image Representation 17*, October 2006.

[3] G. A. M. M. Rodriguez, A., "Hierarchical parallelization of an h.264/avc video encoder," *Proc. Int. Symp. on Parallel Computing in Electrical Engineering*, 2006.

[4] M. Roitzsch, "Slice-balancing h.264 video encoding for improved scalability of multicore decoding.," *27th IEEE Real-Time Systems Symposium.*, 2006.

[5] S. N. Kenneth Vermeirsch, Jan De Cock and P. Lambert, "Macroblock bipartitioning modes for h.264/avc inter coding," *Department of Electronics and Information Systems Multimedia Lab.*

[6] E. G. T. J. E. B. van der Tol and R. H. Gelderblom, "Mapping of h.264 decoding on a multiprocessor architecture," tech. rep., Proc. of SPIE Vol. 5022, 2003.

[7] JVT, "Draft itu-t recommendation and final draft international standard of joint video specification," *ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC*, MAY 2003.

[8] S. W. Lee and C.-C. J. Kuo, "Complexity modeling for motion compensation in h.264/avc decoder," *IEEE Int. Conf. on Image Processing*, MAY 2007.

[9] H. Velayos and G. Karlsson, "Techniques to reduce ieee 802.11b mac layer handover time," tech. rep., Royal Institute of Technology, 2003.