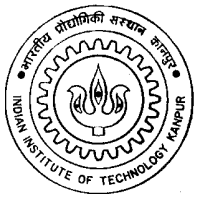
Implementation of MPEG-2 Video Decoder using an associative array processor

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by

Samarpit Bhatia



to the

Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur

May, 2000

Certificate

This is to certify that the work contained in the thesis entitled "*Implementation of MPEG-2 Video Decoder using an associative array processor*", by *Samarpit Bhatia*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

May, 2000

(Dr. Rajat Moona)
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

Acknowledgements

I take this opportunity to express my sincere and deep felt gratitude towards my supervisor Dr. Rajat Moona for his invaluable guidance throughout this work. He has been an immense source of motivation and inspiration. I thank him for giving me the opportunity of working under his guidance.

I would like to thank Neomagic Corporation, California, US , for providing me technical resources and financial support. I would especially like to thank its Vice-president, Engineering Dr. Clement Leung, for his valuable suggestions and guidance throughout this work.

My whole hearted thanks to the faculty members of Computer Science and Engineering department for imparting me with the invaluable knowledge that has brought me to this competent stage.

I have enjoyed the company of all my batch mates especially venky, dharmesh, ghulam, pogde, rahul, unmesh and sarika. I thank venky for his valuable suggestions in my thesis work.

My parents, my sisters and my fiance Rachna have been a constant source of love and affection throughout. I am eternally grateful to them for always being with me whenever I needed them.

Abstract

Emerging multimedia applications, such as digital versatile disk and high definition television, demand higher quality video than ever before. With increased market acceptance of such applications, the Motion Picture Expert Groups' (MPEG) MPEG-2 standard is being widely used. There arises a need for fast and efficient video decoder that can decode MPEG-2 bitstream in real time without compromising on video quality or overloading the system.

In our current work, we have developed a co-hardware/software video decoder model for MPEG-2 mainprofile@mainlevel video bitstreams. The decoder uses an associative array processor - Associative Real Time Vision Machine (ARTVM), for inverse discrete cosine transform (IDCT) computation. The IDCT computation is done in parallel for all the blocks of a single frame. The simulation results show that the proposed decoder model meets the real time decoding requirements very easily.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Digital video is basically collection of related digital frames displayed at a rate of about 25 frames per second. Since its inception digital video has introduced many new technologies. Digital versatile disk (DVD), high definition television (HDTV), video conferencing, video on demand are a few of digital video applications. Digital video provides much more than better picture quality. Ease of manipulation, lossless duplication and transmission are a few of the advantages of digital video.

Digital frames are basically 2-D array of pixels. Each pixel holds color information of a particular small region of the picture. Digital video is usually coded along with digital audio as there are very few application which use digital video alone. Although digitization of video & audio has led to many new applications, digital video & audio in uncompressed form results in enormous amount of data that cannot be handled by present network bandwidths and storage capacities available. Hence a need for compression arises. In fact video and audio data have so much redundancies in it that using them in uncompressed format is a waste of resources. There have been many standards made for encoding of digital video & audio in an efficient format [16].

MPEG family of standards [12] for motion pictures is one of the widely accepted standards. Various versions of MPEG standard have been developed taking in consideration the fact that encoding is usually one time job and decoding needs to be done at many times, hence decoding process should be simpler. Several software and

hardware MPEG decoders ([15, 14, 8, 9]) have been developed so far. The complexity and computational requirements of MPEG decoding process makes the software decoders unsuccessful, as even on very fast processors MPEG decoding is barely real time and it slows down the entire system. Dedicated hardware MPEG decoders have the disadvantage of becoming obsolete as new standards emerge. Further they are not cost effective as they can not be used in applications other than the MPEG decoding.

In this thesis we have simulated a co-hardware/software video decoder model for Main Profile at Main Level (MP@ML) MPEG-2 coded motion pictures. The proposed decoder model uses the processor only for bit-stream parsing and frame display, the computationally most intensive part of Inverse Discrete Cosine Transform (IDCT) is done through an associative array processor called ARTVM (Associative Real Time Vision Machine) [4].

1.2 Video Coding Standards

Table 1.1 gives an overview of a few families of digital standards [16]. The Group name is listed in the left column. Some of the standards in the family are listed by number in the center column and the purpose of each standard is shown in the comments column. This table illustrates how multiple standards are needed to create a complete video application.

These standards have been developed for different specific applications. $p \times 64$ family is a digital teleconferencing standard that operates in the range of 64 kbits/s to 2 Mbits/s. The low-bitrate communication (LBC) teleconferencing family is newer than $p \times 64$ family. It operates at 64 kbits/s or below. Among the various video standards, MPEG is quiet versatile. It has been developed for a very broad range of applications and it is gaining more and more acceptance everyday.

The acronym MPEG stands for Moving Picture Expert Group, which worked to generate the specifications under ISO, the International Organization for Standardization and IEC, the International Electrotechnical Commission. What is commonly referred to as "MPEG video" actually consists of two finalized standards, MPEG-1 and MPEG-2, with a third standard, MPEG-4, in the process of being finalized. The MPEG-1 & -2 standards are similar in basic concepts. They both are based on motion compensated block-based transform coding techniques, while MPEG-4 deviates from

Group name	standards	Comments
$p \times 64$	H.261	Video
	H.221	Communications
	H.230	Initial handshake
	H.320	Terminal systems
	H.242	Control protocol
	G.711	Companded audio (64 kbits/s)
LBC	H.263	Video
	H.324	Terminal Systems
	H.245	Control protocol
	H.223	Multiplexing protocol
	G.723	Speech
MPEG-1	ISO/IEC 11172-1	Systems
	ISO/IEC 11172-2	Video
	ISO/IEC 11172-3	Audio
	ISO/IEC 11172-4	Conformance testing
	ISO/IEC 11172-5	Software simulation
MPEG-2	ISO/IEC 13818-1	Systems
	ISO/IEC 13818-2	Video
	ISO/IEC 13818-3	Audio
	ISO/IEC 13818-4	Compliance testing
	ISO/IEC 13818-5	Software simulation
	ISO/IEC 13818-6	DSM-CC
	ISO/IEC 13818-7	NBC Audio
	ISO/IEC 13818-8	10-Bit Video
	ISO/IEC 13818-9	RTI

Table 1.1: Digital video standards

these more traditional approaches in its usage of software image construct descriptors, for target bit-rates in the very low range, less than 64Kb/sec.

1.2.1 MPEG-1

MPEG-1 was finalized in 1991, and was originally optimized to work at video resolutions of 352x240 pixels at 30 frames/sec (NTSC based) or 352x288 pixels at 25 frames/sec (PAL based). It is often mistakenly thought that the MPEG-1 resolution is limited to the above sizes, but it in fact may go as high as 4095x4095 at 60 frames/sec. The target bit-rate is around 1.5 Mb/sec, but again can be used at higher rates if required. MPEG-1 is defined for progressive frames only, and has no direct provision

for interlaced video applications, such as in broadcast television applications.

1.2.2 MPEG-2

MPEG-2 was finalized in 1994, and addressed issues directly related to digital television broadcasting, such as the efficient coding of field-interlaced video and scalability. Also, the target bit-rate was raised to between 4 and 9 Mb/sec, resulting in potentially very high quality video. MPEG-2 consists of profiles and levels. The profile defines the bit-stream scalability and the colors-pace resolution, while the level defines the image resolution and the maximum bit-rate per profile. The most common descriptor in use currently is Main Profile, Main Level (MP@ML) which refers to up to 720x576 resolution video at 30 frames/sec, at bit-rates up to 15 Mb/sec.

MPEG-2 defines five different profiles, namely Simple profile (SP), Main profile (MP), SNR-scalable profile (SNR), spatially-scalable profile (Spt) and High profile (HP). Four levels are defined in MPEG-2 : low (LL), main (ML), high-1440 (H-14), and high (HL). Of the five profiles and four levels, creating 20 possible combinations, 11 have been defined. The Main profile allows all four levels, the Simple and the Spatial profiles allow one level each, the SNR profile allows two levels, and the High profile allows three levels. For the allowed profile and level combinations the upper bounds on coding rate and picture size parameters are provided in table 1.2 below.

Levels	Profiles				
	SIMPLE nonscalable 4:2:0	MAIN nonscalable 4:2:0	SNR scalable 4:2:0	SPATIAL scalable 4:2:0	HIGH nonscalable 4:2:2
HIGH		1920×1152 80 Mb/s			1920×1152 100 Mb/s
HIGH-1440		1440×1152 60 Mb/s		1440×1152 60 Mb/s	1440×1152 80 Mb/s
MAIN	720×576 15 Mb/s	720×576 15 Mb/s	720×576 15 Mb/s		1440×1152 20 Mb/s
LOW		352×288 4 Mb/s	352×288 4 Mb/s		

Table 1.2: MPEG-2 profile and level structure with parameter upper bounds

1.3 Compression in MPEG

MPEG-1 & -2 are inter- and intra-frame compression system. Frames are coded in one of the three formats namely I-,P- and B-frames. I-frames are intra-coded, i.e. they only remove the spatial redundancies within the frame and serve as a starting point in case of out of order access or transmission errors. P- and B-frames are inter-coded, i.e. they exploit the temporal redundancies also. Spatial redundancies are removed through the use of Discrete Cosine Transform (DCT). DCT with other encoding algorithms gives a very high compression. Temporal redundancies are removed through the use of motion compensation (MC).

1.4 Discrete Cosine Transform

In general, neighboring pixels within an image tend to be highly correlated. As such, it is desired to use an invertible transform to concentrate randomness into fewer, decorrelated parameters. The Discrete Cosine Transform (DCT) has been shown to be near optimal for a large class of images in energy concentration and decorrelating. The DCT decomposes the signal into underlying spatial frequencies, which then allow further processing techniques to reduce the precision of the DCT coefficients consistent with the human visual system model.

The DCT doesn't directly affect compression. In fact for an 8×8 block of 8 bit pixels, the DCT produces an 8×8 block of 11 bit coefficients. The reduction in the number of bits follows from the observation that, for typical blocks from natural images, the distribution of coefficients is non-uniform. The transform tends to concentrate the energy into the low-frequency coefficients and many of the other coefficients are near-zero. The bit rate reduction is achieved by not transmitting the near-zero coefficients and by quantizing and coding the remaining coefficients

1.4.1 Definition of DCT

The $N \times N$ cosine transform matrix C = c(k,n), called *discrete cosine transform*, is defined as

$$c(k,n) = \begin{cases} \frac{1}{\sqrt{N}}, & k = 0,\ 0 \le n \le N-1 \\ \frac{2}{\sqrt{N}} cos\frac{(2n+1)k}{2N}\pi, & 0 \le k \le N-1,\ 0 \le n \le N-1 \end{cases}$$

For a given 2-d data sequence $\{x_{ij} : 0 < i, j < N-1\}$, the 2-d DCT sequence $\{Y_{mn} : 0 < m, n < N-1\}$ is given by the following [2].

$$Y_{mn} = \frac{4}{N^2} u(m)u(n) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x_{ij}\, cos\frac{(2i+1)m}{2N}\pi\, cos\frac{(2j+1)n}{2N}\pi \tag{1.1}$$

and the inverse DCT (IDCT) is given by

$$x_{ij} = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} u(m)u(n)Y_{mn}\, cos\frac{(2i+1)m}{2N}\pi\, cos\frac{(2j+1)n}{2N}\pi \tag{1.2}$$

where

$$u(m) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if m} = 0 \\ 1, & \text{otherwise} \end{cases}$$

The 2-dimensional DCT operation for an 8×8 pixel block generates an 8×8 block of coefficients that represent a "weighting" value for each of the 64 orthogonal basis patterns that are added together to produce the original image.

1.5 Related work

There have been many software as well as hardware approaches for processing of multimedia applications, especially MPEG decoding. Many public domain software decoders for MPEG-1 are available on the net [15], there are many company proprietary software decoders also available [14], proprietary decoders are a bit faster as they usually don't implement the actual Inverse Discrete Cosine Transform (IDCT). Instead they make the decoding faster by making some high frequency components equal to zero and computing the IDCT.

In hardware approaches, there have been designs for fast IDCT/DCT processors [5, 7, 6, 19], full MPEG decoding chips [8, 9], special multimedia instruction

enhancements for general purpose processors (like MMX) and designs of special DSP architectures for multimedia applications [13].

1.6 Current Work

In the current work, we have simulated a co-hardware/software MPEG-2 video decoder for MP@ML. For simulating the IDCT algorithm we have written a simulator for the associative array processor ARTVM. The video decoder uses ARTVM for computation of Inverse Discrete Cosine Transform (IDCT) which is the most computationally intensive part of MPEG video decoding. The bit-stream parsing and frame display is done by the host processor. A parallel IDCT algorithm has been implemented on ARTVM such that the IDCT can be performed on all the blocks which fit in the associative memory in parallel. Hence if ARTVM memory size is sufficient we need to perform IDCT only once per frame.

Total number of ARTVM clock cycles required by our parallel IDCT algorithm is 149197. At 33 Mhz ARTVM processor speed, time required for IDCT is 4.5 ms. The simulator predictions for total time requirement meets the real time decoding requirements very easily. The simulator can handle MPEG bit-streams with or without audio stream but if audio data is present it is discard.

1.7 Organization of this report

The rest of this report is organized as follows. In chapter 2, we describe the associative array processor ARTVM architecture and its instruction set. In chapter 3, we provide a small overview of MPEG-2 video bit-stream structure. In chapter 4, we present the implementation details of ARTVM simulator, discuss design issues involved in designing the IDCT algorithm for ARTVM and provide details of incorporating the parallel IDCT routine with an existing video decoder. In chapter 5, we conclude this thesis and provide test setup, results, conclusions, limitations and scope of future work.

Chapter 2

Associative Array Processor ARTVM

In our work we used an associative array processor developed by Dr. Avidan J. Akerib [3]. In the following sections we describe the processor's architecture and explain its instruction set.

2.1 Overview

There are basically two types of architectural organizations for SIMD processors namely array processors and associative processors. Array processors use random-access memory whereas associative processors use content-addressable memory. Content addressable memory or associative memory includes comparison logic with each bit of storage. A data value is broadcast to all words of storage and compared with the values there. Words which match are flagged in some way. Subsequent operations can then work on the flagged words, e.g. read them out one at a time or write to certain bit positions in all of them.

Associative processors are basically a special type of array processors whose processing elements (PEs) correspond to the words of an associative memory. These PEs operate on each bit cell of every word of the associative memory in parallel and generate the output corresponding to each word. The associative array processor ARTVM used in our implementation of parallel IDCT is based on this associative approach.

2.2 ARTVM Architecture

In ARTVM a word of memory is assigned to each pixel of the image, hence vision algorithms work on all the pixels of an image in parallel. Figure 2.1 describes the core of machine which is a classical associative processor. ARTVM is a fully parallel associative processor. It consists of an associative memory A, Mask and Comparand registers of length equal to associative memory word length ,a Tag register whose size is equal to the associative memory size and an I/O Buffer Array with 16 bit shift register attached to each word. ARTVM provides one words for each pixel in the image to be processed. The pixels are arranged linearly, in the row major order. The machine instruction set is given in table 2.1.

I/O buffer array is used for loading data in to associative memory and for taking out the results. It operates without any intervention from the associative processing. Data can be loaded or read from the I/O buffer array through the I/O bus one word in each ARTVM clock cycle.

Transfer of data from I/O buffer to the associative memory (and vice-versa) is done in parallel through the use of an instruction called TAGXCH. TAGXCH instruction transfers the content of I/O buffer to the tag register one bit slice at a time. Basically the contents of the I/O buffer are circulating right via the tag register.

2.2.1 Operations on associative memory

ARTVM provides a few very basic but very powerful set of instructions on the associative memory. These instructions are capable of executing any logical and arithmetic function. Further the inter connection network between the PEs is very simple as data transfer among the different PEs is possible only through the tag register.

The two most important instructions are COMPARE and WRITE. In COMPARE instruction, comparand register is matched against all words of memory simultaneously and the match is indicated by setting the corresponding tag bits. The comparison is carried out only for the bits indicated by the mask register. Status bit rsp is set to indicate that there was at-least one match (fig 2.2). The WRITE instruction operates in a similar manner. The contents of the comparand are simultaneously written into the words indicated by the tag register. These bits are written only on

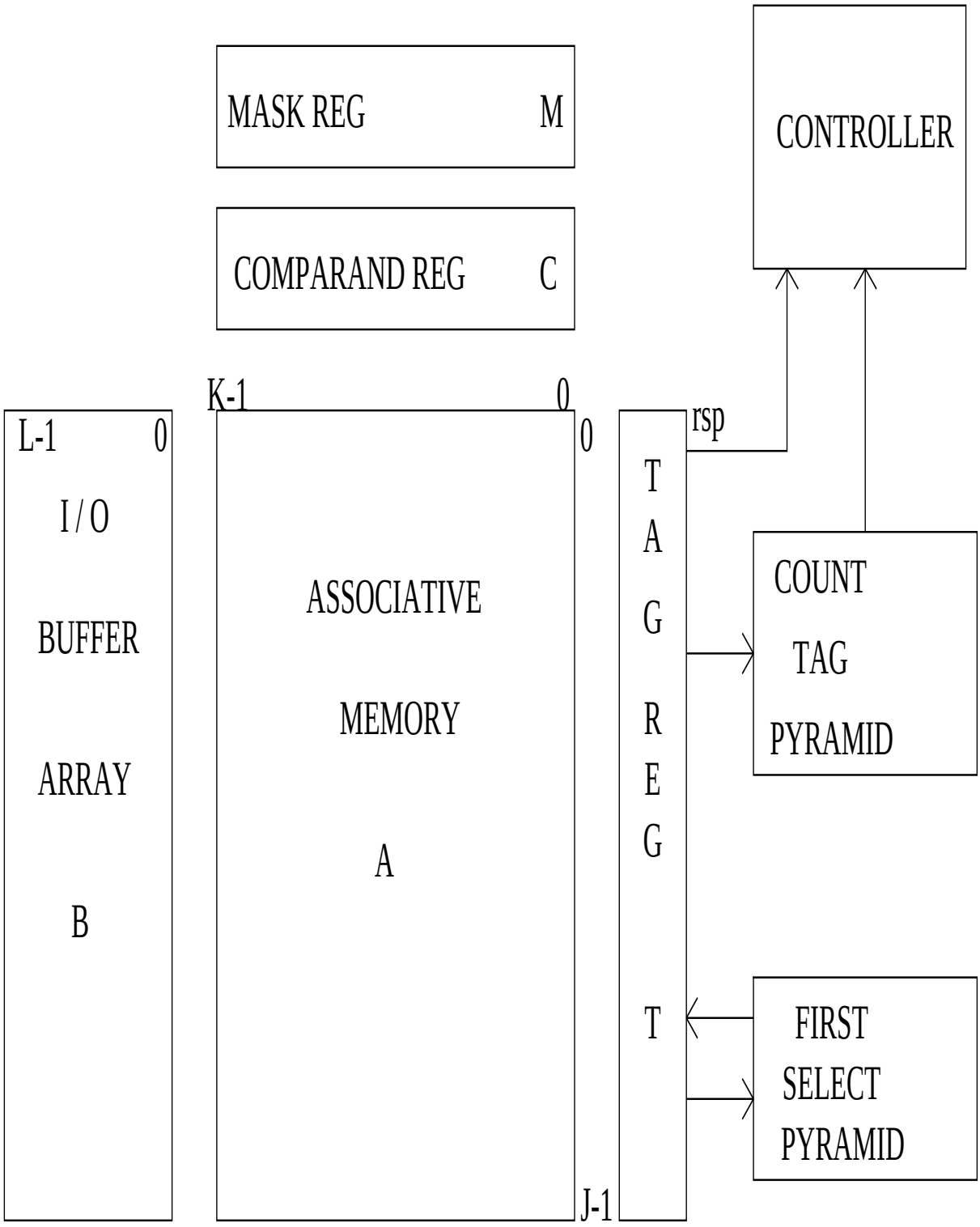


Figure 2.1: The Associative Real Time Vision Machine Architecture

those bit locations that are indicated by the mask register (fig 2.2). The combination COMPARE - WRITE is of type "if condition then action", hence all logical and arithmetic functions can be executed.

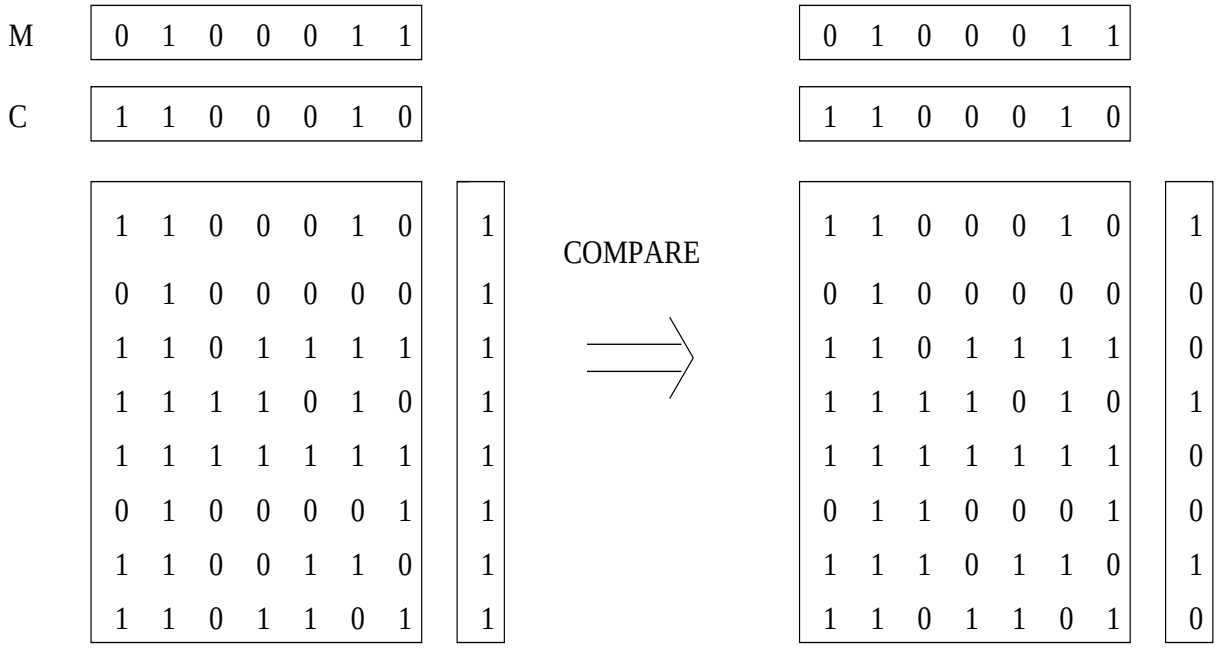
The SETM, SETC, SETMC instructions are used to set the mask and comparand registers. LETM, LETC, LETMC instructions are used for operating on specific bits of the mask and comparand registers. The READ instruction is normally used to bring out a single word, the one pointed to by the tag. Only those bits are read as indicated by the mask register. Data communication among the neighboring pixels is possible only through the tag register, a bit slice at a time, using SHIFTAG instruction. The number of shifts applied to the tag register determines the distance or relation between source and destination. Usually this relation is uniform in image processing applications , hence communication between all processor is simultaneous and the processing is in parallel.

FIRSEL instruction is used to reset the entire tag register excluding the first bit. Another instruction COUNTAG is used to return the number of bits set in the tag register.

Each word in the associative memory acts as a simple processor so that memory and processor are indistinguishable. Input, output and processing go on simultaneously in different fields of the same word. The field to be accessed or processed, is flexible, and can be selected by the application. Hence the processing capabilities (image processing algorithms) can be easily expanded by increasing word length K. It has been shown that for K = 152 almost all the vision algorithms (such as histogram evaluation, convolution, morphological operations etc.) run in real time.

2.3 ARTVM configuration

The designers of ARTVM chose the image size to be 512×512 after analyzing various vision algorithms and real time processing requirements. Thus the associative memory capacity is 256K words (one word per pixel). The word length was fixed at 136 (four 32 - bit sectors and an 8 - bit flag).



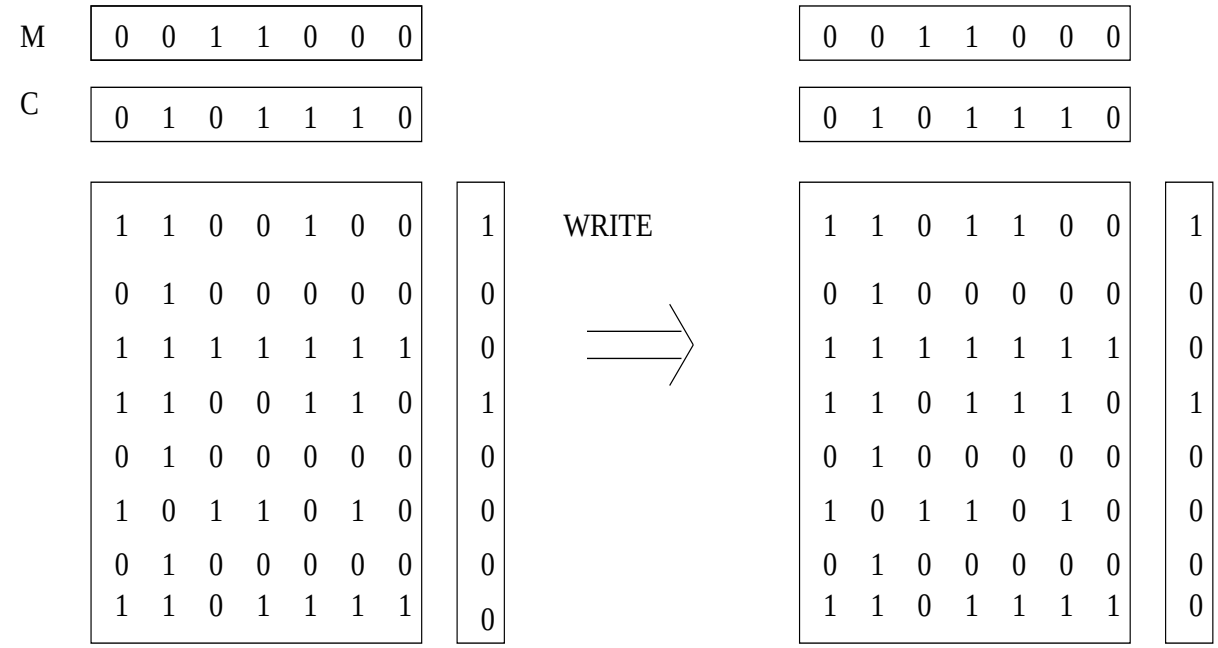


Figure 2.2: Illustration of COMPARE and WRITE instructions, showing condition before and after execution.

ARTVM Instruction Set

The instructions given below are for

$\forall\ j = 0,1,2,...,\text{J-1} \quad \forall\ k = 0,1,2,...,\text{K-1} \quad \text{and} \quad \forall\ l = 1,2,3,...,\text{L-1}$

The symbols used in the instruction have the following meaning.

T $\longrightarrow$ Tag

B $\longrightarrow$ I/O buffer array

A $\longrightarrow$ Associative memory

C $\longrightarrow$ Comparand

M $\longrightarrow$ Mask

Math symbols:

$\bigvee \longrightarrow$ *or*

$\bigoplus \longrightarrow$ *exclusive-or*

$\bigwedge \longrightarrow$ *and*

Tag Operation

SETAG:	$T_j \leftarrow 1$	
SHIFTAG:	$T_{j\pm1} \leftarrow T_j$	(One-Place Shift)
SHIFTAG:	$T_{j\pm b} \leftarrow T_j$	(Long Shift)
TAGXCH:	$T_j \leftarrow B_{j0}$	$B_{j(l-1)} \leftarrow B_{jl}, B_{j(l-1)} \leftarrow T_j$
COUNTAG:	S $\leftarrow \sum_j T_j$	

FIRSEL: $T_j \leftarrow \begin{cases} 1 & \text{if } T_j \text{ is the first "1" of } T \\ 0 & \text{otherwise} \end{cases}$

Comparand and Mask Initialization

Let X denote any one of the following:
M(*Mask*), C(*Comparand*), MC(*Mask and Comparand*)

SETX: $X_k \leftarrow 1$
LETX: *opt1 opt2 opt3*

where:

$opt1 = d(r_1), d(r_2), \ldots d(r_s)$

$opt2 = dseq(u_1, u_2)$

$opt3 = dvar(v_1, v_2, p)$

and options can be intermingled in any order. Then:

$$X_k \leftarrow \begin{cases} 1 & \forall k = r_1, r_2, \ldots r_s & \text{if } opt1 \text{ defined} \\ 1 & \forall k = u_1, u_1 + 1, \ldots u_2 & \text{if } opt2 \text{ defined} \\ p_{k-v_1} & \forall k \in \{v_1, v_2\} & \text{if } opt3 \text{ defined} \\ 0 & & \text{otherwise} \end{cases}$$

Compare Operation

$$\begin{array}{ll} \text{COMPARE:} & T_j \leftarrow T_j \bigwedge (\overline{\bigvee_k M_k \bigwedge (A_{jk} \bigoplus C_k)}) \\ & rsp \leftarrow \bigvee_j T_j \end{array}$$

Read Write Operations

$$\begin{array}{ll} \text{READ} & C_k \leftarrow \bigvee_j (A_{jk} \bigwedge T_j) \\ \text{WRITE} & A_{jk} \leftarrow (\overline{T}_j \bigwedge A_{jk}) \bigvee (T_j \bigwedge (M_k \bigwedge C_k \bigvee \overline{M}_k \bigwedge A_{jk})) \end{array}$$

Table 2.1: ARTVM instruction set

If the 16 - bit image buffer is also included, the total word length becomes 152. Loading an entire comparand or mask word would require 136 bits wide bus. However, associative algorithms operate only on one or two short fields and on a number of flag bits at a time. Hence buses are provided for simultaneous access to the flag field and one sector only. The cycle time used for timing analysis is taken to be 30 nanoseconds (33 Mhz).

The parallel IDCT algorithm that we have designed operates under this word length. For simulating the MPEG decoding process we didn't required such a high memory size, hence we have introduced the option of setting the memory size in the simulator we have designed.

Chapter 3

MPEG VIDEO CODEC

3.1 Introduction

Bringing video pictures into the digital format introduces one major problem. Uncompressed digital video pictures take up enormous amount of storage. For example a CD can hold only about five minutes of an uncompressed movie. A suitable compression algorithm such as MPEG can compress video data many times over, while still managing to retain very high picture quality.

MPEG-1, formally known as ISO/IEC 11172, is a standard in 5 parts. The first three parts represent Systems, Video and Audio, in that order. Two more parts in the standards for suite of MPEG-1 standard are Conformance Testing, which specifies the methodology of conformance verification, and Software Simulation.

The MPEG-2 (ISO/IEC 13818) concept is similar to MPEG-1, but includes extensions to cover a wider range of applications. It is designed for diverse applications. MPEG-2 is used in applications like digital high-definition TV (HDTV), interactive storage media (ISM), cable TV (CATV). MPEG-2 has bitstream scalability. Thus it is possible to extract a lower bitstream rate from the high quality image to get lower resolution or frame rate.

3.2 Compression Algorithms

3.2.1 Overview

In MPEG, video is represented as a sequence of pictures, and each picture is treated as a two-dimensional array of pixels (pels). The color of each pel consists of three components : Y (luminance), U and V (two chrominance components). In order to achieve high compression ratio, MPEG uses hybrid coding techniques to reduce both spatial redundancy and temporal redundancy. These techniques are described as the following.

- **Color Space Conversion and Subsampling of Chrominance.** In general, each pels in a picture consists of three components : R (Red), G (Green), B (Blue). However (R,G,B) must be converted to (Y,U,V) before they are processed. (Y,U,V) representation of pel has low correlation among the components as compared to the (R,G,B) components. Therefore (Y,U,V) components can be coded more efficiently. After color space conversion, each pels is represented as (Y,U,V) components. The human eye is most sensitive to the Y component. Therefore for a good quality picture it is necessary to encode Y component with high resolution. At the same time U and V components, can be stored less frequently by subsampling without compromising the picture quality.

- **Quantization.** Quantization is used to reduce a range of numbers to a single value, requiring fewer bits for representation. For example, we can round off a real number to an integer. Inverse Quantization is the reverse process to reconstruct original value. However the reconstructed value is not the same as the original value. The difference between the actual value and the reconstructed one is called the quantization error.

- **DCT (discrete cosine transform).** As described in section 1.3, DCT is very efficient in encoding natural (highly correlated) images. The DCT coefficient at location (0,0) is called the DC coefficient. Other coefficients represent the AC coefficients at various frequencies. In general, we use large quantization step in quantizing AC coefficients, and use small quantization step to quantize DC coefficient so as to preserve high precision. Below is an example of DCT coefficient array for the corresponding 8×8 pixel block.

88	84	83	84	85	86	83	82
86	82	82	83	82	83	83	81
82	82	84	87	87	87	81	84
81	86	87	89	82	82	84	87
81	84	83	87	85	89	80	81
81	85	85	86	81	89	81	85
82	81	86	83	86	89	81	84
88	88	90	84	85	88	88	81

An 8 x 8 Block

67.5	1	-6	2	-2	0	5	-5
-4	1	2	1	5	1	-3	0
2	3	4	6	-2	2	1	5
-3	-1	0	2	0	-2	2	-4
4	3	1	-1	-2	1	-3	1
1	-2	0	-3	2	-1	1	1
3	0	-1	0	-1	-1	0	-2
-1	-1	-5	5	2	-2	2	0

DCT Coefficients

- **Zig-Zag Scan And Run Length Encoding (RLE).** After DCT and quantization most AC coefficients are close to 0. In fact, higher the frequency, lower is the coefficient in most images. By using a zig-zag scan (figure 3.1) the consecutive zeros are brought together, and then RLE is used to gain compression ratio.

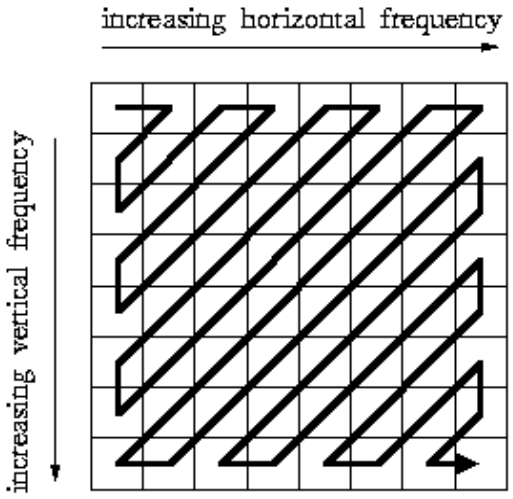


Figure 3.1: Scan Order

The bitstream after zigzag scan is encoded as (skip,value) pairs, where skip is the number of zeros and value is the next non-zero coefficient. Zig-zag scan and

RLE are used only for the AC coefficients. For DC coefficients, DPCM coding method is used as is described next.

- **Predictive Coding.** Predictive coding is a technique to reduce statistical redundancy. That is based on the current value, the next value can be predicted and the difference between the predicted one and the actual can be stored. If the prediction algorithm is good, the prediction error will be small. Thus fewer bits can be used to encode the prediction error. In MPEG, we use Differential Pulse Coded Modulation (DPCM) technique [11] for prediction and storage of errors.

- **Motion Compensation (MC) And Motion Estimation (ME).** Motion Estimation is used to predict a block of pels' value in next picture using a block in current picture. The location difference between these blocks is called the Motion Vector. Similarly the difference between two blocks is called the prediction error. Motion compensation is the inverse of motion estimation. The decoders implement the motion compensation while the encoder implement the motion estimation.

- **Variable Length Coding (VLC).** Variable Length Coding is a statistical coding technique. It uses short codeword to represent the value which occurs frequently and uses long codeword to represent the value which occurs less frequently. This method makes the overall data storage smaller than the original data storage. In MPEG, it is the last step in the encoding process and the first step in the decoding process.

3.3 Frame Coding

All MPEG frames are encoded in one of three different ways - Intra-coded (I-frames), Predictive-coded (P-frames), or Bidirectionally-predictive-coded (B-frames). I-frames are encoded as discrete frames, independent of adjacent frames. Thus, they provide randomly accessible points within the video stream. Because of this, I-frames have the least compression ratio of the three frames. P-frames are coded with respect to a past I-frame or P-frame, resulting in a smaller encoded frame size than that for the I-frames. The decoding of B-frames require a preceding frame and a future frame,

any of which may be either an I-frames or a P-frames. Thus the B-frames offer the highest degree of compression. In MPEG, frames are ordered together as Group of Pictures (GOP). Each GOP begins with an I-frame followed by arrangement of P and B frames. A typical sequence of frames is:

$$...P_0\ B_1\ B_2\ I_3\ B_4\ B_5\ P_6\ B_7\ B_8\ P_9\ B_{10}\ B_{11}\ P_{12}\ B_{13}\ B_{14}\ I_{15}...$$

In the above sequence a GOP in display order is:

$$B_1\ B_2\ I_3\ B_4\ B_5\ P_6\ B_7\ B_8\ P_9\ B_{10}\ B_{11}\ P_{12}$$

The corresponding encoding bit-stream order will be:

$$I_3\ B_1\ B_2\ P_6\ B_4\ B_5\ P_9\ B_7\ B_8\ P_{12}\ B_{10}\ B_{11}$$

3.3.1 Intra-frame Coding

I-frames are coded by intra-frame coding. When encoding I-frame, we only reduce the spatial redundancy in the picture without referencing other pictures. The coding process is much similar to JPEG Standard. Figure 3.2 describes the intra-frame coding process.

3.3.2 Inter-frame Coding

P-frames and B-frames are coded using inter-frame coding techniques. Coding of the P and B frames is more complex than that of the I frames, since motion-compensated macroblocks have to be searched in it. P-frames use the previous I- or P-frame for motion compensation and may be used as a reference for further prediction. Each block in a P-frame can either be predicted or intra-coded. By reducing spatial and temporal redundancy, P-frames offer increased compression compared to the I-frames.

B-frames are introduced for increasing the frame rate without increasing too

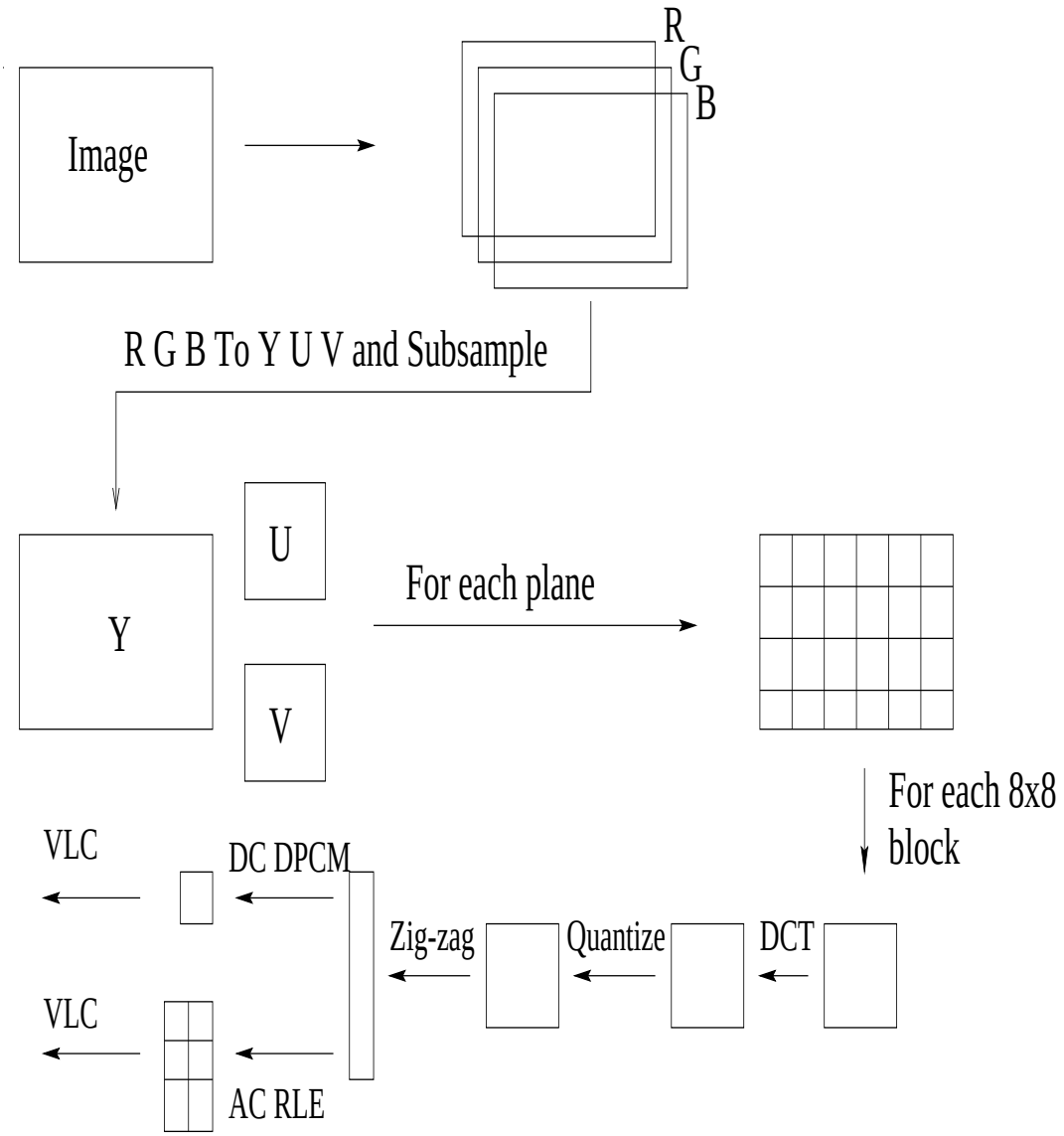


Figure 3.2: Intra-frame Coding

much bitrate. B-frames use the previous and the next I- or P-frames for motion-compensation, and offer the highest degree of compression. Each block in a B-frame can be forward, backward or bidirectionally predicted or intra-coded. To enable backward prediction from a future frame, the coder reorders the frames from natural 'display' order to 'bitstream' order so that a B-frame is transmitted after the previous and the next frames it references. This introduces a reordering delay dependent on the number of consecutive B-frames.

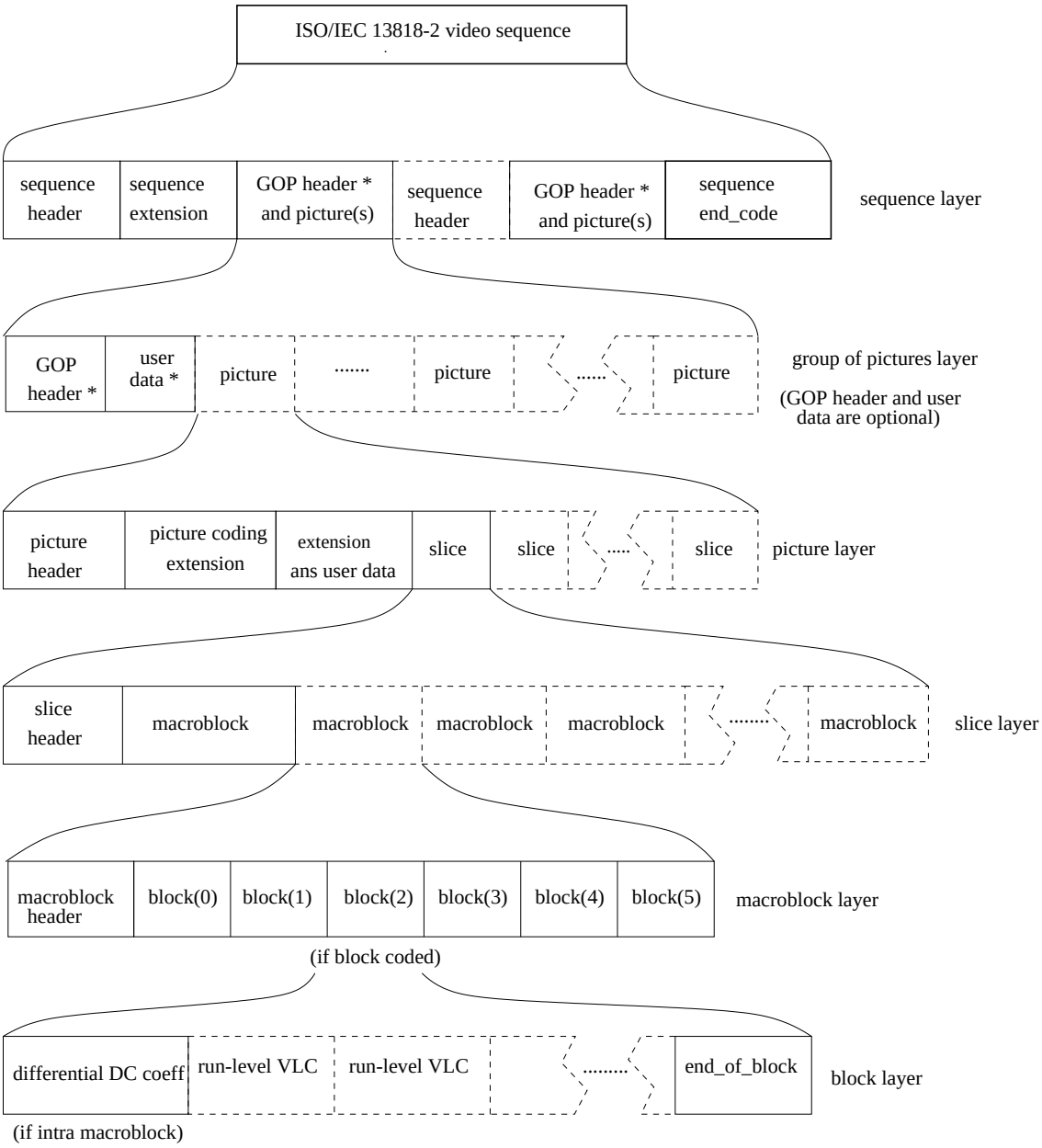


Figure 3.3: MPEG-2 main profile video syntax

3.4 Layered Structure

The syntax of a typical MPEG-2 main profile is given in a hierarchical representation with six layers in the figure 3.3. MPEG video is broken up into a hierarchy of layers to help with error handling, random search and editing, and synchronization for example with an audio bitstream. From the top level, the first layer is known as the sequence layer. It is a self-contained bitstream. For example it may contain a coded movie or an advertisement. The second layer represents the group of pictures. It is composed of one or more intra (I) frames and/or non-intra (P and/or B) frames. The third layer down is the picture layer that represents one of the frames. The next layer is called the slice layer. Each slice consists of macroblocks, which are 16x16 arrays of luminance pixels, and two 8x8 arrays of associated chrominance pixels. The macroblocks are further divided into distinct 8x8 block layers. Each of the layers has its own unique 32 bit start code defined in the syntax.

3.5 MPEG Video Decoder

The structure of a typical MPEG video decoder is shown in the figure 3.4. First the bit-stream is parsed and blocks are extracted before the inverse quantization. Inverse Discrete Cosine Transform (IDCT) is performed on these DCT coefficients. The DCT coefficients of a block are not always the DCT of actual pixel values as many of the MPEG-2 frames (P- and B-frames) are inter-coded to take the advantages of temporal redundancies between successive frames in a normal video sequence.

As shown in the figure 3.4, the IDCT coefficients are either directly sent for post processing for generation of frame or they are added with the form predictor to generate the actual frame data before being sent for post processing. The decoded frame is also stored for the generation of subsequent frames. At most two frames are stored as reference frames.

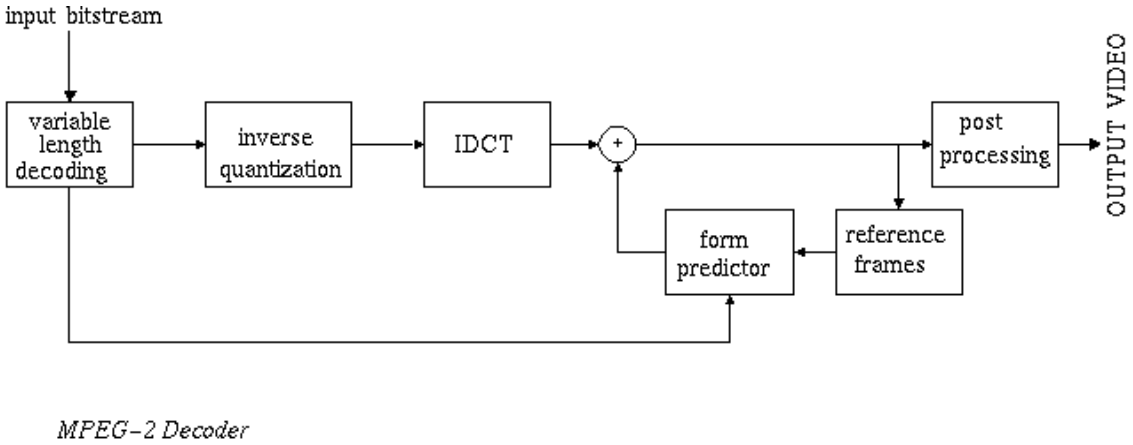


Figure 3.4: A typical MPEG-2 decoder structure

Chapter 4

Implementation Details

In our present work we have designed a parallel IDCT algorithm for the ARTVM processor. In order to verify the algorithm, we also built a simulator for the ARTVM processor. The simulator was used to test the algorithm for its functionality and performance. The parallel IDCT algorithm was tested against IEEE standard 1180-1990 [1] accuracy test for inverse DCT. We have modified an existing MPEG-2 MP@ML video decoder to use the parallel IDCT algorithm. Necessary changes have been made in the video decoder so that instead of performing IDCT on each block of data sequentially, IDCT on all the blocks in a frame is taken in parallel. We have also analyzed the timing performance and calculated the time required for bit-stream parsing, and number of clock cycles required by the ARTVM processor for IDCT. It gives a rough estimate of the time that will be taken if the ARTVM processor is used in the system.

In our simulator we simulate the processor's instructions. The simulated decoder takes considerable amount of time to decode one frame of the video sequence. Since our aim was to perform efficient video decoding, we do not decode the audio data. This data is ignored if it is present in the bit-stream. Further we have limited our work to handling MP@ML MPEG-2 streams only.

4.1 Simulator for ARTVM

The simulator for ARTVM is written in 'C' programming language. It basically consists of three parts - I/O subsystem model, instruction modeler and machine cycles

evaluator. In I/O subsystem model we model the data transfer from the host processor to the ARTVM I/O buffer and from the I/O buffer to the associative memory. Instruction modeler models each instruction of ARTVM as a 'C' function. In machine cycles evaluator the ARTVM processor is modeled as a finite state machine where state transitions represent the cost in machine cycles. It is used for calculating the total number of ARTVM clock cycles taken by an algorithm running on the ARTVM simulator.

4.1.1 I/O subsystem model

There are the following two steps in an I/O operation.

1. Transfer of data into and out of the I/O buffer.
2. Transfer of data from the I/O buffer to the associative array.

In the first step the I/O buffer is filled through an I/O bus by transferring the given number of bits in each clock cycle. Two functions are available in the ARTVM simulator for transfer of data to the I/O buffer. They are named *ioread* and *iowrite*. Function *ioread* is used to read a particular word from the I/O buffer array. Similarly *iowrite* is used to write to a particular word of the I/O buffer.

The second step of date transfer from I/O buffer to the associative array is done through the use of *TAGXCH* instruction in the ARTVM instruction set.

In our application of IDCT implementation, the I/O bus width is decided taking in consideration the timing requirements of MPEG decoding. The total time available to decode and display one frame is *40 ms* for a display requirement of 25 frames per second. Out of this, the time taken by ARTVM for IDCT computation is *4.6 ms*. The average time taken for parsing bitstream on Pentium (233 Mhz) processor is *19 ms* per frame. Therefore about a maximum of *16.4 ms* time is available for the I/O of all the blocks of a frame. For VCD quality video stream (352×288 with 4:2:0 format), maximum number of IDCT blocks in a single frame is 2376. Thus at 33 Mhz bus clock speed the bus width requirement comes out to be 16 bits.

4.1.2 Instruction Modeler

The instruction modeler implements the simulation of ARTVM instructions. All ARTVM instructions are implemented as 'C' functions. These functions effect only the values of members of an external structure called *parameters*. The structure *parameters* is defined in a header file. It's members include the following.

```
A[MEM_SIZE][WORD_LENGTH]
B[MEMSIZE][16]
mask[WORD_LENGTH]
comparand[WORD_LENGTH]
tag[MEM_SIZE]
```

Here, the variable A represents associative memory array, B represents the I/O buffer array and the variables *mask comparand and tag* represent the corresponding registers in the ARTVM processor respectively. The contents of the structure *parameters* change upon simulation of ARTVM instructions. In our simulations, the ARTVM word length is fixed at 128 bits and the memory size is fixed at 160000 (which can store upto a maximum of 2500 8×8 blocks). These parameters can however be changed by editing the header file and recompiling the simulator.

4.1.3 Machine cycles Evaluator

For evaluation of ARTVM cycles for a program, a simplified model is given by the designer of ARTVM [3]. In this model the ARTVM processor is viewed as a simplified *Finite State Machine* (FSM). Each state transition of this FSM is assigned a cost in units of machine cycles. The FSM has only two states: S_0 and S_1. The input alphabet for FSM was selected by grouping the ARTVM instructions into 5 categories as following.

$$I_1 = \text{one of} \begin{cases} letm & d(.) \\ letc & d(.) \\ letmc & d(.) \\ letm & d(.) \quad \text{setc} \\ letc & d(.) \quad \text{setm} \\ setm \\ setc \\ setmc \end{cases}$$

$$I_2 = \text{one of} \begin{cases} setag \\ shiftag(\pm 1) \\ shiftag(\pm \text{b}) \end{cases}$$

$$I_3 = \text{one of} \begin{cases} compare \\ write \\ read \end{cases}$$

$$I_4 = countag$$

$$I_5 = firsel$$

The Finite State Automation used to evaluate machine complexity is given in Fig 4.1

In the original model, TAGXCH instruction is not considered in this grouping of instructions. This model considered ARTVM performance only for associative memory processing and neglected the data transfer from I/O buffer to associative memory. It has however been accounted for in our implementation. We assume that it takes one clock cycles to execute. Therefore we take the state transition cost as one clock cycle when ever it occurs and change the FSM state to S_0 as done for the instructions of groups I_4 and I_5.

To find the total number of ARTVM clock cycles taken by an algorithm a function *clockcount* is introduced in the ARTVM simulator. This function is called at the end of the algorithm to get the total number of clock cycles taken by the algorithm.

In		Out	
Group	State	State	Cost in clockcycles
1	S0	S1	0
1	S1	S1	0.5
2	S0/S1	S0	0.5
3	S0	S0	0.5
3	S1	S0	1
4	S0/S1	S0	12
5	S0/S1	S0	6

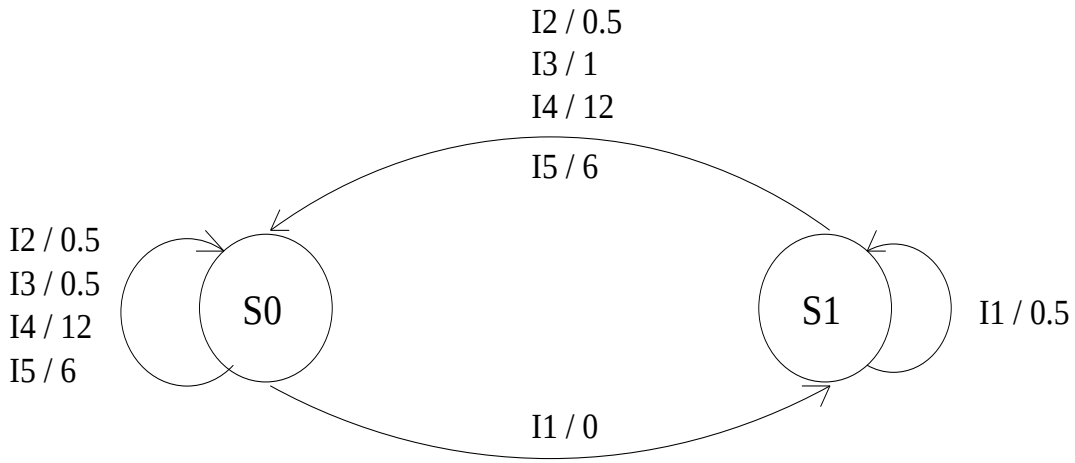


Figure 4.1: Finite Automaton model

4.2 The video decoder with parallel IDCT

We used a video decoder in public domain provided by "MPEG Software Simulation Group" [10]. Necessary changes have been made in the video decoder code so that instead of taking IDCT of the blocks sequentially, the blocks are stored and a full frame IDCT is taken, once for every frame, using the ARTVM simulator. The pseudo-codes below gives the decoder structure.

```
main()
{
        process_options();
        Initialize_Buffers();
        Initialize_Decoder();
        Decode_Bitstream();
}

Decode_Bitstream()
{
        for(; ;)
        {
        /* parse through all the headers till it finds
           Picture_Start_Code or Sequence_End_Code */
        ret = Parse_Headers();
        if(ret == 1)
                Video_sequence();
        else
                return(ret);
        }
}

Video_sequence()
{
        Initialize_Sequence();
        Decode_Picture();
        while(ret=Parse_Headers())
                Decode_Picture();
        Output_Last_Frame_of_Sequence();
        Deinitialize_Sequence();
}
```

```
Decode_Picture()
{
        Update_Picture_Buffers();
        Decode_Picture_data();
        artvm_idct();
        add_Frame();
        frame_reorder();
}
```

The decoder gives timing information to parse the video bit-stream and clock cycles required by ARTVM for IDCT calculation. It also provides total time required for decoding if it is done using a real ARTVM processor. The simulator takes around 30 minutes to decode one frame of video sequence. The decoded frames are stored in TrueVision Targa file format [17].

4.3 Parallel IDCT for ARTVM

Inverse Discrete Cosine Transform (IDCT) is one of the most computationally intensive part of MPEG-2 decoding. It amounts for almost 50 percent of processing requirements of MPEG-2 video stream [13], hence we have concentrated mainly on IDCT computation. We have designed an algorithm for IDCT computation on ARTVM processor which takes IDCT of all the blocks which are stored in the associative memory in parallel.

The 8×8 IDCT is defined as:

$$x_{ij} = \frac{1}{4} \sum_{m=0}^{7} \sum_{n=0}^{7} u(m)u(n)Y_{mn}\, cos\frac{(2i+1)m}{16}\pi\, cos\frac{(2j+1)n}{16}\pi \tag{4.1}$$

$0 \leq i, j \leq 7$.

where

$$u(m) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if m = 0} \\ 1, & \text{otherwise} \end{cases}$$

The time complexity of IDCT as given in equation 4.1 is $O(N^4)$. Many fast algorithms and hardware architectures have been proposed for computing 1- and 2-D DCT/IDCT's ([5, 7, 6, 19]). However, most of them involve high communication complexity. Since nonuniform data communication is very difficult to achieve in ARTVM, none of the fast IDCT algorithms were found suitable for implementation on ARTVM. For ARTVM processor we require an IDCT algorithm with uniform data communication.

We have taken the standard row-column approach for IDCT computation. The 2-D IDCT given in equation 4.1, is broken into 16 1-D IDCT's. In row-column approach we first take 1-D IDCT of all the 8 rows of the 8×8 block, after that we take 1-D IDCT of all the 8 columns of the block. This way the time complexity is reduced to $O(N^3)$ and the data communication is also uniform.

The IDCT computation is made parallel by loading many 8×8 blocks of DCT coefficient into the associative memory, linearly one after the other. The Row IDCT is implemented by taking IDCT of DCT coefficients at i, i+1, i+2..., i+7 indexes in the associative memory. Similarly the column IDCT is implemented by taking IDCT of DCT coefficients at i, i+8, i+16..., i+56 indexes in the associative memory. As the processing is associative, IDCT of all the blocks is taken in parallel. The algorithm takes 149197 ARTVM clock cycles and requires a minimum of 115 bit word length of the associative memory.

4.3.1 Pseudo-code to calculate IDCT

The pseudo-code for performing IDCT for an 8×8 block is given below. In this code the DCT coefficients are stored in a one-dimensional array *data*. The output IDCT coefficients are returned in the same array.

```
artvm_idct(data)
{
     clear_memory();
     load_blocks_in_I/O_buffer();
     transfer_blocks_to_associative_mem();
     for(m = 0;m < 2;m++)
     for(n = 0;n < 8;n++) {
```

```
        load_cosine_terms_in_I/O_buffer();
        transfer_cosine_terms_in_associative_mem();
        if(m==0)
            idctrow();
        else
            idctcol();
    }
        transfer_result_in_I/O_buffer();
        store_result_back();
}
```

The actual ARTVM code for IDCT is given in appendix A.2.

Chapter 5

Results and Conclusions

5.1 Test Setup

The co-hardware/software MPEG-2 decoder simulator is developed on linux operating system in C programming language. It has been tested on many linux versions like Linux 2.0.36, Linux 2.2.9-19mdk, and Linux 2.2.14-15mdk. The decoder was tested on 3 different systems mainly to test it's robustness and the efficiency of simulator. The table 5.1 shows the configuration of the systems.

Resource	System 1	System 2	System 3
Processor	Pentium-II (Klamath)	Pentium-II (i686)	AMD (K6-3)
CPU Clock Speed	233 MHz	350 MHz	450 Mhz
Memory	32 MB RAM	64 MB RAM	64 MB RAM

Table 5.1: Test System configuration

MEM_SIZE of ARTVM was chosen to be 160000 (for a maximum of 2500 8×8 blocks). The simulator performs the timing analysis and provides output such as number of clock cycles required by the ARTVM, time taken to parse the bitstream and the total time required to decode each frame with real ARTVM processor.

5.2 Results

Table 5.2 gives the output of MP@ML MPEG-2 video decoder simulator. The time shown is the maximum time required to decode a frame of the video sequence. It can be seen that the decoder meets real time decoding constraints very easily.

Sample no.	Frame Size	Simulator Output: Maximum time for decoding one frame		
		P-II (Klamath)	P-II (i686)	AMD K6-3
1.	352 × 288	34 ms	28 ms	25 ms
2.	256 × 192	26 ms	21 ms	17 ms
3.	176 × 120	14 ms	11 ms	10 ms

Table 5.2: Simulator Results : Maximum time required to decode one frame

The parallel IDCT algorithm designed for ARTVM was tested against IEEE standard 1180-1990 [1] accuracy test for inverse DCT. The test was run several times, it was found that usually a few (around 12) IDCT values out of the 64 values differed from the reference values by 1. This conforms to the IEEE 1180-1990 standard for worst peak error. The other tests of IEEE standard 1180-1990 accuracy test, like the peak mean square error (pmse) and the mean square error (mse) were not tested for because it required testing for very large numbers of iterations which was not possible with one iteration taking 30 mins of time.

Sample no.	Frame Size	ARTVM MEM_SIZE required	Time required by simulator for IDCT		
			P-II (Klamath)	P-II (i686)	AMD K6-3
1.	352 × 288	115200	2519 s	1721 s	1260 s
2.	256 × 192	76800	1667 s	1123 s	829 s
3.	176 × 120	38400	789 s	524 s	404 s

Table 5.3: Results : Time taken by ARTVM for IDCT calculations

Number of ARTVM clock cycles required by the IDCT algorithm is 149197. This is independent of the number of blocks for which IDCT is taken in parallel. The algorithm can be made more accurate with more number of clock cycles.

The simulator takes considerable amount of time to decode even one frame of video sequence. Table 5.3 below gives simulation time required for various MPEG files over different processors.

5.3 Conclusions

Associative array processing has been found to be a good alternative for IDCT calculations in MPEG decoding. The IDCT algorithm takes only 4.5 ms on the ARTVM processor to compute IDCT of all the blocks loaded in the associative memory in parallel. The IDCT algorithm conformed to the IEEE standard 1180-1990 accuracy test for worst peak errors. Therefore the decoded picture quality is extremely good. The proposed MPEG-2 MP@ML decoder model meets the real time decoding requirements very easily without overloading the system or dropping the frames.

5.4 Limitations

- Audio - MPEG-2 bit-streams usually also contain audio data. Since we were simulating the video decoding process, audio data is discarded.
- Simulation speed - In spite of considerable efforts the ARTVM simulator designed was not very fast. This was a big bottleneck in testing the MPEG decoder simulator for large bit-streams.
- User features - None of the usual user features like random access, fast forward, fast-rewind or other trick modes were implemented in the MPEG decoder.

5.5 Future Work

- Other Applications - We have tested ARTVM processors viability for MPEG decoding process. To make the full use of ARTVM we should make use of its parallel processing capabilities in other applications also.
- MPEG Video - In our present design we have used the ARTVM for only IDCT

calculations. There are a few other parts of video decoding like inverse quantization and variable length decoding which might also be possible through ARTVM. We should look into these possibilities also in future. Further we can also try to parse the video bitstream for more than a single frames data to make the full use of associative memory.

- MPEG Audio - Audio decoding should also be combined with the present decoder design. We should look into audio compression algorithms to see if there is any possibility of making audio decoding parallel so that it can also be done through ARTVM.

Appendix A

A.1 List of Acronyms

ARTVM	Associative Real Time Vision Machine
CBP	Coded Block Pattern
CD	Compact Disk
DCT	Discrete Cosine Transform
DSM	Digital Storage Media
DSP	Digital Signal Processing
FSM	Finite State Machine
FA	Finite Automation
GOP	Group Of Pictures
HDTV	High Definition Television
HP	High Profile
IDCT	Inverse Discrete Cosine Transform
IEC	International Electrotechnical Commission
ISM	Interactive Storage Media
ISO	International Organization for Standardization
ITU	International Telecommunications Union
JPEG	Joint Photographic Experts Group
Kbps	Kilobits per second
LBC	Low Bitrate Communication
LL	Low Level
Mbps	Megabits per second
MB	Macroblock

MC	Motion Compensation
ME	Motion Estimation
ML	Main Level
MP	Main Profile
MPEG	Moving Picture Experts Group
NTSC	National Television System Committee
PE	Processing Element
RAM	Random Access Memory
RLE	Run-Length Encoding
SIF	Source Input Format
SNR	Signal-to-Noise Ratio
VLC	Veriable Length Coding
VLSI	Very Large Scale Integration
WG	Working Group

A.2 Parallel IDCT code for the ARTVM processor simulator

```
/*****************************************************************
 *                                                               *
 *                    Implementation of parallel                 *
 *               Inverse Discrete Cosine Transform(IDCT)         *
 *                                                               *
 *                        for ARTVM processor                    *
 *                                                               *
 *                   By : Samarpit Bhatia (9811118)              *
 *                                                               *
 *               Under the guidance of Dr. Rajat Moona           *
 *                                                               *
 *                        January, 2000                          *
 *****************************************************************/

#include "asslib.h"

void artvm_dct (short *data)
{
 static int COS_TERM[8][8] = {
{ 23170, 32138, 30274, 27246, 23170, 18205, 12540, 6393},
{ 23170, 27246, 12540, -6393,-23170,-32138,-30274,-18205},
{ 23170, 18205,-12540,-32138,-23170, 6393, 30274, 27246},
{ 23170, 6393, -30274,-18205, 23170, 27246,-12540,-32138},
{ 23170,-6393,-30274, 18205, 23170,-27246,-12540, 32138},
{ 23170,-18205,-12540, 32138,-23170,-6393, 30274,-27246},
{ 23170,-27246, 12540, 6393,-23170, 32138,-30274, 18205},
{ 23170,-32138, 30274,-27246, 23170,-18205, 12540,-6393}
};
 static int shift[8][3] = {
```

```
{ -1, -2, -4 },
{  1, -2, -4 },
{ -1,  2, -4 },
{  1,  2, -4 },
{ -1, -2,  4 },
{  1, -2,  4 },
{ -1,  2,  4 },
{  1,  2,  4 }
};

 int DCTi=0,COSi=32,SUMi=64,RESULTi=96;
 int row=112,col=113,tempbit=114,carrybit=115;
 int temp,m=0,n=0,i,j,k,l;

/*clear all of the contents of memory*/

initial();
 for (i = 0;i < MEM_SIZE;i++)
{
if ((i%64)==0)
iowrite(i,3);
else if ((i%8)==0)
iowrite(i,1);
else if ((i%64)<8)
iowrite(i,2);
}
 tagxch();letmc(d(row));a_write();
 tagxch();letmc(d(col));a_write();

/* Transfer all the idct blocks in IO buffer */

for (i = 0;i < MEM_SIZE;i++)
```

```
{
iowrite(i,*(data+i));
}

 for (i = DCTi;i < DCTi+16;i++)
{
tagxch();letmc(d(i));a_write();
}

 while(m<2)
{
letmc(d(DCTi+15));setag();compare();
letmc(dseq(DCTi+16,DCTi+31));a_write();
letm(d(DCTi+15));setag();compare();
letm(dseq(DCTi+16,DCTi+31));
letc();a_write();
n = 0;
while(n<8)
{
if(m == 0)
{
k = row;
l = 1;
}
else
{
k = col;
l = 8;
}

letmc(d(k));setag();compare();
```

```
for(j = 0;j < 8;j++)
{
i = (j+n)%8;
letc(dvar(COSi,COSi+31,COS_TERM[n][i]));
letm(dseq(COSi,COSi+31));a_write();
shiftag(l);
}

/* clear the assoc. mem from SUMi for multiplication */

letm(dseq(SUMi,(SUMi+31)),d(carrybit),d(tempbit));setag();a_write();

/*  Multiply the DCT coeff with it's COS TERM  */

for(i = DCTi;i < DCTi+32;i++)
{
k = i - DCTi;
l = COSi;
letm(d(carrybit));
letc();setag();a_write();
for(j = SUMi+k;j < SUMi+32;j++)
{
letmc(d(i));setag();compare();
letm(d(j),d(carrybit),d(l));
letc(d(carrybit));compare();
letc(d(j));a_write();
letmc(d(i));setag();compare();
letm(d(j),d(carrybit),d(l));
letc(d(carrybit),d(j));compare();
letc(d(carrybit));a_write();
letmc(d(i));setag();compare();
letm(d(j),d(carrybit),d(l));
```

```
letc(d(j),d(l));compare();
letc(d(carrybit),d(l));a_write();
letmc(d(i));setag();compare();
letm(d(j),d(carrybit),d(l));
letc(d(l));compare();
letc(d(j),d(l));a_write();
l = l+1;
}
}
if (m==0)
k = 1;
else
k = 8;

/*Move the multiplication results to other pixels for summation */

for(i = 0;i < 3;i++)
{
for(j = SUMi;j < SUMi+32;j++)
{
letmc(d(j));setag();compare();
shiftag(shift[n][i]*k);
letmc(d(tempbit),d(j-32));a_write();
letm(d(tempbit));
letc();setag();compare();
letm(d(j-32));a_write();
letm(d(tempbit));setag();a_write();
}
letm(d(carrybit));
letc();setag();a_write();

/* summation */
```

```
for(j = SUMi;j < SUMi+32;j++)
{
letm(d(j),d(carrybit),d(j-32));
letc(d(carrybit));setag();compare();
letc(d(j));a_write();
letc(d(j),d(carrybit));setag();compare();
letc(d(carrybit));a_write();
letc(d(j),d(j-32));setag();compare();
letc(d(carrybit),d(j-32));a_write();
letc(d(j-32));setag();compare();
letc(d(j-32),d(j));a_write();
}
}
if (m==0)
k = row;
else
k = col;

/* This will take care of truncation errors */

letm(dseq(COSi+15,COSi+31),d(carrybit));
i = COSi+15;
letc(d(i));setag();a_write();
for(j = SUMi+15;j < SUMi+32;j++)
{
letm(d(j),d(carrybit),d(i));
letc(d(carrybit));setag();compare();
letc(d(j));a_write();
letc(d(j),d(carrybit));setag();compare();
letc(d(carrybit));a_write();
letc(d(j),d(i));setag();compare();
```

```
letc(d(carrybit),d(i));a_write();
letc(d(i));setag();compare();
letc(d(i++),d(j));a_write();
}

j = RESULTi;
for(i = SUMi+16;i < SUMi+32;i++)
{
letmc(d(k));setag();compare();
letmc(d(i));compare();
letmc(d(j));a_write();
j = j+1;
}
letmc(d(k));setag();compare();
if (m==0)
shiftag(1);
else
shiftag(8);
a_write();
letmc(d(tempbit));a_write();
letc();setag();compare();
letm(d(k));a_write();
letm(d(tempbit));setag();a_write();
n = n+1;
letm(dseq(COSi,RESULTi-1));setag();a_write();
}
letm(dseq(DCTi,RESULTi-1));setag();a_write();
j = DCTi;


/* Row IDCT has been done now move the coefficients back to index DCTi */
/* For column IDCT calculations */
```

```
for(i = RESULTi;i < RESULTi+16;i++)
{
letmc(d(i));setag();compare();
letmc(d(j));a_write();
j = j+1;
}

letm(dseq(RESULTi,RESULTi+15));setag();a_write();
m = m+1;
}

/* transfer the results from associative memory to the I/O buffer */

 for(i = DCTi;i < DCTi+16;i++)
{
letmc(d(i));setag();compare();
tagxch();
}

/* print the total number of clock cycles required  */

 clockcount();

/* transfer the results back to the array 'data'  */

 for(i = 0;i < MEM_SIZE;i++)
{
*(data+i) = ioread(i);
}
}
```

Bibliography

[1] IEEE Test for IDCT accuracy
http://www.mpeg1.de/imaa/util/unix/ieee1180/

[2] N.Ahmed, T. Natarajan and K. R. Rao, "Discrete cosine transform", IEEE Trans Commun., vol. COM-23, pp 90-93, Jan. 1974

[3] Avidan J. Akerib, Phd Thesis, "Associative Real Time Vision Machine", Department of Applied Mathematics and Computer Science Weizmann Institute of Science, March 1992

[4] Avidan J. Akerib, Phd Thesis, "Associative Real Time Vision Machine", Department of Applied Mathematics and Computer Science Weizmann Institute of Science, pp 22, March 1992

[5] A. Madisetti & A. N. Willson, "A 100 MHz 2-D 8×8 DCT/IDCT Processor for HDTV Applications", IEEE Trans on Circuits and Systems for Video Tech, Vol 5, No 2, April 1995

[6] Nam Ik Cho and Sang Uk Lee, "Fast Algorithm and Implementation of 2-D DCT", IEEE Trans. on CAS, vol 38, pp 297-305, Mar. 1991

[7] Darren Slawecki and Weiping Li, "DCT/IDCT Processor Design for High Data Rate Image Coding", IEEE Trans on Circuits and Systems for Video Tech, Vol 2, No 2, June 1992

[8] MPEG-1 Decoder Boards:
http://www.visiblelight.com/mpeg/products/p_play3.htm

[9] MPEG-1 Decoder Boards:
http://www.visiblelight.com/mpeg/products/p_com.htm

[10] MPEG Software Simulation Group
ftp://ftp.cstv.to.cnr.it/pub/MPEG2/conformance-bitstreams/video/verifier/

[11] B. G. Haskell, A. Puri, A. N. Nerravali, "DIGITAL VIDEO AN INTRODUCTION TO MPEG-2", in Digital Multimedia Standards Series, Chapman & Hall, pp 116, 1997

[12] MPEG Homepage
http://www.cselt.it/mpeg/

[13] I. Kuroda, T. Nishitani, "Multimedia Processors", Proceedings of the IEEE, Vol 86, No. 6, Jun. 1998.

[14] Software MPEG-1 Decoders:
http://www.visiblelight.com/mpeg/products/p_play4.htm

[15] Public domain MPEG decoders
http://www.mpeg.org/MPEG/video.html

[16] J.L.Mitchell, W.B.Pennebaker, C.E.Fogg and D.J.LeGall, "MPEG Video Compression Standard", in Digital Multimedia Standards Series,Chapman & Hall, pp 396, New York, 1997.

[17] TGA File Format Spec
ftp://ftp.truevision.com/pub/TGA.File.Format.Spec/PC.Version/

[18] MPEG Video Group Homepage
http://bs.hhi.de/mpeg-video/

[19] Chin-Liang Wang and Chang-Yu Chen, "High-Throughput VLSI Architectures for the 1-D and 2-D Discrete Cosine Transform", IEEE Trans on Circuits and Systems for Video Tech, Vol 5, No 1, Feb 1995

Index